Multi-scale Bio-Modeling and Visualization

Chandrajit Bajaj

October 25, 2010

Contents

1 Computational Biology: From Structure to Function **7**
1.1 Representations of Data . . . 8
1.1.1 Images . . . 8
1.1.2 Geometry . . . 8
1.1.3 Meshes . . . 8

2 Image and Signal Processing **9**
2.1 Pixel/Voxel Operations . . . 9
2.1.1 Filtering . . . 9
2.1.2 Contrast Enhancement . . . 23
2.1.3 Classification/Segmentation . . . 31
2.1.4 Symmetry Detection . . . 40
2.1.5 Compression . . . 44
2.2 Alignment . . . 77
2.3 Projections . . . 82
2.3.1 Forward Radon Transform . . . 82
2.3.2 Fourier Transform . . . 82
2.4 Reconstruction . . . 82
2.4.1 Inverse Radon Transform . . . 82
2.4.2 Fourier Based Tomography . . . 88
2.5 Bio-Imaging . . . 108
2.5.1 molecules — electron microscopy . . . 108
2.5.2 cells with organelles —- serial section electron microscopy . . . 108
2.5.3 tissues cells + extra-cellular space/filling —- confocal microscopy/micro-CT . . . 108
2.5.4 organs — CT/MRI . . . 108
2.5.5 systems — live cell imaging, phase-contrast and tagged MRI . . . 123
2.6 Bibliographic Notes . . . 123
2.6.1 Pixel/Voxel Operations . . . 123
2.6.2 Alignment . . . 128
2.6.3 Projections . . . 128
2.6.4 Reconstruction . . . 128
2.6.5 Bio-Imaging . . . 129

3 Elucidating Geometry from Images **143**
3.1 Digital Topology Filtering and Geometric Sampling . . . 143
3.2 Segmentation . . . 143
3.3 Skeletonization . . . 143
3.4 Biological Structure Elucidation . . . 174
3.4.1 molecules — ultrastructure elucidation of viruses, ribosomes from EM . . . 174
3.4.2 cells with organelles —- reconstruction of neuronal circuits from ssTEM . . . 174

3.4.3 tissues cells + extra-cellular space/filling —- reconstruction of T-tubules from, and bone tissue from micro-CT . . . 174
3.4.4 organs — heart reconstruction from CT/ brain reconstruction from MRI . . . 174
3.4.5 systems — developmental cell reconstruction, beating heart parameters from phase-contrast and tagged MRI . . . 174
3.5 Bibliographic Notes . . . 174
3.5.1 Skeletonization . . . 174

4 Geometry of Finite Elements **179**
4.1 Reconstruction . . . 179
4.2 Filtering . . . 212
4.3 Segmentation . . . 221
4.4 Skeletonization . . . 221
4.5 Meshed Models . . . 244
4.5.1 Branching Structures . . . 292
4.5.2 Free-form Surface Models . . . 292
4.5.3 Volumetric Models . . . 292
4.6 Biological Structure Elucidation . . . 292
4.6.1 molecules — meshed models of molecular surfaces and volumes . . . 292
4.6.2 cells with organelles —- meshed models of branching structures, forest, weight functions . . . 292
4.6.3 tissues cells + extra-cellular space/filling —- volumetric meshes of T-tubules, bone tissue . . . 292
4.6.4 organs — surface and volumetric meshed models of heart, brain . . . 292
4.6.5 systems — meshed time dependent geometry . . . 292
4.7 Bibliographic Notes . . . 292
4.7.1 Reconstruction . . . 292
4.7.2 Filtering . . . 292
4.7.3 Segmentation . . . 292
4.7.4 Skeletonization . . . 292
4.7.5 Meshed Models . . . 292
4.7.6 Time Dependent Geometry . . . 294

5 Appendix **305**
5.1 Mesh Decompositions . . . 305
5.2 Interpolants . . . 305
5.3 Topology . . . 305
5.4 Analysis . . . 305
5.5 Statistics . . . 305

6 Appendix **307**
6.1 Computer Graphics . . . 307
6.2 User Interfaces . . . 307
6.3 GPU . . . 307

Preface

A preview into the ever increasing breadth and depth of the algorithms and data structures that form the computational infrastructure to visualize and quantify multi-scale biological structure.

Techniques: filtering, contrast enhancement, classification, alignment, symmetry detection, segmentation, subunit identification, boundary segmentation, variational methods, skeletonization, meshing, multi-dimensional transfer functions, impostor rendering, visible surface and volume rendering, function on surface, Contour Trees, Spectrum, animation, stereo

Applications: (a) structural elucidation—x-ray diffraction-pdb-proteins, rna, –complementary space single particle cryo-em macromol, tomographic-em — mol-cell, ss-tem, comparative analysis

(b) surface, volume properties analysis and visualization

(c) mol. dynamics, cell electrophysiology, tissue – calibration

(d) interpretive animation

Thanks to members of ccv

Chapter 1

Computational Biology: From Structure to Function

Content of Chapter

biomedical visualization (VolRover) ....visual explanations....of change in growth, disease, flexibility, destructive path of an x-ray, heat, bullet,

purpose—> students, researchers, practitioners

students —> learn biomedicine researchers –> visually verify their models, express the results of their finding practitioners –> understand details of the methods they use in constructing - a visual for TV or graphics illustration or computer game

Biological Apps (Drivers)

molecules — targets + drugs.....assembly ...immunology

cells with organelles —- cancer —normal vs disease

tissues cells + extra-cellular space/filling —- cancer — normal vs disease

organs — normal vs disease or inflicted, changes in growth

systems — changes in growth.....

Index to content of chapters:

chap 1 introduces computational algorithms and data structures for structural elucidation, analysis and visualization of biological structure from imaging and simulated dynamics, properties

chap 2 describes representations of images. 3D maps, geometries and finite element meshes, polynomials, piecewise polynomials, algebraic functions, splines over triangulations differential geometry,

chap 3 image processsing : filtering, contrast enhancement, classification, alignment, reconstruction, symmetry detection, segmentation, skeletonization, subunit identification, matching, compression

chap 4 image to geometry extraction: boundary segmentation, variational methods, surfaces, matching

chap 5 describes geometry processing, extracting geometric features and finite element meshing: triangular, tetrahedral, quadrilateral and hexahedral meshes, A-patch and B-Spline models,

chap 6 describes simulation and analysis techniques, including numerical, finite difference, boundary element, finite element, finite volume analysis, and numerical linear system solvers

chap 7 describes quantitative (topological, geometric and statistical) analysis and visualization of imaging and simulation bio-data: contour trees, REEB graphs, multi-dimensional transfer functions, visible surface and volume rendering, function on surface, VisPortal

chap 8 describes analysis and visualization of time dependent imaging / simulations

chap 9 high performance multi-core computing...use of GPU's , CUDA and CILK++ for bio-modeling and analysis

chap 10 concludes with index to appendix, and other sources of comp. bio. imaging/modeling/analysis information. Also online resources and software resources.

Appendix 1 is devoted to some mathematical preliminaries and prerequisites

Appendix 2 is devoted to background and prerequisites in computer graphics

1.1 Representations of Data

This section describes representations of images. 3D maps, geometries and finite element meshes, polynomials, piecewise polynomials, algebraic functions, splines over triangulations differential geometry,

1.1.1 Images

2D greyscale and color
3D greyscale and color

1.1.2 Geometry

boundary, solids

1.1.3 Meshes

triangulations, hexahedral decompositions

Bibliographic Notes

Bibliography

Exercises

Chapter 2

Image and Signal Processing

chap 2 image to image processsing : filtering, contrast enhancement, classification, alignment, reconstruction, symmetry detection, segmentation, skeletonization, subunit identification, matching, compression, image synthesis

2.1 Pixel/Voxel Operations

operations on a single image returning a processed image

2.1.1 Filtering

The following algorithm was taken from [167] (jbwbc2003:jsb).

Introduction

Many image filters have been developed to suppress the noise, such as low pass, Wavelet transforms, median filters, etc. (Gonzalez and Woods, 2002). A fundamentally important property for any filter to possess, but often lacking, is the ability to maximally suppress noise solely without affecting much of the true signal. The bilateral filter is designed to achieve this goal by striking a fine balance between minimizing noise and losing signal. Bilateral filter is a relatively new filter proposed originally to denoise 2-D photographic images and has been shown to be very effective in achieving this goal (Tomasi and Manduchi, 1998). In this study, we continue to explore filtering by investigating the effectiveness of the bilateral filter in denoising biological electron microscopy images and 3-D reconstructions.

The kernel of a bilateral filter is composed of an inner product of two low pass filters in real space (Tomasi and Manduchi, 1998). The first of these filters is a normal low pass filter, which basically averages the neighboring pixel intensity values with decreasing weights for pixels at larger distances. The second one is also a type of low pass filter, except the weights for the neighboring pixels are derived from the pixel intensity value differences to the center pixel intensity value instead of geometric distances. Therefore, the larger the pixel value difference, the smaller the pixels contribution during averaging. This second filter component distinguishes the bilateral filter and results in a significant advantage over other types of filters. In practice, both filter components in bilateral filter are realized using Gaussian functions:

$$I_{\text{out}}(\vec{r}) = \int e^{-\frac{|\vec{\xi}-\vec{r}|^2}{2\sigma_1^2}} e^{-\frac{|I_{\text{in}}(\vec{\xi})-I_{\text{in}}(\vec{r})|^2}{2\sigma_2^2}} I_{\text{in}}(\vec{\xi}) d\vec{\xi}$$

where $I_{\text{in}}(\vec{r})$ is the input noisy image, $I_{\text{out}}(\vec{r})$ is the output de-noised image, $e^{-\frac{|\vec{\xi}-\vec{r}|^2}{2\sigma_1^2}} e^{-\frac{|I_{\text{in}}(\vec{\xi})-I_{\text{in}}(\vec{r})|^2}{2\sigma_2^2}}$ is the bilateral filter kernel, where σ_1 and σ_2 control the width of the two Gaussian functions respectively. σ_1 controls the extent of the normal spatial low pass filtering where a larger σ_1 causes severe smoothing. σ_2 controls the discrimination power between true features and noises with the assumption that larger pixel intensity value variations are mainly from true features and smaller pixel intensity

value variations are contributed by noise. In the above equation, the normalization factor is ignored. To keep the local mean, the weights $e^{-\frac{|\vec{\xi}-\vec{r}|^2}{2\sigma_1^2}} e^{-\frac{|I_{\text{in}}(\vec{\xi})-I_{\text{in}}(\vec{r})|^2}{2\sigma_2^2}}$ should be normalized by the sum of all the weights $\int e^{-\frac{|\vec{\xi}-\vec{r}|^2}{2\sigma_1^2}} e^{-\frac{|I_{\text{in}}(\vec{\xi})-I_{\text{in}}(\vec{r})|^2}{2\sigma_2^2}} d\vec{\xi}$.

The advantages of the bilateral filter can be illustrated using the example of "smoothing without edge blurring" shown in Figure 2.1. If the noisy function is a step function (Fig. 2.1a) with noise (Fig. 2.1b-d), a normal low pass filter will suppress noise but blur the edges, with more severe blurring caused by wider Gaussian half widths σ_1 (Fig. 2.1e σ_2=∞ column). In contrast, the bilateral filter is capable of smoothing out the noise in the plateau regions while maintaining sharp edges (Fig. 2.1e σ_2=0.3 column). In the plateau regions, the pixel intensity value differences from noises are much smaller than the appropriately chosen σ_2, thus the second low pass component in bilateral filter will be close to 1.0. This results in making the bilateral filter effectively equivalent to normal low pass filters, capable of smoothing out the noise. However, when the pixels at the edge are filtered, the intensity value differences of neighboring pixels in the plateau regions are significantly larger than σ_2, and thus cause the second component in bilateral filter to approach 0.0. This essentially cancels the contributions for those pixels and minimizes the blurring effects found in low pass filters. When σ_2 is chosen inappropriately, the bilateral filtering is similar to normal low pass filter for too large σ_2 (Fig. 2.1e σ_2=1.0 column), or no filtering for too small σ_2 (Fig. 2.1e σ_2=0.01 column). When σ_2 is infinite, the bilateral filter is reduced to a normal low pass filter (Fig. 2.1e σ_2=∞ column). It is worthy noting that bilateral filter is not very effective against "singular" type noises which have a single or a few noise pixels with very different values than the rest pixels. Median filter will be more effective for this type of noises (Gonzalez and Woods, 2002).

Figures

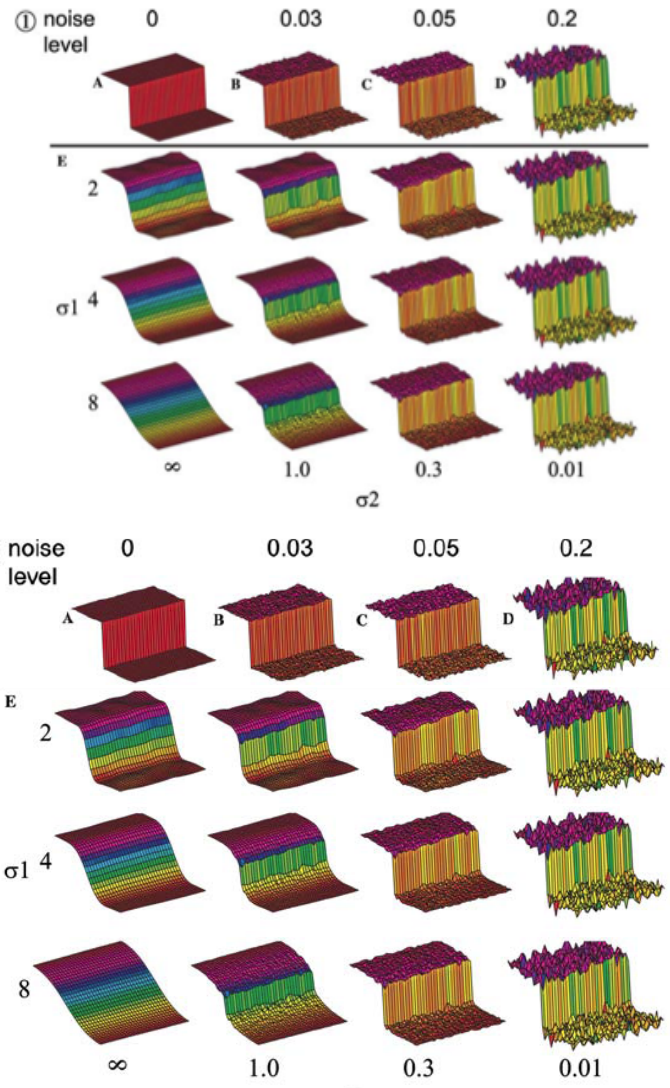


Figure 2.1: Illustrations of bilateral filter. (a) the step function with 0 for one half side and 1 for the other half side. (b-d) the noise-free step function in (a) was added varying levels of random noises of Gaussian distribution with the half-width at 0.03 (b) 0.05 (c) and 0.2 (d) respectively. (e) results of bilateral filtering with combinations of varying σ_1 (2, 4, 8 pixels half width) and σ_2 (∞, 1.0, 0.3, 0.01 half width) on the noisy step function in (d).

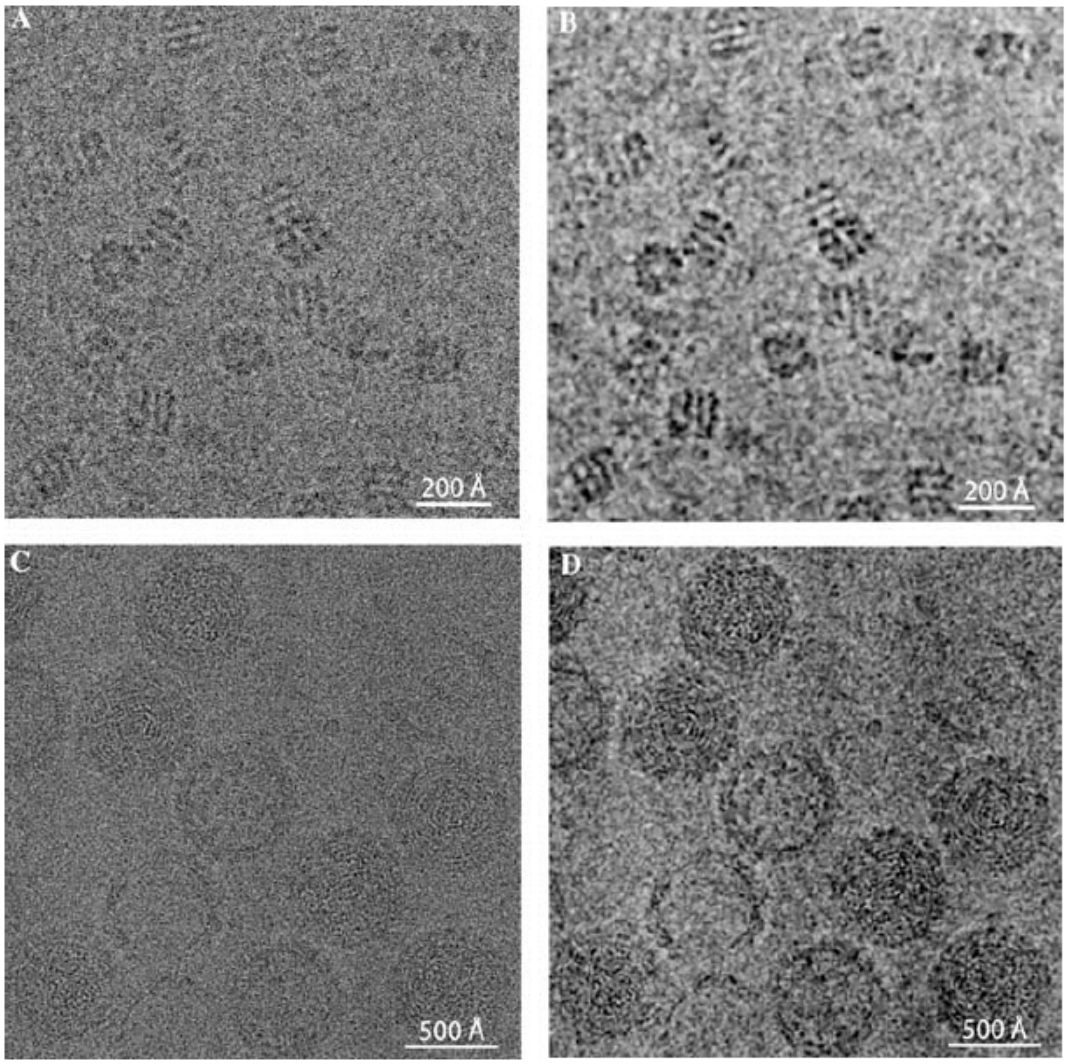


Figure 2.2: Bilateral filtering of raw micrograph. (a) the original micrograph of GroEL imaged on a JEOL4000 microscope at 400 kV and 50,000 magnification. The density variance of the whole micrograph is 0.22. (b) the bilateral filtered image of (a) with σ_1= 2 pixels and σ_2=1.1. (c) the original micrograph of RDV imaged on a JEOL4000 microscope at 400 kV and 50,000 magnification. (d) the bilateral filtered image of (c) with σ_1= 3 pixels and σ_2=5.

The following algorithm was taken from [34] (BajajWu_noyear) - Anisotropic Diffusion.

Noise is always present in 3D imaging data; how to smooth noise in large 3D data has been an important research topic in the image processing field. In this paper, we present our volumetric anisotropic diffusion model and its parallel implementation on a PC cluster. Our model extends work in [38] to construct a 3D anisotropic diffusion tensor for filtering 3D volumetric image. To improve the accuracy of pre-filtering, we replace isotropic diffusion pre-filtering with bilateral pre-filtering. To assess our method, efficient solver for the resulting linear systems has been implemented under MPI [218] on a PC cluster.

Volumetric Anisotropic Diffusion

In [38], Bajaj and Xu propose a method to smooth the function defined on the surface using anisotropic geometric diffusion. Their method is described by

$$\partial_t \varphi - \|\nabla\varphi\| \mathrm{div} D^\sigma \nabla\varphi) = C(\varphi - \varphi_0) \tag{1.1}$$

We can extend their method to smooth the function defined on 3D volume, namely, 3D volumetric data. Actually, the 3D volumetric image is considered as the function defined on the hyper-surface in 3-manifold. The initial noisy function can be smoothed by the volumetric anisotropic diffusion, and the 3D image function is independent of hyper-surface geometry. Because curvature is a measurement of how much the curve bends in the direction of curvature, we can associate the feature with large curvature. To preserve the salient feature, our goal is to inhibit the diffusivity along the dominant principal direction of curvature. Based on the 3D curvature derived from Riemannian theory [See Appendix], curvatures and principal directions of curvature can be estimated, and the anisotropic curvature diffusion tensor can be constructed to smooth the 3D image. We can incorporate anisotropic diffusion based on a proper anisotropic diffusion tensor D^ε, which will decrease the diffusion along the dominant principal direction of curvature. The diffusivities along the two subdominant principal directions of curvature are determined by their corresponding curvatures. We can define the anisotropic diffusion tensor D^ε with respect to the orthogonal basis $\upsilon^1, \upsilon^2, \upsilon^3$ of tangent space by

$$D^\varepsilon = [\upsilon^1, \upsilon^2, \upsilon^3] \begin{bmatrix} G(\kappa_1) & 0 & 0 \\ 0 & G(\kappa_2) & 0 \\ 0 & 0 & G(\kappa_3) \end{bmatrix} [\upsilon^1, \upsilon^2, \upsilon^3]^T \tag{1.2}$$

In (1.2), G(.) is a function to control the diffusivity along each principal direction of curvature, we define function G(.) as follows:

$$G(x) = \frac{1}{1 + x^2/\lambda^2} \tag{1.3}$$

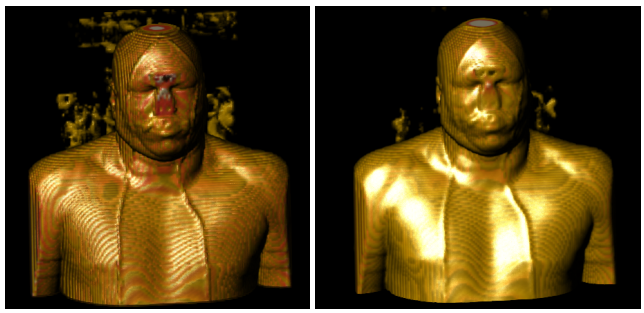
Here parameter λ determines the feature based on the curvature. If the curvature is large, the diffusivity will be inhibited along the principal direction of such curvature. So it is vital to choose a suitable λ for different 3D imaging data, we show how λ plays a role in smoothing in Figure 2.4.

Bilateral Pre-filtering

To avoid inexact estimation of curvature from noisy data, we need to implement the anisotropic diffusion process in two steps [85]. The first step is to apply linear diffusion on original data.

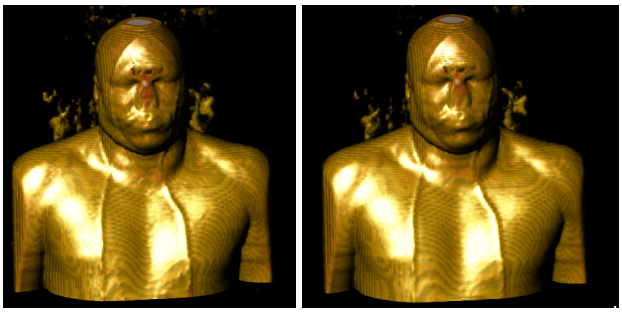
The purpose of this step is to generate more accurate computation of curvature for the anisotropic diffusion tensor. But linear isotropic diffusion cannot provide very precise estimation for edge or curvature information. As You in [323] points out, pre-filtering by linear isotropic diffusion is obviously against the spirit of anisotropic diffusion.

The nature of bilateral filtering resembles that of anisotropic diffusion [50]. In [295] Tomasi and Mandichi first introduce bilateral filtering as a nonlinear filter that combines domain and range filtering. In addition, bilateral filtering does not involve the solution of partial differential equation and can be implemented in a single iteration. In [50], Barash presents that both bilateral filtering and anisotropic diffusion are related to adaptive smoothing. Since bilateral filtering can remove noise as Gaussian filtering can do, bilateral filtering has the advantage of preserving edge or curvature information, which is better for

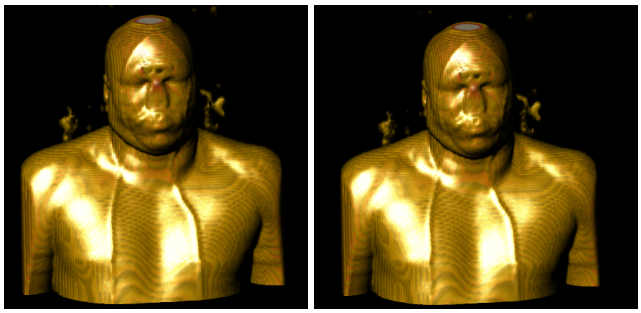


(a) (b)

Figure 2.3: Volume rending of visible human being. (a) Before filtering. (b) After Isotropic filtering (timestep 0.00002).

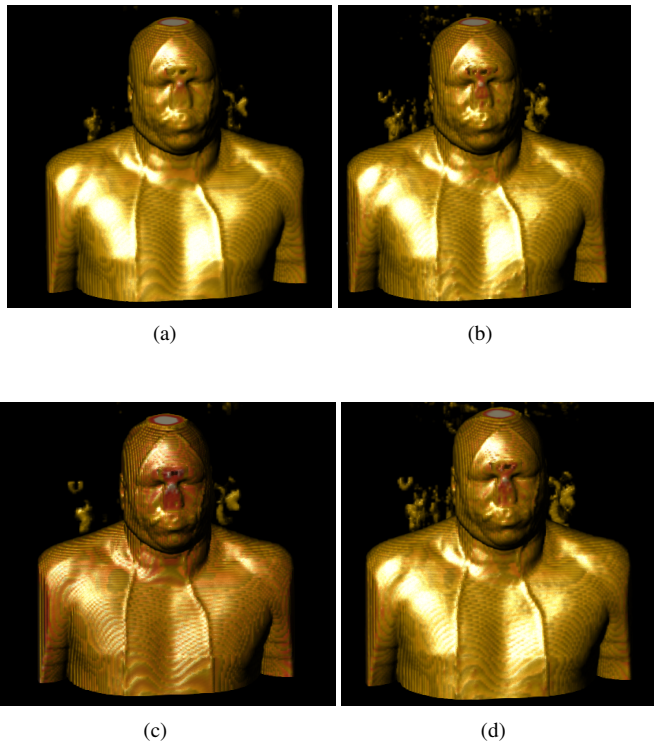


(a) (b)



(c) (d)

Figure 2.4: Volume rending of visible human being after Anisotropic Diffusion Filtering (timestep=0.00002). (a) Lamda=0.3. (b) Lamda=3.0. (c) Lamda=300.0. (d) Lamda=30000.0.



(a) (b)

(c) (d)

Figure 2.5: Volume rendering of visible human being. (a) After Isotropic Pre-filtering. (b) Isotropic Pre-filtering + Anisotropic Diffusion. (c) Bilateral Pre-filtering. (d) Bilateral Pre-filtering + Anisotropic Diffusion.

constructing the robust anisotropic diffusion tensor. Here we replace Gaussian pre-filtering with bilateral pre-filtering. We show different filtering results in Fig. 2.5.

The following was taken from [35] (aniso02).

Pre-filtering with bilateral prefiltering. To assess our method, efficient solver for the resulting linear systems has been implemented under MPI[218] on PC cluster.

Main contributions of this paper are: (1) Our 3D anisotropic diffusion tensor allows smoothing along normal direction and two curvature directions of local structure, and enhances 1D features (curves) and 2D features (surfaces) also. (2)Bilateral prefiltering is coupled with anisotropic diffusion process to improve filtering performance. (3) Efficient parallel implementation is done on PC cluster.

Volumetric Anisotropic Diffusion

In [324], You etc. present that anisotropic diffusion is a process that dissipates energy with time, and its behavior depends on the shape of the energy surfaces. They consider the following energy functional defined on the space of smooth images:

$$E(\varphi) = \int_{\Omega} f(\nabla\varphi) d\Omega \tag{1.4}$$

where $f(|\nabla\varphi|) \geq 0$ is an increasing function of $|\nabla\varphi|$, as follows:

$$f^{'}(|\nabla\varphi|) > 0 \tag{1.5}$$

This energy functional is a measure of image smoothness and its minimization is equivalent to smoothing. The global minimum of [to latex] $E(phi)$ occurs when

$\varphi(x,y,z) \equiv$ constant, for all (x,y,z) $\in \Omega$ However, this trivial solution is not in agreement with our will to preserve features in image. Our goal is to preserve the features in image while decreasing the energy of image as much as possible. Based on the model in [239], we use a more general model as follows:

$$\partial_t \varphi - \|\nabla\varphi\| \mathrm{div} D^{\sigma} \frac{\nabla\varphi}{\|\nabla\varphi\|}) = C(\varphi - \varphi_0) \tag{1.6}$$

$$n^T (D^{\sigma} \nabla\varphi) = 0 \tag{1.7}$$

The diffusion tensor determines diffusion in local structure, like [239], we decompose any local vector into three directions: two principal directions of curvature and normal direction of local structure. Let B_{σ} be the basis transformation from the regularized frame of principal directions of curvature and the normal $\{\nu^{1,\sigma}, \nu^{2,\sigma}, N^{\sigma}\}$ onto the canonical basis $\{e_1, e_2, e_3\}$. We construct diffusion tensor as:

$$D^{\sigma} = B_{\sigma}^T \begin{pmatrix} G_1(\kappa^{1,\sigma}) & & \\ & G_2(\kappa^{2,\sigma}) & \\ & & G_3(\|\nabla\varphi\|) \end{pmatrix} B_{\sigma} \tag{1.8}$$

Our anisotropic diffusion tensor is different from that of previous work[239], the diffusion tensor in [239] is constructed from surface and smoothing is achieved along curvature directions. However, smoothing along normal direction is also helpful in smoothing image, Therefore smoothing along normal direction should be considered in constructing a 3D diffusion tensor, we call our method volumetric anisotropic diffusion model.

In [239], the function of G1(.) and G2(.) in (1.8) are selected to do diffusion along curvature directions to smooth 1D features (curve). G1(.) and G2(.) are defined as:

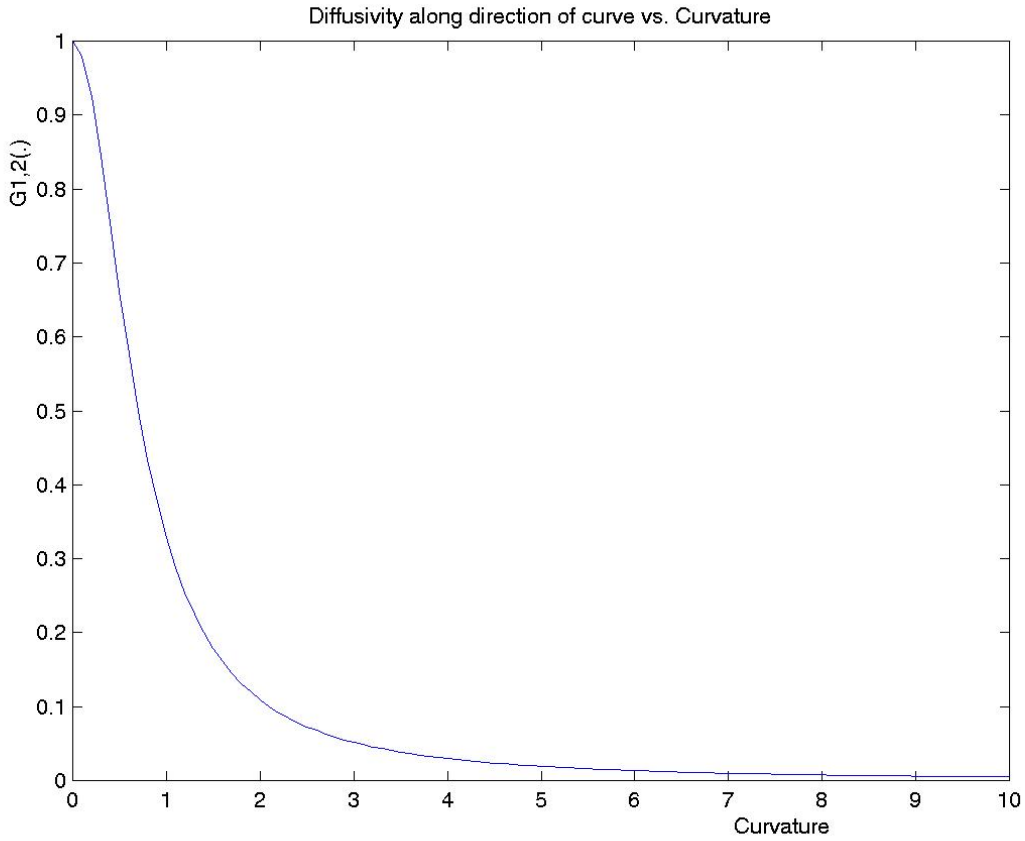


Figure 2.6: The diffusion along the curve direction is function of curvature of local structure.

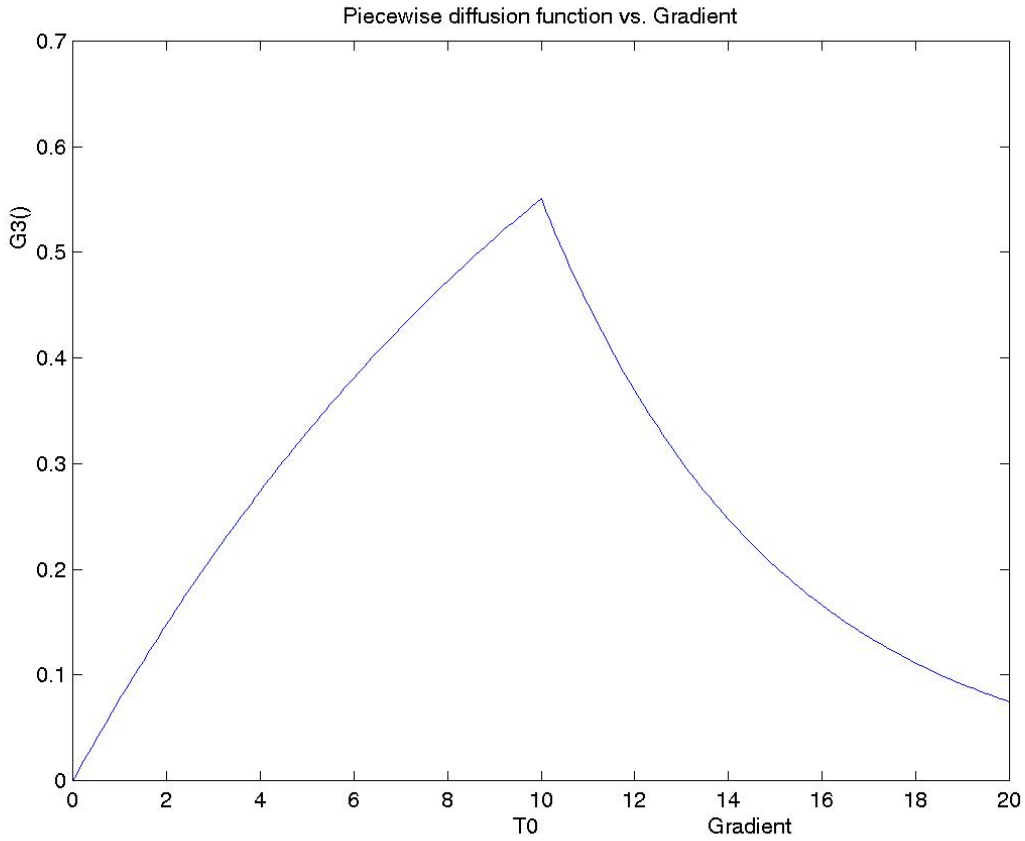


Figure 2.7: The diffusion along normal direction is function of gradient of local structure.

$$G_{1,2}(s) = \frac{1}{1+s^2/\lambda^2} \tag{1.9}$$

$\kappa^{1,\sigma}$, $\kappa^{2,\sigma}$ are curvatures along principal curvature direction of $v^{1,\sigma}$ and $v^{2,\sigma}$ respectively, smoothing is inhibited along the dominant principal direction of curvature. See Fig 2.6 for plot of function G1(.)

In order to enhance 2D features (surfaces), we construct diffusion tensor based on the following idea: Noise along normal direction can be eliminated by encouraging diffusion along the normal direction. In order to decrease the energy of image, we propose the diffusivity along the normal direction as $G_3(\|\nabla\varphi\|) = 1 - e^{-\beta\|\nabla\varphi\|}$, which can enhance coherence in homogeneous structure. In order to enhance the surfaces between heterogeneous structures, we adapt diffusion function as a piecewise function of gradient $\|\nabla\phi\|$, the definition is as follows:

$$G_3(\|\nabla\varphi\|) = \begin{cases} 1 - e^{-\beta_1\|\nabla\varphi\|} & \text{if } \|\nabla\varphi\| \le T_0 \\ (1 - e^{-\beta_1 T_0})e^{-\beta_2(\|\nabla\varphi\| - T_0)} & \text{otherwise} \end{cases} \tag{1.10}$$

where β_1 and β_2 are parameters to control the diffusion rate, T_0 are threshold associated with image. See Fig 2.7 for the plot of G3 (.)

By steering diffusion along the normal direction we can also decrease the gradient $\|\nabla\varphi\|$, according to (6.7), which will lead to decrease of energy of image. In other words, smoother images are generated through diffusion along three directions: normal direction and two curvature directions. Fig. 2.8 shows that applying diffusion along normal direction can fasten the smooth process.

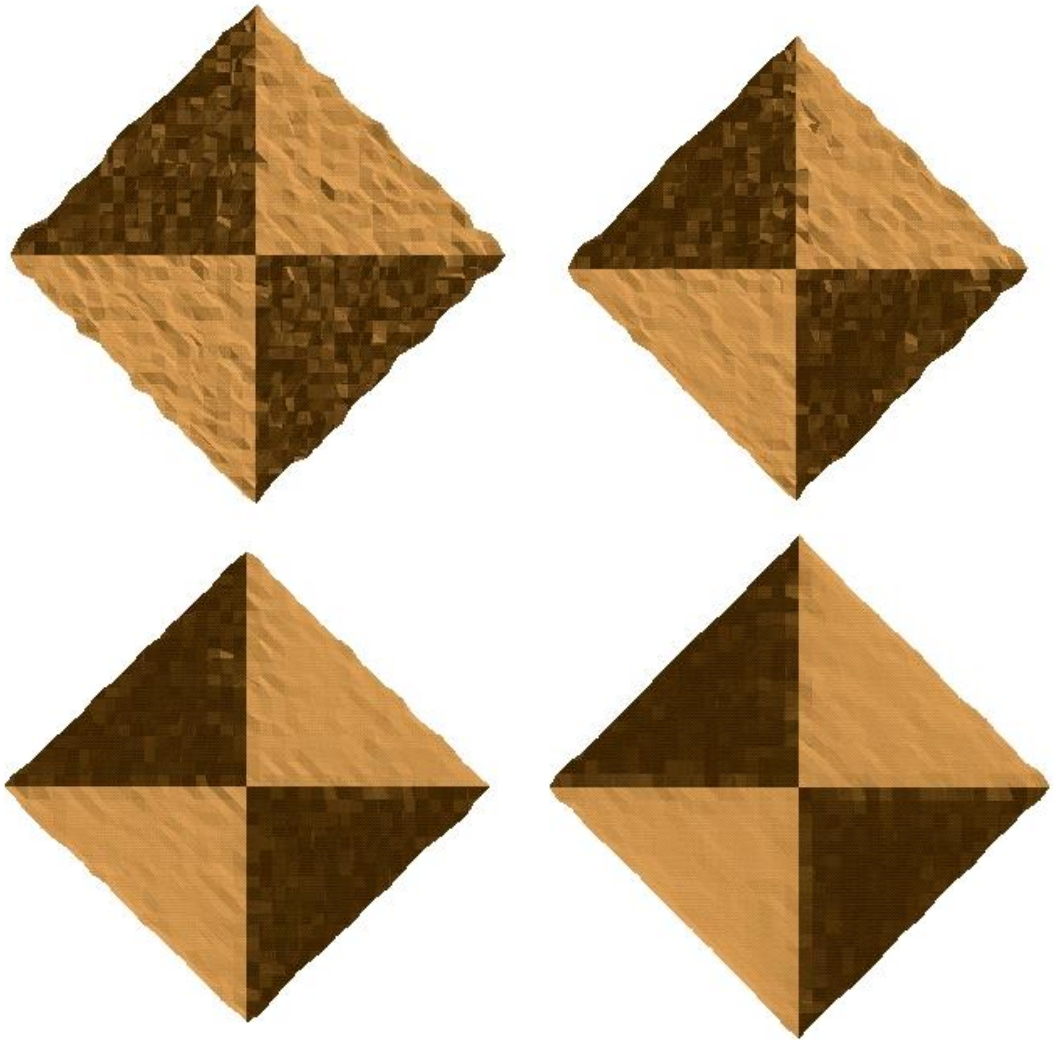


Figure 2.8: (a)-(d), the isosurface whose isovalue is 0.3 3 (a) is from the volume data that is defined as function |x|+|y|+|z| +noise 3(b) is from the filtered data after applying the method in [324] by setting diffusion along normal direction 0, the timestep is 3 3 (c) is from the filtered data after applying ansiotropic tensor(1.10), the timestep is 3 3(d) is from the filtered data after applying ansiotropic tensor(1.10), the timestep is 6

Finite Element Discretization and Bilateral Prefiltering

Finite Element Methods (FEM) are widely used to discretize and implemented the diffusion based models[239, 85, 173]. We also use FEM to solve (1.5) in our implementation.

Let $\Phi = \Phi(x,y,z,t)$ be the intensity value of the 3D image defined on the domain $\Omega = [0,1]^3$, where t serves as a multiscale parameter. For the reason of simplicity, the linear base function (hat function) is used as the basis function to approximate the original function, as [239], i.e,

$$\Phi(x,y,z,t) = \sum_{i=0}^{m-1} X_i(t) N_i(x,y,z) \tag{1.11}$$

where $X_i(t)$ are unknowns to be determined with the initial value $X_i(0)$.Then the spatial discretization is

$$\sum_p \partial_t X_p(t)(\frac{N_p}{\|\nabla\Phi\|}, N_q) + \varepsilon \sum_p X_p(t)(D^\rho \frac{\nabla N_p}{\|\nabla\Phi\|}, \nabla N_q) = 0 \tag{1.12}$$

where the inner product is defined as

$$(f,g) = \int_\Omega \text{fgdxdydz}$$

In each step of the discrete evolution we have to solve a linear system

$$(M^n + \tau L^n(D_\varepsilon^n))X^{n+1} = M^n X^n \tag{1.13}$$

for the new function vector X^{n+1} at time $t_{n+1} = (n+1)\tau$

Here, lumped mass matrix

$$M = (\frac{N_p}{\|\nabla\Phi\|}, N_q)_{p,q} \tag{1.14}$$

and nonlinear stiffness matrix

$$L = \frac{(D\nabla N_p, \nabla N_q)_{p,q}}{\|\nabla\Phi^\varepsilon\|} \tag{1.15}$$

To avoid inexact estimation of curvature from noisy data, we have to implement anisotropic diffusion process (1.13) in two steps[85]: The first step is to apply linear diffusion on initial data.

The purpose of this step is to have more accurate computation of curvature for anisotropic diffusion tensor. But linear isotropic diffusion can't provide very precise estimation for edge or curvature information. As You in [85] point out, prefiltering by linear isotropic diffusion is obviously against the spirit of anisotropic diffusion.

The nature of bilateral filtering resembles that of anisotropic diffusion[50]. In [295] Tomasi and Mandichi firstly introduced bilateral filtering as a nonlinear filter which combines domain and range filtering. In addition, bilateral filtering does not involve the solution of partial differential equation and can be implemented in a single iteration. In [50], Barash has presented that both bilateral filtering and anisotropic diffusion are related to adaptive smoothing.

Next we describe the basics of bilateral filtering, more details are available in [137].Consider a low-pass domain filter applied to an image, the input image f_1 and output image f_2 can be described as

$$f_2(x) = k_d^{-1} \int_y f_1(y) c(y-x) dy \tag{1.16}$$

where normalizing coefficient

$$k_d = \int_y c(y-x) dy \tag{1.17}$$

y is neighbor of x in certain neighborhood. Range filtering is defined as:

$$f_2(x) = k_r^{-1} \int_y f_1(y) s(f_1(y) - f_1(x)) dy \tag{1.18}$$

In this case, the kernel measures the similarity between pixels. The normalization coefficient in this case is

$$k_r = \int_y s(f_1(y) - f_1(x)) dy \tag{1.19}$$

Weight function c(.) and s(.) can be any monotically decreasing function, like (6.5) or (6.6), if c(.) takes the form of (6.5) , (1.17) becomes Guassian filtering. Combined domain and range filtering will be denoted as bilateral filtering,

$$f_2(x) = k_b^{-1} \int_y f_1(y) s(f_1(y) - f_1(x)) c(y-x) dy \tag{1.20}$$

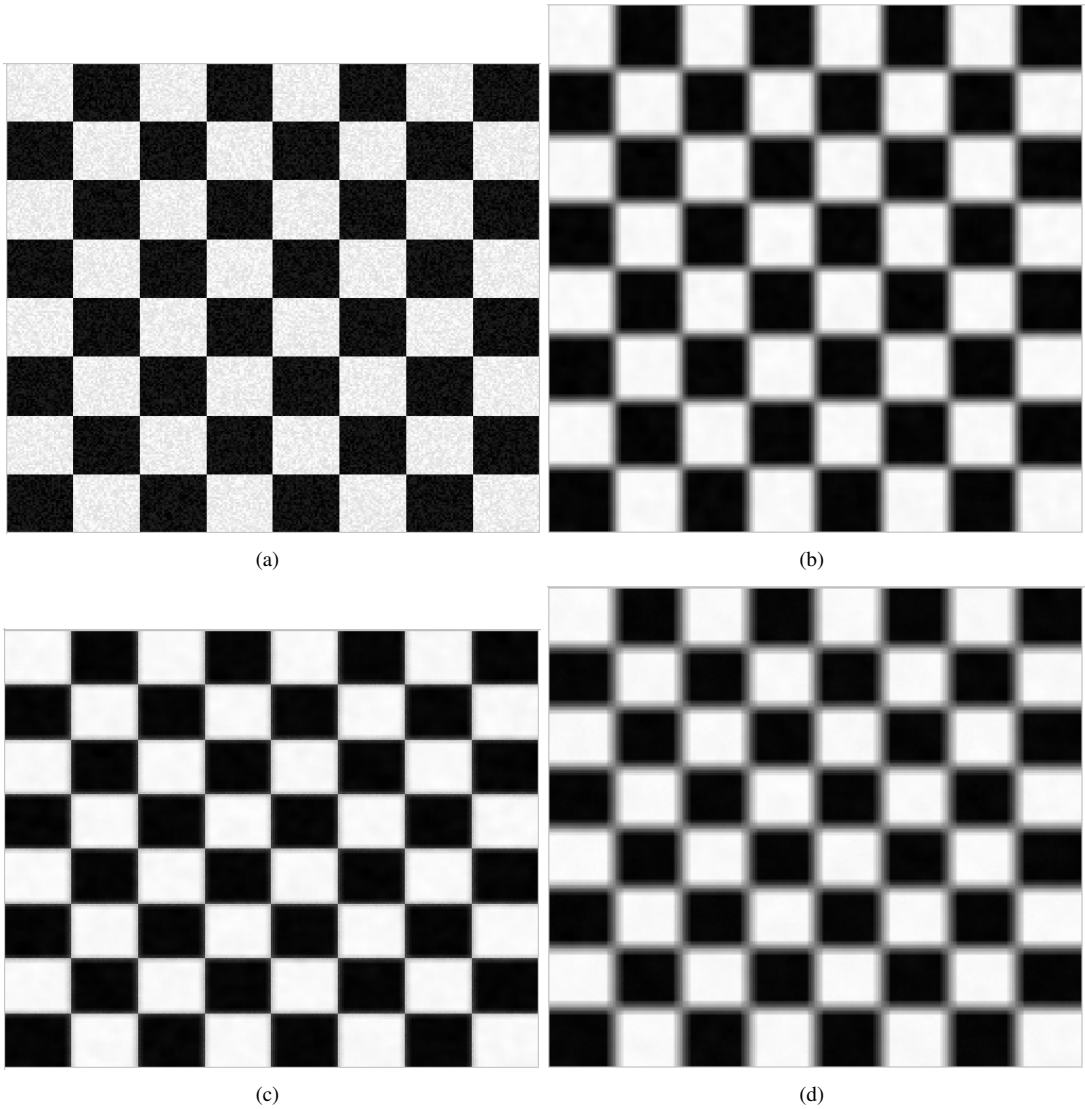
where

$$k_b = \int_y c(y-x) s(f_1(y) - f_1(x)) dy \tag{1.21}$$

Fig 2.9 shows that bilateral filtering outperforms Gaussian filtering by applying different filtering techniques on a simple 2D image.

Since bilateral filtering can remove noise as Gaussian filtering can do, here we replace Gaussian prefiltering with bilateral prefiltering, anisotropic diffusion process in step 2 can be expected to be well posed. Additionaly, bilateral filtering has the advantage of preserving edge or curvature information, which is better for constructing anisotropic diffusion tensor.

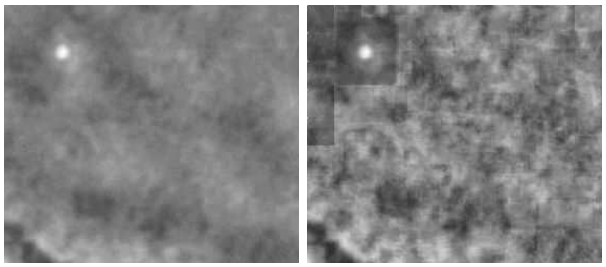
Based on the calculations of curvature from the prefiltered data in the first step, the second step is to do anisotropic diffusion by solving (1.13).



(a) (b)

(c) (d)

Figure 2.9: Bilateral Filtering vs. Gaussian Filtering. (a) Noisy Image (b) Filtered by Gaussian Filter (c) Filtered by Bilateral Filter (d) Filtered by Isotropic Diffusion



(a) Original image (b) Enhanced image

Figure 2.10: Enhanced image using local searching method

2.1.2 Contrast Enhancement

The following algorithm was taken from [332] (Yu2004:ICIP).

Image enhancement is a process used to improve the image quality for better visual appearance or for other specific applications (e.g., segmentation, feature recognition, 3D visualization, etc.). In general, image enhancement includes image contrast enhancement and image noise removal. The first technique is used in low-contrast images while the second one is often required for noisy images. In this paper we shall concentrate on image contrast enhancement.

Many images, such as medical images, remote sensing images, electron microscopy images and even our real-life photographic pictures, suffer from poor contrast. For example, some images taken with electronic microscopy have very low contrast such that the biological particles are very hard to see by eyes [133, 209]. Low contrast images are also commonly seen in medical images (such as chest CT and mammography images [202, 186]) as well as in many other application areas. Therefore, it is very necessary to enhance the contrast of such images before further processing or analysis can be conducted.

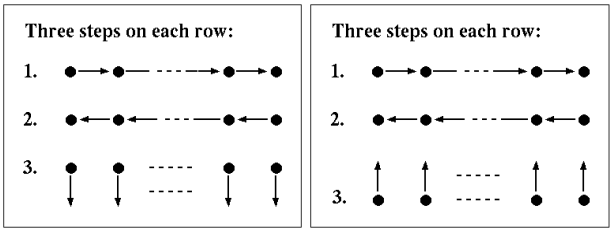
In this section we describe the details of our algorithm for image contrast enhancement. In our method, new intensity is assigned to each pixel based on the local minimum/maximum intensity as well as the local average intensity at that pixel. Hence, in the following, we will first describe how to compute the local min/max/avg maps for each pixel, and then we will explain how to design a stretching function, which is then used to compute a new intensity for each individual pixel.

Compute Local Min/Max/Avg by Conditional Propagation Scheme

The local min/max/avg of a pixel can be simply defined as the minimal, maximal and averaging intensities within a local window with a fixed size (e.g., 11×11 window). By this definition we can easily compute the local min/max/avg by working on the intensity values of all the pixels within the local window. This technique, called *local searching* in the following, is straightforward to implement but it turns out to have two problems. First, it takes a lot of time to search for the local min/max or compute the local average for each pixel (it might be possible to solve this problem by considering the correlation between the local min/max of two neighboring pixels). Secondly, the computed min/max maps by local searching are not continuously defined over the image domain. In other words, the resulting min/max maps may contain many "steps", yielding lots of block-like artifacts in the enhanced images. Fig. 2.10 shows an example, where part (b) is the enhanced image by our method except that the local min/max/avg maps are computed by *local searching method*. In the following we will first consider the computation of local min/max maps. The computation of local average will be given at the end of this subsection.

One way to remedy the second problem is to use a Gaussian filter on the obtained min/max maps, yielding smoothed min/max maps over the image domain. However, this requires additional time. According to [101, 326], the the exponential and Gaussian filters can be implemented very fast by propagation scheme. The question is: "can we combine the computation of local min/max maps together with the smoothing of these maps by using a unified propagation scheme?". If we can, then the smoothed min/max maps could be computed very fast. To achieve this goal, we introduce in the following a *conditional propagation scheme*.

The basic idea of our propagation scheme is that, to compute the local min/max at pixel (m, n), we just need to check the local min/max at its four neighbors, namely, $(m-1, n)$, $(m+1, n)$, $(m, n-1)$ and $(m, n+1)$, instead of checking all the pixels



(a) Top-down (b) Bottom-up

Figure 2.11: Two scans to compute local minimum and maximum by propagation

within the pre-defined local window. The propagation rule from a neighboring pixel, say, $(m-1, n)$ to the pixel (m, n) is defined as follows:

$$\begin{cases} if(\text{lmin}[m-1,n] < \text{lmin}[m,n]) \\ \quad \text{lmin}[m,n] = (1-C) \times \text{lmin}[m,n] \\ \qquad\qquad +C \times \text{lmin}[m-1,n] \\ \\ if(\text{lmax}[m-1,n] > \text{lmax}[m,n]) \\ \quad \text{lmax}[m,n] = (1-C) \times \text{lmax}[m,n] \\ \qquad\qquad +C \times \text{lmax}[m-1,n] \end{cases} \tag{1.22}$$

where C is called *conductivity factor*, which is chosen from the interval $[0,1]$. The two matrices *lmin* and *lmax* stand for the local minimum map and local maximum map, respectively. Both of these two matrices are initialized with the image intensity values at corresponding pixels.

The propagation rule defined in (1.22) is repeatedly used in two scans on the entire image. One is top-down scan and the other is bottom-up scan. Top-down scan is illustrated in fig. 2.11(a), where the scan starts from the first row and ends on the last row. On each row three steps are implemented. First, the rule defined in (1.22) is repeatedly applied from the first pixel to the last pixel on this row. Secondly, the same rule is repeatedly applied from the last pixel to the first pixel on this row. Thirdly, the same rule is repeatedly applied to all the pixels on the current row to the next lower row. Similarly, for the bottom-up scan, the propagation starts from the last row and ends on the first row and on each row three steps are implemented in a similar way (see fig.2.11(b)).

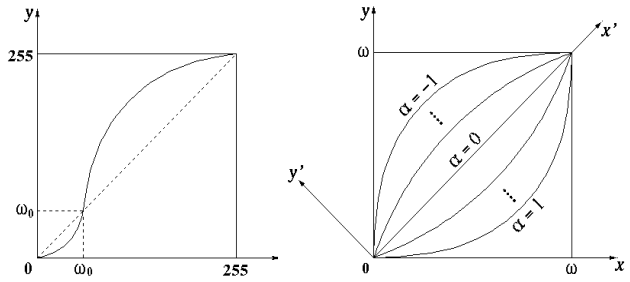
The *conditional propagation scheme* has three advantages compared to the local searching method. First, it is very fast since the computation of min/max at a pixel only requires six propagations (three for top-down scan and three for bottom-up scan). Secondly, the resulting min/max maps by *conditional propagation scheme* are smoothed over the image domain. This smoothing effect, however, is different from Gaussian filtering in that the weighting function used here is an exponential function instead of a Gaussian function. [101]. Thirdly, the *conditional propagation scheme* does not explicitly need any local windows as seen in local searching method. Theoretically any pixel has influence to all other pixels if it satisfies the conditions seen in (1.22).

To conclude this subsection, we briefly describe the computation of local average intensity of each pixel. Actually the local average intensity can be computed together with the computation of local minimum and maximum using a similar propagation scheme as seen in (1.22) except that all the if statements are omitted here:

$$\begin{aligned} \text{lavg}[m,n] = (1-R) \times \text{lavg}[m,n] \\ +R \times \text{lavg}[m-1,n] \end{aligned} \tag{1.23}$$

Determine Transfer Function

As we said before, the idea of our contrast enhancement algorithm is to assign a new intensity value to each pixel based on the obtained minimum, maximum and average intensities locally defined at that pixel. Therefore, before we actually implement the assignment, we need to design the stretching functions for each pixel. This includes two aspects: scaling the domains of functions and designing the transfer functions. Keep in mind that all the following calculations are done pixel by pixel.



(a) Stretching window (b) Transfer function

Figure 2.12: Determine the transfer functions

The fundamental idea for contrast enhancement seen in most existing approaches is to take advantage of range stretching (either directly or indirectly). In other words, the original low contrast image has a narrow intensity range, which is expanded to a much broader range such that the resulting image is enhanced. This mapping from a narrow range to a broad range is called range stretching or the stretching of the function domain. In our method, we also use range stretching, which, however, differs from other approaches in that the stretching function in our method are different from pixel to pixel. In our method, the original range at a pixel is given by the absolute difference between the obtained local minimum and maximum intensities at that pixel, that is, $|lmax - lmin|$. Then this value is mapped to a new value according to the functional curve illustrated in fig.2.12(a), where the x-coordinate represents the input range while the y-coordinate stands for the output range (expanded in most cases). This function curve is composed of two circular arcs:

$$y = \begin{cases} w_0 - \sqrt{w_0^2 - x^2}, & \text{if } x \le w_0 \\ w_0 + \sqrt{(255 - w_0)^2 - (255 - x)^2}, & \text{else} \end{cases} \tag{1.24}$$

where w_0 is a fixed value, which might be useful for suppressing image noises while enhancing image features. However, in our current implementation we are not using w_0 (just let $w_0 = 0$).

After the mapping from the original narrow range $|lmax - lmin|$, we obtain a broader range (denoted by ω) such that $lmin$ and $lmax$ are mapped to 0 and ω, respectively. In the meanwhile, the original image intensity I_{old} and average A_{old}, which satisfy $lmin \le I_{old}, A_{old} \le lmax$, must be linearly stretched to their new values I_{new}, A_{new}:

$$\begin{cases} I_{new} = \omega \times \frac{I_{old} - lmin}{lmax - lmin} \\ A_{new} = \omega \times \frac{A_{old} - lmin}{lmax - lmin} \end{cases} \tag{1.25}$$

To achieve better contrast enhancement, the value I_{new} needs to be further updated by a certain transfer function defined on $[0, \omega]$. Those transfer functions are adaptively defined in the sense that they may be different from pixel to pixel based on a parameter α, where $\alpha \in [-1, 1]$, such that the functions are convex if $\alpha < 0$ and concave if $\alpha > 0$ (as seen in fig.2.12(b)). The parameter α for each pixel is given by

$$\alpha = \frac{(A_{new} - I_{new})}{128} \tag{1.26}$$

where A_{new} and I_{new} are given by (1.25) for each individual pixel. The value of α defined in (1.26) satisfies $\alpha \in [-1, 1]$ in most cases. Otherwise, we let $\alpha = 1$ if $\alpha > 1$ and $\alpha = -1$ if $\alpha < -1$. Note that the intension for choosing α in (1.26) is as follows. If the image intensity at a pixel is lower than the local intensity average, then we further decrease the image intensity using a concave transfer function. On the other hand, if the image intensity at a pixel is higher than the local intensity average, then we further increase the image intensity using a convex transfer function.

Given a parameter α, the transfer function can be defined as a segment of parabolic curve. To derive the explicit formulas of those parabolic functions, we introduce a new coordinate system, namely, X'Y'-system (see fig.2.12(b)). In the new system, the parabolic curves can be represented as:

$$y' = \frac{\alpha}{\sqrt{2}\omega} x'(x' - \sqrt{2}\omega) \tag{1.27}$$

where $\alpha \in [-1, 1]$ and $x' \in [0, \sqrt{2}\omega]$. Note that the weight $\frac{1}{\sqrt{2}\omega}$ is used in (1.27) such that all the curves defined in (1.26) are entirely enclosed by the rectangle box and the two curves associated with $\alpha = \pm 1$ are exactly tangent to the box at both ends (see fig. 2.12(b)).

To derive the representation of those curves in XY-coordinate, we have the following coordinate transformation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \tag{1.28}$$

Using (1.28) we can replace x', y' in (1.27) with x, y:

$$-\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y = \frac{\alpha}{\sqrt{2}\omega}(\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y)(\frac{\sqrt{2}}{2}x + \frac{\sqrt{2}}{2}y - \sqrt{2}\omega) \tag{1.29}$$

Or equivalently,

$$\frac{\alpha}{2\omega}y^2 + (\frac{\alpha}{\omega}x - \alpha - 1)y + (\frac{\alpha}{2\omega}x^2 - \alpha x + x) = 0 \tag{1.30}$$

So we have the explicit expression to compute y from x:

$$y = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1.31}$$

where

$$\begin{cases} a = \frac{\alpha}{2\omega} \\ b = \frac{\alpha}{\omega}x - \alpha - 1 \\ c = \frac{\alpha}{2\omega}x^2 - \alpha x + x \end{cases} \tag{1.32}$$

where $x \in [0, \omega]$ and α is given by (1.26).

Non-Linear Contrast Enhancement

After we obtain a transfer function for each pixel, we can use it to compute the new value for that pixel. Without loss of generality, let us just consider the pixel (x, y) and the original image intensity at that pixel is I_{old} which is then replaced by I_{new} using (1.25). Then I_{new} is further updated to its final value I_{enh} using the obtained transfer function for pixel (x, y), say, f_{xy}. I_{enh} is given by the following equation:

$$I_{enh}(x, y) = lmin(x, y) + f_{xy}(I_{new}(x, y)) \tag{1.33}$$

where *lmin* is the obtained local minimum at pixel (x, y). In next section, we will see that the addition of *lmin* in (1.33) is very important to correctly enhance the image contrast.

We summarize the whole procedure of our algorithm in the following.

First step (two scans):

(i) Compute local min/max maps *lmin* and *lmax* using equation (1.22).

(ii) Compute local average maps I_{avg} using equation (1.23).

Second step (for every pixel):

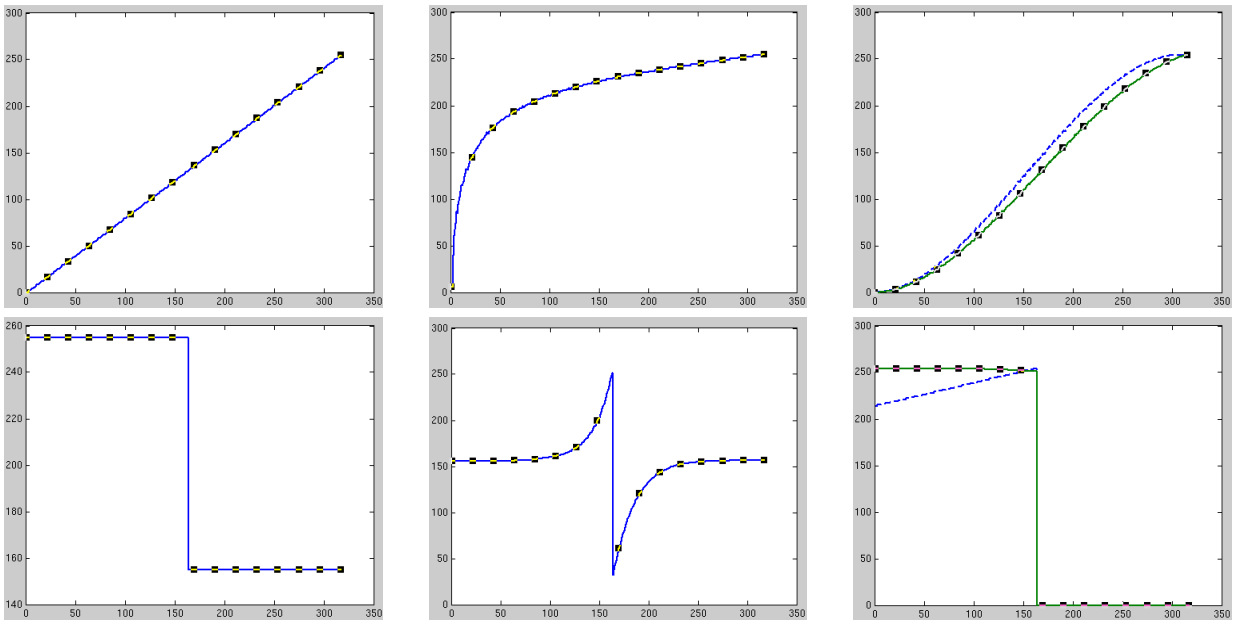


Figure 2.13: Two simple images shown by intensity variation from left to the right. Left: original intensity variation. Middle: by retinex model. Right: by our method. Top-row: our first case. Bottom-row: our second case.

(i) Compute the expanded window size ω using equation (1.24).
(ii) Compute the corrected image intensity I_{new} and local average A_{new} using equation (1.25).
(iii) Compute the parameter α using equation (1.26) to specify the transfer function.
(iv) Compute a, b, c using equation (1.32) based on the obtained α, ω and I_{new}.
(v) Compute the new value using (1.31).
(vi) The final value is given by the value obtained in step 2(v) plus *lmin* obtained in step 1(i) at that pixel.

Discussion

In this section we discuss several issues on our proposed approach. First we will study two special cases in which different methods behave in different ways. Then we analyze the histograms of the original images and the enhanced images by histogram equalization and by our method. Finally we briefly discuss some other issues such as extension to 3D images, extension to anisotropic propagation and the multiscale property of our method.

Two simple images are discussed in the following. The first image has intensities increasing evenly from the left (dark area) to the right (bright area) but the intensity values are constantly defined in vertical direction. The intensity variation (from the left to the right) of this image is shown in fig.2.13(top-left). Obviously histogram equalization does not change this figure because in this case the histogram is already equalized. The intensity variation after applying single-scale retinex model is illustrated in fig.2.13(top-middle), from which we can see that this method tends to brighten the original images but causes an effect of intensity saturation in bright areas. The solid curve in fig.2.13(top-right) shows the result by our method. We can see that our result nearly matches the original intensity variation. The dashed line in fig.2.13(top-right) corresponds to the result by our method except that the term *lmin* in equation (1.33) is omitted.

Our second special case is shown in fig.2.13(bottom-left) where the original image has a sudden change of intensity from the left (bright area) to the right (dark area). Traditional histogram equalization technique may change the relative intensity values of those two areas but the resulting figure remains a perfect step. However, the result by single-scale retinex model (see fig.2.13(bottom-middle)) comes up with two 'pulses' near the boundary between those two areas. This kind of reality loss could be clearly seen from examples in [169]. The behavior of our method is illustrated in fig.2.13(bottom-right), where the step between two areas is correctly kept (see the solid line). Again the dashed line in fig.2.13(bottom-right) is the result by our method without the term *lmin* as used in equation (1.33). From both of the above cases we know that the additional term *lmin* in equation (1.33) is quite necessary to guarantee correct enhancement.

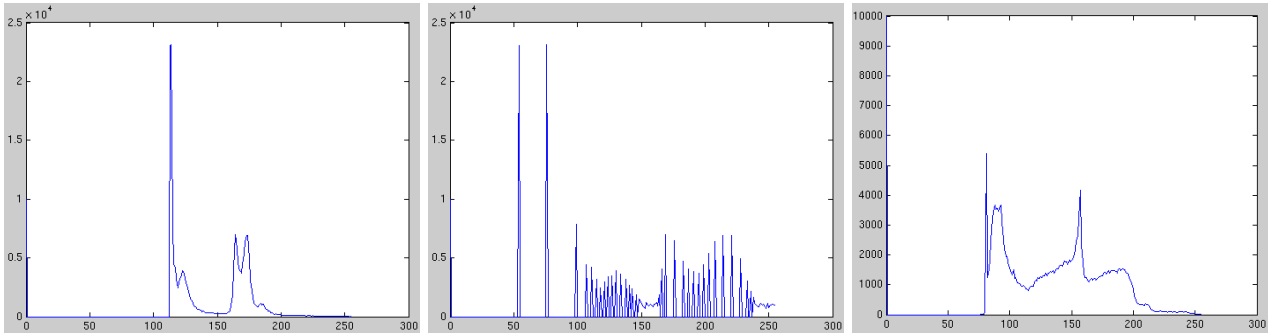


Figure 2.14: Histogram analysis. Left: histogram of original image (see fig.8 for the original image). Middle: histogram after histogram equalization. Right: histogram after applying our method.

Histogram Analysis

It is quite interesting to study different contrast enhancement techniques based on histogram analysis. As we know, images with low contrast usually have a very narrow intensity range in their histograms (see fig.2.14(left) for example; the original image comes from the chest CT image shown in fig.2.17(left)). To enhance the contrast, we can simply broaden the intensity range. The intensity range can be broadened by histogram equalization, which, however, often causes an effect of intensity saturation due to the reduced number of intensity values (see fig.2.14(middle)). As comparison, our method produced a more "continuous" histogram (fig.2.14(right)) such that the enhanced image looks more comfortable.

Other Issues

The extension of our approach to 3D images is quite straightforward. The determination of the transfer functions is the same as we saw in section 2 for 2D images. The only difference is the computation of the local min/max/avg maps for each pixel. There are also two-direction scans: top-down and bottom-up. For top-down scan, we start from the top slice and compute the min/max/avg maps within that slice using 2D algorithm as seen before and propagate the results to the next lower slice. This process is repeated until we reach the bottom slice. Then the bottom-up scan is done in a similar way.

The *conductivity factor C* in equation (1.22) is a constant value chosen from the interval $[0,1]$. It is this *conductivity factor* that makes the computed min/max/avg maps vary smoothly over the image domain. However, this is an isotropic smoothing, which, similar to Gaussian filtering, makes some small features disappear after the enhancement (this will be shown in next section). A commonly used technique to remedy this problem is called anisotropic diffusion [230, 307]. Borrowing the idea from [230] we developed an anisotropic propagation for our contrast enhancement. Equation (1.22) and (1.23), called isotropic propagation, are now modified and combined into the following equations.

$$\begin{cases} if(\text{lmin[m-1, n]} < \text{lmin[m, n]}) \\ \quad \text{lmin[m, n]} \mathrel{+}= (\text{lmin[m-1, n]} - \text{lmin[m, n]}) \\ \quad\quad *exp(\text{-R} * \|\text{lmin[m-1, n]} - \text{lmin[m, n]}\|) \\ \\ if(\text{lmax[m-1, n]} > \text{lmax[m, n]}) \\ \quad \text{lmax[m, n]} \mathrel{+}= (\text{lmax[m-1, n]} - \text{lmax[m, n]}) \\ \quad\quad *exp(\text{-R} * \|\text{lmax[m-1, n]} - \text{lmax[m, n]}\|) \\ \\ \text{lavg[m, n]} \mathrel{+}= (\text{lavg[m-1, n]} - \text{lavg[m, n]}) \\ \quad\quad *exp(\text{-R} * \|\text{lavg[m-1, n]} - \text{lavg[m, n]}\|) \end{cases} \tag{1.34}$$

where R is called *resistance factor* and generally chosen from the interval $[0.01, 0.1]$ in our experiments.

Another interesting issue about our method is the multiscale property. Briefly speaking, both *conductivity factor C* and *resistance factor R* can be chosen differently, yielding different scales of image details that are enhanced. In next section we shall see an example of this property.

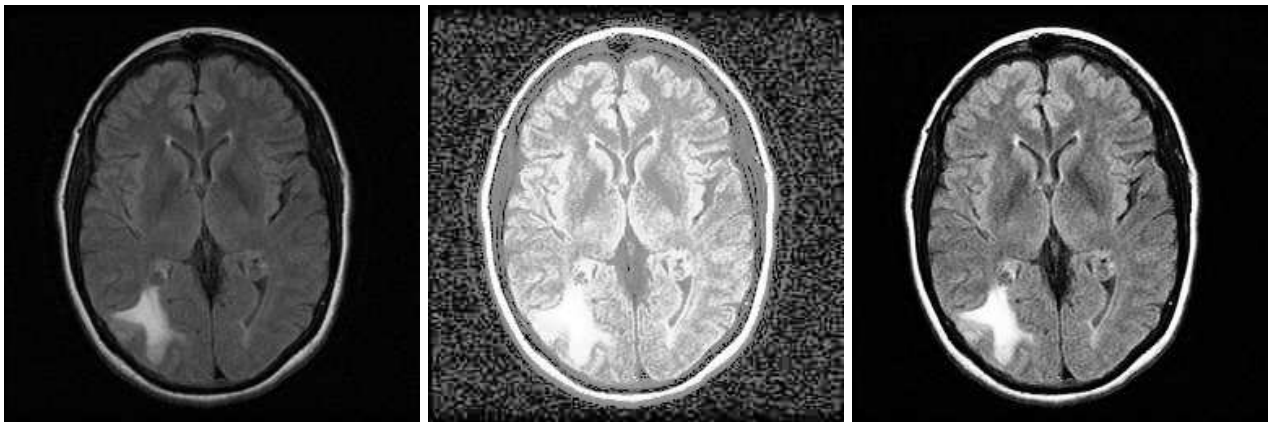


Figure 2.15: Brain MR image —– Left: original image with low contrast. Middle: enhanced image by histogram equalization. Right: enhance image by our method (isotropic propagation with $C = 0.95$).

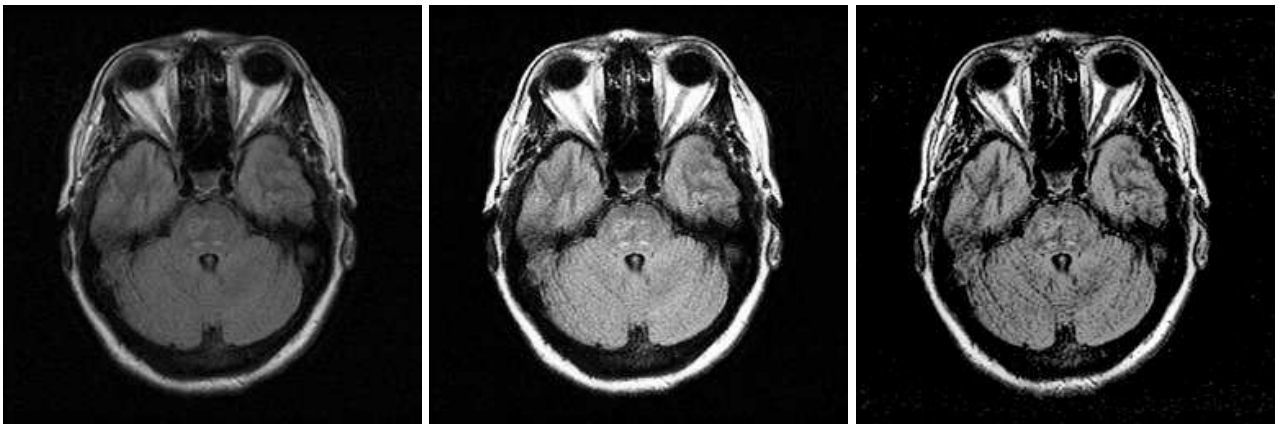


Figure 2.16: Brain MR image —– Left: original image with low contrast. Middle: enhanced image by our method with isotropic propagation ($C = 0.95$). Right: enhance image by our method with anisotropic propagation ($R = 0.02$).

Results

We have tested our method on various types of images including medical images, electron microscopy images, remote sensing images as well as many real-life photographs and it turns out that our approach always gives very good enhancement results. However, due to the space limit, we will only show in this paper several examples of medical images and demonstrate their enhanced results by our method. From the left to the right, the images in Fig.6 respectively show a brain MR image with low contrast, the enhanced image after applying histogram equalization and the result by our method (isotropic propagation with $C = 0.95$). Obviously the enhanced image by histogram equalization shows an effect of intensity saturation. The image on the left in Fig.7 shows another example of brain MR image. From the other two images in Fig.7, we can see the difference of the enhanced images using our method with isotropic and anisotropic propagations. Clearly anisotropic propagation behaves better in preserving image features than isotropic propagation. Fig.8 shows an example of chest CT image. From the left to the right, we show the original image, the enhanced image by histogram equalization and enhanced image by our method (anisotropic propagation with $R = 0.1$). Finally we demonstrate the multiscale property of our method (by isotropic propagation) in Fig.9. The original image (the left one) is a mammography image with very low contrast. The other images illustrate results by our method using different *conductivity factors*.

Figure 2.17: Chest CT image —– Left: original image with low contrast. Middle: enhanced image by histogram equalization. Right: enhance image by our method (anisotropic propagation with $R = 0.1$).

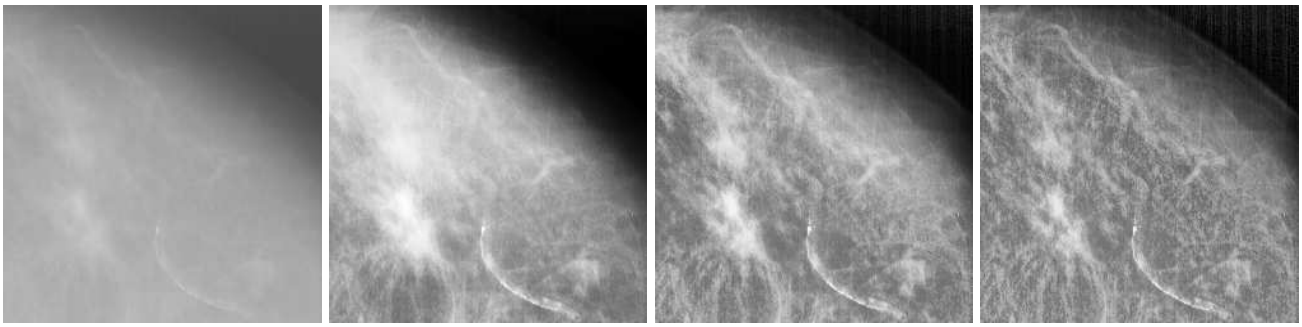


Figure 2.18: Mammography image —– Left: original image with very low contrast. Second from left: enhanced image by our method with isotropic propagation ($C = 0.95$). Third from left: enhanced image by our method with isotropic propagation ($C = 0.85$). Right: enhanced image by our method with isotropic propagation ($C = 0.75$).

2.1.3 Classification/Segmentation

The following is taken from [334]. (YuBajaj2005)

Detection of Critical Points

In general, the critical points of a scalar map include three types: maximal, minimal, and saddle. However, only the maximal critical points are of our interest to represent the features of molecular density maps. This type of critical points can be simply computed from the local maxima of a given scalar map. Since noise is always present in the original maps, a pre-filtering process should be applied. A filter can be either linear or nonlinear. A linear filter (e.g., Gaussian filtering) may destroy some weak features and hence eliminate some critical points. A nonlinear filter [230, 306], however, tends to "flatten" a region, yielding many unwanted critical points. In this section we propose another type of filters, based on the gradient vector diffusion, and extract the critical points from the diffused vector fields.

In general, the gradient vector field generated directly from the original map suffers from noise and cannot be used to detect the critical points. In [313], the authors described a diffusion technique to smooth gradient vector fields. The gradient vectors are represented by Cartesian coordinates and a set of partial differential equations (PDEs) are separately applied to each component of the vectors. The equations are linear or isotropic, and therefore inherit the drawbacks of most linear systems. Another way to diffuse a gradient vector field is based on the polar-coordinate representation of the vectors [329, 330]. The drawback of this method is its computational burden due to the efforts that have to be made to deal with the periodicity of orientations. In the following we propose a set of anisotropic diffusion equations and attempt to address the afore-mentioned problems:

$$\begin{cases} \frac{du}{dt} = div(g(\alpha)\cdot\nabla u) \\ \frac{dv}{dt} = div(g(\alpha)\cdot\nabla v) \\ \frac{dw}{dt} = div(g(\alpha)\cdot\nabla w) \end{cases} \tag{1.35}$$

where (u,v,w) are initialized with the negative gradient vectors of the original maps. $g(\cdot)$ is a decreasing function and α is the angle between the central vector and its surrounding vectors. For instance, we can define $g(\alpha)$ as follows:

$$g(\vec{c},\vec{s}) = \begin{cases} e^{\kappa\cdot(\frac{\vec{c}\cdot\vec{s}}{\|\vec{c}\|\|\vec{s}\|}-1)} & if\ \vec{c}\neq 0\ and\ \vec{s}\neq 0 \\ 0 & if\ \vec{c}=0\ or\ \vec{s}=0 \end{cases} \tag{1.36}$$

where κ is a positive constant; $\vec{c}$ and $\vec{s}$ stand for the central vector and one of the surrounding vectors, respectively.

Once the gradient vector field is generated and diffused, we can detect the *source* points, as defined below:
Definition: A point is called *source* if none of its neighbors points to it. In other words, a point A is a *source* if and only if, for any neighbor B of A:

$$\vec{v}_B\cdot\vec{BA}\leq 0 \tag{1.37}$$

where $\vec{v}_B$ is the diffused gradient vector at B and $\vec{BA}$ is the vector from B to A. All the *source* points are regarded as the critical points, which will be frequently used below in the symmetry detection as well as the segmentation.

Segmentation of Asymmetric Subunits

Before we describe the algorithm for segmenting the asymmetric subunits of an icosahedral map, we briefly review the well-known fast marching method and introduce a variant of this method.

The Fast Marching Method: A Variant

The fast marching method [205, 260, 259] is a simplified variant of the level set method [259] but it is much faster than the latter one. The basic idea of this method is that a contour is initialized from a pre-chosen seed point, and the contour is allowed

to grow until a certain stopping condition is reached. Every voxel is assigned with a value called *time*, which is initially zero for seed points and infinite for all other voxels. Repeatedly, the voxel on the marching contour with minimal *time* value is deleted from the contour and the *time* values of its neighbors are updated according to the following equation:

$$||\nabla T(\vec{r})|| \cdot F(\vec{r}) = 1 \tag{1.38}$$

where $F(\vec{r})$ is called the *speed function* that is usually determined by the gradients of the input maps (e.g., $F(\vec{r}) = e^{-\alpha\|\nabla I\|}$, where $\alpha > 0$ and I is the original map). The updated neighbors, if they are updated for the first time, are then inserted into the contour.

The traditional fast marching method is designed for a single object segmentation. In order to segment multiple objects, such as the 60-component virus shells or 3-component molecular trimmers, one has to choose a seed for each of the objects. However, assigning only one seed to each object may cause a problem. As shown in Figure 2.19(a), we consider an image consisting of two components. Two seeds A and B are chosen as seen in the figure. If we let two contours, starting from A and B respectively, grow simultaneously and independently, we may see that the contour from B reaches C before the other contour from A does so. This apparently causes a wrong segmentation of the two components. To remedy this problem, we describe in the following an approach based on an idea of "re-initialization". Before we go into the details, we introduce a concept of *marching distance*.

Definition: Given a 3D scalar function $f(\vec{r})$, the *marching distance* between two points A and B in the function domain is defined by:

$$MD_f(A,B) = Min\{\int_{A\to B} e^{\|\nabla f(\vec{r})\|} ds\} \tag{1.39}$$

where $\int_{A\to B}$ is the integral along a path from A to B. The minimization is conducted over all the paths from A to B. In particular, if $f(\vec{r})$ is constant, the *marching distance* is degraded to *Euclidean distance* or *geodesic distance*, depending on whether the paths are constrained or not. It can be shown that the *marching distance* satisfies the properties of a classical metric:

- $d(x,y) \geq 0$, and $d(x,y) = 0 \Leftrightarrow x = y$.
- $d(x,y) = d(y,x)$.
- $d(x,y) + d(y,z) \geq d(x,z)$.

The discrete form of Equation 1.39 is as follows:

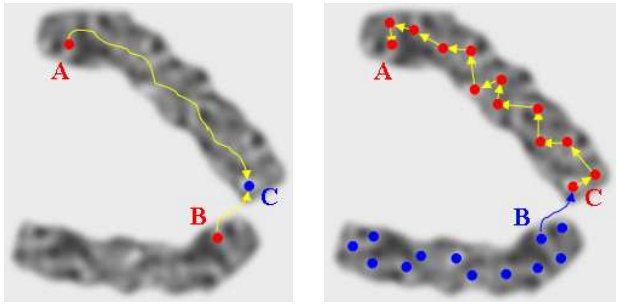
$$MD_f(A,B) = Min\{\sum_{\vec{r}=A}^{B} e^{\|\nabla f(\vec{r})\|} - \frac{e^{\|\nabla f(A)\|} + e^{\|\nabla f(B)\|}}{2}\} \tag{1.40}$$

The second term on the right-hand side of Equation 1.40 is used to guarantee the *discrete marching distance* satisfying the properties of a metric. Again, the minimization in Equation 1.40 is conducted over all the paths from A to B, where the paths may be defined on the basis of 6- or 26-voxel neighborhoods. Since the volumes under consideration are always digitized, we always refer to the *discrete marching distances* when we speak of the marching distance. By definition, the marching distance is analogous to the arrival time of a contour, starting from a seed, at an arbitrary point.

The problem seen in Figure 2.19(a) can be rewritten as: $MD(A,C) > MD(B,C)$. To remedy this problem, one can divide the marching distance $MD(A,C)$ into a number of small pieces by casting some critical points, denoted by $D_i, i = 1,2,\cdots,n$, orderly on the path from A to C (see Figure 2.19(b)). In terms of the fast marching method, the arrival time of the contour starting from A is re-initialized to 0 at the nearest critical point D_1 of A. Similarly, the arrival time of the contour starting from D_1 is re-initialized to 0 at the nearest critical point D_2 of D_1, and so on. By this way, the correct segmentation can be achieved as long as enough critical points are used such that the following condition holds (let $D_0 = A$ and $D_{n+1} = C$):

$$Max\{MD(D_i, D_{i+1}), \text{for all } i = 0,1,\cdots,n\} < MD(B,C) \tag{1.41}$$

The above "re-initialization" idea can be simply implemented by regarding all the critical points as seeds. In other words, each component (object) will be assigned a number of seeds instead of just one seed. Every seed initiates one contour and all contours start to grow simultaneously and independently. Two contours corresponding to the same object should merge into one contour, while two contours corresponding to different objects should stop on their common boundaries. Unfortunately, we do not have any explicit clue about the correspondence of the critical points (seeds) to the objects. In case of the example seen



(a) problem (b) solution

Figure 2.19: A variant of the fast marching method

in Figure 2.19(b), all we know for sure is that A and B belong to the upper and the lower objects, respectively. Hence we have to find a way to assign the membership of each of the other critical points to one of the objects that we want to segment. Given that at least one seed has been assigned to each object, we summarize in the following the overall algorithm of our proposed variant of the fast marching method:

1. **Detection of the critical points**
   This has already been discussed in Section 2.

2. **Classification of the critical points**

   (a) Let $C_i, i = 1, 2, \cdots, K$ denote all the critical points with unknown membership. If $K > 0$, we take all $C_i, i = 1, 2, \cdots, K$ as the seeds and run the traditional fast marching method until a point with known membership is visited. Denote this point by P.

   (b) The fast marching method produces a *time* map, representing the arrival time on every voxel (*time* is infinite for unvisited voxels). We start from P and trace downhill by deepest gradient scheme to the valley of this map. We can prove that we must end up at a critical point $C_{i'}, i' \in \{1, 2, \cdots, K\}$, whose arrival time is zero. Apparently the arrival time at P gives the *marching distance* between P and $C_{i'}$. In other words, $C_{i'}$ is the nearest critical point of P, where P has a known membership while $C_{i'}$ does not. Based on our previous discussion, we can assign the membership of P to $C_{i'}$.

   (c) Repeat **(a)**-**(b)** until $K = 0$. This iterative process can be seen from Figure 2.19(b), where initially A has known membership. All the other (red) critical points are assigned memberships by marching back to A. We call this process the *fast anti-marching method*, in contrast to the traditional fast marching method. Note also that the obtained graph of the critical points for each object may not necessarily form a linear path, but instead form a forest of trees in general.

3. **Multi-seeded fast marching method**
   Once we classify all the critical points (seeds), we can follow the traditional fast marching method except the following modifications. First, each object may correspond to a number of seeds instead of just one. Secondly, since each seed initiates a marching contour, we must attach to each seed (and accordingly, the marching contour) with a membership index based on the classification of seeds. Once a voxel is conquered by a marching contour, it should be assigned with the same index of the marching contour. Thirdly, two marching contours with the same index should merge into one when they meet, while two marching contours with different indexes should stop on their common boundaries. This multi-label idea has been used elsewhere (e.g., [268]), but apparently the detection and classification of seeds are quite different in our approach, as discussed above.

The above approach can be readily applied to the segmentation of layers from the whole molecular density maps. We shall give examples in Section 5. In the following we describe how this approach is used to segment asymmetric subunits of an icosahedral map.

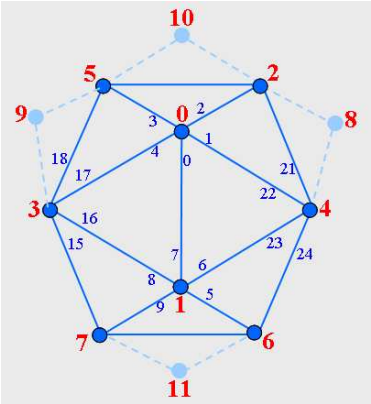


Figure 2.20: Index numbers of an icosahedral map

Asymmetric Subunits of Icosahedral Maps

The segmentation of asymmetric subunits of an icosahedral map follows the proposed variant of the fast marching method except that, in the second step (classification of critical points), we should take account of the icosahedral symmetry. In other words, once a critical point is assigned with a membership i ($i \in \{0,1,\cdots,59\}$), its corresponding 59 symmetric points (may or may not be a critical point) should be assigned with an index differently from $\{0,1,\cdots,i-1,i+1,\cdots,59\}$. We label the 12 symmetry axes, as detected in Section 3, as $\{S_j, j=0,1,\cdots,11\}$ and sort them such that $\{S_1,S_2,S_3,S_4,S_5\}$ are the nearest neighbors of S_0, $\{S_6,S_7,S_8,S_9,S_{10}\}$ are the next nearest neighbors of S_0, and S_{11} is opposite to S_0, as seen in Figure 2.20 (numbers in red). Initially, all the critical points have unknown membership (indexes). We assign 0 to the critical point, denoted by P, which has the highest density value among all the critical points. Then we find all the other 59 symmetric points of P and assign them with indexes orderly and differently from 1 to 59, based on the following pseudo-C code (see the blue numbers in Figure 2.20 for part of these indexes):

$$
\left\{
\begin{array}{l}
index = 1; \\
for\,(i = 1; i < 5; i{+}{+})\,\{ \\
\quad Q = R^0_{2i\pi/5}(P); \\
\quad \text{assign } index \text{ to } Q; \\
\quad index{+}{+};\} \\
\\
for\,(j = 1; j < 11; j{+}{+})\,\{ \\
\quad P' = R_{0\to j}(P); \\
\quad for\,(i = 0; i < 5; i{+}{+})\,\{ \\
\quad\quad Q = R^j_{(2i+1)\pi/5}(P'); \\
\quad\quad \text{assign } index \text{ to } Q; \\
\quad\quad index{+}{+};\}\} \\
\\
P' = R_{0\to 11}(P); \\
for\,(i = 0; i < 5; i{+}{+})\,\{ \\
\quad Q = R^{11}_{2i\pi/5}(P'); \\
\quad \text{assign } index \text{ to } Q; \\
\quad index{+}{+};\}
\end{array}
\right. \tag{1.42}
$$

where $R^j_\alpha(P)$ is the rotation of P about the axis S_j by an amount of α, and $R_{j_1\to j_2}(P)$ is the rotation of P from S_{j_1} to S_{j_2}. Note that the total number of critical points with unknown indexes should be deducted by one every time when Q is a critical point.

Equation 1.42 gives the first round of assignment, which guarantees that each of the 60 asymmetric subunits should have at least one critical point with membership assigned. Then we count all the critical points with unknown membership and if the number is positive, we should regard all such points as seeds and run the traditional fast marching method until one of the points with known membership is visited. Denote this point by A. Then, similar to **step (2.b)** of the algorithm discussed in

Section 4.1, we should start from A and trace back to the nearest critical point of A that has unknown membership. Denote this critical point by B. Apparently B should be assigned the same index of A. Now the question is how to assign the indexes to the other 59 symmetric points of B? If the index of B is 0, we can simply apply Equation 1.42 to B. Otherwise, we have to "map" B back to its symmetric point B_0 that corresponds to index 0 (Note that mapping B_0 to B is exactly what we did in Equation 1.42). Let k be the index of B, $j = k/5$, and $i = k - 5 * j$. We can see that j is the index of the symmetry axis that B corresponds to, while i indicates how much we should rotate about the specified axis S_j in order to map B_0 to B. The following algorithm finds B_0, given B:

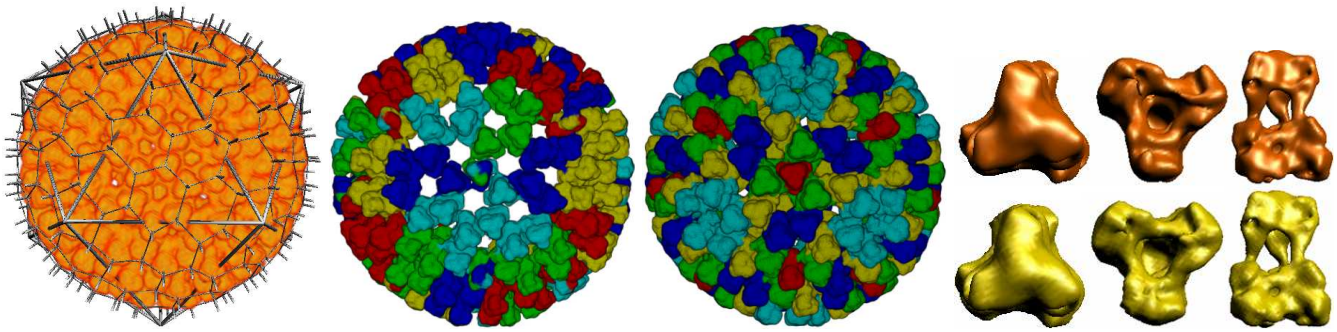
$$\begin{cases} if\,(j == 0) \\ \quad B_0 = R^0_{-2i\pi/5}(B); \\ \\ else\ if\,(j < 11)\{ \\ \quad B' = R^j_{-(2i+1)\pi/5}(B); \\ \quad B_0 = R_{j\to 0}(B');\} \\ \\ else\ if\,(j == 11)\{ \\ \quad B' = R^{11}_{-2i\pi/5}(B); \\ \quad B_0 = R_{11\to 0}(B');\} \end{cases} \tag{1.43}$$

where $R^j_\alpha(P)$ and $R_{j_1\to j_2}(P)$ are defined the same as those in Equation 1.42. Once we find B_0, we can assign 0 to B_0 and run Equation 1.42 by replacing P with B_0. The above steps are repeated until all critical points have been assigned their corresponding indexes. Finally we run the multi-seeded fast marching method, as discussed in Section 4.1, to segment the 60 asymmetric subunits.

Results

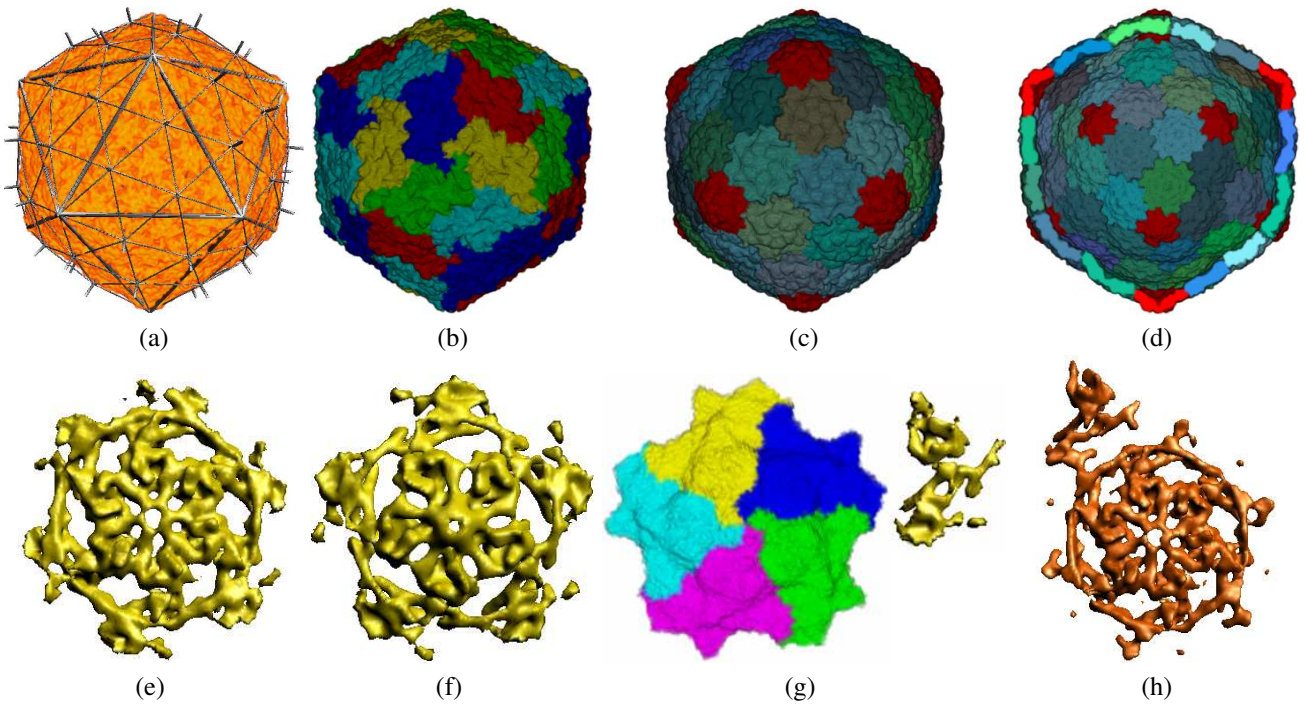
In this section we present results on two simulated icosahedral maps and one real map reconstructed using cryo-EM single particle method. All the segmentations, including seed generation and symmetry detection, are automatic, except that a few parameters, varying from data to data, are given by the users. Our first experiment is on the blurred map of the bluetongue virus model [144]. The crystal structure of this virus, consisting of two layers, is available on Protein Data Bank (PDB) [54]. Here we only consider the outer layer which has much more complicated geometry than the inner one. The triangulation number of this layer is $T = 13$ ($h = 3$ and $k = 1$). The file (PDB-ID = 2BTV) taken from PDB only provides an asymmetric unit of the structure. Hence we need to make 60 copies of this subunit to construct the complete shell. The transformation matrices are available on Viper [242] (http://mmtsb.scripps.edu/viper/). Once we obtain the full shell, we then blur the crystal structure into a lower resolution map at about 12 Å, which is volume-rendered in Fig. 2.21(a), where the symmetries detected by our method, including the icosahedral (global) symmetry and the 3-fold local symmetry, are also shown. The sixty asymmetric units are segmented using our approach and shown in Fig. 2.21(b) in five different colors. To better see the results, we chopped out a small portion of the sphere in the back. From this figure, we can see that each asymmetric unit consists of $4\frac{1}{3}$ trimers. This type of segmentation requires solely the global symmetry. To segment each trimer, we need to incorporate the local 3-fold symmetry into the segmentation and the results are shown in Fig. 2.21(c). This picture shows that there are five different types of subunits (colored differently) which have slightly different biochemical environments on the virus shell. To demonstrate the performance of our algorithm, we can compare one of the segmented trimers (bottom row in Fig. 2.21(d)) with the blurred map of the trimer of the crystal structure (top row in Fig. 2.21(d)). Visually, our results agree quite well with the true map.

The second example is also a blurred map generated from a crystal structure model but has a higher resolution. This structure is known as the bacteriophage HK97 (PDB-ID = 1FH6) [309]. The full shell is first constructed from the single subunit provided by PDB and then blurred at a resolution of about 7 Å, as shown in Fig. 2.22(a), where the symmetries detected by our method (including the icosahedral symmetry and the 6-fold local symmetry) are also shown. The triangulation number of this map is 7 ($h = 2$ and $k = 1$). The difficulty with this map is that the subunits are closely overlapped with each other, yielding indistinguishable boundaries between them. For this reason, the asymmetric units segmented by our algorithm may not be biologically accurate, see Fig. 2.22(b). However, by incorporating the 6-fold local symmetry into the segmentation, we can correctly segment all the pentons and hexons, colored respectively in red and other (randomly assigned) colors in Fig. 2.22(c) and (d). To inspect the accuracy of the automatic segmentation on this map, we consider one of the segmented hexons, shown in Fig. 2.22(e), and one of the segmented pentons, shown in Fig. 2.22(f). We then segment the monomers of the selected penton, shown in Fig. 2.22(g). The blurred map of the asymmetric unit of the crystal structure of bacteriophage HK97, consisting of



(a) (b) (c) (d)

Figure 2.21: Blurred map (512^3 voxels) of the bluetongue virus (PDB-ID = 2BTV) at resolution of about 12 Å. (a) Original map with symmetries detected. (b) Asymmetric units (60 in total). (c) Segmentation into trimers (260 in total). (d) Comparison between our segmentation result (bottom row) and the blurred map of the trimer crystal structure (top row). Left: top view; Middle: bottom view; Right: side view.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 2.22: Blurred map (400^3 voxels) of the bacteriophage HK97 (PDB-ID = 1FH6) at resolution of about 7 Å. (a) Original map with symmetries detected. (b) Asymmetric units. (c) Segmentation into pentons and hexons (viewed from outside). (d) Segmentation into pentons and hexons (viewed from inside). (e) One of the hexons by iso-surfacing. (f) One of the pentons by iso-surfacing. (g) Segmentation into monomers from the selected penton. Five colors isolate different monomers while one of the monomers is shown by iso-surfacing in the upper-right corner. (h) The iso-surface of the blurred map of the crystal structure, consisting of one monomer of penton and one full hexon.

one penton monomer and one full hexon (see Fig. 2.22(h)), is visually compared against the segmented monomer (upper-right of Fig. 2.22(g)) and the segmented hexon (Fig. 2.22(e)). We can see that both the monomer and the hexon segmented using our method agree well with the blurred map of the crystal structure, although they differ slightly in some areas probably due to the overlap of the subunits when the full shell is constructed.

The third example we would like to show in this paper is a true electron density map of the semliki forest virus [206], reconstructed at resolution 9 Å using the cryo-EM single particle technique. Fig. 2.23(a) shows the central slice of the 3D map. This map contains two icosahedral layers that we are interested in: one is the nucleocapsid layer and the other is the glycoprotein spike layer, shown in green and red respectively in Fig. 2.23(b). Given the original density map, we first need to segment different layers from the entire map. For some viruses, such as the semliki forest virus, this in general can be achieved solely based on the radial distance from the center. For some other viruses, in which the radial distances cannot separate the layers but the layers do have clear enough boundaries between each other, we can segment each layer without much difficulty using the segmentation method described in Section IV.A. Otherwise, it becomes much harder if different layers overlap each other too much (hence the boundaries become very hard to distinguish). In such cases, if two layers have different local symmetries, which happens quite often in many multi-layer viruses, we may be able to use the local symmetry scores to distinguish between the layers. Once we segment each icosahedral shell from the entire density map, the problem then becomes the segmentation of asymmetric units. Fig. 2.23(c) and (d) show the segmented icosahedral shells, corresponding to the capsid shell and the spike shell respectively, and the detected symmetry axes, including the global and local symmetries. Both shells have the same triangulation number: $T = 4$ where $h = 2$ and $k = 0$. Sixty asymmetric units of the capsid layer are shown in different colors in Fig. 2.23(e) (again, part of the shell in the back has been chopped out to avoid confusions). Fig. 2.23(f) and (g) show the segmented pentons and hexons of the capsid: one is viewed from outside and the other is viewed from inside. Each penton and hexon can be further segmented into individual monomer proteins, as shown in Fig. 2.23(h). Similarly, the asymmetric units of the spike layer can be segmented based only on the global symmetry (see Fig. 2.23(i)). The trimers are segmented using both the global and local 3-fold symmetries, as shown in Fig. 2.23(j) and (k). Furthermore, each trimer can be segmented into monomers based on its 3-fold symmetry, as seen in Fig. 2.23(l).

To demonstrate the better performance of our multi-seed fast marching method than the traditional fast marching method, we show an example in Fig. 2.24. We consider the P8 trimer pre-segmented from cryo-EM reconstructed Rice Dwarf Virus (RDV) map. Our task here is to segment P8 trimer into monomers using our method and the traditional method. The original P8 trimer map is shown in top and side views in (a) and (b), respectively. (c) and (d) show the segmentation of the trimer. Three monomers are indicated in different colors and a clipping plane is applied to both pictures. The top picture (c) is the result by our multi-seed method while the bottom one (d) is the result by the traditional method. We can see that, although both methods can segment the trimer into exactly symmetric monomers, the segmented monomers are spatially very different between our results and the results by the traditional method. (e) and (f) show another view of the same maps as (c) and (d), where (e) is the result by our method while (f) is obtained by the traditional method. The accuracy of both segmentation methods can be clearly judged from the segmented monomer maps. (g) is the segmented monomer by our method while (h) is the result by the traditional method. By Comparing the segmented monomer with the manually segmented monomer map [343], we can say our method performs much better than the traditional method.

There are five important parameters that the users must provide to the program: *h_num*, *k_num*, *t_high*, *t_low*, and $fold_num$. The first two parameters are the h and k numbers (as explained in Section III.A), used to calculate the number of independent local symmetry axes. The parameter *t_high* is used as a threshold to choose the subset of the critical points that will be used in the symmetry detection and segmentation. The parameter *t_low* is used as a stopping criterion in the segmentation procedure. When a marching contour collides with other contours with different membership, the marching is stopped. However, in some cases (e.g., the subunits facing outside) the marching contours (or part of them) may never collide with any other contours. In this case, the parameter *t_low* is used such that the marching contours stop when the intensities at the fronts are less than *t_low*. The last parameter $fold_num$ tells the program what type(s) of symmetries the subunits have. This kind of information will be incorporated into the asymmetric unit segmentation procedure.

Finally we would like to give a short summary on the timings of the segmentation on the three datasets we presented here. Depending on the complexity and size of the shells, the time for the three datasets varies from a few minutes (for the small and simple capsid layer of the semliki forest virus), to one hour (for the medial size and medial complexity such as the blurred map of bacteriophage HK97 which has 400^3 voxels), to four hours (for large maps with complex geometries such as the blurred map of the bluetongue virus which has 512^3 voxels). Again, all the timings are estimated based on our SGI Onyx2 with single processor (400 MHz MIPS R12000).

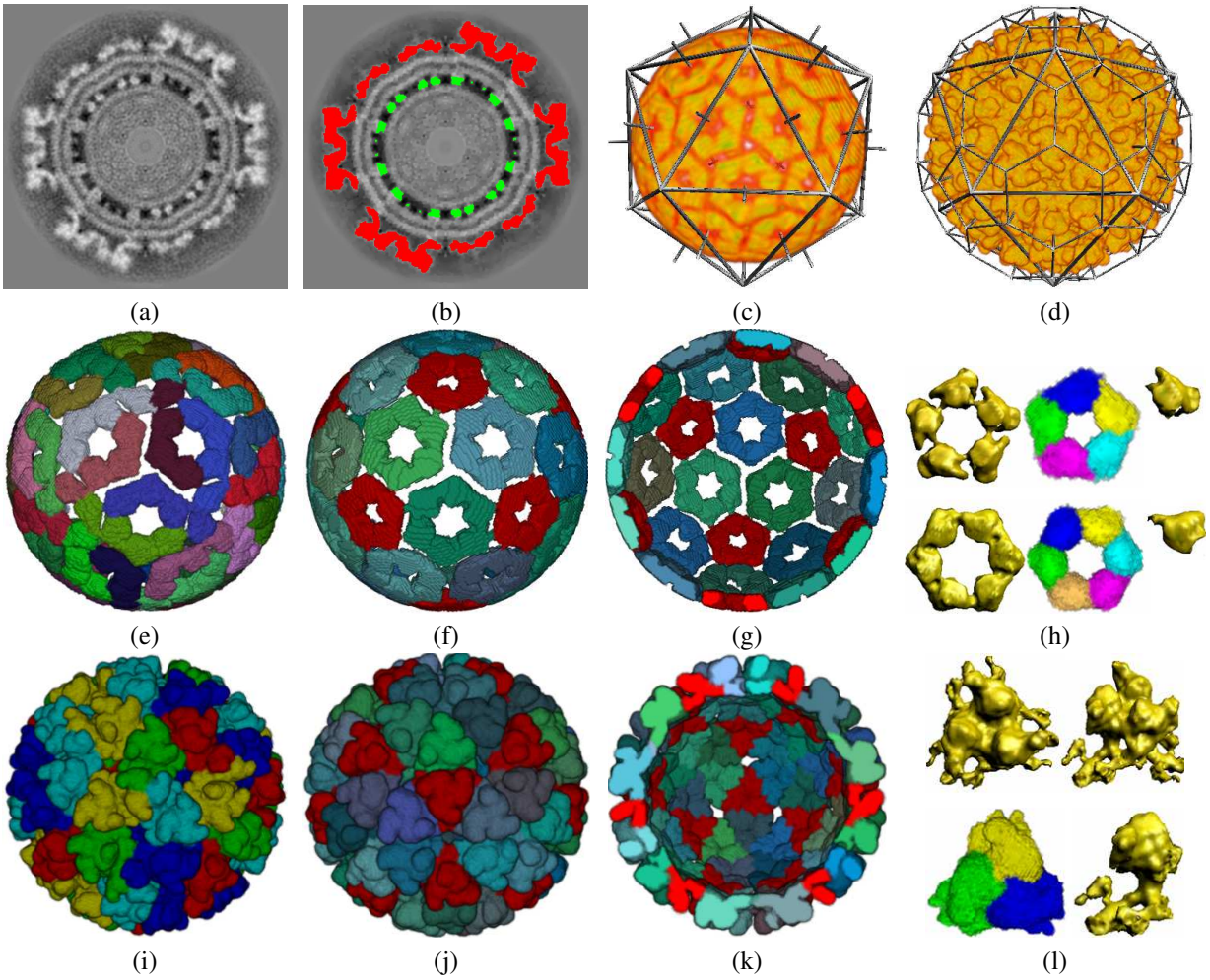


Figure 2.23: Semliki forest virus map (300^3 voxels, at resolution 9.0 Å). (a) Original map (middle slice). (b) Segmentation of capsid and spike layers (middle slice). (c) Capsid layer (3D volume-rendering) and its symmetries. (d) Spike layer (3D volume-rendering) and its symmetries. (e) Asymmetric units of the capsid. (f) Segmentation into pentons and hexons (from outside). (g) Segmentation into pentons and hexons (from inside). (h) Segmentation into monomers from penton and hexon. Left: penton/hexon (iso-surfacing); Middle: segmentation into monomers; Right: segmented monomers. (i) Asymmetric units of the spike. (j) Segmentation into trimers (from outside). (k) Segmentation into trimers (from inside). (l) Segmentation into monomers. Upper: two views of the trimer (by iso-surfacing). Lower: segmentation into monomers and one of the segmented monomers.

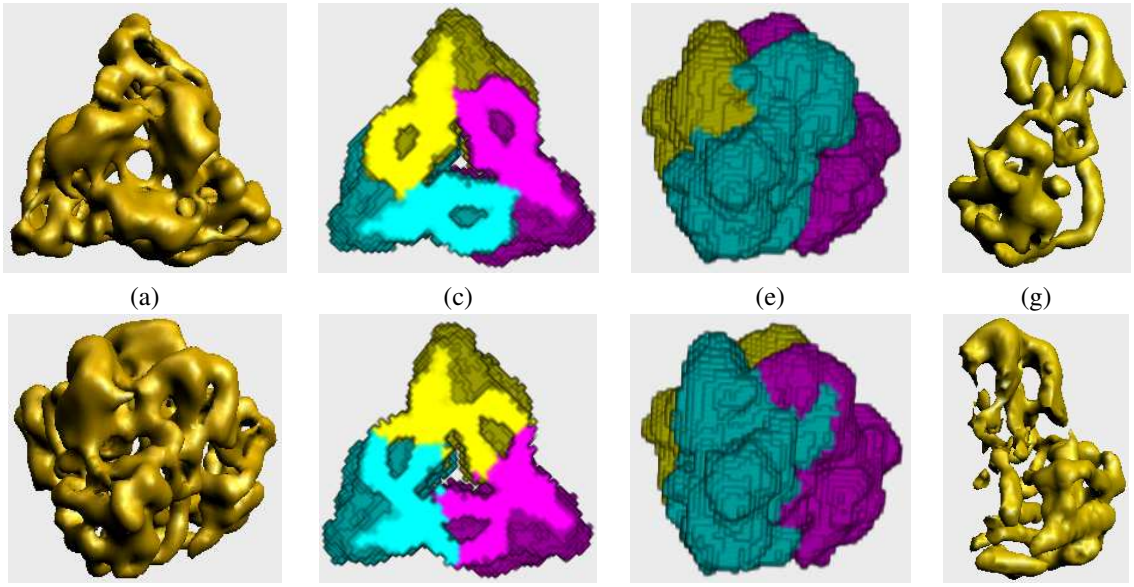


Figure 2.24: A 3D segmentation example demonstrating the improvements of our modified multi-seed fast marching approach over the traditional fast marching method. We consider the segmentation of Rice Dwarf Virus (RDV) trimer into monomers. (a)&(b) show the segmented trimer from cryo-EM reconstructed RDV map, in top and side views, respectively. (c)&(d) show the segmentation of the trimer. Three monomers are indicated in different colors and a clipping plane is applied to both pictures. The top picture (c) is the result by our multi-seed method while the bottom one (d) is the result by the traditional method. (e)&(f) show another view of the same maps as (c)&(d), where (e) is the result by our method while (f) is obtained by the traditional method. The accuracy of both segmentation methods can be clearly judged from the segmented monomer maps. (g) is the segmented monomer by our method while (h) is the result by the traditional method. By Comparing the segmented monomer with the manually segmented monomer map [343], we claim our method performs much better than the traditional method.

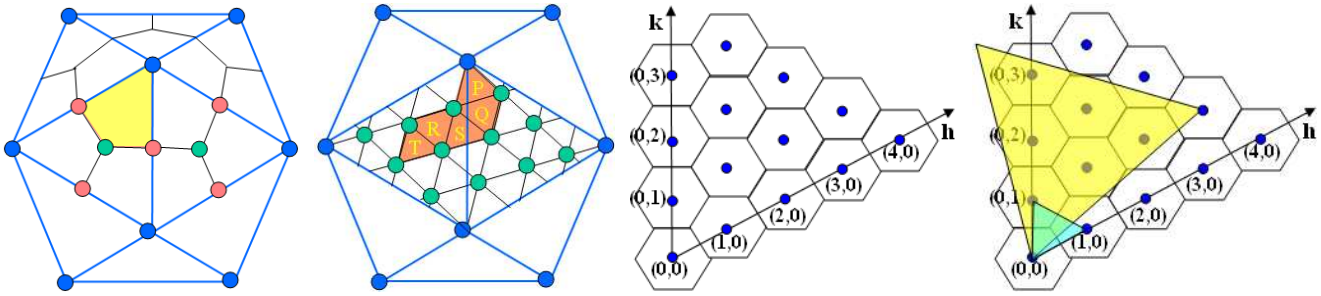
2.1.4 Symmetry Detection

Do we want Bajaj2005 (Geometric Processing of Reconstructed 3D Maps...) here? – TODO: Extract portions and merge with appropriate subsections.

The following is taken from [334]. (YuBajaj2005)

Symmetry Detection

The symmetry of a shape or object provides fundamental information for shape recognition. The problem of symmetry detection is hence critical in many applications, and has been extensively studied for decades in image processing and analysis [279, 210, 321, 336, 102, 264, 201]. Most of the past work, however, were focused on simple objects, e.g., points, curves, polygons. In case of the virus maps being studied in this paper, we presume that the maps exhibit an globally icosahedral symmetry. In addition to that, some local n-fold rotational symmetries are also frequently seen in many virus structures. In the following, we shall first give a brief review on the symmetry of a virus map. More details can be found in [49].



(a) ideal icosahedra (b) quasi-equivalence (c) hexagonal lattice (d) triangular number

Figure 2.25: Detection of 5-fold Symmetry axes

Symmetry of Virus

An ideal icosahedra has 12 vertices (5-fold), 20 faces (3-fold), and 30 edges (2-fold), as seen in Fig. 2.20(a). The simplest virus structure is hence composed of 60 identically interacting subunits. One of such subunits is indicated in yellow in Fig. 2.20(a). However, as pointed out by Caspar and Klug [70], when more than 60 subunits form a closed shell, they no longer share biochemically identical environments and hence have to self-organize into a quasi-equivalent icosahedra. In other words, not all of the subunits interact identically but they do share a similar biochemical environment. An example is illustrated in Fig. 2.20(b), where the viruses (e.g., rice dwarf virus) consist of five types of subunits. Interestingly the subunits form a quasi-equivalent icosahedra in a purely mathematical way, controlled by a magic number known as triangulation number [49]. This number is defined based on a hexagonal lattice (Fig. 2.20(c)). The coordinates of the lattice, denoted by h and k, define the triangular number by: $T = h^2 + hk + k^2$. The simplest case, where $T = 1$ ($h = 1$ and $k = 0$), gives the simplest virus structure, as shown in light blue in Fig. 2.20(d), which corresponds to the case shown in Fig. 2.20(a). A more complicated case, where $T = 13$ ($h = 3$ and $k = 1$) as shown in light yellow in Fig. 2.20(d), corresponds to the virus structure shown in Fig. 2.20(b). Although viruses demonstrate various local symmetries (mostly, 6-fold or 3-fold rotational symmetries), they do obey the icosahedral symmetry, which is globally defined on the whole maps.

In the following subsections, we shall describe how to detect the icosahedral symmetry as well as the local symmetries, followed by the performance analysis of the proposed algorithms. It is worthwhile pointing out that the icosahedral symmetry axes are in general available when the 3D maps are reconstructed through cryo-EM single particle technique. We put our efforts on it here for two reasons. First, our algorithm for local symmetry detection will be a slight modification of the one for global symmetry detection. We believe that it is better understandable to explain our symmetry detection algorithm first on the simpler case. Secondly, when the given maps do not have icosahedral symmetry available, our algorithm can be helpful.

Global Symmetry Detection

To detect the global (icosahedral) symmetry, we only need to locate all twelve 5-fold rotational symmetry axes. Without much trouble in real data, we assume that the center of the given map is always identical to the center of its corresponding icosahedra.

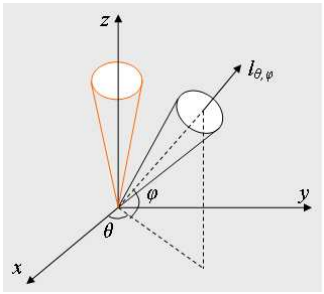


Figure 2.26: Rotation about an arbitrary axis

Given an axis $l_{\theta,\varphi}$ passing through the origin, where θ and φ are defined in a classical way (see Figure 2.26) such that $\theta \in [-\pi,\pi]$ and $\varphi \in [-\pi/2,\pi/2]$, a 3D scalar map $f(\vec{r})$ is said to have a 5-fold rotational symmetry about $l_{\theta,\varphi}$ if the following equation holds:

$$f(\vec{r}) = f(R_{(\theta,\varphi,2\pi/5)} \cdot \vec{r}), \quad \text{for} \quad \forall \vec{r} \tag{1.44}$$

where the 3×3 matrix $R_{(\theta,\varphi,\alpha)}$ is defined as the coordinate transformation that rotates a point counterclockwise about an axis $l_{\theta,\varphi}$ by an angle of α. In particular, the matrix $R_{(\theta,\varphi,\alpha)}$ is composed of five fundamental coordinate transforms:

$$R_{(\theta,\varphi,\alpha)} = A^{-1} \cdot B^{-1} \cdot \begin{pmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot B \cdot A \tag{1.45}$$

where matrices A and B are defined as:

$$A = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.46}$$

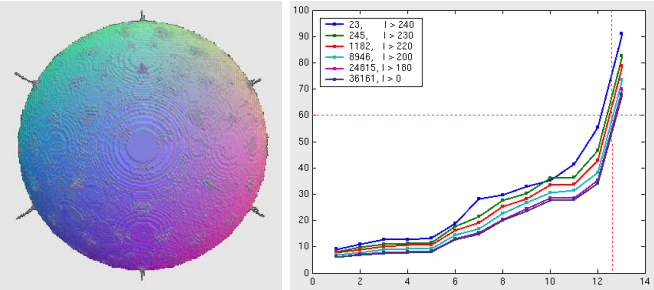
and

$$B = \begin{pmatrix} cos(\frac{\pi}{2}-\varphi) & 0 & -sin(\frac{\pi}{2}-\varphi) \\ 0 & 1 & 0 \\ sin(\frac{\pi}{2}-\varphi) & 0 & cos(\frac{\pi}{2}-\varphi) \end{pmatrix} \tag{1.47}$$

In order to detect all twelve symmetry axes, corresponding to the twelve vertices of an icosahedra, one can simply correlate the original map with its rotated map and search in the resulting correlation map for the peaks [210]. Obviously this method has a very high computational cost, as the time complexity is $O(NM)$, where N is the number of voxels and M is the number of angular bins used. In our application, N may be as big as 512^3 and M is about 46,000 (we take a uniform sampling on a sphere with a radius of 200-voxel size). Although a number of techniques can be employed to speed up the searching process by reducing the number of the angular bins (e.g., principal axis method [279] or hierarchical approach), it is still expensive if N is very big. In the following, we introduce another way for fast detection of rotational symmetries, given that the folding number is known. Basically the idea is to reduce N, the number of voxels to be tested, by restricting the correlation only to a subset of the critical points instead of the whole set of volume. Let us denote by $C_i, i = 1,2,\cdots,p$ all the qualified critical points and by $B_j, j = 1,2,\cdots,q$ all the angular bins. As detailed later, p could be very small by taking only a small subset of the critical points. The algorithm to detect the 5-fold rotational symmetry axes is described as follows:

1. **Compute the scoring function**
   For every $B_j, j = 1,2,\cdots,q$, compute the corresponding θ_j, φ_j and do **(a)**-**(b)**.

   (a) For every $C_i, i = 1,2,\cdots,p$, do **(i)**-**(ii)**.



(a) scoring function (b) six sets of critical points

Figure 2.27: Detection of 5-fold Symmetry axes

i. compute the rotated points of C_i:

$$\vec{r}_k(C_i, B_j) = R_{(\theta_j, \varphi_j, 2k\pi/5)} \cdot \vec{C}_i \tag{1.48}$$

where $k = 0, 1, 2, 3, 4$, and R is defined in Eq. 1.45.

ii. Calculate the deviation of $\{f(\vec{r}_k), k = 0, \cdots, 4\}$ by:

$$Dev(C_i, B_j) = \frac{1}{5} \sum_{k=0}^{4} |f(\vec{r}_k(C_i, B_j)) - \overline{f}| \tag{1.49}$$

where $\overline{f}$ is the average of $\{f(\vec{r}_k), k = 0, \cdots, 4\}$.

(b) Compute the scoring function for each bin B_j:

$$SF(B_j) = \frac{1}{p} \sum_{i=1}^{p} Dev(C_i, B_j) \tag{1.50}$$

Apparently a smaller score indicates a better chance for that bin to be selected as a 5-fold symmetry axis.

2. **Locate the symmetry axes**
Search in the scoring function map $SF(B_j), j = 1, 2, \cdots, q$ for the twelve angular bins with the smallest scores. However, these bins cannot be too close to each other. Otherwise, the one with larger score is discarded and the searching continues until all twelve bins are located.

As an example, we show in Figure 2.27(a) the (inverted and normalized) scoring function of the outer capsid layer of the rice dwarf virus (RDV) map [343]. The "peaks" with high contrast can be clearly seen and hence easily detected by conducting a "peak-searching" procedure.

Performance Analysis

To demonstrate the performance of our algorithm, we test on the outer capsid layer of RDV map [343] with different numbers of critical points. The total number of critical points on this map is 36,161. We can choose only a subset of the critical points by requiring that the density on the critical points should be larger than a threshold (e.g., $I > t_0$). In Figure 2.27(b), six sets of critical points are tested. For each set, we can detect the twelve best symmetry axes by comparing their scores. In Figure 2.27(b), the scores of the selected symmetry axes are shown in the lower-left window, while the maximal score over all the angular bins for each of the six tests is shown in the upper-right window. The legend gives the number of critical points and the corresponding threshold for each of the six tests, provided that the original density map is normalized to [0, 255]. From this figure, we can see that the number of critical points does not significantly affect the resulting scoring functions. Interestingly the detected symmetry axes (data are not shown here) agree perfectly on all six tests, meaning that the total displacement of all twelve axes on the sampling sphere is less than 0.001 voxel between any two of the tests.

The following is taken from [34] (BajajWu_noyear).

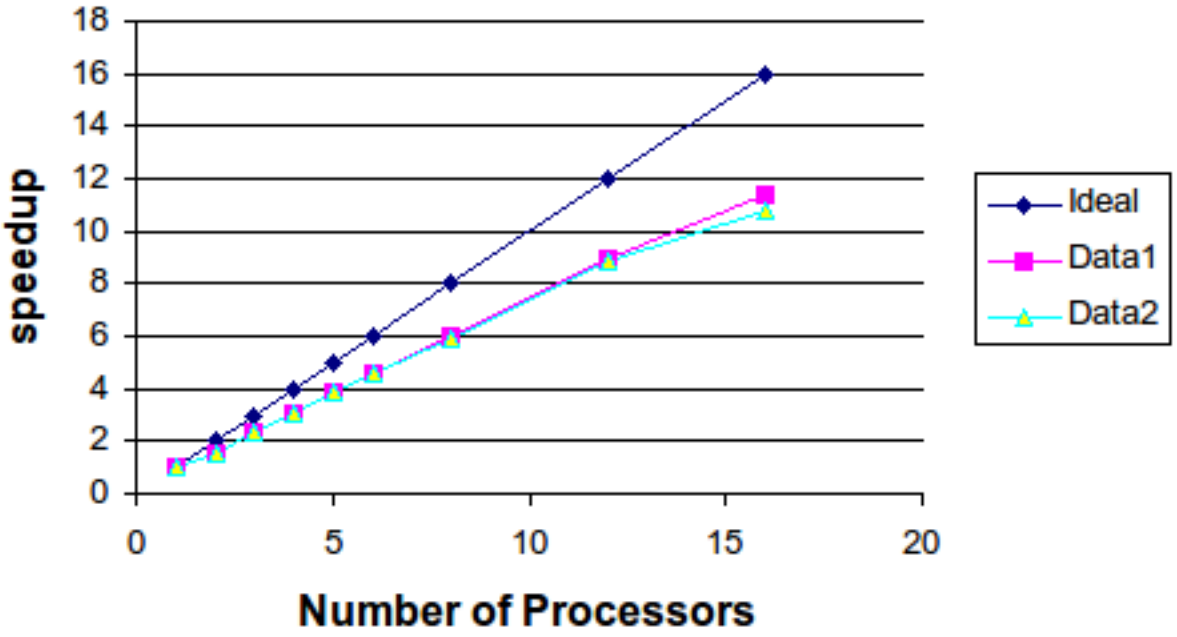


Figure 2.28: Speedup factor as function of number of processors.

Local Symmetry Detection

Applying FEM on (1.1) leads to the problem of solving numerically linear equation

$$Ax = b \tag{1.51}$$

Due to the problem size (A is a n×n matrix, n being the number of voxels of 3D image). Assemblying stiffness matrix A and vector b is a computationally intensive process, we try to parallelize matrix assemblying (for matrix A) and matrix-vector multiplication (for vector b) in parallel. Because the base functions in (1.10) are linearly independent and have compact support, matrix A is highly sparse matrix, solving (1.17) exhibits an inherent parallelism. Bilateral prefiltering is also suitable for parallelization because of it is only involved with local data. So, evolution of (1.2) exhibits a high degree of data locality and parallelism and maps quite readily to specialized massively parallel computing hardware.

Our code is implemented using a server/slave scheme, a sever program reads an image file from disk, then partitions data into equally sized pieces, and sends the pieces to programs running on slave machines in the cluster.

To evaluate a local function $\Phi(x,y,z,t)$, each processor requires its local portion of the vector X as well as its ghost values – or bordering portions of X that are owned by neighboring processors.

Data distribution also has impact on parallel performance. We employ slice partitioning as partition scheme and each processor locally owns a sub-matrix of contiguously numbered global slices. This trivial partition scheme achieves load balance among processors and can make ghost point updates and indexing easy.

From (1.13),(1.14) and (1.15) we can see that both lumped mass matrix M and nonlinear stiffness matrix L are symmetric, matrix M is positive definite, and matrix L is nonnegative definite. So matrix A is symmetric and positive definite. The well-known conjugate gradient(CG) with preconditioning is used to solve (1.13)[255], each processor solves the sub-vector it owns. At the end of each iteration, each processor has to update ghost points via communicating with its neighboring processor(s), so each processor has to wait until all other processors complete computing, one slower processor will slow down the performance of whole system, which is overhead of parallel solution.

It should be noted that parallel solutions from multiple processors converge to the unique solution, see Fig. 6 for our experiment results.

The timing results in Table 1 are done on a 129×129×129 size Kidney dataset.

2.1.5 Compression

The following is taken from [16] (Bajaj05).

The molecular surface of bio-molecules can also be generated as an iso-surface (implicit solvation model) of a smooth volumetric electron density function representation. The accurate computation of electron density representations from the PDB requires computations at the quantum mechanical level ([62]). One usually approximate the electron density distribution of the i^{th} atom with a Gaussian function ([215, 143, 58]) as $\rho_i(\mathbf{r}) = \exp\left(\frac{\mathscr{B}r^2}{R_i^2} - \mathscr{B}\right)$, where $\mathscr{B} < 0$ is the rate of decay parameter, R_i is the Van der Waals radius of the i^{th} atom and $r^2 = (x-x_{ci})^2 + (y-y_{ci})^2 + (z-z_{ci})^2$ ($\{x_{ci}, y_{ci}, z_{ci}\}$ is the center of the i^{th} atom.). A volumetric representation of the molecule may now be obtained by summing the contributions from each single atom, thus the electron density $I(x)$ for M atoms is described as

$$I(x) = \sum_{k=1}^{M} \rho_k(r) = \sum_{k=1}^{M} e^{(\frac{\mathscr{B}r^2}{R_k^2} - \mathscr{B})}. \tag{1.52}$$

Volumetric maps of bio-molecules are additionally reconstructed from either tomographic electron microscopy reconstruction or from single particle reconstructions via Cryo-Electron Microscopy techniques. Several images are taken and averaged to produce high resolution of below 10 angstroms. The molecular surface of such reconstructed maps of bio-molecules can also be generated as an isosurface of this map. In Figure 2.29(f) an example of the Rice Dwarf Virus (RDV) map, reconstructed from single particle Cryo-Electron Microscopy, is shown.

Isosurfaces can be computed and visualized from these volumetric maps by the well-known Marching cubes algorithm ([200]) which extract isosurfaces from scalar fields by traversing and triangulating each mesh cell. There has been work done to accelerate searching for only contributing cells ([29, 197, 310]) and later extended to time-varying fields ([265, 280, 33]). In Figure 2.29(a-c) examples of the isosurface of the electron density map of the ribosomal subunit 50S are shown at different resolutions.

Some of the relevant molecular properties that we deal in this paper are hydrophobicity and electrostatics. We shall briefly describe how they where calculated.

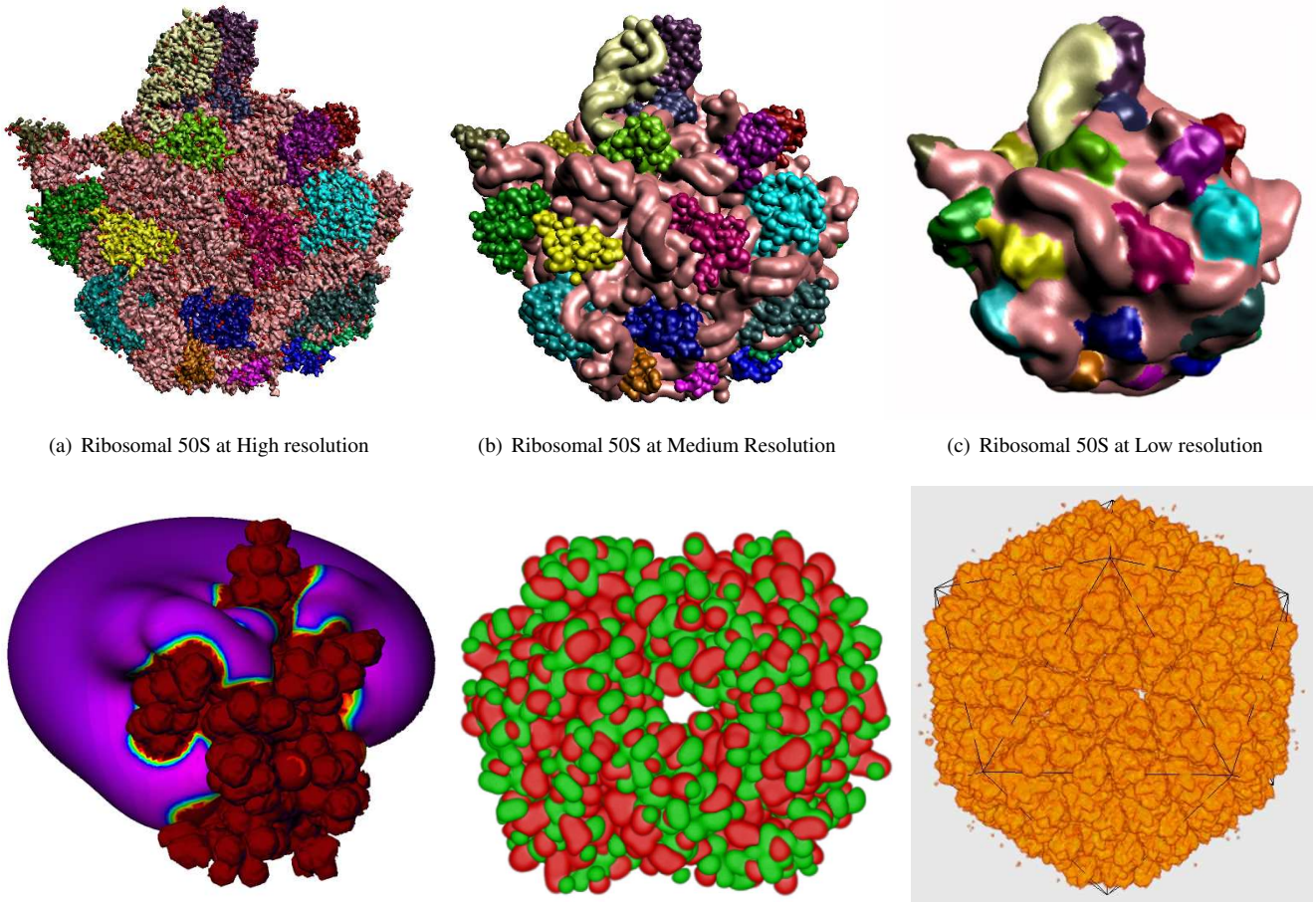
The affinity towards water by different parts of the molecule plays an important part in its interaction with itself in folding and other proteins in docking. Hydrophilic / hydrophobic regions are those which have an affinity / dislike to water. It is generally seen that hydrophobic regions tend to align together, away from the solvent boundary. If we are given per atom hydrophobicity values h_i, for each atom i in a protein, one can define the hydrophobicity function ϕ as shown below

$$\phi(x,y,z) = \sum_{i=1}^{M} h_i e^{\alpha_i}$$

where $\alpha_i = \mathscr{B} \times \frac{(x-x_{ci})^2+(y-y_{ci})^2+(z-z_{ci})^2}{R_i^2} - \mathscr{B}$. An example of the hydrophobicity map is shown in 2.29(e).

Associated properties of bio-molecules such as electrostatic potential fields are computed by solving the Poisson-Boltzmann equation. The numerical solutions to the equation we use here for as examples were produce by the Adaptive Poisson-Boltzmann Solver (APBS) code ([48]). Again isosurface extraction and rendering provide one way of visualizing these maps. In Figure 2.29(d) an isosurface of the electrostatic map of glucose is produced. Electrostatic Poisson solvation energy estimations can also be created from the Generalized Born model ([189]).

The rest of the paper is as follows: In our main section 2.1.5 we first show how to construct HB wavelets of all polynomial orders and then concentrate on a linear HB wavelet scheme as one possessing near very good tradeoffs with compression rates and decompression times. An encoding and decoding scheme is described to achieve fast random access. Additionally, we derive error bounds on the accuracy of the linear HB wavelet representation for general molecular maps and synthetic Radial Basis Function (RBF) molecular maps. In the penultimate section 2.1.5 we present results of our volumetric compression results for various resolutions of bio-molecules and their associated properties along with rate distortion comparisons with non-HB wavelet methods. In Appendix A we introduce a fully error bounded HB wavelets compression scheme and in Appendix B, we derive a-priori mesh width estimations and local wavelet updates for synthetic Gaussian maps of varying resolutions.



(a) Ribosomal 50S at High resolution

(b) Ribosomal 50S at Medium Resolution

(c) Ribosomal 50S at Low resolution

(d) Electrostatics Potential

(e) Hydrophobicity of hemoglobin. Green - hydrophobic. Red - hydrophilic.

(f) Rice Dwarf Virus Cryo-Em

Figure 2.29: Visualization of Biomolecules and associated properties. (a)-(c) Electron density map at different resolution levels for the Ribosomal 50s Large Subunit. Each color represents a different chain in the molecule. For example, pink and light yellow correspond to the RNA.(d) Electrostatic potential field for glucose. (e) 256^3 Hydrophibicity map of hemoglobin, where green represents the hydrophobic part and red the hydrophilic. (e) Cryo-EM map of Rice Dwarf Virus. Note that (a)-(e) are synthetic and (f) is a natural map.

Results and Discussion

In this section we construct our interactive wavelet HB compressed molecular representation and apply it to a series of natural and synthetic molecular maps. First, we discuss wavelet theory and **construction the HB representation**. Second, a zero-bit **encoding, decoding scheme** to achieve **fast random access** is derived for this basis. Finally, we obtain **error bounds and timing, visualization results**.

Wavelet Function Spaces with Hierarchical Basis We construct Hierarchical Basis (HB) ([327]) over 3D domains in the context of *second generation wavelets* ([282]). In the following subsections 2.1.5-2.1.5 we show the construction of HB for the general case. In section 2.1.5, linear HB is explicitly constructed. In section 2.1.5 error bounds and our compression strategy are discussed, then followed by an appropriate encoding scheme in section 2.1.5.

Construction of the Hierarchical Basis Our objective is to explicitly build wavelets in three dimensions with the following desirable properties:

- *Adapted to the local geometry.*
- *Easy to implement.*
- *high degree of accuracy.*
- *high degree of compression.*

To this end we construct a class of interpolating wavelet bases built from piecewise polynomials and of compact support from the class of second generation wavelets, introduced by Sweldens ([282]). Many of the details of the following section can be found in ([282, 72, 71, 5]). First we shall establish some useful notation.

Suppose we partition the domain $S_{0,0} = S \subset R^{\eta}$ into the disjoint union (partition) of K_0 parts $\{S_{1,0}, \ldots, S_{1,K_0}\}$. Every single part $\{S_{1,i}\}$ is further divided into a partition of the form $\{S_{2,0}, \ldots, S_{2,K_1}\}$ for some K_1. This process is repeated ad infinitum for every single partition. Now that we have partitioned our domain in a series of "multiresolution" patches, we define a set of nodes on each $\{S_{j,v}\}$. Denote $x_{0,k}$ as the nodes on the level zero partition $S_{0,0}$ were $k = 1...N_0$, for some positive integer N_0. Similarly to the multiresolution partition decomposition, more nodes such as $x_{1,k}$ are added to all the level 1 partitions $\{S_{1,i}\}$. This process is then repeated for all $\{S_{j,v}\}$ ad infinitum.

Define $K(j)$ as the index set at each level j of refinement and $X_j = \{x_{j,k} \in S | k \in K(j)\}$ be the set of vertex values defined over the index set $K(j)$. The vertices are constrained such that $X_j \subset X_{j+1}$ and $x_{j,k} = x_{j+1,k}$. The index sets are also constrained such that $K_j \subset K_{j+1}$ and the difference between the index sets from one level to another is given by $M(j) = K(j+1) \backslash K(j)$. Now that we have a multi resolution partitioning of our domain we are interested in constructing an efficient functional representation of an arbitrary function $f(\cdot)$ over the domain S. To this end we construct a series of basis function $\phi_{j,k}$ and nested *scaling function* subspaces V_j such that

$$V_j = \text{closure span}\{\phi_{j,k} \mid k \in K(j)\} \quad \text{and} \quad V_j \subset V_{j+1},\ \forall j \in Z^*.$$

This implies that each basis function $\phi_{j,k}$ at level j can be written as a linear combination of $\phi_{j+1,k}$ from level $j+1$:

$$\phi_{j,k} = \sum_{l \in K(j+1)} h^0_{j,k,l} \phi_{j+1,l},\ k \in K(j). \tag{1.53}$$

Now, let the *wavelets* subspaces $W_j = \text{closure span}\{\psi_{j,m} \mid m \in M(j)\}$ be the difference between V_j and V_{j+1}, i.e.

$$V_{j+1} = V_j + W_j.$$

Since $W_j \subset V_{j+1}$ then

$$\psi_{j,m} = \sum_{l \in K(j+1)} h^1_{j,m,l} \phi_{j+1,l},\ m \in M(j). \tag{1.54}$$

The wavelet and scaling function subspaces are closely linked to a series of *dual* subspaces. Similar to wavelets and scaling functions we define a set of *dual scaling functions* $\tilde{\phi}_{j,k}$ such that

$$\tilde{V}_j = \text{closure span}\{\tilde{\phi}_{j,k} \mid k \in K(j)\} \quad \text{and} \quad \tilde{V}_j \subset \tilde{V}_{j+1},$$

for all $j \in Z$. Moreover, we define a series of *dual wavelet subspaces*

$$\tilde{W}_j = \text{closure span}\{\tilde{\psi}_{j,m} \mid m \in M(j)\},$$

such that

$$\tilde{V}_{j+1} = \tilde{V}_j + \tilde{W}_j.$$

It can now be seen that the dual scaling and wavelet functions satisfy

$$\begin{aligned} \tilde{\phi}_{j,k} &= \sum_{l \in K(j+1)} \tilde{h}^0_{j,k,l} \tilde{\phi}_{j+1,l},\ k \in K(j) \\ \tilde{\psi}_{j,m} &= \sum_{l \in K(j+1)} \tilde{h}^1_{j,m,l} \tilde{\phi}_{j+1,l},\ m \in M(j). \end{aligned}$$

The subspaces $V_j, W_j, \tilde{V}_j$ and $\tilde{W}_j$ are all interrelated and satisfy the following biorthogonal conditions

$$V_j \perp \tilde{W}_j, W_j \perp \tilde{V}_j. \tag{1.55}$$

In addition, the basis functions $\phi_{j,k}, \tilde{\phi}_{j,k}, \psi_{j,m}$ and $\tilde{\psi}_{j,m}$ satisfy the orthogonality conditions:

$$\langle \phi_{j,k}, \tilde{\phi}_{j,l} \rangle = \delta_{k-l}, \quad \langle \psi_{j,m}, \tilde{\psi}_{j,l} \rangle = \delta_{m-l}, \tag{1.56}$$

where

$$\delta_{k-l} = \begin{cases} 1 & k = l \\ 0 & k \neq l \end{cases}.$$

We can now represent an arbitrary function $f(\cdot) \in V_j$ as a linear combination of the single scale basis functions

$$f = \sum_{k \in K(j)} \lambda_{j,k} \phi_{j,k} + \varepsilon,$$

where $\lambda_{j,k} = \langle f, \tilde{\phi}_{j,k} \rangle$ and some small error ε, or in terms of its equivalent *multi resolution* representation:

$$V_j = V_0 \oplus \sum_{i=0}^{j-1} W_i, \Rightarrow f = \sum_{k \in K(0)} \lambda_{0,k} \phi_{0,k} + \sum_{i=0}^{j-1} \sum_{m \in M(i)} \gamma_{i,m} \psi_{i,m} + \varepsilon,$$

where $\gamma_{j,m} = \langle f, \tilde{\psi}_{j,m} \rangle$. In general, the multi resolution form leads to a more compact representation of the function f.

In Figure 2.30(a) an example of a simple multiresolution representation over 1D domain is shown. The primary scaling and wavelet equations are well suited for the representation of smooth functions in f with linear accuracy. Notice, that the duals are Dirac's and in general will not be well suited for functional representation. Other smoother duals can be constructed, (see Chapter 3 in [71]). However, the duals will be used implicitly for showing that the multi resolution decomposition of the primary functions exists and for estimating the decay of the wavelet coefficients.

This is one of the most simple wavelet representations and has been successfully employed for adaptive meshing and preconditioning of finite element matrices ([327]). Moreover, in a departure to traditional wavelet theory, biorthogonality and the wavelet structure is preserved even if the nodes are irregularly placed. This class of wavelet is an example of an interpolating wavelet basis. In the following section we shall concentrate further into this class of wavelets and show how to construct linear interpolating wavelets in R^3.

Interpolating wavelets basis An interpolating function $\phi_{j,k}$, satisfies the following condition

$$\phi_{j,k}(x_{j,k'}) = \delta_{k-k'}, \forall k, k' \in K(j).$$

Since all the scaling functions are interpolating the refinement equation (1.53) may be written in a more compact form as

$$\phi_{j,k} = \phi_{j+1,k} + \sum_{m \in n(j,k)} h^0_{j,k,m} \phi_{j+1,m}, \ \forall j \text{ and } \forall k \in K(j), \tag{1.57}$$

where the local neighborhood $n(j,k)$ is defined as follows:

- $n(j,k) \subset M(j)$ corresponds to the set of indices in $M(j)$ needed to construct $\phi_{j,k}$.

Similarly we define the counterpart to $n(j,k)$ as

- $N(j,m) \subset K(j)$ is equal to $\{k \in K(j) \mid m \in n(j,k)\}$.

$N(j,m)$ will be useful to apply the wavelet transforms.

For the linear interpolating basis shown in Figure 2.30 only two coefficients are needed to satisfy the refinement equation (1.57), thus for each j and k, $n(j,k) = \{m_1, m_2\}$ and the coefficients $h^o_{j,k,m_1} = h^o_{j,k,m_2} = \frac{1}{2}$, see Figure 2.30(b). In figure 2.30(b) we show how to construct the neighborhood $N(j,m)$.

Assuming that we can find a series of scaling functions and coefficients $h^0_{j,k,m}$ that satisfy the refinement equation (1.57) a simple *interpolating* wavelet can be constructed. At every single node $m \in M(j)$ choose $h^1_{j,m,l} = \delta_{l,m}$; this implies the wavelet $\psi_{j,m} = \phi_{j+1,m}$. Although we can build more complex wavelets by applying the *lifting scheme* ([281]), in this paper we restrict ourselves to this simple wavelet.

Having set up the framework for interpolating wavelets we can move from single scale at level j to a multi scale representation for an arbitrary function f by applying the wavelet transform

$$\begin{aligned} \forall m \in M(j), \;\; \gamma_{j,m} &= \lambda_{j+1,m} - \sum_{k \in N(j,m)} h^0_{j,k,m} \lambda_{j+1,k}. \\ \forall k \in K(j), \;\; \lambda_{j,k} &= \lambda_{j+1,k}. \end{aligned}$$

or vice versa by applying the inverse wavelet transform

$$\begin{aligned} \forall k \in K(j), \;\; \lambda_{j+1,k} &= \lambda_{j,k}, \\ \forall m \in M(j), \;\; \lambda_{j+1,m} &= \gamma_{j,m} + \sum_{k \in N(j,m)} h^0_{j,k,m} \lambda_{j+1,k}. \end{aligned}$$

(See [282] for a proof). Both transforms require only $\mathcal{O}(N)$ computations, where N is the total number of vertices at level $K(j)$. Note that if $f \in V_n$, then $\lambda_{n,k} = \langle f, \tilde{\phi}_{j,k} \rangle = f(x_{j,k})$. The last step allows us to to approximate f by using samples of the function f at the highest resolution level n and avoid explicitly constructing the dual scaling functions (see [72] on how to construct them), this leads to a more simple implementation. In the following section we describe a large class of coefficients $h_{j,k,m}$ that satisfy equation 1.57.

Spatially Adapted MultiWavelets We now flesh out the framework introduced in the previous section, by showing how to construct interpolating piecewise scaling functions of arbitrary degree. Such constructions are introduced in ([72]).

Let $\{S_{j,v}\}$ be a multi resolution partitioning of the domain $S \subset R^3$ and choose a set of linearly independent functions $\{P_i(x)\}$, $i = 0, 1, \ldots, M-1$. Moreover, for all the nodes $x_{z,k}$ in $S_{j,v}$, $z \geq j$, all the neighboring nodes $x_{z+1,m}$, such that $m \in n(z,k)$, are contained in $S_{j,v}$. Furthermore, assume that the number of elements of $N(z,m)$ is equal to some positive integer M. If the collection of nodes $\{x_{z,k}; z \geq j\}$ in $S_{j,v}$ are distinct and form a dense subset in $S_{j,v}$ then there exists an interpolating scaling function (see [72] for a proof)

$$\phi_{j,k}(x) = \sum_{i=0}^{M-1} a_i P_i(x) \qquad a.e.$$

over every $S_{j,v}$ for some set of unique coefficients $\{a_0, a_1, \ldots, a_{M-1}\}$, such that

$$\sum_{i=0}^{M-1} a_i P_i(x_{j,k'}) = \begin{cases} 1 & k' = k \\ 0 & k' \neq k \end{cases} \quad k' \in K(j). \tag{1.58}$$

and

$$\int_S P_i(x) \tilde{\psi}_{j,k}(x) \, dS(x) = 0. \tag{1.59}$$

The support of $\phi_{j,k}$ is equal to union of all the neighboring partitioning of the node k and the coefficients of the refinement equation are given by

$$h^0_{j,k,l} = \begin{cases} \delta_{k-l} & l \in K(j) \\ \sum_{i=0}^{M-1} a_i P_i(x_{j,l}) & l \in n(j,k), x_{j,l} \in S_{j,v} \end{cases}. \tag{1.60}$$

The wavelets may now be constructed by letting $\psi^{old}_{j,m} = \phi_{j+1,m}$. In this paper, for simplicity, we construct a linear interpolating wavelet basis.

Construction of linear Volumetric Rectangular Wavelets on 3D domains We now have the framework to construct a multi-resolution decomposition of a volumetric mesh of the form

$$V_j = V_0 \oplus \sum_{i=0}^{j-1} W_i,$$

over trilinear functions. Such constructions provide high degree of accuracy while simple to implement.

Linear Scaling function construction

1. Choose the order of approximation of the scaling functions. For a linear approximation basis we choose $\{P_0,\ldots,P_7\} = \{1,\ x_1,\ x_2,\ x_3,\ x_1x_2,\ x_1x_3,\ x_2x_3,\ x_1x_2x_3\}$, thus M = 8.

2. Denote the initial level of resolution $j = 0$ and define an initial coarse cube mesh C_0 scaled to one.

3. Subdivide the mesh equally into eight new cubes and form the set C_1 as all the cubes at resolution level 1.

4. Place all 27 nodes at level $K(1)$ on the cubes C_1. The eight nodes of $k \in K(0)$ are put on on each corner of C_0 and the rest of the 19 nodes $m \in M(1)$ are placed according to the pattern in Figure 2.31(a) i.e. mid points. All, 19 nodes $m \in M(1)$ form $n(0,k)$ and $h^0_{0,k,l} = \sum_{i=0}^{M-1} a_i P_i(x_{0,l})$, $\forall l \in n(j,k)$. Since $\{a_0,\ldots a_{M-1}\}$ are chosen to make the trilinear polynomial over C_0 interpolating, only six nodes are non-zeros, which correspond to:

   - $h^0_{0,k,l} = \frac{1}{2}$ if $l = a$ (edge node)
   - $h^0_{0,k,l} = \frac{1}{4}$ if $l = b$ (face node)
   - $h^0_{0,k,l} = \frac{1}{8}$ if $l = c$ (interior node)
   - $h^0_{0,k,l} = 0$ otherwise.

   (See Figure 2.31(a)). The neighborhood $N(0,m)$ cardinality is also M, however, in practice we also need less since the polynomial $\sum_{i=0}^{M-1} a_i P_i(x)$ is zero at many of the corners. Thus, depending if the node is an edge, the interior of a face of the neighborhood $N(0,m)$ is chosen accordingly. See Figures 2.31(b), (c) and (d).

5. For each of the subdivided cube at level C_1 we repeat the above steps and place the new nodes $m \in M(1)$. The neighborhoods $N(1,m)$, $n(1,k)$ and the coefficient $h^0_{1,k,l}$ are are also calculated on the subdivided cubes.

6. The process is repeated ad infinitum such that the set of $\{x_{j,k}\}$ forms a dense subset of the volumetric domain. Notice that each V_j on C_j has a dimension of $(2^j+1)^3$.

7. Apply the spatially adapted multiwavelet existence theorem from section 2.1.5 (or [72]). This implies that every cube $v \in C_j$ consists of a trilinear polynomial. The scaling functions $\phi_{j,k}$ at any vertex $x_{j,k'}$ is equal to the union of all neighboring cubes in C_j. Denote this union of cubes as $C^{\phi}_{j,k}$.

Linear Wavelet Construction

With all the scaling functions and refinement equation constructed a simple wavelet can be formed at every single node $m \in M(j)$, by letting $h^1_{j,m,l} = \delta_{l,m}$. This implies the wavelet has the form $\psi_{j,m} = \phi_{j+1,m}$ and support $C^{\psi}_{j,m} = C^{\phi}_{j+1.m}$.

Wavelet transform

We can now interchange between the single scale and multi scale representations $V_n \leftrightarrow V_0 \oplus \sum_{i=0}^{n-1} W_i$ by applying the wavelet transform. Since $\lambda_{n,k} = f(x_{n,k})$ are the coefficients for the trilinear approximation at the highest resolution n and $\lambda_{j,k} = \lambda_{j+1,k}$ for all $j \le n$ then

$$\forall m \in M(j),\ \ \gamma_{j,m} = f(x_{j+1,m}) - \sum_{k \in N(j,m)} h^0_{j,k,m} f(x_{j+1,k})$$

and the inverse

$$\forall m \in M(j), \;\; f(x_{j+1,m}) = \gamma_{j,m} + \sum_{k \in N(j,m)} h^0_{j,k,m} f(x_{j+1,k})$$

since $\lambda_{n,k} = f(x_{n,k})$ are the coefficients for the trilinear approximation at the highest resolution n. Notice, that since $\gamma_{j,m}$ only depends on samples of the density function the wavelet transform can be done in any order. In fact, the wavelet transform at level j is exactly a trilinear interpolation of the data at level j minus the values of data at the node of the wavelets, which is exactly the interpolation error for a trilinear grid.

Adapted Compression Strategy for General Volumetric Maps The compression scheme can be broken in two steps. First, the data has to be sparsified using a suitable functional representation as described in this section. The second step is to produce an *encoding* which tells where the non-zero terms are located. In this paper we adapt the zero-bit encoding scheme that we have developed ([25]) to HB.

The linear HB is well suited for the representation of smooth data with localized discontinuities. We shall first derive some bounds on the decay of the wavelet. These bounds indicate that the wavelets we have constructed are very well suited to compress data on a local and global scope. We also show some strategies for compression of molecular data.

Let $f : V_n(C) \subset \mathbb{R}^3 \rightarrow \mathbb{R}$ be a density function twice differentiable defined on a unit cube C. Let n be the maximum multi-resolution level and $V_1 \subset V_2 \subset \ldots V_n$ the scaling function spaces defined on C, then the density function can be expanded in the single scale formulation in V_n as

$$f = \sum_{k \in K(n)} \lambda_{n,k} \phi_{j,k} + \varepsilon.$$

For the interpolating projection of f onto V_n the coefficients are equal to $\lambda_{n,k} = f(x_{n,k})$ for all $k \in K(n)$. For the previously derived trilinear function space V_n the global error ε is bounded as

$$|\varepsilon| \leq 2^{-2n-3} \left(\| \frac{\partial^2 f}{\partial x_1^2} \|_{\mathscr{L}^\infty(C)} + \| \frac{\partial^2 f}{\partial x_2^2} \|_{\mathscr{L}^\infty(C)} + \| \frac{\partial^2 f}{\partial x_3^2} \|_{\mathscr{L}^\infty(C)} \right),$$

where $\|f\|_{\mathscr{L}^\infty(C)} = \sup_C |f|$. However, the coefficients $\lambda_{n,k}$ strictly depend on the value of the function f at the vertex $x_{n,k}$ and in general will not decay quickly, thus not suitable for compression. An alternative representation is the multiresolution form

$$f = \sum_{k \in K(0)} \lambda_{0,k} \phi_{0,k} + \sum_{i=1}^{n-1} \sum_{m \in M(i)} \gamma_{i,m} \psi_{i,m} + \varepsilon.$$

Since both the single scale and multi scale representation span the same space, the error ε will be the same. However, the advantage of the multiresolution representation is that the coefficients $\gamma_{j,m}$ vanish quickly.

Recall that the wavelet coefficients $\gamma_{j,m}$ can be obtained from a few samples of the original function:

$$\gamma_{j,m} = f(x_{j,m}) - \sum_{k \in N(j,m)} h^0_{j,k,m} f(x_{j+1,k}). \tag{1.61}$$

A more careful examination of Equation (1.61) and the neighborhood $N(j,m)$ shows that the term $\sum_{k \in N(j,m)} h^0_{j,k,m} f(x_{j+1,k})$ is the interpolating estimate of the $f(x_{j,m})$. For the trilinear wavelets derived previously, the value of edge node m is estimated from the two adjacent nodes, for a face node the estimate is from all four corners, and for the interior node all eight corners of the cube are used (See Figure 2.31(b),(c) and (d)). This implies that

$$|\gamma_{j,m}| \leq 2^{-2j-3} \left(\| \frac{\partial^2 f}{\partial x_1^2} \|_{\mathscr{L}^\infty(C^\psi_{j,m})} + \| \frac{\partial^2 f}{\partial x_2^2} \|_{\mathscr{L}^\infty(C^\psi_{j,m})} + \| \frac{\partial^2 f}{\partial x_3^2} \|_{\mathscr{L}^\infty(C^\psi_{j,m})} \right).$$

The wavelet coefficient values depend on the level of resolution j and the maximum of the local second derivative and are exactly the trilinear error. This estimate can be used to derive a strategy for the construction of hierarchical adaptive representation of the density function f. If f contains a discontinuity the neighborhood wavelet coefficients are in general large, but away from the discontinuity the decay is very fast. This makes wavelets very well suited for representing local and global variations. In contrast, Fourier and spherical harmonics have very slow decays due to the bias toward global representations.

It should be pointed out that this linear HB representation is essentially an adaptive mesh representation. However, most adaptive meshes are discontinuous unless a computational expensive step is used a-priori to avoid hanging nodes. These discontinuities lead to artifacts for visualization of molecular volumes and surfaces, in particular the iso-surface extraction would be very poor.

One strategy for compression is to compute $\gamma_{j,m}$ for every single m and level of resolution j and store only the values above a predetermined level dependent value ε_j. Thus depending on the level of resolution and the derivative $\|\partial^2 f\|_{\mathscr{L}^\infty(C^\psi_{j,m})}$ we sparsify the representation.

1. Start at level $j = 0$
2. $\forall m \in M(j)$ compute $\gamma_{j,m}$
3. Store wavelet coefficients only if $\gamma_{j,m} < \varepsilon_j$
4. $j := j + 1$
5. Go to step 2 until $j = n$

The downside of this interpolating representation is that the linear interpolating basis is not *Reisz stable* ([99]), thus the traditional choice of removing the wavelet coefficients with the minimum energy does not lead to an optimal L^2 representation of the density function f.

Since the wavelets are the error between one level of resolution and the next one choice is to pick $\varepsilon_j = \varepsilon * \sup|f|$, where $\varepsilon \in [0,1]$, thus eliminating the wavelets with a low contribution to the error. However, low level wavelets have a higher weight in the total error. Instead we weight the threshold according to the level of resolution, i.e. $\varepsilon_j = \frac{\varepsilon * \sup|f|}{2^{n-j-1}}$.

Although this scheme does produce highly compressed maps with good accuracy, it does not lead to an error bounded scheme. In the Appendix A, we present a fully error bounded compression wavelet scheme. Moreover, in Appendix B error bounded mesh sizes and local wavelet updates for Radial Basis Function molecular maps are derived.

Encoding As with all compression schemes, a suitable encoding of the non zero wavelets is a necessary step. Many such schemes have been developed for efficient transmission of compressed data streams . However, very few encoding schemes have been developed for visualization purposes like volume rendering and iso-contouring. Such encoding schemes should aim for high compression ratios and fast random access. The zero-bit encoding scheme, introduced by [24], uses a two-stage significance map to achieve a good compromise between these two aspects. Whether wavelet coefficients are null or not is stored in "zero-bits" while the significance information of the detail coefficients of each non-null detail node is stored in additional bits. Nevertheless, since the encoding is limited to Haar wavelets, unpleasant and biologically incorrect visual artifacts appear at moderately high compression ratios. To solve this problem, a novel encoding scheme is created on the basis of zero-bit encoding for HB linear wavelets.

The wavelet coefficients are assumed to be mostly zeros, with a few localized non-zero coefficients, as shall be the case for a large number of Cryo-EM and molecular synthetic maps. Under this assumption, a multi-resolution encoding scheme can be built to optimally access the concentrated information. In this encoding scheme the rectangular 3D volume data is subdivided into blocks of equal size (which can be predetermined by the user). Each block is further segmented into cells which contain a fixed number of voxels. Now, we run the wavelet transform on each cell separately. Since the transformed data usually has large swaths of zeros, a large number of cells or blocks can be ignored. By constructing an appropriate multi resolution indexing scheme, an efficient encoding of the non-zero coefficients is built.

A 3D volume of N^3 voxels is divided into blocks of equal size n^3 where N is a multiple of n. The blocks are subdivided into cells of size m^3, where n is a multiple of m. The next step is to index the blocks and cells appropriately. Provided that there are $p = \frac{N^3}{n^3}$ blocks in the 3D volume, a p-bit length word is defined. A non-empty block is tagged as a '1' bit, while empty block as a '0' bit. The '0' indicates that further searches in that block are not required. Similarly, for each non empty block, a word is constructed such that a '1' bit indicates a non empty cell. (See Figure 2.32(a)).

Now, each block contains a byte to represent the Number of Non-Null Cells (NNNC) in it. When NNNC is zero, the block is a null block, which means that all coefficients in the block are zero. Otherwise, a Cell Bit Flag Table(CBFT), in which one bit is zero if and only if its corresponding cell is null, is used to indicate whether each cell in the block is null or not.

When a cell is not null, a one or two tier scheme is exploited to represent it. To encode the non-zero wavelet coefficients in each cell for wavelet level 0 and 1, we use one bit to index a voxel. For level 2 we employ a two tier "zero-bit" tree. The

coefficients are separated in groups of two bytes (16 bits). If all 16 coefficients are empty, we indicate it with a '0' at the first level of the tree, otherwise a '1' label is used. For each non-empty group, a 16 bit word is used to indicate the non zero coefficients. (See Figure 2.32(b)).

The non-zero values are stored in a data stream, which can be retrieved by a combination of Zero-bit, Significance Map Stream (ZSMS) table and the Cell Information Array (CIA). The two tier encoding scheme is stored in the ZSMS table. For each cell, the CIA contains the Data Offset that gives the starting position in the data stream, in addition to the Zero-bit Offset, which indicates the beginning position in the ZSMS table for a cell.

Decoding With this encoding scheme we can efficiently recover any original data coefficient. The decoding is obtained by the following steps (see Figure 2.32(c)):

1. Identify the wavelet coefficients contributing to that data point. For each coefficient, follow the next three steps.
2. Find out the block that contains the coefficient. Check the NNNC of the block. If it is zero, discontinue and return a zero. Otherwise, go to the next step.
3. Check the bit in CBFT associated with the cell containing the coefficient. If it is '0', the cell is a zero cell and discontinue. Otherwise, go to the next step.
4. Get the bit associated with the coefficient from ZSMS table by Zero-bit Offset. If the bit is zero, the coefficient has the value of zero. Otherwise, add the position with the Data Offset of the cell to get the position of the wavelet coefficient in the data stream and fetch its value accordingly.
5. Once all the significant coefficients are retrieved, the original data point is calculated with the wavelet basis.

Random Access An important feature of our encoding scheme is fast random access with high compression rates. Here, the worst case reconstruction and lookup computational costs for decoding a voxel are analyzed. In addition, we also present the average number of sparsified wavelets needed to represent the entire domain.

Let γ_i represent the ratio of the number of non zero wavelet coefficients to the total number of vertices's at level i for all non zero cells. Similarly, let α be the ratio of the number of non zero blocks to total number of blocks and β be the ratio of the number of non zero cells to total number of cells located in non zero blocks. In section 2.1.5, we use 9 as the cell size, so i is no more than 3. To decode the value of any voxel in the cell, we first identify the number of wavelets that contribute to the reconstruction. For every wavelet, the ZSMS table is searched to provide the coefficient value. The total decoding cost is the sum of the lookup cost in the ZSMS tree and the computational reconstruction cost. To decode a voxel, the following procedure is executed:

1. If the voxel to be reconstructed is located at scaling function level 0, then it is the value of the cell and no reconstruction is needed. But we still need to perform one lookup. Otherwise suppose that the voxel is located at a level j wavelet vertex.
2. Subdivide the cube into eight new subcubes and let $i = 0$.
3. Locate the subcube that contains the voxel to be reconstructed at level i and retrieve the other seven coefficients which belong to that cube.
4. Add the wavelet contributions to the original voxel location's value.
5. Go to the next level, i.e. $i := i + 1$.
6. Subdivide each subcube into eight again.
7. Repeat Steps 3-6 until $i = j$.

To reconstruct a voxel in V_0, at most 15 arithmetic operations for a interior node are needed, where an arithmetic operation denotes addition, subtraction, multiplication or division. To reconstruct a voxel in $W_0 = V_1 - V_0$, 3*3+7*3+15=45 more arithmetic operations are needed in the worst case. In the previous formula, 3 arithmetic operations for an edge node, 7 arithmetic operations for a face node and 15 arithmetic operations for a interior node are needed. So 15+45=60 arithmetic operations are

used to reconstruct a voxel in W_0 in the worst case. The worst case analysis can also be extended to W_1 and W_2 in a similar way and shown in Table 2.3(a).

In general, we are more interested in the average cost to decode a single voxel, since this would determine the speedup due to our encoding scheme. To reconstruct the voxel at level j, then $7j+1$ wavelets are needed at most. Since γ_i is the ratio of non zero wavelets to the total number of vertices at level i for non zero cells, on average only $8+7\gamma_0+7\gamma_1+7\gamma_2$ wavelets and scaling functions are needed to decode the voxel in any non-empty cell. This leads to an upper bound of the average arithmetic cost, which is

$$\alpha\beta(15+60\gamma_0+105\gamma_1+150\gamma_2).$$

In addition to the computational cost, memory lookup costs can also be significant. In the following analysis we estimate the average memory lookup cost.

To retrieve any wavelet picked at random, we must first check that blocks and cells are non-zero. Since $(1-\alpha)$ is the probability that a block is zero and $(1-\beta)$ is the probability that the cell is zero in the non zero blocks, we must at least have an average lookup cost of $(1-\alpha)+2\alpha(1-\beta)$. After identifying the block and cells, any random wavelet we choose can land in level 0, 1, or 2 with probability 19/721, 98/721 and 604/721. Since Level 2 has a two-tier zero-bit encoding scheme, at most 5 lookups are needed, which implies that the average cost for landing in level 2 is $3020\alpha\beta(1-\gamma_2)/721$. A similar analysis is done for level 1 and 0, however only a one level zero-bit encoding, and hence only a maximum of 4 lookups are needed. The total lookup memory cost is

$$(1-\alpha)+2\alpha(1-\beta)+\frac{3020\alpha\beta\gamma_2+392\alpha\beta\gamma_1+76\alpha\beta\gamma_0}{721}.$$

Timings, errors and visualizations of molecular maps To gauge the effectiveness of the HB representation (zerobit and non zerobit) to molecular maps, we apply it to different types of synthetic and natural data. In particular, electron density, electrostatic and Cryo-EM maps that have been described in section 2.1.5. Moreover, theoretical and actual random access timings for the zero-bit decoder are presented.

We first test the HB representation with zero-bit encoding on the electron density map of the NMR Structure of Ectodomain of Siv Gp41 (2EZP in the PDB). The molecular map is created by Equation 1.52 with a rate of decay $\mathscr{B}=-2.3$, which corresponds to 1-3 angstroms resolution. In Table 2.1(a) file compression ratio (CR) results for 2EZP are shown. As the threshold level ε is increased we observe that CR increased but the PSNR of the reconstructed image degrades. This can be observed in Figure 2.33(a)-(d), however, notice that even for high compression rates, the molecular surface is retained very well. We obtain similar results for compression of the 50s Large Ribosomal Subunit electron density and electrostatic map, as shown in Figure 2.34(a)-(h) and Table 2.1(b).

A Cryo-EM map of the Rice Dwarf Virus was used as an example of a natural map to test our compression scheme. The original dataset dimension is 241×481^2. In Figure 2.33(e), we show a volume rendering of the original map. We again used three different thresholds for wavelet compression as shown in Figure 2.33(f)-(h). High visual accuracy of the structure of the virus can be observed even at a compression ratio of 37:1. In Table 2.1(c), we present details of our compression algorithm for more threshold levels.

Another interesting feature to observe is the compression capabilities of our scheme with respect to maps of varying resolutions. In table 2.2(a)-(c) compression results for the 50s Large Ribosomal Subunit electron density at rates of decay $\mathscr{B}=-1,-2,-5$. For a fixed threshold level, we observe that both the PSNR and file Compression Ratio increases with increase in $\mathscr{B}$ (i.e. decrease in resolution).

We now compare our zero-bit continuous HB scheme with zero-bit Haar compression, which to our knowledge, is the only other interactive compression format. In table 2.2(d) we directly compare our scheme with Haar for the same file compression ratios. It is clear from the data, that our scheme is superior. This is supported visually in Figure 2.35. Notice that the visual quality for HB is much superior.

Random Access Times To give some meaning to the costs that we have derived in section 2.1.5 we study the 513^3 electron density map of the 50s ribosomal large unit with decay parameter $\mathscr{B}=-1$ (see Table 2.2(a)-(c)). For each cell we obtain the parameters $\alpha,\beta,\gamma_0,\gamma_2$ and γ_3 and average them for all the cells. The following values are obtained: $\alpha=0.421$, $\beta=0.784$, $\gamma_0=0.384$, $\gamma_1=1.651\times 10^{-3}$ and $\gamma_2=0.0$. This yields an average arithmetic cost of 12.61 and an average lookup memory cost of 1.145 for accessing a single voxel. Considering that the optimal lookup cost is 1, we obtain very good access speeds. Moreover, arithmetic operations in many computer architectures are much faster than memory lookups.

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.50	32.73	99.50	86.23	201.65	27.76	51.56	23.47	539321	125.89
0.35	34.53	99.34	86.23	153.54	21.14	41.04	19.08	666472	101.87
0.15	38.65	98.80	86.23	83.56	11.50	18.93	11.87	1139150	59.60
0.05	45.19	97.20	86.23	35.73	4.92	6.53	5.59	2512429	27.02
0.01	60.65	93.19	86.23	14.70	2.02	1.26	0.94	5764216	11.77
0.00								67898440	

(a) 2EZP Electron density map

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.500	38.50	98.84	0.58	85.91	85.40	63.01	3.48	8263922	64.96
0.35	40.24	98.78	0.58	82.39	81.91	45.67	2.85	8572631	62.62
0.15	44.18	98.60	0.58	71.85	71.43	19.75	1.81	9675507	55.48
0.05	50.29	98.06	0.58	51.56	51.26	6.28	0.90	12972064	41.38
0.01	60.36	96.13	0.58	25.85	25.70	1.29	0.28	24344104	22.05
0.00								536870980	

(b) Ribosome 50s Large Subunit Potential Map 513^3

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.50	31.10	99.15	72.61	117.83	32.27	41.11	25.61	4771349	46.74
0.35	32.97	98.88	72.61	89.49	24.51	36.45	20.64	6042206	36.91
0.150	36.03	98.08	72.61	52.29	14.32	18.95	14.53	10031643	22.23
0.05	42.69	94.73	72.61	19.01	5.20	6.68	6.74	26590966	8.38
0.01	55.69	86.26	72.61	7.27	1.99	1.31	1.48	67075200	3.32
0.00								223032072	

(c) Rice Dwarf Virus Cryo-EM map (481*481*241)

Table 2.1: Zero Bit encoded HB Compression results for 2EZP electron density map, Large Ribosomal subunit 50s and a Cryo-EM map of the Rice Dwarf Virus. Legend: ε as defined in 2.1.5, PSNR=Peak Signal to noise ratio, TX: wavelet transform, TCC: Total Coefficient Compression ratio, NZC: Non Zero Compression ratio, e_∞:maximum l^∞ percent error between reconstructed compressed version and the original, e_2= l^2 percent error between reconstructed compressed version and the original. CR:File Compression Ratio.

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.50	30.15	99.10	77.67	111.73	24.94	54.13	27.84	6376816	84.19
0.35	31.87	98.81	77.67	84.07	18.77	43.37	22.82	8025803	66.89
0.15	36.13	97.73	77.67	44.13	9.85	19.09	13.97	14473038	37.09
0.05	42.52	94.71	77.67	18.91	4.22	6.63	6.63	32247352	16.64
0.01	58.49	87.39	77.67	7.93	1.77	1.30	1.04	72738554	7.38
0.00								536870980	

(a) $\mathscr{B}$=-5

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.50	30.82	99.19	74.77	123.72	31.20	29.80	19.76	5807819	92.43
0.35	32.76	98.92	74.77	93.06	23.47	25.71	15.73	7315547	73.38
0.15	37.35	97.99	74.77	49.80	12.56	17.18	9.31	12469751	43.05
0.05	42.15	95.76	74.77	23.63	5.96	6.53	5.31	25731023	20.86
0.01	56.88	86.17	74.77	7.23	1.82	1.31	0.96	79575816	6.74
0.00								536870980	

(b) $\mathscr{B}$ = -2

$\mathcal{E}$	PSNR	% 0's after TX	% 0's original	TCC	NZC	e_∞	e_2	Size(Bytes)	CR
0.50	33.16	99.35	71.58	153.85	43.72	22.24	11.93	4586097	117.06
0.35	34.52	99.22	71.58	128.86	36.62	16.91	10.23	5460302	98.32
0.15	39.18	98.54	71.58	68.56	19.48	9.27	5.92	9432914	56.91
0.05	44.58	97.06	71.58	34.07	9.68	5.77	3.18	17706006	30.32
0.01	54.79	89.12	71.58	9.19	2.61	1.31	0.96	63375575	8.47
0.00								536870980	

(c) $\mathscr{B}$ = -1

Zero-bit HB				Zero-bit Haar			
CR	PSNR	e_∞	e_2	CR	PSNR	e_∞	e_2
98.32	34.53	16.91	10.23	98.32	29.76	65.60	28.88
56.91	39.19	9.27	5.92	56.91	32.94	52.13	20.03
30.32	44.59	5.78	3.18	30.22	40.93	25.33	7.96

(d) Zero-bit HB and zero-bit encoded Haar compression

Table 2.2: Compression and comparison results for 50s Ribosomal Large unit with different Rate of Decay parameter $\mathscr{B}$.(a)-(c) Compression results with HB basis. (d) Compression results for 50s Ribosomal large unit in 513^3 grid with $\mathscr{B} = -1$ produced by zero-bit HB and zero-bit encoded Haar compression. It is clear from the results that for the same file compression ratios the zero-bit HB is about 5db higher that zero-bit HB, which is very significant at high compression ratios. Legend: ε as defined in 2.1.5, PSNR=Peak Signal to noise ratio, TX: wavelet transform, TCC: Total Coefficient Compression ratio, NZC: Non Zero Compression ratio, e_∞:maximum l^∞ percent error between reconstructed compressed version and the original, e_2= l^2 percent error between reconstructed compressed version and the original. CR:File Compression Ratio.

level	*numbas*	*numrec*	*arith*
0 (V_0)	8	8	15
0 (W_0)	19	15	60
1 (W_1)	98	22	105
2 (W_2)	604	29	150
total	729		

(a)

Rate of Decay	t_1	t_2	t_3	t_4
$\mathscr{B} = -1$	5.049×10^{-5}	1.456×10^{-4}	2.58×10^{-6}	26.679
$\mathscr{B} = -2$	5.379×10^{-5}	1.427×10^{-4}	2.58×10^{-6}	25.835
$\mathscr{B} = -5$	5.404×10^{-5}	1.375×10^{-4}	2.58×10^{-6}	24.829

(b)

Table 2.3: (a) Worst case reconstruction costs for a Cell with three wavelet levels. Let *numbas* be the number of basis functions at level j. In addition, let *numrec* be the worst case decoding cost and *arith* be the worst case arithmetic cost to decode a single voxel. (b) Reconstruction random access timings for the 50s Ribosomal large unit. The timings are in seconds. Legend: t_1= The average time of 1000 random accesses of a voxel from the compressed format. t_2= The average time of 1000 random accesses of a full cell from the compressed format. t_3= The average time of 1000 random accesses of a voxel inside a cell, where the cell was chosen at random from the uncompressed format. t_4= The average time to decode the entire compressed volume for 10 runs.

In Table 2.3(b) we show random access timing test cases for our method tested on the 50s Ribosomal large unit with varying rates of decay. The four test cases are

- t_1: The average time of 1000 random accesses of a voxel from the compressed format.
- t_2: The average time of 1000 random accesses of a full cell from the compressed format.
- t_3: The average time of 1000 random accesses of a voxel inside a cell, where the cell was chosen at random from the uncompressed format.
- t_4: The average time to decode the entire compressed volume for 10 runs.

The code was run on a SGI ONYX2 system with R12000 processors. The timing results are for one processor. As observed, to decode a single voxel the slowest case is t_1. However, for most volume rending and iso-contouring applications it is more efficient to decode an entire cell since many neighboring points are used.

Conclusions

We have demonstrated the value of HB wavelet compressed volumetric representations for macromolecular structures and associated properties, especially in the age of rapidly increasing structures and medium resolution maps in both the PDB and EBI. The added value of fast decompression time yields the necessary ingredient for enabling interactive visualization performance on current desktop computers. Our future work is predominantly in the development of computational analysis algorithms that work directly from the HB compressed volumes of bio-molecules and associated properties, with fast decompression. We are also interested in the development of interactive visualization techniques which optimally utilize the faster bandwidth of programmable graphics hardware, commonplace on most desktop computers.

This work was supported in part by NSF grants INT-9987409, ACI-022003, EIA-0325550 and a grant from the NIH 0P20 RR020647.

Appendix

Error bounded compression

By eliminating only zero and *low valued* wavelets, we can balance compression and accuracy. We would like to ensure that the error due to the lossy compression is within a user defined threshold $\varepsilon > 0$. The following algorithm describes a top down approach to providing error bounded wavelet compression.

Algorithm 1: Top down error-guided wavelet compression The maximum error $\varepsilon_T(C)$ between a function f and the trilinear approximation on C is given as.

$$|\varepsilon_T(C)| \leq 2^{-2n-3}\left(\|\frac{\partial^2 f}{\partial x_1^2}\|_{\mathscr{L}^\infty(C)} + \|\frac{\partial^2 f}{\partial x_2^2}\|_{\mathscr{L}^\infty(C)} + \|\frac{\partial^2 f}{\partial x_3^2}\|_{\mathscr{L}^\infty(C)}\right) \tag{1.62}$$

Given a predefined maximum number of levels for the wavelets in a block, we perform an adaptive top down decomposition of the domain as follows:

1. Let $i = 0$ correspond to the initial level where we have a single cube $C_0^1 \in C_0$. If $|\varepsilon_T(C_0^1)| \leq \varepsilon$, we are done. Else, we decompose and continue with the next steps.
2. Decompose the current cube $C_i^k \in C_i$ into 8 new subcubes C_{i+1}^k, where $k \in \{1..8\}$.
3. Obtain the wavelets associated with the nodes $M(i+1)$ that correspond to the cube C_{i+1}^k.
4. For each subcube C_{i+1}^k, if the corresponding trilinear error $\varepsilon_T(C_{i+1}^k)$ is within the user defined error ε, we do not decompose C_{i+1}^k any further. Otherwise, we need to decompose the subcube, and continue in a hierarchical fashion with step 2.
5. This error driven adaptive decomposition is continued till we reach the maximum level or the error is satisfied everywhere.

A subcube C_{i+1}^k with error $|\varepsilon_T(C_{i+1}^k)| \leq \varepsilon$ is not decomposed in the above algorithm. But if a neighboring subcube C_{i+1}^{j+k} is decomposed, then the shared node $m \in M(i+1)$ adds a new function, the compactly supported trilinear associated with a node at level $i+1$ centered at m, to the subcube C_{i+1}^k. This changes the function in C_{i+1}^k, which can potentially violate the requirement $|\varepsilon_T(C_{i+1}^k)| \leq \varepsilon$. Hence the above algorithm in fact does not satisfy the error bound required by the user in a global sense and is modified as shown below.

Let us consider a subcube C_l^k which is surrounded by cubes at level $i \geq 2$ on all its 6 faces. For each level, each face can be subdivided into 4 new faces. Each time, 5 new nodes are added per old face, 4 of which are shared by 2 faces and 1 is on the center (It should be noted that any cube can be surrounded by all sides only if it is defined after level 1. Hence a cube fully surrounded by fully subdivided cubes cannot have 8 new nodes at level 1). The total number of nodes that could surround a cube, given i levels is bounded by

$$\frac{4(6\times 4^{i-2})}{2} + 6\times 4^{i-2} = \frac{9}{8}4^i$$

Let $\varepsilon'_T(node, C_l^k)$ be the trilinear error in the cube containing *node* (if *node* is a corner of C_l^k). If, on the other hand, *node* was a hanging face node for the cube C_l^k, let $\varepsilon'_T(node, C_l^k)$ represent the bilinear error associated with that face. This error $\varepsilon'_T(node, C_l^k)$, from equation 1.62 is bounded as follows:

- Node is a corner vertex of C_l^k:

$$\varepsilon'_T(node, C_l^k) = \varepsilon_T(C_l^k)$$

- Node is a hanging node on a face C_l^k: Let the node be created due to the recursive subdivision of a face of C_l^k at some level $\hat{l} > l$. (The node itself is at level $\hat{l}+1$ and is created from a face subdividing at level $\hat{l}$.)

$$\varepsilon'_T(node, C_l^k) \leq |2^{-2*(\hat{l}-l)-3}\varepsilon_T(C_l^k)|$$

The error due to the new nodes on the faces of C_l^k is hence bounded by

$$\begin{aligned}
\varepsilon_{nodes}(C_l^k) &= \sum_{k=l}^{L} \sum_{node=1}^{\frac{9}{8}4^k} \varepsilon'_T(node, C_l^k) \\
&\leq \left| 2^{-2l-3} \left(\| \frac{\partial^2 f}{\partial x_1^2} \|_{\mathscr{L}^\infty(C_l^k)} + \| \frac{\partial^2 f}{\partial x_2^2} \|_{\mathscr{L}^\infty(C_l^k)} + \| \frac{\partial^2 f}{\partial x_3^2} \|_{\mathscr{L}^\infty(C_l^k)} \right) \right. \\
&+ \left. \sum_{k=l}^{L} \frac{9}{8} \left(\| \frac{\partial^2 f}{\partial x_1^2} \|_{\mathscr{L}^\infty(face(C_l^k))} + \| \frac{\partial^2 f}{\partial x_2^2} \|_{\mathscr{L}^\infty(face(C_l^k))} + \| \frac{\partial^2 f}{\partial x_3^2} \|_{\mathscr{L}^\infty(face(C_l^k))} \right) \right| \qquad (1.63)
\end{aligned}$$

where $face(C_l^k))$ corresponds to the faces of C_l^k and L is the maximum number of levels.

Hence the algorithm is modified to continue to subdivide the subcube C_l^k to satisfy the error due to the possible addition of face nodes due to subdivision of its neighbors. For each new subdivision, we see a reduction in at least 4 in the error $\varepsilon_{nodes}(C_l^k)$ as given by equation 1.62. Hence we need to subdivide a cube at the most C_l^k: $log_4(L-l)$ more levels after $|\varepsilon_T(C_l^k)| \leq \varepsilon$ is satisfied.

Algorithm 2: Top down error-bounded wavelet compression Here we summarize the modified algorithm, which guarantees that the error in the wavelet compression is within the user defined error.

1. Compute the second derivatives in the volume to help us calculate equation 1.63.
2. Let $i = 0$ correspond to the initial level where we have a single cube C_0. If $|\varepsilon_T(C_0)| \leq \varepsilon$ flag the cube C_0 and proceed to the last step. Else, we decompose and continue with the next 3 steps.
3. Decompose the current cube C_i^k into 8 new subcubes C_{i+1}^k.
4. Obtain the wavelets associated with the nodes $M(i+1)$ that correspond to the cube C_{i+1}^k.
5. For each subcube C_{i+1}^k, if $|\varepsilon_T(C_{i+1}^k)| \leq \varepsilon$ flag the cube C_{i+1} and proceed to the last step. Otherwise, we need to decompose the subcube, and continue in a hierarchical fashion with step 3.
6. This error driven adaptive decomposition is continued till we reach the maximum level or the error is satisfied everywhere.
7. For each node not at the last level and hence flagged in the above steps, we subdivide further to satisfy the theoretical error due to addition of new functions from possible subdivision of neighbors. Hence for each flagged node C_i, we continue to subdivide till

   $$|\varepsilon_{nodes}(C_i^k)| + |\varepsilon_T(C_i^k)| < \varepsilon$$

   From the previous discussion, we also know that the above inequality is always satisfied if we subdivide $log_4(L-i)$ more levels. This step is placed separately from the previous steps just for clarity. It could be performed in step 2 and 5 itself.

Note that the cost of this algorithm is $\mathscr{O}(Nlog(N))$, where N is the total number of voxels in C.

Although we have a fully error bounded scheme, it is not yet optimal. In many cases we might over shoot the required accuracy. This algorithm will be further optimized to obtain a tighter bounded.

Apriori Mesh Width Estimation and local Wavelet Updates for RBF Gaussian Maps

Most compression schemes require the computation of all the wavelets and then subsequently threshold them. Due to the size of the problems involved, with grid sizes well into the 256^3 voxel range, this can be a costly step. However, for a class of synthetic molecular maps such as RBF Gaussian summations, a-priori mesh size estimates and significant wavelet coefficients can be computed. In particular, we are interested in compression of data maps of the form

$$f = \sum_{i=1}^{M} c_i e^{-\|\bar{x}_i\|^2/B_i},$$

where $\|\bar{x}_i\|^2 = (x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2$, M is the number of atoms and $B_i = (\mathscr{B}/R_i^2)^{-1}$, where $\mathscr{B}$ is the rate of decay and R_i is the radius of the i^{th} atom. Moreover, fast updates of the wavelets coefficients with respect to the rate of decay are desirable.

The source of error in the compression scheme is broken up into trilinear interpolation plus error associated with the wavelet. The trilinear error estimates provides a suitable mesh size to accurately represent the RBF map. The wavelet error estimates lead to the fast computation of the projection coefficients.

Mesh size estimation. First, we assume that $B_{min} \leq B_i \leq B_{max}$ for all $i = 1 \ldots M$, and and let the ratio $Q = B_{max}/B_{min}$. Note that Q is $1 \ldots 9$, since the Van der Waals radius of the atoms in a protein vary between 1 to 3 angstroms. The second partial derivative of each summation term of $f(\bar{x}) = f(x,y,z)$ takes the form

$$\frac{\partial^2 e^{-\|\bar{x}_i\|^2/B_i}}{\partial x \partial x} = \frac{2}{B_i} e^{-\|\bar{x}_i\|^2/B_i} \left(\frac{2(x-x_i)^2}{B_i} - 1 \right),$$

where $\bar{x}_i = (x-x_i, y-y_i, z-z_i)$ for each Gaussian centered around (x_i, y_i, z_i). The minimum critical point is located at $\bar{x} = (x_i, y_i, z_i)$ with a value of $-\frac{2}{B_i}$. The maximum critical point is located at $\bar{x} = (x_i \pm \sqrt{\frac{3B_i}{2}}, y_i, z_i)$ with a value of $\frac{4}{B_i} e^{-3/2}$, then

$$\begin{aligned}
|\varepsilon| &\leq \frac{h^2}{8} \left(\| \frac{\partial^2 f}{\partial x^2} \|_\infty + \| \frac{\partial^2 f}{\partial y^2} \|_\infty + \| \frac{\partial^2 f}{\partial z^2} \|_\infty \right) \\
&\leq \frac{h^2}{4B_{min}} \sum_{i=1}^{M} \left(\| c_i e^{-\|\bar{x}_i\|^2/B_i} (\frac{2}{B_i}(x-x_i)^2 - 1)) \|_\infty \right. \\
&+ \| c_i e^{-\|\bar{x}_i\|^2/B_i} (\frac{2}{B_i}(y-y_i)^2 - 1)) \|_\infty \\
&+ \left. \| c_i e^{-\|\bar{x}_i\|^2/B_i} (\frac{2}{B_i}(z-z_i)^2 - 1)) \|_\infty \right).
\end{aligned}$$

Now, let M_r be the maximum number of expansion points in a sphere $\mathbb{S} \subset \mathbb{R}^3$ of radius $x_r = 9\sqrt{B_{max}}$ around a desired point (x,y,z), then

$$\begin{aligned}
\| \sum_{i=1}^{M} c_i e^{-\|\bar{x}_i\|^2/B_i} (\frac{2}{B_i}(x-x_i)^2 - 1)) \|_\infty &\leq \| \sum_{i=1}^{M_r} |c_i| \frac{2}{B_{min}} + \sum_{i=M_r+1}^{M} |c_i| \| e^{-\|\bar{x}_i\|/B_{min}} (\frac{2(x-x_i)^2}{B_{min}} - 1) \|_{l^\infty(\mathbb{R}^3/\mathbb{S})} \\
&\leq \| \sum_{i=1}^{M_r} |c_i| \frac{2}{B_{min}} + \sum_{i=M_r+1}^{M} |c_i| e^{-x_r^2/B_{min}} (\frac{2x_r^2}{B_{min}} - 1) \\
&\leq \| \bar{c}_r \|_1 \frac{2}{B_{min}} + \| \bar{c} \|_1 e^{-x_r^2/B_{min}} (\frac{2x_r^2}{B_{min}} - 1),
\end{aligned}$$

where $\|\bar{c}\|_1 = \sum_{i=1}^{M} |c_i|$ and $\|\bar{c}_r\|_1 = \sum_{i=1}^{M_r} |c_i|$. The second term on the right hand side becomes less than $10^{-33}\|\bar{c}\|$. This is below machine precision for molecules up to 10^{16} atoms, thus it can be safely ignored. This implies then that

$$|\varepsilon| \leq h^2 \left(\| \bar{c}_r \|_1 \frac{3}{2B_{min}^2} \right) \leq \varepsilon.$$

where the desired accuracy is ε. By setting the mesh size

$$h = B_{min} \sqrt{\frac{2\varepsilon}{3\|\bar{c}_r\|_1}},$$

then the absolute error is lower than ε.

Fast Updates of Mesh size and Wavelet coefficients with Respect to Changes in the Rate of Decay. As we increase the rate of decay B_i we can easily update the grid size to maintain the same accuracy. Suppose that B_{min} increases to $B^{'}_{min}$, then

$$h^{'} = h\frac{B^{'}_{min}}{B_{min}}\sqrt{\frac{\|\bar{c}_r\|_1}{\|\bar{c}^{'}_r\|_1}}.$$

A fast update of the wavelet coefficients can also be achieved with increasing (or decreasing) rate of decay. In particular we provide bounds on the wavelet coefficients for which the error is below a predetermined threshold. For each wavelet $\psi_{j,m}$ with support bounded by 2^{-j} along each dimension

$$|\psi_{j,m}| \leq \|\frac{2^{-2j-3}}{B_{min}}\sum_{i=1}^{M} c_i e^{-\|\bar{x}_i\|^2/B_{min}}\left(\frac{2}{B_{min}}((x-x_i)^2+(y-y_i)^2+(z-z_i)^2)-3\right)\|_\infty.$$

We are now interested in bounding the magnitude of the the wavelet coefficient. For each vertex $\bar{x}_i$ any wavelet support that intersects with a sphere of radius $\sqrt{\frac{3B_{max}}{2}}$ is computed if

$$|\psi_{j,m}| \leq \frac{2^{-2j-3}}{B^2_{min}}\sum_{i\in M_{j,m}}|c_i| \leq \varepsilon_j,$$

where $M_{j,m}$ is an index set for all the vertices $\bar{x}_i$ such that $d(\psi_{j,m},\bar{x}_i) \leq \sqrt{\frac{3B_{max}}{2}}$, where

$$d(\psi_{j,m},\bar{x}_i) = \min_{\bar{x}\in supp(\psi_{j,m})}\|x-\bar{x}_i\|_2.$$

Now, for wavelets such that $d(\psi_{j,m},\bar{x}_i) > \sqrt{\frac{3B_{max}}{2}}$, pick the minimum distance r_{min} from the support of the wavelet $\psi_{j,m}$ to the closest Gaussian, then

$$|\psi_{j,m}| \leq \frac{2^{-2j-3}}{B_{min}}\|\bar{c}\|_1 e^{-r^2_{min}/B_{min}}\left(\frac{2}{B_{min}}r^2_{min}-3\right).$$

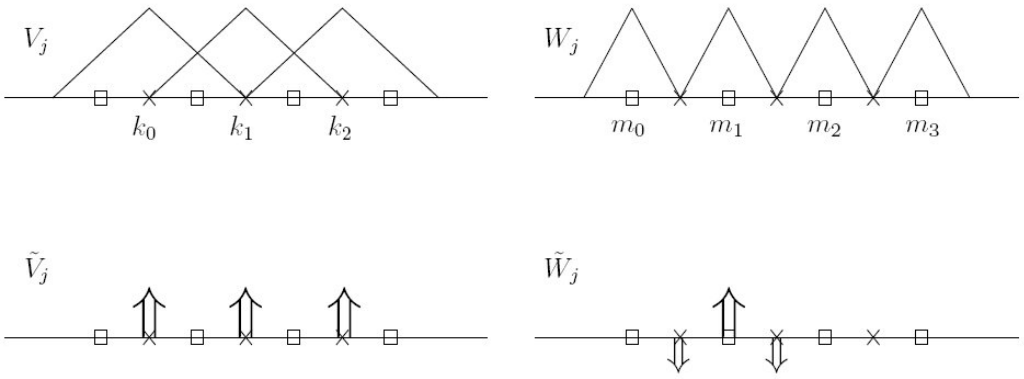
We now choose the wavelets which are bounded by ε_j, then

$$|\psi_{j,m}| \leq \frac{2^{-2j-3}}{B_{min}}\|\bar{c}\|_1 e^{-r^2_{min}/B_{min}}\left(\frac{2}{B_{min}}r^2_{min}-3\right) \leq \varepsilon_j.$$

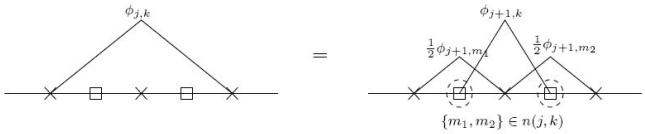
This condition is satisfied if

$$r_{min} \geq \max\left(\sqrt{-B_{min}W\left(-\frac{\varepsilon_j B_{min}}{2^{-2j-2}\|\bar{c}\|_1}\right)}, -\sqrt{-B_{min}W\left(-\frac{\varepsilon_j B_{min}}{2^{-2j-2}\|\bar{c}\|_1}\right)}\right)$$

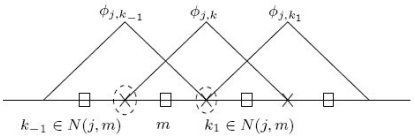
where W is the LambertW function. Thus only the T wavelets that are contained in the radius r_{min} from any Gaussian center with vertex x_i is computed. Note that they can be computed in $\mathcal{O}(M+T)$ time, by using the *Fast Gauss Transform*. ([236, 52, 13]).



(a)



(b)



(c)

Figure 2.30: Construction of Second Generation Wavelets. (a) Simple example of second generation wavelets and scaling functions. The primary scaling functions $\phi_{j,k}$ are constructed as linear hats on $K(j)$ and the wavelets are chosen such that $\psi_{j,m} = \phi_{j+1,m}$, we let $h_{j,m,l} = 0$ for all $l \neq m$. The dual scaling functions and wavelets are easily constructed as delta functions, i.e. $\tilde{\phi}_{j,k_1} = \delta(x - x_{j,k_1})$ and $\tilde{\psi}_{j,m_1} = \delta(x - x_{j,m_1}) - \frac{1}{2}\delta(x - x_{j,k_0}) - \frac{1}{2}\delta(x - x_{j,k_1})$. It is easy to see that these four spaces of function satisfy biorthogonality. Other smoother duals can be constructed ([71]). (b) In this linear "hat" example, the scaling function $\phi_{j,k}$ is constructed from three scaling functions at the next level of resolution $j+1$. The neighborhood $n(j,k)$ is formed from nodes in $M(j)$ that are needed to construct $\phi_{j,k}$. (c) This figure illustrates the neighborhood $N(j,m)$ for the index point $m \in M(j)$. Both scaling functions $\phi_{j,k_{-1}}$ and $\phi_{j,k}$ use the node m for their respective refinement equation, thus m belongs to $n(j,k_{-1})$ and $n(j,k)$. We say then that $\{k_{-1},k\}$ belongs to $N(j,m)$.

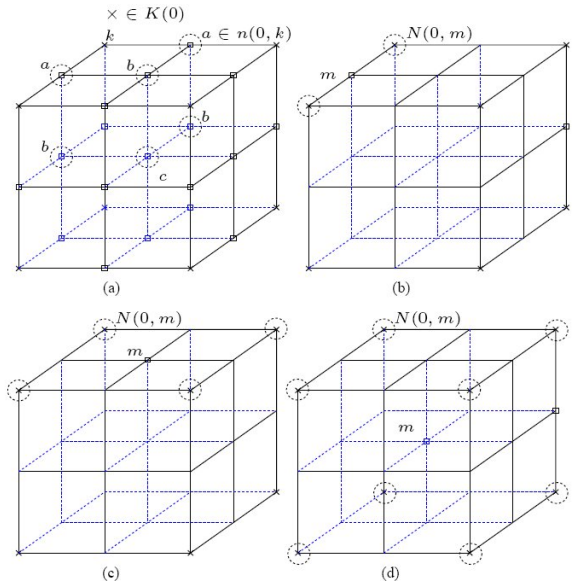


Figure 2.31: Trilinear HB wavelet construction. (a) The distribution of nodes for the construction of trilinear scaling functions on a rectangular mesh. For these functions we pick $\{P_0,\ldots,P_7\}$ to be equal to all the terms in the expansion of the shape function $(1+x_1)(1+x_2)(1+x_3)$. The 8 starting nodes in $K(j)$ are marked as $\times$. The mesh is refined into eight elements and the new nodes belonging to $M(0)$ are placed in the middle (although this is not a hard restriction). As we observe, the cube is subdivided into 8 elements with identical starting node patterns. The new nodes that belong to $M(0)$ form the set $n(j,k_i)$ for every single node $k_i \in K(j)$. However, many of the coefficients $h_{j,k,m}$ are equal to zero, thus for any node $\times \in K(0)$ only six nodes in $m \in M(0)$ are needed (a,b,c). (b) Example of the nodes belonging to $N(0,m)$. If m is an edge node then only the two immediate neighbor nodes are needed to completely determine the polynomial in equation 1.60.(c) For a interior face node 4 nodes are needed for $N(0,m)$.(c) For an interior cubic nodes $N(0,m)$ has eight nodes.

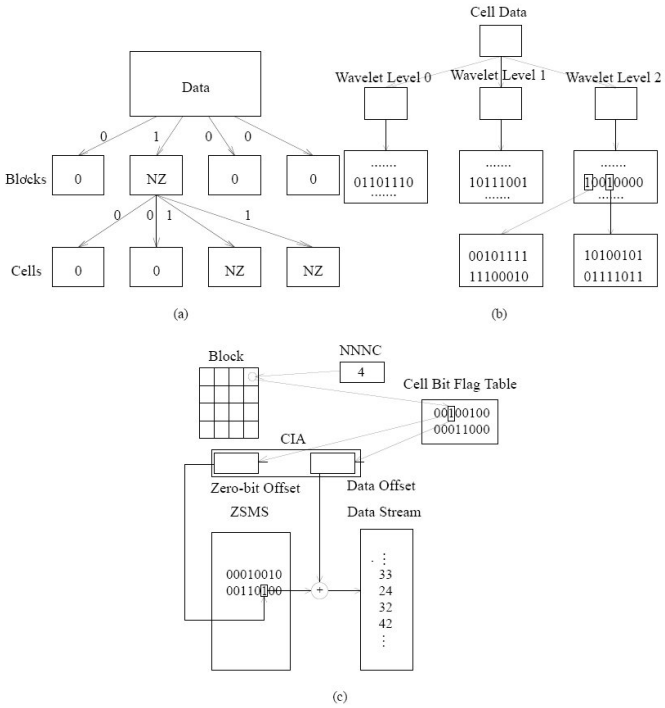
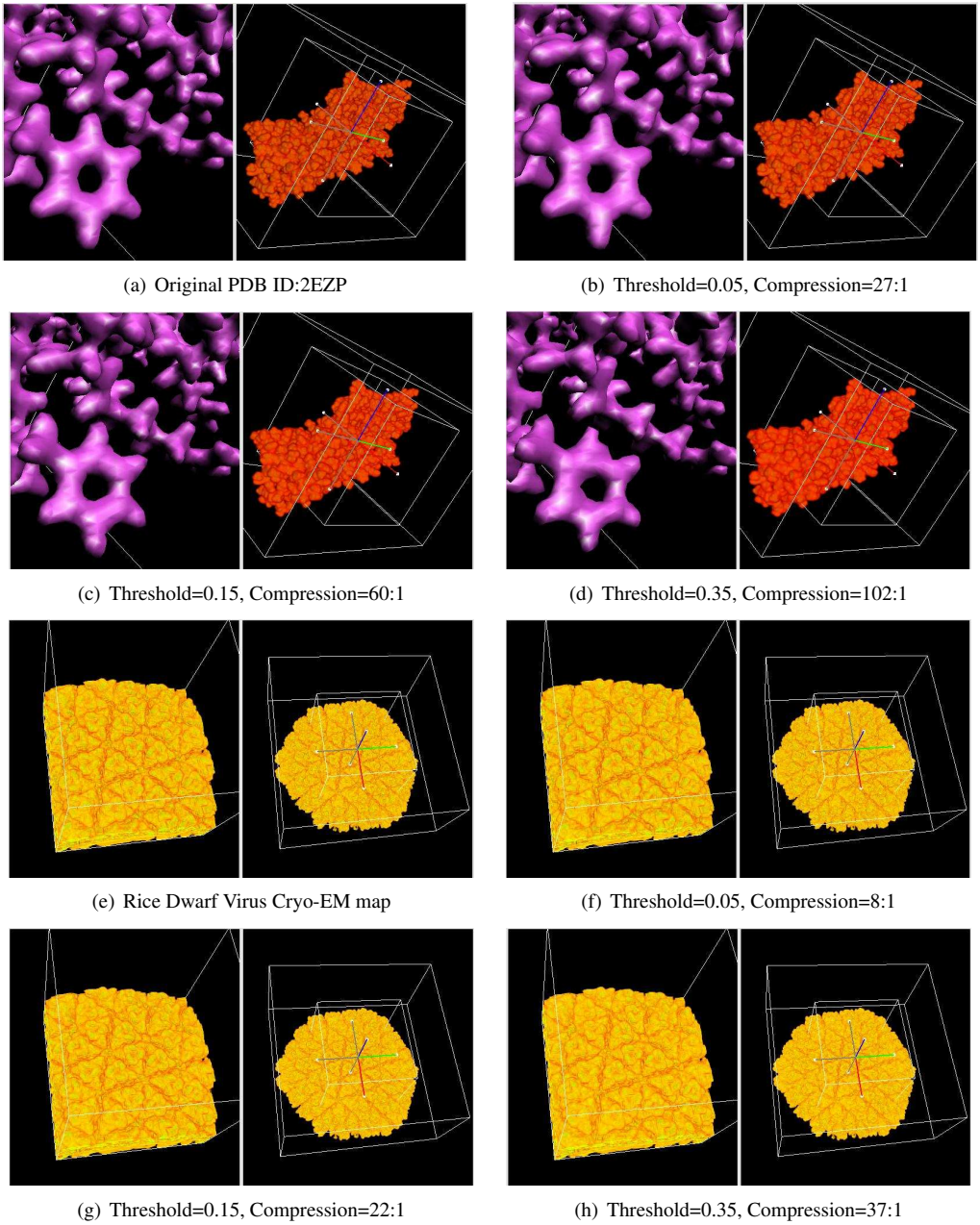


Figure 2.32: Scheme for Hierarchical Encoding and Decoding. (a)The data is decoded into hierarchically into blocks and cells. Bit vectors are used to show if a block or cell is empty or not. (b) Encoding of wavelets in each cell of variable size, although in our implementation we used cells of 9^3. (c) Procedures and structures used to decode wavelet coefficients.



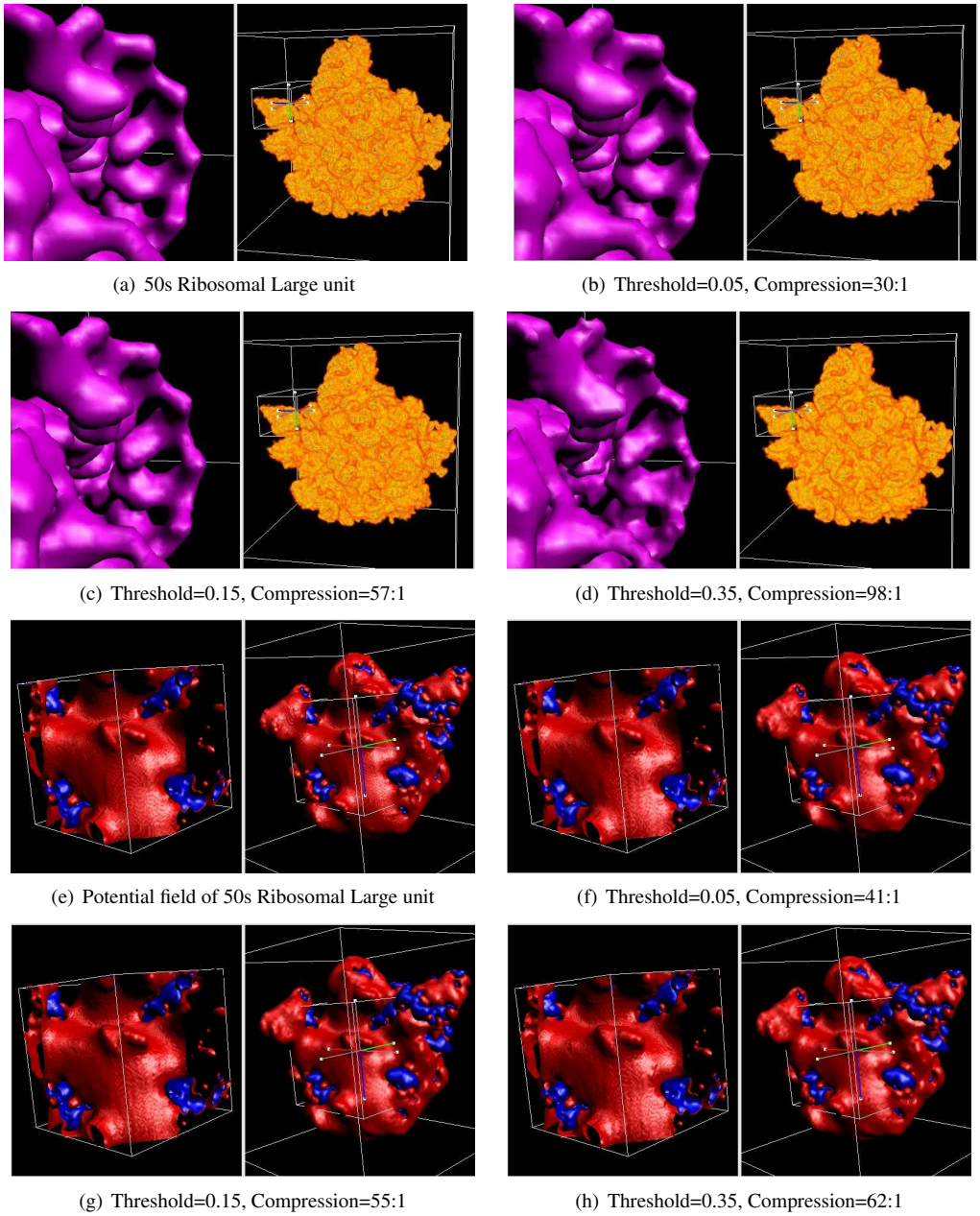
(a) Original PDB ID:2EZP (b) Threshold=0.05, Compression=27:1

(c) Threshold=0.15, Compression=60:1 (d) Threshold=0.35, Compression=102:1

(e) Rice Dwarf Virus Cryo-EM map (f) Threshold=0.05, Compression=8:1

(g) Threshold=0.15, Compression=22:1 (h) Threshold=0.35, Compression=37:1

Figure 2.33: Compression examples of synthetic and Cryo-EM molecular maps. (a)-(d) Electron density map of PDB 2EZP (Siv Gp41) with 3057 atoms in a 257^3 grid. (e)-(h) Rice Dwarf Virus Cryo-EM compressed maps in $241 * 481^2$ voxels.



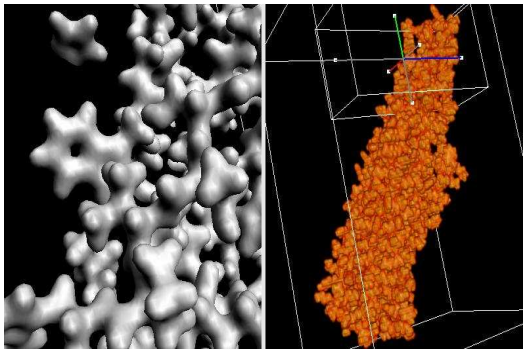
(a) 50s Ribosomal Large unit (b) Threshold=0.05, Compression=30:1

(c) Threshold=0.15, Compression=57:1 (d) Threshold=0.35, Compression=98:1

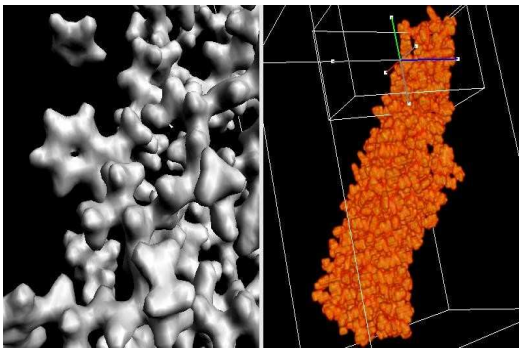
(e) Potential field of 50s Ribosomal Large unit (f) Threshold=0.05, Compression=41:1

(g) Threshold=0.15, Compression=55:1 (h) Threshold=0.35, Compression=62:1

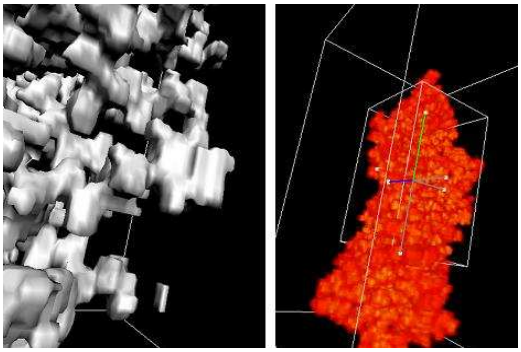
Figure 2.34: Compression examples of synthetic molecular maps.(a)-(d) Electron density map of 50s Ribosomal subunit (PDB:1JJ2) with 98543 atoms in 512^3 grid. (e)-(h) Potential field of 50s Ribosomal subunit in 513^3 grid created from the Adaptive Poisson Boltzmann Solver (*https://gridport.npaci.edu/apbs/*)



(a) Original electron density map (2EZP.pdb)



(b) non zero-bit 207:1 coefficient compression using HB



(c) non zero-bit 207:1 coefficient compression using Haar

Figure 2.35: Compression comparison between HB and Haar schemes. In (a) we have a view of the iso surface corresponding to the molecular surface of the original electron density map. By setting the compression ratio for both hierarchical methods to 207:1 we can readily see that HB leads to better feature preserving (as shown in (b)), in contrast to the poor performance of Haar (as shown in (c)).

The following algorithm is taken from [263] (Sharma2010Segmentation).

Introduction

The segmentation problem addressed here is to subdivide a three dimensional image $I(x,y,z) : \Omega \mapsto \mathbb{R}(\Omega \subset \mathbb{R}^3)$ into non-overlapping partitions $\Omega_i(i = 1..n_c)$ such that $\cup\Omega_i = \Omega$, where each partition is homogeneous in the sense that it minimizes a certain quantity. Each region is said to produce a class representing the partition.

Implicit surfaces (or level sets of functions on $\mathbb{R}^3$) naturally capture the arbitrary topology of the boundary of subdomains (Ω_i) in contrast to explicit or parameterized surfaces. Deformable level set surfaces under mean curvature flow (by solving a partial differential equation that minimizes some sort of an energy functional) provide a direct solution methodology for the segmentation problem. Pioneering work by Osher and Sethian [223] present an effective implicit representation for evolving curves. Later, the work was developed in context of the Mumford-Shah functional by Chan and Vese [79] for 2D images that do not contain prominent edges. In a subsequent paper, the authors suggest a multi-domain segmentation [301] using the same level set framework. Other variants of the same method exist for applications like image de-noising based on total variation minimization [247, 306].

Conventional level set methods solve the interface evolution equation with linear interpolation of the implicit function and its derivatives (at sampled grid points). The level set surface from this results in a C^0 continuous surface. Bajaj et al. [40, 41] present a tri-cubic B-spline based level set method that produces a C^2 continuous level set surface solution. Due to the high computational intensity of the level set method and inherent parallelism in the solution of the involved PDEs, a parallel compute environment is the most appropriate . In particular, in this paper we explot the multi-core parallelism of Graphics Processing Units (GPU's). Schemes for fast evaluation of PDEs are suggested by Weickert et al. [306]. Multigrid methods are also suitable for a fast solution of differential equations. An active contour model using multigrid methods is suggested by Papandreou and Maragos [224]. Of particular interest is a solution to the level set equations for segmenting large volumes. Work by [277, 190] show GPU based segmentation and visualization. Lefohn et al. [190] demonstrated an efficient sparse GPU segmentation using linear level set methods.

In this work we propose a streaming solver framework suited to large volume segmentation. With 3D textures available to the commodity graphics hardware, we show that a 2D slicing is no longer required for a solution. This is also in contrast to [190] where the authors use a compact representation of the active volume packed into 2D textures. We solve the governing partial differential equations (PDEs) for a general case of any number of segmentation sub-domains (or *classes*). The number of *classes* is determined as the level set evolves, creating new *classes* while merging some of the existing ones. Every single class then gives rise to a partition of the volume. Our solver operates in a streaming fashion and makes optimal use of the available GPU memory. Global algorithms cannot be directly applied in a streaming compute environment. We employ streaming counterparts of various algorithms and show memory optimizations via a *host memory manager*. The result of the streaming solver is demonstrated with multi-domain segmentation along with speedup benchmarks for tri-linear and tri-cubic level set computations.

Background

The main idea behind a level set based segmentation method is to minimize an energy term over a domain by numerically solving the corresponding time varying form of the variational equation. Let us represent a volume by a scalar field $I(x,y,z) : \Omega \mapsto K$, where Ω is a bounded open subset of $\mathbb{R}^3$, and $K \subset \mathbb{R}$ is a bounded set of discrete intensity values sampled over a regular grid. In this setup, motion by mean curvature provides a deformable level set formulation where the surface of interest moves in the direction of normal at any point with velocity proportional to the curvature [223].

The deformable surface is represented by a level set of an implicit function $\phi(x,y,z) : \Omega \mapsto \mathbb{R}$. In level set methods, ϕ is generally chosen to be a signed distance function for certain properties that they offer [223, chap. 6]. Toward a segmentation approach, various energy formulations are possible. The energy functional is further penalized by a regularizing term that introduces smoothness in the resulting surface. In this paper, we show how the modified Mumford-Shah functional of Chan and Vese [79] can be minimized in a higher order fashion and also used to perform a multi-domain segmentation. The Mumford-Shah energy functional has a distinct advantage of producing better segmentation regions in absence of sharp edges as compared to an edge based energy functional. Consider an evolving interface $\Gamma = \{(x,y,z) : \phi(x,y,z) = 0\}$ in Ω, denoting $\Gamma^+ = \{(x,y,z) :$

$\phi > 0\}$ as the interior of the volume bounded by Γ and $\Gamma^- = \{(x,y,z) : \phi < 0\}$ as the exterior of the volume bounded by Γ. A modified Mumford-Shah energy functional with regularization can be written as:

$$\begin{aligned} F(c_1,c_2,\Gamma) &= \mu \cdot Area(\Gamma) + \nu \cdot Volume(\Gamma^+)) \\ &\quad +\lambda_1 \int_{\Gamma^+} |I-c_1|^2\,dxdydz \\ &\quad +\lambda_2 \int_{\Gamma^-} |I-c_2|^2\,dxdydz, \end{aligned} \tag{1.64}$$

where $\mu \geq 0, \mu \geq 0, \lambda_1 > 0$, and $\lambda_2 > 0$ are fixed scalar control parameters. A time varying variational form of equation 1.64 is:

$$\begin{aligned} \frac{\partial \phi}{\partial t} = \quad \delta_\varepsilon(\phi) \quad & \left[\mu \nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|}\right) - \nu \right. \\ & \left. - \lambda_1 (I_0 - c_1)^2 + \lambda_2 (I_0 - c_2)^2 \right], \end{aligned} \tag{1.65}$$

where, c_1 and c_2 are averages in Γ^+ and Γ^- respectively. Bajaj et al. [40] propose to solve the higher order regularizing term in equation 1.65 by tri-cubic B-spline interpolation to compute accurate higher order derivatives of ϕ.

Multi-domain segmentation Equation 1.65 defines two regions with respect to the zero level set surface of ϕ, i.e., $\phi > 0$ and $\phi < 0$. Often in segmentation, we need more than two partitions of the input signal. Vese and Chan [301] show that multiple level set evolutions can be used to keep track of multiple regions in the signal. In a multi-domain setup a single implicit function ϕ is replaced by a vector valued $\Phi = \{\phi_0, \phi_1, \ldots, \phi_{m-1}\}$ function where m is the total number of implicit functions that are combined to give a maximum of $n = 2^m$ partitions of Ω. Equation 1.65 is replaced by a system of m PDEs. We compactly write this system as:

$$\begin{aligned} \frac{\partial \phi_i}{\partial t} = \delta_\varepsilon(\phi_i) \left[\mu \nabla \cdot \left(\frac{\nabla \phi_i}{|\nabla \phi_i|}\right) - \nu - \sum_{k=0}^{2^{m-1}-1} \left\{ \left((I - c^1_{i,k})^2 \right.\right.\right. \\ \left.\left.\left. - (I - c^0_{i,k})^2\right) \prod_{p=0}^{m-1, p\neq i} \left(b_{k,p} + (-1)^{b_{k,p}} H_\varepsilon(\phi_p)\right) \right\} \right], \end{aligned} \tag{1.66}$$

for $i \in [0, m-1]$, where

$$b_{k,p} = p^{th} \text{ bit in binary representation of } k, \in \{0,1\}, \tag{1.67}$$

$$H_\varepsilon(z) = \frac{1}{2}\left(1 + \frac{2}{\pi}\tan^{-1}\left(\frac{z}{\varepsilon}\right)\right), \tag{1.68}$$

$$c^0_{i,k} = mean(I) \text{ in } \begin{cases} (x,y,z): & \phi_p > 0 \text{ if } b_{q,p} = 1, \\ & \phi_p < 0 \text{ if } b_{q,p} = 0, \text{, and} \\ \text{with} & q = 2k - k \bmod 2^i \end{cases} \tag{1.69}$$

$$c^1_{i,k} = mean(I) \text{ in } \begin{cases} (x,y,z): & \phi_p > 0 \text{ if } b_{q,p} = 1, \\ & \phi_p < 0 \text{ if } b_{q,p} = 0, \\ \text{with} & q = 2k + 2^i - k \bmod 2^i \end{cases} \tag{1.70}$$

$\forall p \in [0, m-1]$.

Consider $q \in \mathbb{Z}^+$ such that its i_{th} bit is 1 for Γ_i^+, 0 for Γ_i^-, and 0 if $i > (m-1)$. In this way, q spans the n possible regions induced by Φ. In equation 1.66, $c^1_{i,k}$ (or $c^0_{i,k}$) represents the average of intensity values of I in Γ_i^+ (or Γ_i^-) where the other bits determine inside or outside for rest of the implicit functions. To generate the index for a region, we enumerate the possible 2^{m-1} values and insert a 1 or a 0 at the i_{th} bit. Alternative expressions for q in equations 1.69 and 1.70 are

$$q = (k - (k \wedge (2^i - 1))) \ll 1 + k \wedge (2^i - 1), \text{and} \tag{1.71}$$

$$q = (k - (k \wedge (2^i - 1))) \ll 1 + 2^i + k \wedge (2^i - 1) \tag{1.72}$$

respectively. We find that this representation for keeping track of various regions of Φ is not only compact but also efficient for implementation on a constrained compute environment like the GPU.

Solver framework

We provide a framework for streaming computation of the solution to the differential equations (1.65 or 1.66) based on NVIDIA's parallel framework called CUDA [262]. The complexity of the solver is increased by the fact that m simultaneous PDEs need to be solved to arrive at a solution to equation 1.66. Our solver has a very small memory footprint on the *device* (GPU) compared to the actual volume that it can handle. On the CPU side, memory requirement is of the order of the number of implicit functions. Our GPU computational setup consists of a host memory manager to handle data streaming and a set of CUDA kernels to operate on parts of the data fetched to the *device* by the host.

Host memory manager The data on the *device* is handled in manageable chunks of a 3D sub-volume called a *computational volume*. The memory manager splits the entire volume into minimum possible number of sub-volumes of size of the computational volume. The size of such a computational volume is chosen such that:

- computations are performed by one thread per voxel.
- the computational volume fits into the *device* memory.

The computational volume is further divided into the CUDA grid and block for thread invocation. CUDA allows for a 3D block of threads, but not a 3D grid of blocks. We create a logical hierarchy by creating a 3D grid and a 3D block of voxels and assigning each voxel with a thread to process it. A linear grid of blocks is logically mapped to a 3D grid of blocks. For a 128^3 computational volume, a typical grid size could be 16^3 blocks with each block consisting 8^3 threads. Note that the number of threads in a block cannot exceed 512 with the version 2.0 of CUDA. The hierarchy of thread blocks, grid and the computational volume is shown in Figure 2.36.

For operations that involve accessing voxel neighbors (e.g., finite differencing or convolution filtering), the host memory manager appropriately pads the computational volume to enable required number of shared voxels around the border of the computational volume. This effectively reduces the size of the volume to incorporate neighbors along the border of the volume. Further, the full volume might not be an exact multiple of the computational volume, therefore the memory manager pads the computational volume on the boundary with null values in the empty space.

Memory copies between *device* and host are performed in size of the computational volume. Special care is taken while copying data along the border of the full volume. The memory manager also dynamically allocates and frees the *device* memory if required by a kernel. Figure 2.37 shows a sketch of the setup.

Current sub-volumes from the image, the implicit functions and cubic coefficients are first transferred to the GPU memory as 3D textures. Individual kernels then operate on these sub-volumes and the results are stored on the GPU global memory, which are then transferred back to the host volumes. Details of PDE evolution algorithms and respective kernels are presented in the next section.

Level set evolution

Following is our general approach in solving equation 1.66:

1. Interface initialization.
2. Signed distance field computation.
3. Average values computation.
4. Cubic coefficients computation.
5. PDE time stepping.
6. Reinitialization.
7. Repeat steps (3) - (6) until convergence.
8. Level set extraction and domain labeling.

Majority of these steps can be executed in a streaming fashion with an exception of average value computation.

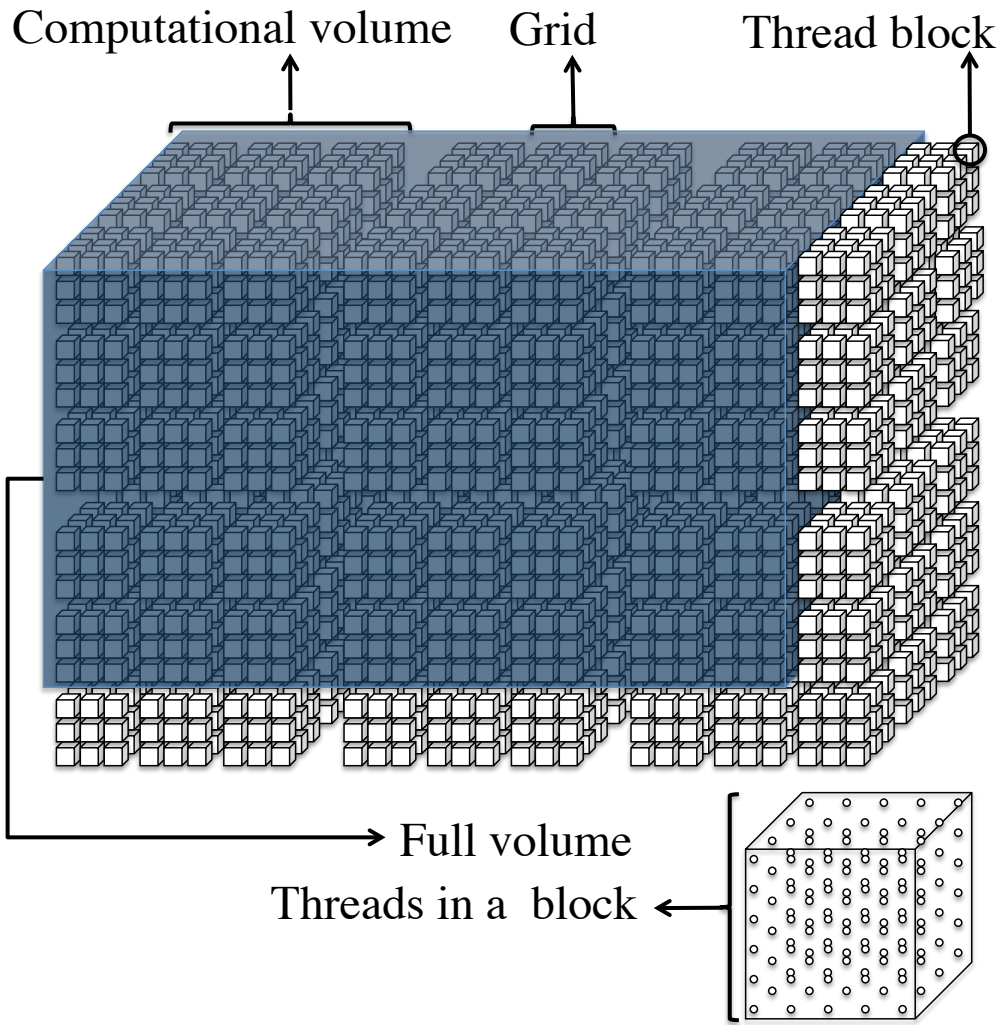


Figure 2.36: Volume hierarchy for spawning CUDA threads.

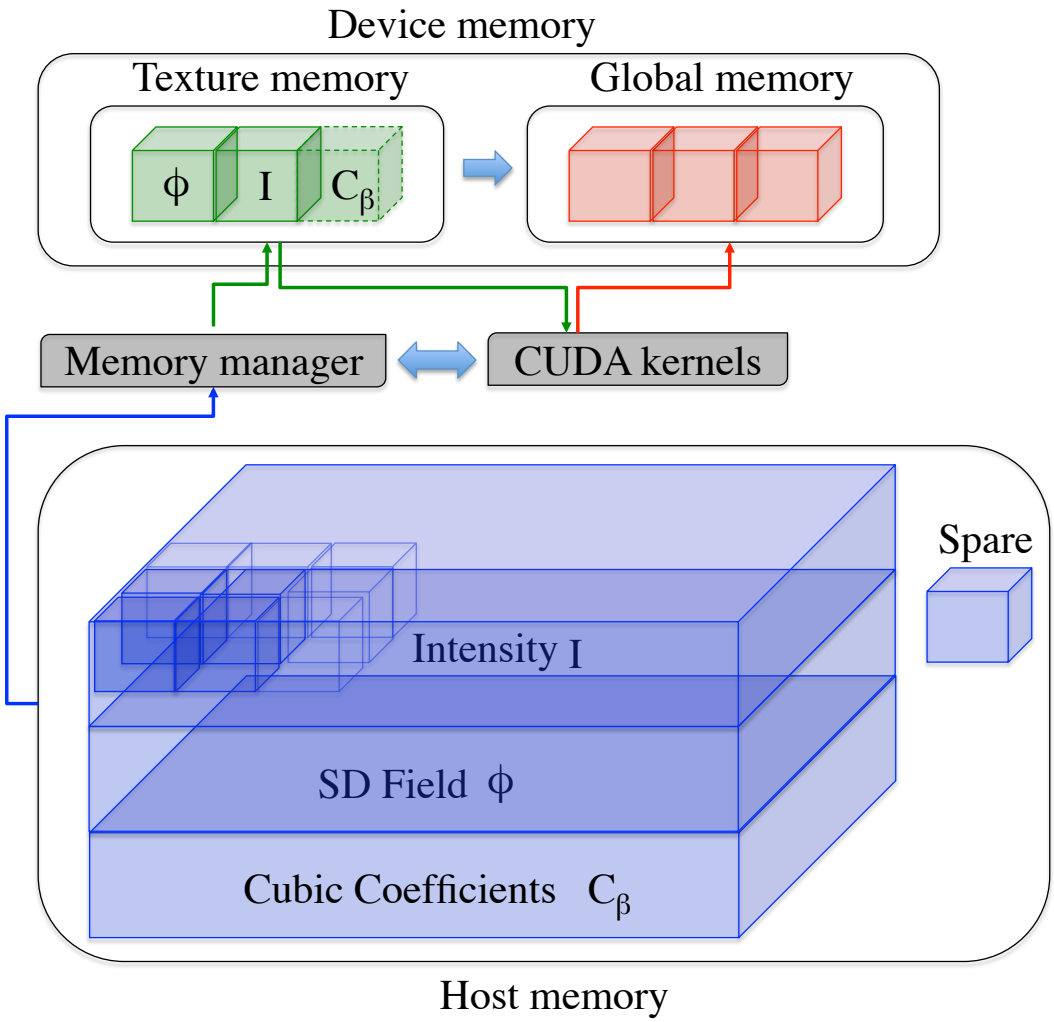
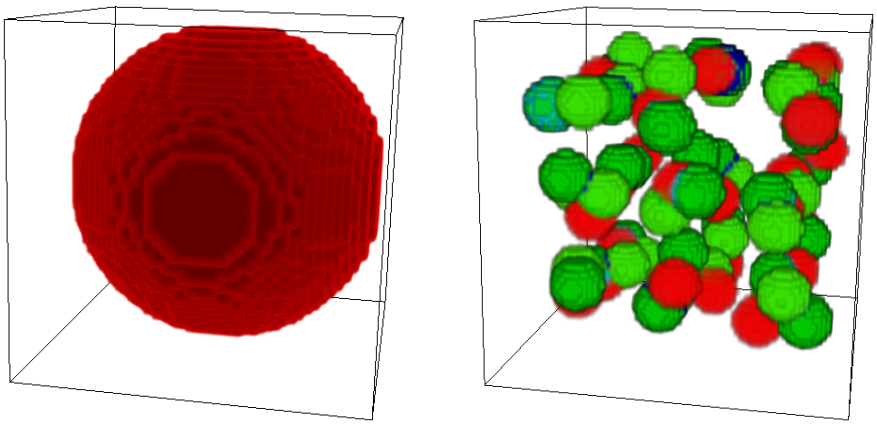


Figure 2.37: The memory management software architecture

Interface initialization We choose to initialize the level set interface Γ to a bounding box or to a super-ellipsoid for a two domain segmentation. Multi-domain initialization should ensure that all the possible *classes* occupy non-zero region in space at the start so that all the domains have a scope of evolution. The domain Ω is partitioned into smaller sub-domains and each sub-domain is assigned small super-ellipsoids (with randomized centers) that form the interface Γ for each implicit function (see Figure 2.38). Chan and Vese [301] observe a better and faster convergence in a 2D case for an initialization of the latter kind. Our tests confirm this observation for the 3D case.



(a) Two domain interface (b) Multi-domain interface

Figure 2.38: Interface initialization for multi-domains with different colorss

The kernel module for interface initialization computes the implicit function Φ such that:

$$\phi_i(x,y,z) = \begin{cases} k, & \text{if } (x,y,z) \in \Gamma^+ \\ -k, & \text{otherwise,} \end{cases} \tag{1.73}$$

where $i \in [0, m-1]$, and $k \in \mathbb{R}^+$ is a constant.

Signed distance field We create a narrow band signed distance field on GPU using the d-pass approach by Sharma and Anton [262] to compute the distance field in d layers where $2d$ is the integer width of the narrow band. This is a streaming algorithm to create a Chamfer distance field [59] that uses optimal values of coefficients for distance multipliers to minimize accuracy error with an actual distance field. The algorithm has complexity $O(dN)$, where N is the total number of voxels in the volume.

The distance field is constructed layer by layer until the d layers are formed, making a narrow band of width $2d$. Every voxel is updated based on the values of the neighbors. The resulting layer has distance values that are locally Euclidean. The kernel to compute a signed distance layer operates on the computational volume that has a shared 1-voxel border, thus updating 126^3 voxels in a computational volume.

Average values In a two-domain segmentation, the zero level set of ϕ divides Ω into two regions. Average values c_1 and c_2 can be easily computed over these regions.

Multi-domain segmentation creates more than two regions corresponding to every class of segmentation. We use binary indexing (as explained in subsection 2.1.5) to keep track of inside and outside in every implicit function. Thus, for any $q \in [0, n]$, we can compute the average value of image intensity.

Computation of average values is a serial operation and requires parsing all the values in the dataset. Reduction algorithms do exist for a parallel computation of sum like operations [57]. A CUDA implementation of the same exists as the CUDPP library by Harris and Sengupta [150]. With CUDPP, however, it is difficult to sum up datasets that cannot fit into the *device* memory, thus requiring some sort of data slicing. In our experience, the overhead of data slicing, setting up the prefix sum and computing average values turns out to be more expensive than a fast CPU computation of the averages.

Cubic coefficients Let

$$\beta^3(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}|x|^3, & 0 \le |x| < 1, \\ \frac{1}{6}(2-|x|)^3, & 1 \le |x| < 2, \\ 0, & 2 \le |x|. \end{cases} \tag{1.74}$$

be the univariate cubic B-spline basis function over the knots $\{-2,-1,0,1,2\}$, and let

$$s(x) = \sum_{i=1}^{n-1} c_i \beta^3(x-i) \tag{1.75}$$

be a cubic spline function. Then the spline interpolation problem is to determine the coefficients $\mathbf{c} = \{c_i\}_{i=1}^{n-1}$ such that the following interpolation conditions

$$s(k) = \sum_{i=1}^{n-1} c_i \beta^3(k-i), \;\; k = 1, \cdots, n-1 \tag{1.76}$$

are satisfied for any given function values $\{s(k)\}_{k=1}^{n-1}$. To avoid inversion of a large matrix to solve the linear system 1.76, a better approach is to compute these coefficients in a recursive fashion (see [40]).

Cubic coefficients are computed for a set of data values sampled along any direction. Tri-cubic spline coefficients can be derived from these coefficients by computing cubic coefficients along each direction sequentially. These correspond to tensor product B-splines in $\mathbb{R}^3$. Rather than operating in the computational volume, we derive coefficients along an axis in planar sections orthogonal to one of the co-ordinate axes (see Figure 2.39). The computation takes place in three steps, sequentially along each coordinate direction.

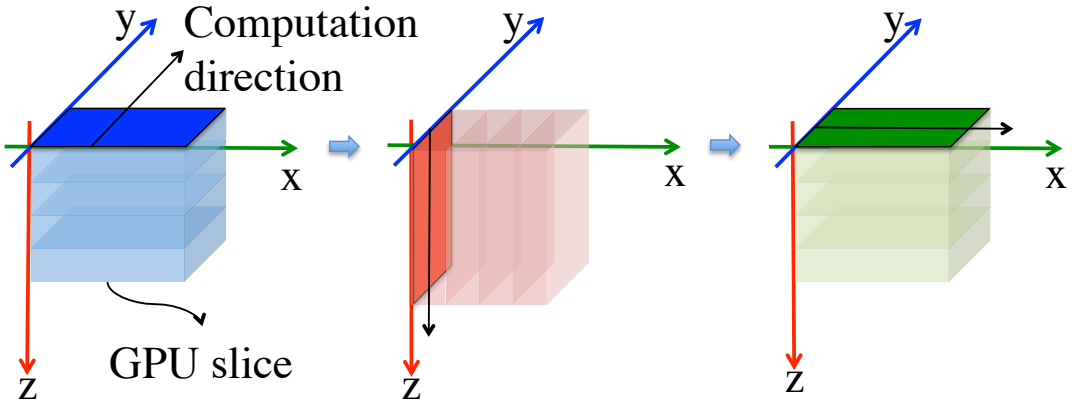


Figure 2.39: Tri-Cubic tensor product B-spline coefficient computation.

The host memory manager determines the largest possible volume slice (*GPU slice*) that can fit into the available *device* memory. The full volume is then processed in sizes of the GPU slice. For every plane in the GPU slice, cubic coefficients are computed along the segments parallel to one of the coordinate axes. The resulting coefficients are written to *device* array of same size as the GPU slice and copied back to the host array holding the coefficients. The CUDA kernel for computing coefficients works on a linear section per thread. The intermediate sequences are not stored, but recomputed during the recursive process to evaluate c_i.

PDE time stepping The PDEs are solved by discretizing equation 1.66 to compute the lower order term and by computing the spline derivatives for the higher order curvature term. Each PDE has a single higher order term, and 2^{m-1} terms involving average values. We again use binary indexing to enumerate the lower order terms in every PDE.

Since all the PDEs must be updated simultaneously (i.e., every voxel in all implicit functions must be updated in parallel), we need cubic coefficients for all the implicit functions in the *device* memory at all times. However, CUDA does not allow

dynamically creating texture references. Furthermore, a 4D texture (array of 3D textures) is also not possible in CUDA. Therefore, we simulate a 4D texture by a large 3D texture containing computational volume sized coefficient sub-volumes for all implicit functions.

There is a possibility of wasting some *device* memory here for very large number of implicit functions, since for some m ($>$16) it might not be possible to have a rectangular 3D array. This is because the largest 3D texture in CUDA can be of size $2^{11} \times 2^{11} \times 2^{11}$, and we hit this limit along one dimension for a computational volume of size $128 \times 128 \times 128$ and $m > 16$. In such a case, we try to minimize the unusable *device* memory locked in the texture by computing optimal number of three positive factors m_x, m_y and m_z for a number $\hat{m} \geq m$ such that $(\hat{m} - m)$ is minimized and $\hat{m} = m_x \times m_y \times m_z$. The three factors are the number of computational volumes along the coordinate axes. In doing so, we try to make m_x as large as possible, followed by a similar heuristic for m_y.

With sub-volumes of Φ, I and the coefficients cached on the GPU, PDE update is computed for every voxel and for all the implicit functions.

Results We present the results of our multi-domain segmentation on a CT (Computed Tomography) volume of a human thoracic region, followed by GPU performance statistics. The tests are produced on an NVIDIA Tesla C870 machine with Dual-Core AMD Opteron™ processor 2218 running at clock speed of 2.6 GHz, and a physical memory of 2 GB. The GPU has 16 multiprocessors (128 processor cores) running at clock speed of 1.35 GHz and an onboard memory of 1.6 GB. The program has been implemented in the freely available software UT-CVC image processing and visualization software called VolRover [14].

Figure 2.40 shows a volume visualization of our human thoracic region dataset. The CT volume has a size of $256 \times 256 \times 256$ voxels. Segmentation parameters for this volume are: $\lambda_1 = \lambda_2 = 1$, $\mu = .000005 \times 255 \times 255$, $\varepsilon = 1$, and $m = 3$. A time stepping $\Delta t = 0.01$ is used. The interface is initialized to a super-ellipsoid of power $p = 2$ with a bounding box offset of 5 voxels from all sides. A total of 60 solver iterations produced the segmentations shown in Figure 2.41. The segmentation yields four prominent *classes*. These *classes* separate regions of ribs, spinal chord, lungs and bronchioles as shown in Figure 2.41(a), (b), (c), and (d) respectively.

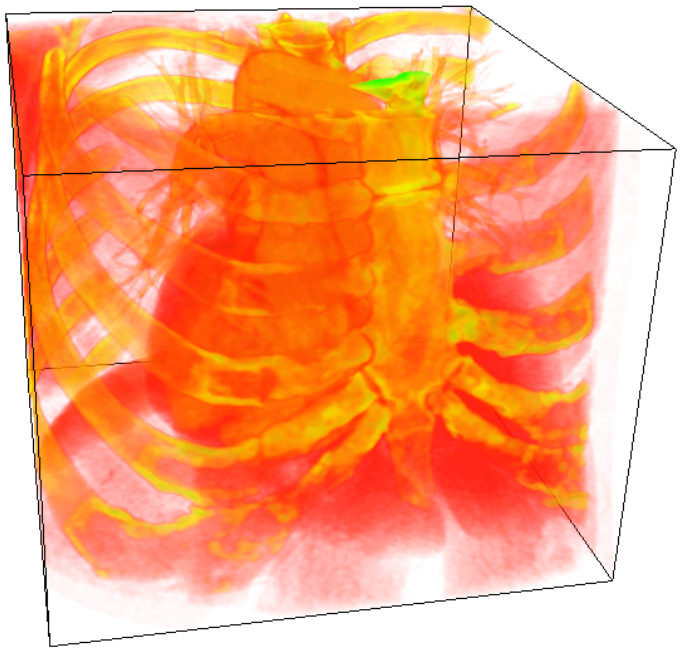
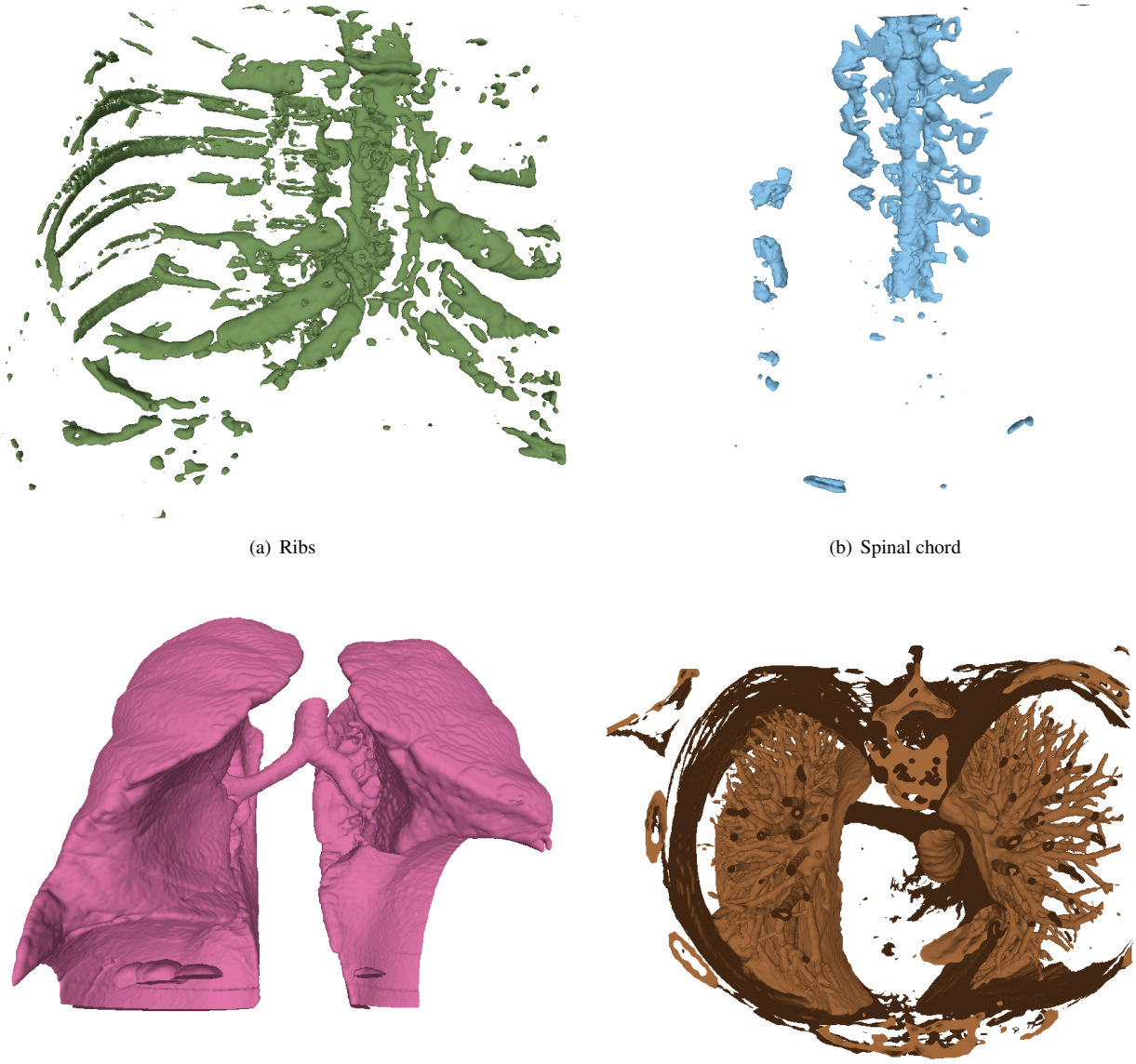


Figure 2.40: Volume visualization of computed Tomography (CT) volume of a human thoracic region dataset.



(a) Ribs (b) Spinal chord

(c) Lungs (d) Bronchioles

Figure 2.41: Multi-domain segmentation of the CT volume into 4 *classes*.

Interface initialization CUDA kernel has low arithmetic intensity, therefore a very high speedup of about 24x to 25x is achieved. On the other hand, PDE updates are expensive in terms of arithmetic operations, thus giving a speedup of about 3x with tri-linear update and that of 11x with tri-cubic update. It should be noted that the tri-cubic PDE update is faster than the tri-linear one since the tri-linear uses finite differencing to compute double derivatives, while the tri-cubic uses texture lookups and fewer computations. Tri-cubic B-spline coefficients are expensive to compute and we obtain a speedup of about 3x here.

Performance speedups of GPU computations compared to CPU ones are shown in Figures 2.42 and 2.43. The speedups show a general trend (non-linear) of increase in performance with increase in size of volume. However, for very large volumes (e.g. $1602 \times 1125 \times 195$ volume), both the host and the *device* computations slow down by a large extent due to excessive memory paging. Since the *device* computation reads the volume in small chunks of memory, we surmise that this access pattern

hits the performance even more by increasing the number of page faults.

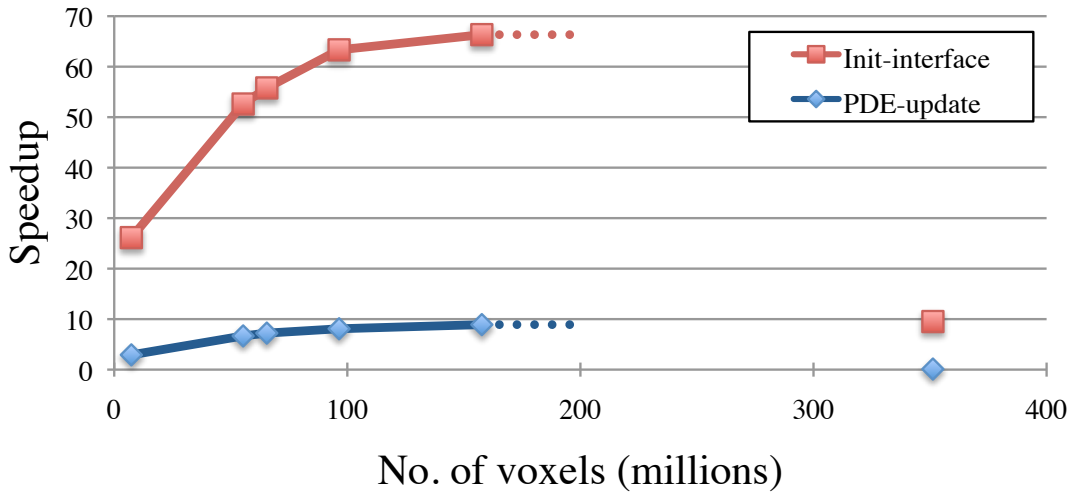


Figure 2.42: GPU speedup for tri-linear level set segmentation

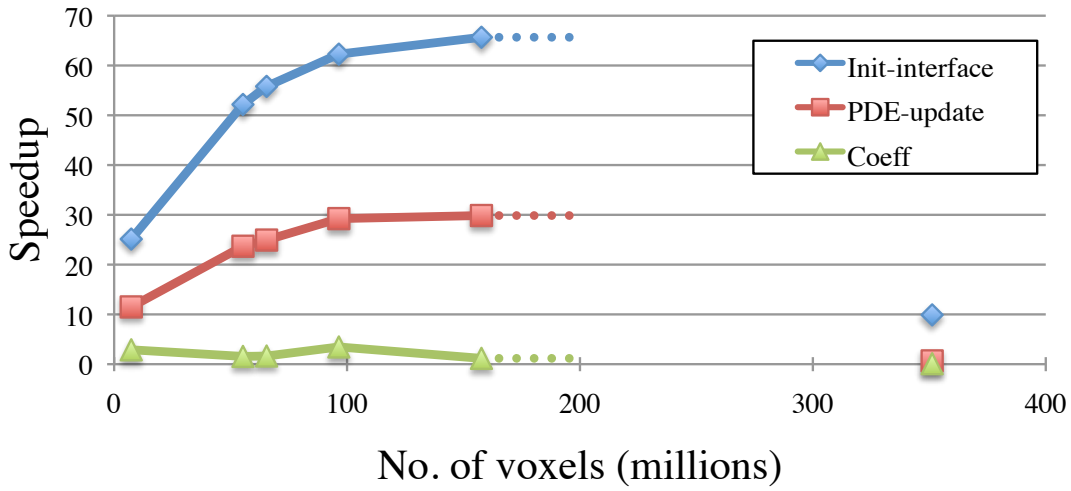


Figure 2.43: GPU speedup for tri-cubic level set segmentation

Conclusions

In this work we present an efficient parallel (multi-core) CUDA computation of a multi-domain, higher order level set method applied to a Mumford-Shah like energy functional. The higher order framework can be easily used with other energy functional as well. We show results of the segmentation on CT volumes along with performance speedups obtained with our GPU based implementation. The overall performance gains we obtain is 20x for the two-domain segmentation and 10x for the multi-domain segmentation.

2.2 Alignment

operations of matching two images, returning the transformations

The following algorithm is taken from [10] (AAKSB2006).

Formulation of the problem. In many areas of science and engineering, we have two images $I_1(\vec{x})$ and $I_2(\vec{x})$ (2-D or 3-D) which represent the same object but viewed from different angles and positions, in a possibly different scale. Since these images represent the same object, they can be obtained from each other by an appropriate shift $\vec{x} \to \vec{x} + \vec{a}$, rotation $\vec{x} \to R\vec{x}$, and scaling $\vec{x} \to \lambda \cdot \vec{x}$, i.e., $I_2(\vec{x}) \approx I_1(\lambda \cdot R\vec{x} + \vec{a})$.

In many practical situations, we do not know the relative orientation of the two images. In such situations, it is desirable to *register* these images, i.e., to find the rotation and the shift after which the images match as much as possible.

A similar problem occurs when we have the images of two different objects whose shapes should match. For example, we may have images of two bioactive molecules. We know that in vivo, these molecules interact because one of these molecules "docks" to the other one, i.e., gets into a position where their surfaces match. In such situations, it is also important to find orientation and shift corresponding to this match.

Comment. Sometimes, the images also differ in lighting conditions, as a result of which we may have $I_2(\vec{x}) \approx C \cdot I_1(\lambda \cdot R\vec{x} + \vec{a})$ for some unknown factor C.

General image registration problem vs. its particular cases. In the *general* problem, images may differ by shift, by orientation, and by scale. In such a general case, we need to find all three groups of parameters: shift $\vec{a}$, rotation R, and scaling λ.

In practice, we sometimes do not need to find all three of these parameters. For example, we sometimes know that the images are already similarly oriented and they are in the same scale, so all we need to do is find the shift $\vec{a}$ between the two images.

In other practical problems, we know that the images are scaled right, but we need to find the shift $\vec{a}$ and the rotation R. For example, when a molecules docks into another molecule, it can shift and rotate, but it cannot seriously shrink or expand. Thus, in the protein docking problem, we must find the shift $\vec{a}$ and the rotation R – but not the scaling.

In short, in general, all three groups of parameters are unknown: the shift $\vec{a}$, the rotation R, and the scaling λ. In practice, in addition to this complex general image registration problem, we sometimes face (somewhat) simpler particular cases of the general image registration problem, cases in which we only need to determine some of these parameters.

First Case: Registration of Similarly Oriented 2-D and 3-D Images

Formulation of the Problem. We have two images $I_1(\vec{x})$ and $I_2(\vec{x})$, we know that they are obtained from each other by an (unknown) shift $\vec{a}$, i.e., that $I_2(\vec{x}) \approx I_1(\vec{x} + \vec{a})$, and we want to find this shift.

How an image is represented. In the 2-D case, each image on a 2-D $n \times n$ grid consists of n^2 intensity values at different grid points. In the 3-D case, an image on a 3-D $n \times n \times n$ grid consists of n^3 intensity values. In general, an image is represented by n^d intensity values, where $d = 2$ or 3 is the dimension of the image.

Registration in time $O(n^d)$. In some real-life problems, we can register the two similarly oriented images in the smallest possible time $O(n^d)$. One example if when we are registering the two images which have clear *landmarks*: e.g., points where the intensity have the largest possible value, or saddle points which are useful in describing the structure of the image.

In such situations, to match the two images, it is sufficient to find, for each of the two images, the point $\vec{x}$ with the largest possible intensity value; this point can be found by searching over all n^d points $\vec{x}$ so it can be done in $O(n^d)$ steps. Once the corresponding landmark locations $\vec{d}_1$ and $\vec{d}_2$ are found, we can find the desired shift $\vec{a}$ as $\vec{a} = \vec{d}_2 - \vec{d}_1$.

For many important images, we cannot use this fast algorithm for image registration, because there are no easy-to-find landmark locations – namely, for every special point in the image, there are several similar points in the same image. For example, in multi-cellular biological images, whether we have a point on the border between several cells, or a special point of

a special structure within a cell, there are many different points with the similar values of intensity at different cells. Similarly, in a satellite image, if the maximum of the image intensity corresponds to, e.g., road intersection or a river, we may have many road intersections and several rivers within the same image.

In some such situations, however, we can still use a fast ($O(n^d)$) algorithm for image registration. For example, in astronomical images, we often have an image of an object surrounded by empty space. In this case, we can use the moments to find out the shift between the images. It is easy to see that when we apply a shift to an image, i.e., replace the original image $I_1(\vec{x})$ with a shifted image $I_2(\vec{x}) = I_1(\vec{x}+\vec{a})$, then the 0-th order moment (overall intensity) $M_i = \int I_i(\vec{x})\,d\vec{x}$ does not change ($M_2 = M_1$), while the 1-st order moments $M_{ij} = \int I_i(\vec{x}) \cdot x_j \cdot d\vec{x}$ change to $M_{2i} = M_{1i} - a_i \cdot M_1$. Thus, we can find the desired shift $\vec{a}$ by comparing the corresponding moments: $a_i = (M_{1i} - M_{2i})/M_1$. An integral is, in effect, a sum over all n^d grid points, so computing each of the integrals M_1 and M_{ij} requires $O(n^d)$ steps. Thus, we still need computation time $O(n^d)$.

This method has a natural geometric meaning: the ratio M_{ji}/M_1 is the i-th coordinate of the image's center of gravity. Thus, in effect, this algorithm consists of the following three steps:

- first, we compute the center of gravity $d_{1i} = M_{1i}/M_1$ of the image $I_1(\vec{x})$;
- then, we compute the center of gravity $d_{2i} = M_{2i}/M_2$ of the image $I_2(\vec{x})$;
- finally, we find the desired shift as the difference between the centers of gravity: $\vec{a} = \vec{d}_1 - \vec{d}_2$.

Comment. This reformulation works also when the images are not only shifted, but also "re-scaled", i.e., $I_2(\vec{x}) \approx C \cdot I_1(\vec{x}+\vec{a})$ for some (unknown) constant $C > 0$, due, e.g., to different lighting conditions related to these two images.

Situations when $O(n^d)$ algorithms are not applicable. In many practical situations, none of the above $O(n^d)$ algorithms is applicable. For example, in satellite and multi-cellular images, not only we do not have landmark points, we also do not have empty space around an image. As a result, e.g., the shifted satellite image contains the values which are not present in the original image. When both images cover the same reasonably homogeneous area, then the intensity is of about the same value throughout both images, and hence, the center of gravity of each image is close to the geometric center of the image. Thus, if we apply the above center-of-gravity algorithm to detect the shift between the images, then, no matter what the actual shift is, the above algorithm will return (approximately) 0.

Straightforward approach: computational complexity. A natural idea is to look for a vector $\vec{a}$ for which the distance

$$\int (I_2(\vec{x}) - I_1(\vec{x}+\vec{a}))^2\,d\vec{x} \tag{2.77}$$

attains the smallest possible value.

For this formula, a straightforward approach would mean that we compute the value of this scoring function for all possible shifts $\vec{a}$, and then find the shift for which this value is the smallest.

Each image on a 2-D $n \times n$ grid consists of n^2 intensity values at different grid points. Computing each integral requires time $O(n^2)$: for each pixel $\vec{x}$, we need one subtraction and one multiplication to compute the value $(I_2(\vec{x}) - I_1(\vec{x}+\vec{a}))^2$, and then we must add all these values to get the integral.

According to the straightforward approach, we must compute the value of the L^2-norm for all possible shifts $\vec{a}$. For an $n \times n$ grid, it is reasonable to consider n^2 possible shifts. Computing each integral requires time $O(n^2)$, so overall, we need time $O(n^2) \cdot O(n^2) = O(n^4)$.

For many real-life images, the size n is approximately 10^3, so $n^4 \approx 10^{12}$ computational steps require a large mount of time. It is therefore desirable to find faster algorithms for image registration.

It is known that many signal and image processing techniques can be made faster if we use Fast Fourier Transform (FFT); see, e.g., [91]. FFT of an image of size N requires $N \cdot \log(N)$ steps, so for an image of size $N = n^d$, we need $O(n^d \cdot \log(n))$ steps.

Using Fast Fourier Transform for shift detection. For a shifted image $I_2(\vec{x}) = I_1(\vec{x}+\vec{a})$, the Fourier transform has the form $F_2(\vec{\omega}) = \exp(\mathrm{i} \cdot \vec{\omega} \cdot \vec{a}) \cdot F_1(\vec{\omega})$. Thus, in the presence of measurement noise and inaccuracy, when we have $I_2(\vec{x}) \approx I_1(\vec{x}+\vec{a})$, we have $F_2(\vec{\omega}) \approx r(\omega) \cdot F_1(\vec{\omega})$, where $r(\omega) \approx \exp(\mathrm{i} \cdot \vec{\omega} \cdot \vec{a})$.

The complex number $\exp(\mathrm{i}\cdot\vec{\omega}\cdot\vec{a})$ has magnitude 1; so, it is reasonable, for every spatial frequency $\vec{\omega}$, to find the value $r(\vec{\omega})$ for which the approximation error $|F_2(\vec{\omega}) - r(\vec{\omega})\cdot F_1(\vec{\omega})|^2$ is the smallest possible under the constraint that $|r(\vec{\omega})| = 1$.

By using the Lagrange multiplier method, we can show that the solution to this constrained optimization problem is given by

$$r(\vec{\omega}) = \frac{F_1(\vec{\omega})\cdot F_2(\vec{\omega})^*}{|F_1(\vec{\omega})\cdot F_2(\vec{\omega})|}. \tag{2.78}$$

In the ideal case, $r(\vec{\omega})$ is a sinusoidal wave, and its inverse Fourier transform is an impulse function $\delta(\vec{x}-\vec{a})$, i.e., a function which is only non-zero for $\vec{x} = \vec{a}$. It is therefore reasonable to find the shift $\vec{a}$ by applying the inverse FFT to the above function $r(\vec{\omega})$ and to find the shift as the point at which this inverse FFT attains the largest possible value.

We thus arrive at the following algorithm.

Resulting algorithm. We are given two images $I_1(\vec{x})$ and $I_2(\vec{x})$; we must find the shift $\vec{a}$ between these images. For that, we do the following:

- first, we apply FFT to both images $I_1(\vec{x})$ and $I_2(\vec{x})$, and compute the Fourier transforms $F_1(\vec{\omega})$ and $F_2(\vec{\omega})$;
- then, for each frequency $\vec{\omega}$, we compute the ratio (2.78);
- after that, we apply the inverse FFT to the resulting function $r(\vec{\omega})$ and obtain a new function $P(\vec{x})$.

Finally, we find the shift $\vec{a}$ as the value at which the function $P(\vec{x})$ attains its largest possible value.

Comment. This algorithm was first proposed in [183] (without the least squares justification); for 2-D images, it requires time $O(n^2\cdot\log(n))$. This same algorithm can be used for 3-D images, it then requires time $O(n^3\cdot\log(n))$.

Comment. The above method – based on traditional FFT techniques – is applicable when we have data on a rectangular grid. In many practical problems, e.g., in biomedical imaging, we often have areas with few grid points and areas with a large number of grid points. For such non-equispaced data, the traditional Fast Fourier Transform algorithm is not applicable.

What can we do in this situation? The idea behind Fourier transform is that instead of representing each image $I(\vec{x})$ by its values at different pixels, we use one of the possible bases – namely, the basis of sinusoid functions – and describe the image by the coefficients of this expansion:

$$I(\vec{x}) = \frac{1}{2\pi}\cdot\int F(\vec{\omega})\cdot\exp(\mathrm{i}\cdot\vec{\omega}\cdot\vec{x})\,d\vec{\omega}.$$

For non-equidistant data, we cannot use only this expansion to get an efficient algorithm. Thus, it is natural to use the expansion over other bases – e.g., polynomials – to help computations.

Several fast algorithms for nonequispaced Fourier transform have been proposed; see, e.g., [8, 55, 123, 118, 123, 237, 236, 274, 278, 305]. The main difference between the standard FFT and the nonequidistant FFT (NFFT) is that the standard FFT produces the *exact* Fourier transform, while the nonequidistant FFT only compute the values of the Fourier transform with a given accuracy.

Many of the NFFT algorithms use (local) polynomial approximations as an intermediate computational step; for example:

- in [278], an interval domain is divided into subintervals, and Chebyshev polynomials are used in each subinterval;
- in [8], Taylor polynomials are used instead of the Chebyshev ones;
- splines (a special class of locally polynomial functions) are used in [123, 237, 274].

In [17, 32, 73, 19], the spline-based methods from [123, 237, 274] and several other similar techniques have been successfully applied to the docking problem; the comparison seems to indicate that the spline-based methods are indeed the fastest.

Registration of General 2-D and 3-D Images

Formulation of the problem. We have two d-D images $I_1(\vec{x})$ and $I_2(\vec{x})$; we must find the shift $\vec{a}$ and the rotation R for which $I_2(\vec{x}) \approx I_1(R\vec{x}+\vec{a})$.

Registration in time $O(n^d)$. To find the shift between the two images, it is sufficient to find one landmark point. To find both shift and rotation, we can use two different landmarks:

- we can use the first landmark to find the shift between the two images, and then
- we can compare the location of the second landmark in both images, and thus find the corresponding rotation.

Finding the landmarks in the image requires time $O(n^d)$.

If there are no easily detectable landmarks, but the images are objects surrounded by an empty space, then we can use the moments method to determine both shift and rotation. The values of the 0-th and 1-st order moment M_i and M_{ij} could only determine the shifts. So, to find the rotation, we must also use the second order moments $M_{ijk} = \int x_j \cdot x_k \cdot I_i(\vec{x})\, d\vec{x}$.

To compare the second order moments (also called moments of inertia) of the two images, it is necessary to first shift both images to the same point of origin, e.g., to the center of gravity. As a result, we replace the original value M_{ijk} with the new value $M_{ijk} - d_j \cdot d_k \cdot M_i$, where $d_j = M_{ij}/M_i$ is the j-th coordinates of the center of gravity.

Second order moments of each image form a symmetric non-negative definite matrix. Thus, this matrix has orthogonal eigenvectors. When the image rotates, these eigenvectors rotate as well. So, we can find the rotation angle by comparing the orientations of the eigenvectors.

New algorithm for the general case: motivation and description. In general, the images may not have clear landmarks and they may not be surrounded by an empty space, so the above $O(n^d)$ methods may not be applicable. Let us show that in these case, we can use FFT-based techniques.

It is known that if the two images $I(\vec{x})$ and $I'(\vec{x})$ differ not only by shift but also by a rotation R and a scaling λ, the absolute values $M(\vec{\omega})$ and $M'(\vec{\omega})$ of their Fourier transforms $F(\vec{\omega})$ and $F'(\vec{\omega})$ differ from each other only by the corresponding rotation and scaling.

While the original images were not surrounded by an empty space, in many practical situations, the corresponding magnitudes of Fourier transforms are actually rapidly decreasing as the spatial frequency ω increases; see, e.g., [256, 257].

Comment. For smooth images, the magnitudes of their Fourier transforms do decrease with frequency. However, for non-smooth images, e.g., for a point source $I(\vec{x}) = \delta(\vec{x} - \vec{a})$ located at a point $\vec{a}$, its Fourier transform is equal to $\exp(-\mathrm{i} \cdot \vec{\omega} \cdot \vec{a})$. So, the magnitude of this Fourier transform is equal to 1 for all the frequencies and thus, does not decrease with frequency. Thus, if both given images contain a bright spot on a reasonably uniform background, then we cannot use this moments technique to find the rotation that registers these images. However, the case of bright spots (landmarks) is exactly the case when we can use landmark algorithms.

In situations when the magnitude of the Fourier transform decreases with frequency, we can use the moments method to find the appropriate rotation. Namely, based on the function $M_1(\vec{\omega})$, we compute the second moments $M_{1ij} \stackrel{\text{def}}{=} \int M_1(\vec{\omega}) \cdot \omega_i \cdot \omega_j\, d\vec{\omega}$; similarly, based on the function $M_2(\omega)$, we compute the second moments

$$M_{2ij} = \int M_2(\vec{\omega}) \cdot \omega_i \cdot \omega_j\, d\vec{\omega}.$$

For each of these matrices, we compute the eigenvectors corresponding to the largest eigenvalue; by comparing the orientations of these eigenvectors, we can then find the desired rotation R in time $O(n^d)$.

Comment. Images are usually given on a 2-D or 3-D grid, in the shape of a box. When we rotate the images an transform them back into a box shape, the corner information disappears. The only values which are preserved after all rotations are the values inside a circle (sphere) subscribed into the box. Since the corner values cannot match anyway, it is reasonable, before computing the moments, to only consider values $M(\vec{\omega})$ inside this circle (sphere).

Once we know the rotation R, we can align the images – e.g., by rotating the Fourier transform $F_1(\vec{\omega})$ of one of the image $I_1(\vec{x})$ to $F_1(R\vec{\omega})$. Now, the images $I_1(R\vec{x})$ and $I_2(\vec{x})$ differ only by shift, so we can use the above $O(n^d \cdot \log(n))$ method to find this shift.

Our preliminary results show that the resulting method indeed works well for registering 3-D images.

New algorithm: computational complexity. Overall, in the d-D case, we thus need $O(n^d \cdot \log(n))$ time to compute the original Fourier transforms, $O(n^d)$ time to find the rotation, and then $O(n^d \cdot \log(n))$ time to find the shift – to the total of

$$O(n^d \cdot \log(n)) + O(n^d) + O(n^d \cdot \log(n)) =$$

$$O(n^d \cdot \log(n))$$

computational steps.

	$g = 0$	$g = \|\|\nabla f\|\|$	$g = 1$
FSC	0.052	0.085	0.068
Resolution	19.23Å	11.76Å	14.71Å

Table 2.4: Resolution compare with different regularization terms.

2.3 Projections

operations that produce multiple projections from a single 3D map

2.3.1 Forward Radon Transform

2.3.2 Fourier Transform

2.4 Reconstruction

operations that inverts projections, i.e. reconstructs a single 3D map from multiple images

2.4.1 Inverse Radon Transform

The following algorithm is taken from [194] (Li2009Reconstruction).

Preliminary and Notations

A lot is cut out here because converting from the pdf gave unprintable characters, causing problems in latex. We need the original latex source for all of the math!

This section introduce notations and the necessary matterial used in this paper. B-spline Functions Let rect(t) denote the rectangular function defined as

Illustrative Examples

Simulated Data We first applied our algorithm to a large ribosomal subunit(PDB-ID=1FFK). This dataset contained 5000 projections. The image size was 143 x 143 with 2:8Å pixel size. Part of the projections were shown in figure 5.1(a). We use the algorithm with J1 term, J1+J2 and J1+J2+J3 respectively. Figure ?? show part of the slices along the z- axis of the reconstructed functions. From these pictures we can easily see that the result using J1,J2 and J3 have higher contrast and lower noise. Furthermore, we randomly split the data set in two equal parts, reconstructed two volume using the two subset respectively and computed the resolution using Fourier Shell Correla- tion(FSC) function with 0.5 cutoff. We compared our result using J1+J2+J3 with WBP and Fourier methods. It is easily seen from figure5.2 that our method can achieve higher resolution than WBP and Fourier's. A 3D volume iso-value surface was shown in Figure5.3.

Experimental Data Our second experiment use a dataset contained 7600 projections of a mutant of p53. This molecule has a C2 symmetry along the principle axis(z-axis).Figure 5.4 shows part of the 2D noisy projections.

In this experiment, we first computed three reconstructed functions using J1 ,J2 and J3 with three different regularization terms g(x) = 0 (means no regularizer), g(x) = krf(x)k and g(x) = 1 respectively. For each case, we randomly divide the data set in two equal parts, calculated its FSC and resolution with 0.5 cutoff. These results were put into table 5.1. In this

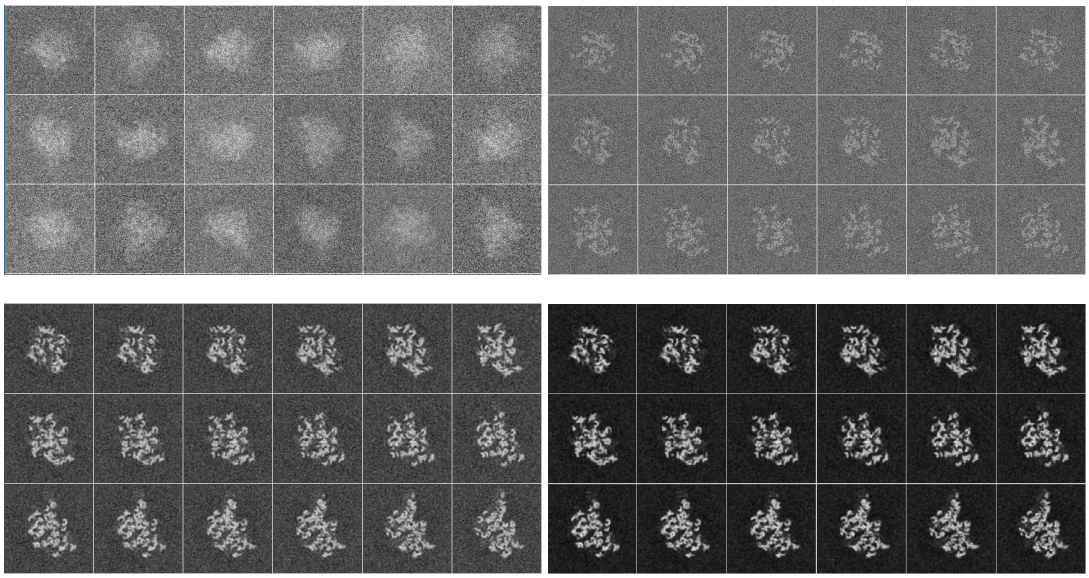


Figure 2.44: (a) 2D noisy projections of 1FFK. (b) reconstructed result using J1. (c) reconstructed result using J1 + J2. (d) reconstructed result using J1 + J2 + J3.

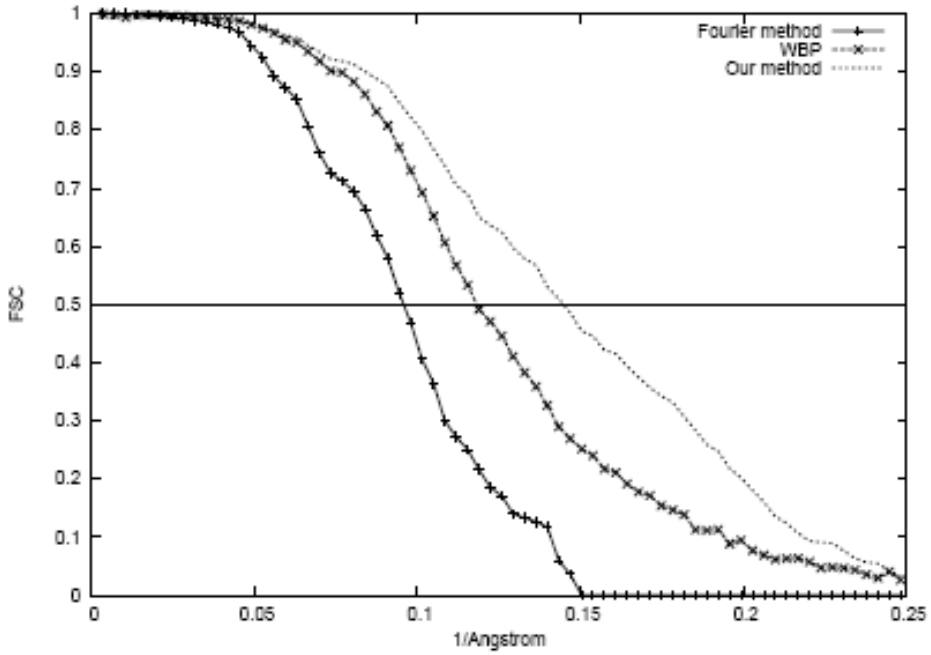


Figure 2.45: FSC curves.

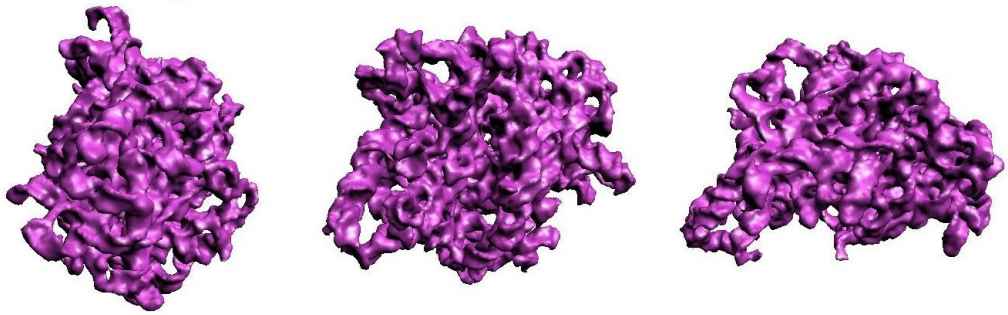


Figure 2.46: Reconstructed 1FFK Volume. From left to right: Side view; Side view; Top view

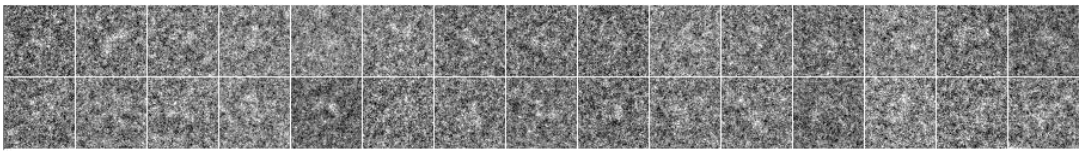


Figure 2.47: Noisy projections of p53.

table, the highest resolution was obtained by using regularization term g = krfk. Next, we reconstructed the p53 3D volume using J1 + J2 + J3 with regularization term g = krfk, WBP and Fourier method respectively with the total 7600 projections. After that, their resolutions were calculated using FSC in Fig. 5.5. From this figure, we can see that our method using J1+J2+J3 obtained the highest resolution(10.48Å). This reconstructed volume was shown in figure 5.6. In the end, we applied our algorithm to a virus named as Adenovirus with icosahedral symmetry. This data set contained 4810 noisy projections with image size 408 x 408. Part of the slices along the z-axis of the reconstructed volume and iso-surface rendering were shown in Fig.5.7 and 5.8 respectively.

Conclusions

In this paper, we proposed a new 3D single particle reconstruction algorithm using L2 gradient flow. We employed the minimal support property of B-spline and Schmidt orthogonalization process to accelerate the algorithm. In the end, simulated and experimental experiments show our algorithm using regulation term g = krfk could achieve the highest resolution compared with WBP and Fourier method.

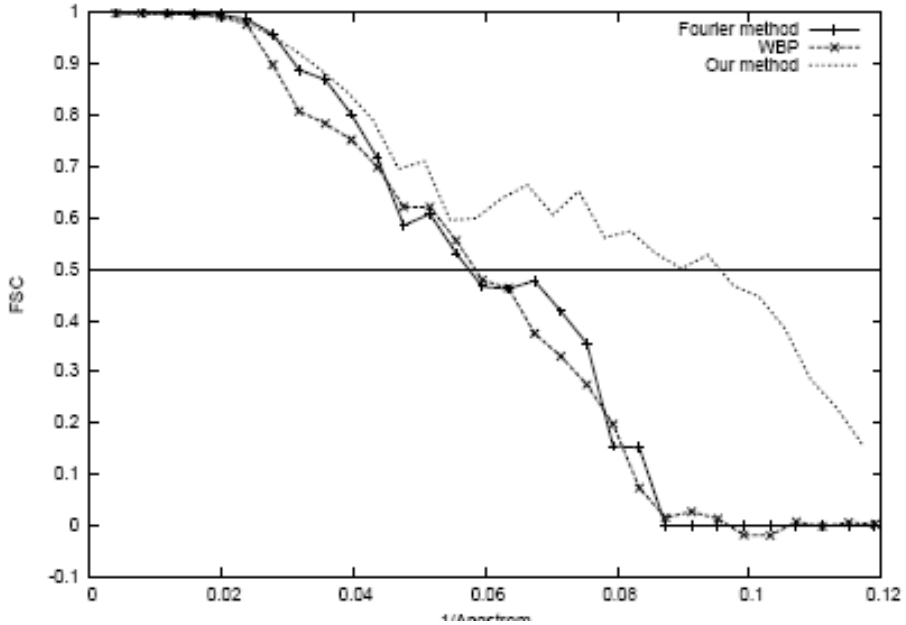


Figure 2.48: FSC curves.

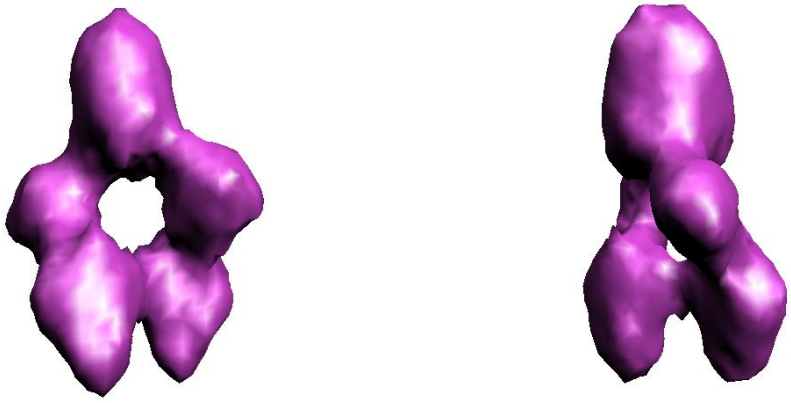


Figure 2.49: Reconstructed p53 volume.(a) (b): Views from different side.

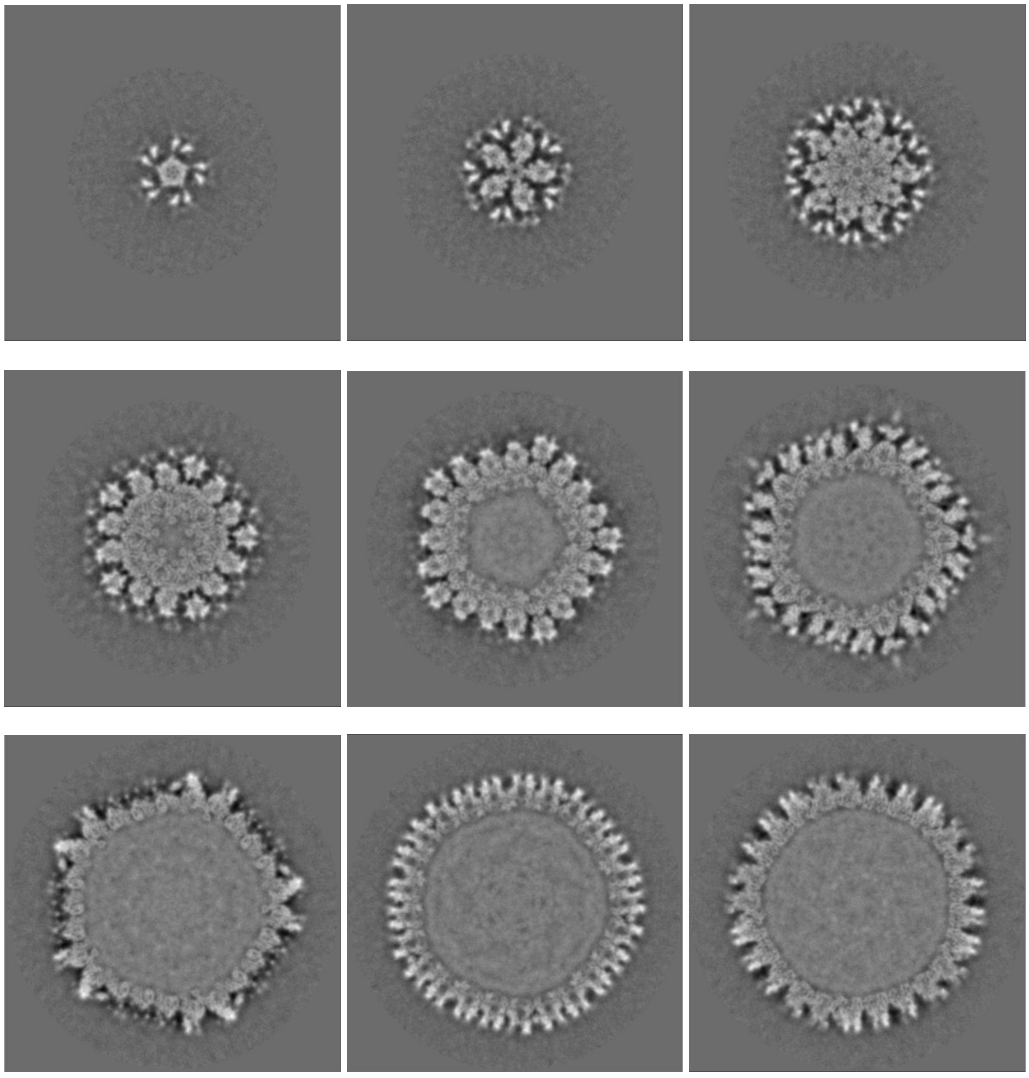


Figure 2.50: Part of the slices along the z-axis of reconstructed Adenovirus.

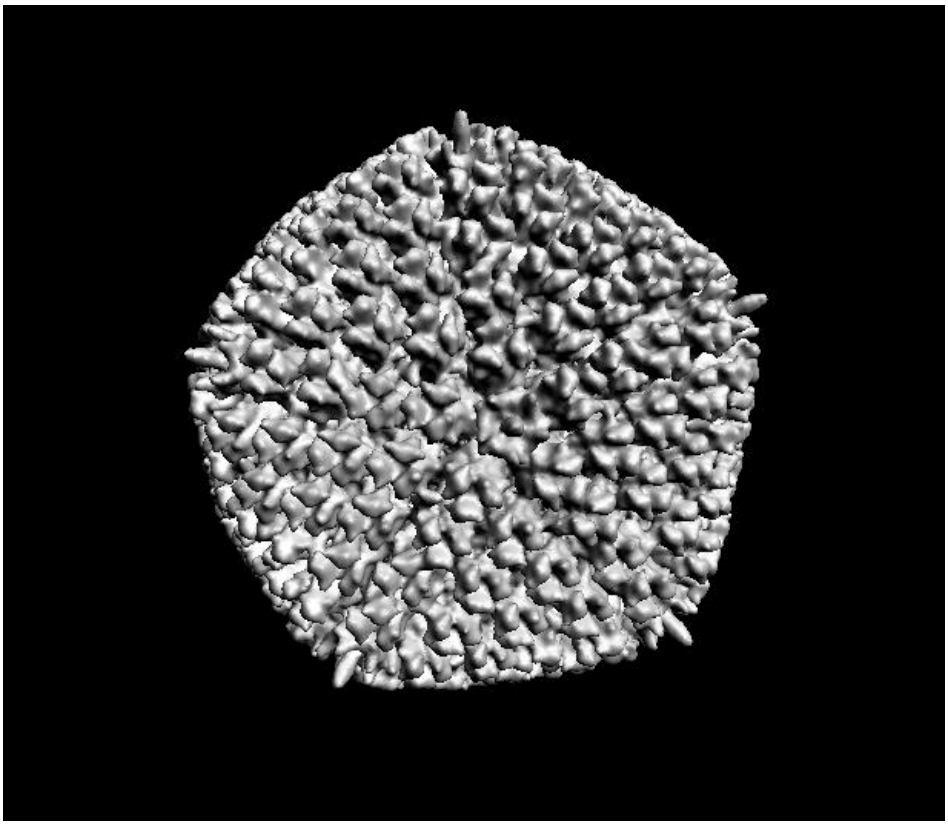


Figure 2.51: Reconstructed volume of Adenovirus.

2.4.2 Fourier Based Tomography

The following algorithm is taken from [315] (Xu2009Inversion).

A Review of Existing ET Reconstruction Techniques

Reconstruction methods try to solve the following system of equations that describe the projection process also called the *forward problem*,

$$I_\theta = \sum_{n=1}^{N} w_{\theta n} f_n \tag{4.79}$$

where I_θ is a 2D projection image at angle θ through the space of unknown voxels $f_n \in \mathbb{R}^3 : n < N$ where $N = I * J * K$ and I, J, K are dimensions of 3D image. $w_{n\theta}$ is a set of basis or weights that describe the contribution of each voxel to a particular projection. Given I_θ, the measured data and $w_{\theta n}$, the projection matrix (forward operator), the goal or the *reverse problem* is to estimate f_n. Kak et al. [174] assume each ray to have a thickness and the weight w for a voxel is the area of the voxel that is intersected by this ray. Matej et al. [211] use blobs to define the basis, they center a set of overlapping blobs as a Body Centered Cube in the $\mathbb{R}^3$ reconstruction space. $w_{n\theta}$ would be the line integral of the overlapping blobs for voxel n at angle θ.

The popular iterative approach to solve the reverse problem is to use the Kaczmarz method [174] for solving the set of forward projection equations. In this technique each tomogram image I_θ is treated like a hyperplane with the equation $w_\theta f$ in an N dimensional space. A solution to the entire equation system for all tomograms I is a point in the N dimensional space where all the hyperplanes intersect. To solve for this solution using an iterative method, the algorithm starts off with a random solution (typically zero). This initial solution is then projected onto the first hyperplane. The projected point on the first hyperplane is then projected onto the next hyperplane and so on for all the images I_θ. The iterative process will converge to the point of intersection of all the hyperplanes [174]. If there is no unique solution which is typically the case, then the iterative scheme will not converge and will oscillate. To work around this problem a regularizing term λ is used in the update equation [67].

$$f^{(\theta+1)} = f^{(\theta)} - \lambda \frac{w_{\theta+1} \cdot f^{(\theta)} - I_{\theta+1}}{||w_{\theta+1}||^2} w_{\theta+1}. \tag{4.80}$$

The general update equation for a n^{th} pixel element at i^{th} iteration is given by the correction value $\Delta f_n^{(i)}$

$$\Delta f_n^{(i)} = f_n^{(i)} - f_n^{(i-1)} = \frac{I_i - q_i}{||w_i||^2} w_{in}. \tag{4.81}$$

The $i-1$ solution is projected on the i hyperplane and corrects the value of the n^{th} pixel by Δf_j^i and $q_i = w_i f^{(i-1)}$.

Algebraic Reconstruction Technique(ART) In this technique the update of $f^{\theta+1}$ with Δf occurs at each iteration. Every projection ray will update the entire estimate of the solution for f. Some variants of this technique assume all the weights w to be 1's [174], which speeds up the iteration process however results in errors. If the weights are assumed to be 1's then $\sum_{n=1}^{N} n_{n\theta}^2 = N_\theta$ where N_θ is the number of image cells that a ray at the projection angle θ pass through. The update reduces to:

$$\Delta f_j^{\theta+1} = \frac{I_\theta - q_\theta}{N_\theta}. \tag{4.82}$$

The advantages of this technique is faster convergence since the solution is obtained within a single pass through all the projections. The disadvantage is that noisy measurements they can alter the solution by updating f with a value that will take it away from the true solution.

Simultaneous Iterative Reconstruction Technique(SIRT) In this technique, voxels are updated after completely iterating through all the projections. The update for the *j*th pixel caused by the *i*th equation is not applied immediately. Instead the update for *j*th pixel is averaged over the updates for that pixel from all the equations (projection rays), making it less prone to noise. This constitutes one iteration and is repeated over several iterations each time the update is done at the end of that iteration.

$$\Delta f_j^{iteration} = \sum_{i=1}^{N} \frac{\Delta f_j^i}{N}, \tag{4.83}$$

where Δf_j^i which is the correction for the j^{th} voxel at the i^{th} projection is calculated for all projections and the averaged value is used as the update for the j^{th} voxel. This process is performed for N iterations.

Weighted Back Projection(WBP) In this technique, the projection data is weighted in Fourier space giving more weight to higher frequencies and inverted using back-projection into the spatial domain. The weighting step is equivalent to a high pass filter since it highlights the edges and removes blurring resulting in better contrast in the reconstruction. Herman [154] describes the implementation details of the back-projection operator and Radermacher [241] provides details of the general weighting function for arbitrary tilt geometry. Let $\tilde{I}_\theta$ be the 2D Fourier transform of the 2D projection image I_θ at tilt angle θ. A weighting function is used to weight the higher frequencies of $\tilde{I}_\theta$ and this weighted Fourier transform is inverted and back-projected. $I_\theta^{weighted} = F_{2D}^{-1}(|weight|\tilde{I}_\theta)$

$$f(x,y,z) = \sum_\theta I_\theta^{weighted}(x\cos(\theta) + y\sin(\theta), z) \tag{4.84}$$

According to the Fourier slice theorem, used by Crowther [97] in a seminal paper on tomography, the 2D Fourier transform of the projection of a 3D function I onto a plane is equal to a slice through the origin of the 3D Fourier transform of that function I parallel to the projection plane. Due to the radial nature of the Fourier space, at higher frequencies the sampling is much smaller than at lower frequencies. To offset this, a weighting function is used which gives more weight to the higher frequencies than the lower frequencies. Further discussion and mathematical details on iterative and Fourier based methods is available in [221].

Classical regularization methods Typical classical regularization involves using a Total Variation (TV) or a norm based regularizer [155]. Herman and Davidi [155] use a modified ART based method incorporating either a TV or a 2-norm based regularizer.

$$||f||_{TV} = \sum_{i,j,k}^{N-2} \sqrt{(f(i+1,j,k) - f(i,j,k))^2 + (f(i,j+1,k) - f(i,j,k))^2 + (f(i,j,k+1) - f(i,j,k))^2} \tag{4.85}$$

$$||f||_2 = \sqrt{\sum_{i,j,k} f(i,j,k)^2} \tag{4.86}$$

At each iteration (block ART), the TV or 2-norm based regularizer is evaluated and the iterations are stopped when the regularizer is less than a predetermined value. Lei et al. [192] use a Tikhonov based regularizer in a Newton's method scheme to perform reconstruction. The Tikhonov regularizer is used in the first update iteration and subsequent iterations are calculated using Newton's method.

$$\begin{aligned} f_{(0)} &= (H^T H + \alpha I)^{-1} H^T g \\ f_{(k+1)} &= f_{(k)} - [\nabla^2 f_{(k)}]^{-1} \nabla f_{(k)} \end{aligned}$$

Software packages for tilt series alignment and tomographic reconstruction

Problem Setting

Let $\{g_{\mathbf{d}}\}$ be a set of the two-dimensional images measured from an unknown three-dimensional function (electric-potential) f by the projection $X_{\mathbf{d}}$ in the direction $\mathbf{d} \in \mathbb{D} \subset S^2$, here S^2 stands for the unit sphere in $\mathbb{R}^3$, $\mathbb{D}$ is the collection of all the projection directions. Our problem is to construct $f(\mathbf{x})$, $\mathbf{x} = [x,y,z]^T \in \Omega \subset \mathbb{R}^3$, such that $X_{\mathbf{d}} f$ is as close to $g_{\mathbf{d}}$ as possible and satisfies the boundary condition $f(\mathbf{x}) = 0$ for any $\mathbf{x} \in \partial\Omega$.

Name of Package	Alignment Method	Reconstruction Algorithms	Platforms
TomoJ	Fiducials and Cross correlation	SIRT, ART, WBP	Plugin to ImageJ
XMIPP	Landmark based	SIRT, ART, WBP	Linux, command line
IMOD	Cross correlation	WBP	Mac and linux
Tom Toolbox	Fiducials and Cross correlation	WBP	Matlab
EM3D	Tracking and Cross correlation	WBP	Mac and linux

Table 2.5: The above table shows some of the popular open source software packages and the algorithms they use for preprocessing and reconstruction.

We assume that all the measured images have the same size $(n+1)\times(n+1)$, the pixel values $g_{\mathbf{d}}$ of each image are defined on integer grid points $(i,j)^T \in \left[-\frac{n}{2},\frac{n}{2}\right]^2$ (we assume n is an even number). A resampling step can be taken if these assumptions are not satisfied. Since $g_{\mathbf{d}}$ are the projection of f, we therefore define Ω as

$$\Omega = \left[-\frac{n}{2}-1,\frac{n}{2}+1\right]^2 \times \left[-\frac{\tau}{2}-1,\frac{\tau}{2}+1\right],$$

where τ/n is a small number since the sample to be reconstructed is thin. $\tau+2$ is the thickness of the Ω. Again, we assume τ is an even integer. Setting one layer offset in Ω is for having room to satisfy the zero boundary condition.

For a given $\mathbf{d}\in\mathbb{D}$. The image values of $g_{\mathbf{d}}$ at the grid points are defined as

$$g_{\mathbf{d}}(i,j) = \int_{-\infty}^{\infty} f(i\mathbf{e}_{\mathbf{d}}^{(1)} + j\mathbf{e}_{\mathbf{d}}^{(2)} + t\mathbf{d})\,\mathrm{d}t, \quad (i,j)^T \in \left[-\frac{n}{2},\frac{n}{2}\right]^2$$

for the unknown function f, where $\mathbf{e}_{\mathbf{d}}^{(1)}$ and $\mathbf{e}_{\mathbf{d}}^{(2)}$ are two directions satisfying

$$\|\mathbf{e}_{\mathbf{d}}^{(1)}\| = \|\mathbf{e}_{\mathbf{d}}^{(2)}\| = 1, \quad ,\langle \mathbf{e}_{\mathbf{d}}^{(1)}, \mathbf{e}_{\mathbf{d}}^{(2)}\rangle = 0, \quad \langle \mathbf{e}_{\mathbf{d}}^{(1)}, \mathbf{d}\rangle = 0, \quad \langle \mathbf{e}_{\mathbf{d}}^{(2)}, \mathbf{d}\rangle = 0. \tag{4.1}$$

$\mathbf{e}_{\mathbf{d}}^{(1)}$ and $\mathbf{e}_{\mathbf{d}}^{(2)}$ also determine the in-plane rotation.

Given an even and positive integer $m = 2(l+2)$, let

$$h = \frac{n+2}{m}, \quad t = \frac{\tau+2}{2h} - 2.$$

For simplicity, we require h and t are integers. The following two cases are the ones we often use:

1. Take $m = n+2$, then $h = 1$, $l = \frac{n}{2}-1$, $t = \frac{\tau}{2}-1$.

2. Take $m = \frac{n+2}{2}$, then $h = 2$, $l = \frac{n+2}{4}-2$, $t = \frac{\tau+2}{4}-2$.

Using spacing h, the domain Ω can be uniformly partitioned with grid point $(i,j,k)h$ for $(i,j,k) \in [-l-2, l+2]^2 \times [-t-2, t+2]$. The function f to be reconstructed is represented as

$$f(\mathbf{x}) = \sum_{i=-l}^{l}\sum_{j=-l}^{l}\sum_{k=-t}^{t} f_{ijk} N_i(x) N_j(y) N_k(z), \quad \mathbf{x} = (x,y,z)^T \in \Omega, \tag{4.2}$$

where N_α are one dimensional cubic B-spline basis functions (see Appendix A) defined on the uniform grid $[-2h+\alpha h, -h+\alpha h, \cdots, 2h+\alpha h]$.

Reconstruction of f

Let $\Gamma_c = \{\mathbf{x} \in \mathbb{R}^3 : f(\mathbf{x}) = c\}$ be a level-set of function f. To reconstruct function f, we minimize the following energy functional.

$$J(f) = J_1(f) + \alpha J_2(f), \tag{4.1}$$

where

$$\begin{aligned} J_1(f) &= \sum_{\mathbf{d}\in\mathbb{D}} \int_{\mathbb{R}^2} (X_{\mathbf{d}} f - g_{\mathbf{d}})^2 \, \mathrm{d}u\mathrm{d}v, \\ J_2(f) &= \int_{-\infty}^{\infty} \int_{\Gamma_c} g(\mathbf{x}) \mathrm{d}A \, \mathrm{d}c, \end{aligned}$$

and $\alpha \geq 0$ is a given number balancing the two $J's$. We first explain the roles played by J_1 and J_2.

1. J_1 is used to minimize the total error of the measured images and the projections of the reconstructed function. Hence, this term is usually called the data fidelity term.

2. J_2 is a regularization term that makes the iso-surface Γ_c smooth in certain sense, depending on what kind g is used. From co-area formula (see [125]), we have

   $$J_2(f) = \int_{\mathbb{R}^3} g(\mathbf{x}) \|\nabla f(\mathbf{x})\| \mathrm{d}\mathbf{x},$$

   where ∇ is the usual gradient operator acting on a three dimensional function. In this paper, we take $g(\mathbf{x}) = 1$ or $g(\mathbf{x}) = \|\nabla f(\mathbf{x})\|$. When $g(\mathbf{x}) = 1$, J_2 represents the total surface area, and minimizing J_2 leads to the level sets of the solution close to minimal surfaces. When $g(\mathbf{x}) = \|\nabla f(\mathbf{x})\|$, minimizing J_2 leads to f approaching a constant function.

Since the energy functional $J(f)$ is nonlinear with respect to the unknown function f, we use the following iterative algorithm to minimize the energy:

Algorithm 3.1 *Reconstruction of f.*

1. Load images $g_{\mathbf{d}}$ for all $\mathbf{d} \in \mathbb{D}$.

2. Set $k = 0$, and choose an initial $f^{(0)}$.

3. Compute $f^{(k+1)}$ by minimizing $J(f)$ by solving an L^2-gradient flow.

4. Check the termination conditions, if they are satisfied, stop the iteration, otherwise set k to be $k+1$ and then go back to step 3.

5. Output $f = f^{(k+1)}$.

In principle, our initial $f^{(0)}$ can be arbitrarily chosen, including taking it to be zero. If a rough approximation of f is available, e.g. from application of other ET reconstruction methods, that can be used as an initial $f^{(0)}$. The choice of the initial $f^{(0)}$ can influence the overall running time, but cannot affect the final reconstructed f. Hence our method can also be considered as a further refinement of prior ET reconstruction schemes. The termination conditions are set as

$$|J(f^{(k+1)}) - J(f^{(k)})| < \varepsilon,$$

where ε is a chosen residual error threshold.

Obviously, the central task in the above algorithm is to minimize $J(f)$. This goal is achieved by solving an L^2-gradient flow of $J(f)$ in the B-spline function space. We construct the flow by a variation of $J_1(f)$ and $J_2(f)$. Let $\delta(J_i(f), \psi)$ be the variation of $J_i(f)$ with respect to a trial function ψ. Then it is easy to derive that

$$\delta(J_1(f), \psi) = 2 \sum_{\mathbf{d}\in\mathbb{D}} \int_{\mathbb{R}^2} (X_{\mathbf{d}} f - g_{\mathbf{d}}) X_{\mathbf{d}} \psi \mathrm{d}u\mathrm{d}v.$$

For $g(\mathbf{x}) = 1$ and $g(\mathbf{x}) = \|\nabla f(\mathbf{x})\|$, $\delta(J_2(f), \psi)$ are respectively given as follows:

$$\delta(J_2(f), \psi) = \int_{\mathbb{R}^3} \frac{(\nabla f)^T \nabla \psi}{\|\nabla f\|} d\mathbf{x}, \tag{4.2}$$

and

$$\delta(J_2(f), \psi) = 2 \int_{\mathbb{R}^3} (\nabla f)^T \nabla \psi d\mathbf{x}. \tag{4.3}$$

Using these first order variations, we construct the following weak form L^2-gradient flow.

$$\int_{\mathbb{R}^3} \frac{\partial f}{\partial t} \psi d\mathbf{x} + \delta(J_1(f), \psi) + \alpha \delta(J_2(f), \psi) = 0, \tag{4.4}$$

for all ψ in the B-spline function space. We solve (4.4) using a numerical methods. For the discretization in the temporal direction, we use the explicit forward Euler scheme. For the discretization in the spatial direction, we use the finite element method in a finite element space consisting of tri-cubic B-splines. Computational details are presented in the next section.

We choose B-splines as the finite element space for several reasons. First of all, our regularizer involve partials computation. Hence smooth functions are required. Secondly, B-spline functions have local support and a closed form Fourier transform. Finally, fast algorithms exist for transforming discrete sampled (i.e. imaging) data to B-spline representation. These properties make our finite element computation using B-splines very efficient.

Computational Details

Let $N_i(s)$ be the one-dimensional cubic B-spline basis functions defined on the 1-D uniform grid $[ih-2h, \cdots, ih+2h]$ (see Appendix for details). Then over the volume Ω, we represent $f(\mathbf{x})$ as

$$f(\mathbf{x}) = \sum_{i=-l}^{l} \sum_{j=-l}^{l} \sum_{k=-t}^{t} f_{ijk} \phi_{ijk}(\mathbf{x}), \quad \phi_{ijk}(\mathbf{x}) = N_i(x) N_j(y) N_k(z), \ \ \mathbf{x} = [x, y, z]^T, \tag{4.1}$$

with f_{ijk} as unknown coefficients. Taking the test function ψ in (4.4) as $\phi_{\alpha\beta\gamma}(\mathbf{x})$, we then obtain a matrix form $MX = B$ for (4.4). The elements of matrix M are in the form

$$\int_{\mathbb{R}^3} \phi_{ijk}(\mathbf{x}) \phi_{\alpha\beta\gamma}(\mathbf{x}) \mathrm{d}\mathbf{x} = \int_{\mathbb{R}} N_i(x) N_\alpha(x) \mathrm{d}x \int_{\mathbb{R}} N_j(y) N_\beta(y) \mathrm{d}y \int_{\mathbb{R}} N_k(z) N_\gamma(z) \mathrm{d}z.$$

The one-dimensional integrals above can be computed by Gaussian quadrature formulas (see [1, 316]). Using the fact that the B-spline basis is locally supported, these one-dimensional integrals can be efficiently calculated. Note that matrix M is fixed during the entire reconstruction process. Hence, M^{-1} could be pre-computed. The elements of the vector B are

$$-[\delta(J_1(f), \phi_{\alpha\beta\gamma}) + \delta(J_2(f), \phi_{\alpha\beta\gamma})]$$

that are represented as the integrations of f and basis functions and their derivatives. In the following, we discuss how matrix M and vector B are efficiently computed.

Computation of M Let L and T be the numbers of the utilized basis functions $N_i(x)$ (and $N_j(y)$) and $N_k(z)$, respectively. Then the total number of tensor product basis functions $\phi_{ijk}(x, y, z)$ is TL^2. Hence the number of the elements of M is T^2L^4. It is very possible that the required space for storing the matrix M is beyond the main memory capacity of the computer. To overcome this difficulty, we orthogonalize the basis functions $N_i(s)$ using the Schimidt orthogonalization process, obtaining a set of new basis functions $\tilde{N}_i(s)$, such that

$$\int_{\mathbb{R}} \tilde{N}_i(s) \tilde{N}_j(s) \mathrm{d}s = \delta_{ij}, \quad i, j = -l, -l+1, \cdots, l.$$

Let

$$\tilde{\phi}_{ijk}(\mathbf{x}) = \tilde{N}_i(x) \tilde{N}_j(y) \tilde{N}_k(z).$$

Then by representing the function f using the new basis $\tilde{\phi}_{ijk}$ and taking the test function $\psi = \tilde{\phi}_{\alpha\beta\gamma}$, we obtain a unit matrix M. Hence there is no need to store and invert the matrix.

Now we have representation (4.1) of f and the representation

$$f(\mathbf{x}) = \sum_i \sum_j \sum_k \tilde{f}_{ijk} \tilde{\phi}(\mathbf{x}), \quad \mathbf{x} = [x, y, z]^T. \tag{4.2}$$

Since there is a lower-triangular matrix A connecting the old and the new basis,

$$[N_1, N_2, \cdots, N_n]^T = A[\tilde{N}_1, \tilde{N}_2, \cdots, \tilde{N}_n]^T.$$

The two representations of f can be converted easily from one to the other. We take the advantageous of having the above dual form representations to reduce the cost of the computation of B.

Computation of $\delta(J_1, \psi)$ Consider the computation of the term $\delta(J_1(f), \psi)$ in (4.4). Using (4.1) and taking $\psi = \tilde{\phi}_{\alpha\beta\gamma}$, we have

$$\delta(J_1(f), \tilde{\phi}_{\alpha\beta\gamma}) \quad = \quad 2 \sum_{\mathbf{d} \in \mathbb{D}} \int_{\mathbb{R}^2} (X_{\mathbf{d}} f - g_{\mathbf{d}}) X_{\mathbf{d}} \tilde{\phi}_{\alpha\beta\gamma} \mathrm{d}u\mathrm{d}v. \tag{4.3}$$

We explain how (4.3) can be efficiently and accurately computed. The computation mainly includes two steps:

1. Compute $\delta(J_1(f), \phi_{\alpha\beta\gamma})$ for all α, β, γ.
2. Convert the data set $\{\delta(J_1(f), \phi_{\alpha\beta\gamma})\}$ to the data set $\{\delta(J_1(f), \tilde{\phi}_{\alpha\beta\gamma})\}$.

Since the conversion from $\{\delta(J_1(f), \phi_{\alpha\beta\gamma})\}$ to $\{\delta(J_1(f), \tilde{\phi}_{\alpha\beta\gamma})\}$ is straightforward, we focus our attention on the computation of $\delta(J_1(f), \phi_{\alpha\beta\gamma})$. We compute these scalar values using Parseval's theorem. Let $f_1(t)$ and $f_2(t)$ be two integrable functions in the L^2-sense, $F_1(\lambda)$ and $F_2(\lambda)$ the Fourier transforms of $f_1(t)$ and $f_2(t)$, respectively. Then Parseval's theorem yields

$$\int_{\mathbb{R}} f_1(t) f_2(t) \mathrm{d}t = \int_{\mathbb{R}} \bar{F}_1(\lambda) F_2(\lambda) \mathrm{d}\lambda,$$

where $\bar{F}_1$ stands for the complex conjugate of F_1. This theorem also holds for higher dimensional integrals. Let $\Phi_{\alpha\beta\gamma}$ be the three dimensional Fourier transform of $\phi_{\alpha\beta\gamma}$. Then by Parseval's theorem and the central slice theorem (see Appendix), we have

$$\begin{aligned}
\delta(J_1(f), \phi_{\alpha\beta\gamma}) &= 2 \sum_{\mathbf{d} \in \mathbb{D}} \int_{\mathbb{R}^2} (X_{\mathbf{d}}(f) - g_{\mathbf{d}}) X_{\mathbf{d}} \phi_{\alpha\beta\gamma} \mathrm{d}u\mathrm{d}v \\
&= 2 \sum_{\mathbf{d} \in \mathbb{D}} \int_{\mathbb{R}^2} \overline{\mathscr{F}_2(X_{\mathbf{d}}(f) - g_{\mathbf{d}})} \mathscr{F}_2(X_{\mathbf{d}} \phi_{\alpha\beta\gamma}) \mathrm{d}u\mathrm{d}v \\
&= 2 \sum_{\mathbf{d} \in \mathbb{D}} \int_{\mathbb{R}^2} \left(\overline{\mathscr{F}_3(f)}\Big|_{P_{\mathbf{d}}} - \overline{\mathscr{F}_2(g_{\mathbf{d}})} \right) \Phi_{\alpha\beta\gamma}\Big|_{P_{\mathbf{d}}} \mathrm{d}u\mathrm{d}v,
\end{aligned} \tag{4.4}$$

where $P_{\mathbf{d}}$ is the plane defined by $\{\mathbf{x} : \mathbf{x}^T \mathbf{d} = 0\}$. To speedup the computation of the double integral in (4.4), we utilize the decay property of $\Phi_{\alpha\beta\gamma}$. It follows from (4.1) that

$$\Phi_{\alpha\beta\gamma}(\omega_1, \omega_2, \omega_3) = h^3 e^{-ih(\alpha\omega_1 + \beta\omega_2 + \gamma\omega_3)} \prod_{i=1}^{3} \left(\frac{sin(h\omega_i/2)}{h\omega_i/2} \right)^4.$$

Hence the magnitude of $\Phi_{\alpha\beta\gamma}(\omega_1, \omega_2, \omega_3)$ is bounded by $h^3 \prod_{i=1}^{3} \left(\frac{1}{h|\omega_i|/2} \right)^4$. Since the maximal value of $\Phi_{\alpha\beta\gamma}(\omega_1, \omega_2, \omega_3)$ is h^3, we therefore, restrict the range of $[\omega_1, \omega_2, \omega_3]$ in the following region.

$$\left(\frac{1}{h|\omega_i|/2} \right)^4 < \varepsilon, \ \ i = 1, 2, 3, \quad \prod_{i=1}^{3} \left(\frac{1}{h|\omega_i|/2} \right)^4 < \varepsilon.$$

We define a restricted sub-domain Ω_ω on which $\Phi_{\alpha\beta\gamma}(\omega_1,\omega_2,\omega_3)$ are evaluated and outside this domain the function is ignored.

$$\Omega_\omega = \left\{(\omega_1,\omega_2,\omega_3)\in\mathbb{R}^3:\ |\omega_i|<2\varepsilon^{-\frac{1}{4}}h^{-1}\ \text{ and }\ |\omega_1\omega_2\omega_3|<8\varepsilon^{-\frac{1}{4}}h^{-3}\right\}.$$

It is not difficult to calculate that the volume of Ω_ω for $h=2$ is

$$V(\Omega_\omega) = 16\varepsilon^{-\frac{1}{4}}\log(\varepsilon^{-\frac{1}{4}})[1+\log(\varepsilon^{-\frac{1}{4}})]+8\varepsilon^{-\frac{1}{4}}.$$

Comparing with the volume $8\varepsilon^{-\frac{3}{4}}$ of the domain $\Omega_\varepsilon=[-\varepsilon^{-\frac{1}{4}},\varepsilon^{-\frac{1}{4}}]^3$, the ratio of the volumes of the two domains is decreasing to zero. For instance, if $\varepsilon=(2\pi)^{-4}$, the ratio is 0.289559. Hence, there is more than 70% save by restricting the computation to the domain Ω_ω. In table 2.6, we list $V(\Omega_\omega),V(\Omega_\varepsilon)$ and the ratios of the two volumes for different ε. As we know, the

Table 2.6: Volumes and Ratios of Ω_ω and Ω_ε

ε	$V(\Omega_\omega)$	$V(\Omega_\varepsilon)$	$V(\Omega_\omega)/V(\Omega_\varepsilon)$
10^0	8.0	8.0	1.000
10^{-1}	40.03	44.98	8.899e-1
10^{-2}	150.61	252.98	5.953e-1
10^{-3}	468.70	1244.62	3.294e-1
10^{-4}	1296.71	8000.0	1.620e-1
10^{-5}	3318.25	44987.30	7.376e-2

approximation order of tri-cubic B-spline is $O(h^4)=O(n^{-4})$. Hence, if we require $\varepsilon=O(n^{-4})$, then ratio $V(\Omega_\omega)/V(\Omega_\varepsilon)=O\left(\frac{\log(n)}{n}\right)^2$.

The first term of $\delta(J_1,\psi)$ Given a projection direction $\mathbf{d}$, the plane $P_\mathbf{d}$ is defined by

$$[\omega_1(\mathbf{u}),\omega_2(\mathbf{u}),\omega_3(\mathbf{u})]^T = u_1\mathbf{e}_\mathbf{d}^{(1)}+u_2\mathbf{e}_\mathbf{d}^{(2)},\quad \mathbf{u}=[u_1,u_2]^T\in\mathbb{R}^2.$$

Denote

$$\mathbf{e}_\mathbf{d}^{(1)}=[e_x^{(1)},e_y^{(1)},e_z^{(1)}]^T,\ \ \mathbf{e}_\mathbf{d}^{(2)}=[e_x^{(2)},e_y^{(2)},e_z^{(2)}]^T.$$

Without loss of generality, we may assume that the two dimensional vectors $[e_x^{(1)},e_y^{(1)}]^T$ and $[e_x^{(2)},e_y^{(2)}]^T$ are linearly independent. This assumption is surely satisfied for ET data, since there are no projections in x- or y-directions. Then

$$\mathbf{u}=E_\mathbf{d}^{-1}[\omega_1,\omega_2]^T\quad\text{with}\quad E_\mathbf{d}=\begin{bmatrix} e_x^{(1)} & e_x^{(2)}\\ e_y^{(1)} & e_y^{(2)}\end{bmatrix} \tag{4.5}$$

defines a linear transform between uv-plane and $\omega_1\omega_2$-plane, and

$$\mathrm{d}\mathbf{u}=\mathrm{d}u_1\mathrm{d}u_2=\det(E_\mathbf{d}^{-1})\mathrm{d}\omega_1\mathrm{d}\omega_2,\quad \omega_3(\omega_1,\omega_2)=[e_z^{(1)},e_z^{(2)}]E_\mathbf{d}^{-1}[\omega_1,\omega_2]^T.$$

Therefore, the first term of $\delta(J_1(f),\phi_{\alpha\beta\gamma})$, denoted by $T_{\alpha\beta\gamma}$, can be represented as

$$\begin{aligned}
T_{\alpha\beta\gamma} &= 2\int_{\mathbb{R}^2}\left[\overline{\mathscr{F}_3(f)}\Phi_{\alpha\beta\gamma}\right]\Big|_{P_\mathbf{d}}\mathrm{d}\mathbf{u}\\
&= 2\int_{\mathbb{R}^2}\overline{\mathscr{F}_3(f)}(\omega_1(\mathbf{u}),\omega_2(\mathbf{u}),\omega_3(\mathbf{u}))\Phi_{\alpha\beta\gamma}(\omega_1(\mathbf{u}),\omega_2(\mathbf{u}),\omega_3(\mathbf{u}))\mathrm{d}\mathbf{u}\\
&= 2\int_{\mathbb{R}^2}\overline{\mathscr{F}_3(f)}(\omega_1,\omega_2,\omega_3(\omega_1,\omega_2))\Phi_{\alpha\beta\gamma}(\omega_1,\omega_2,\omega_3(\omega_1,\omega_2))\det(E_\mathbf{d}^{-1})\mathrm{d}\omega_1\mathrm{d}\omega_2\\
&= 2h^3\det(E_\mathbf{d}^{-1})\int_{\mathbb{R}}e^{-ih\alpha\omega_1}\mathrm{d}\omega_1\int_{\mathbb{R}}Q(\omega_1,\omega_2)R_{\beta\gamma}(\omega_1,\omega_2)\mathrm{d}\omega_2,
\end{aligned}\tag{4.6}$$

where

$$\begin{aligned} Q(\omega_1,\omega_2) &= \overline{\mathscr{F}_3(f)}(\omega_1,\omega_2,\omega_3(\omega_1,\omega_2))\left[\prod_{k=1}^{2}\frac{sin(h\omega_k/2)}{h\omega_k/2}\right]^4\left[\frac{sin(h\omega_3(\omega_1,\omega_2)/2)}{h\omega_3(\omega_1,\omega_2)/2}\right]^4, \\ R_{\beta\gamma}(\omega_1,\omega_2) &= e^{-ih(\beta\omega_2+\gamma\omega_3(\omega_1,\omega_2))}. \end{aligned}$$

Note that $Q(\omega_1,\omega_2)$ does not depend on α,β and γ, but depends on f. To calculate the integrations in (4.6), we define a uniform grid on the $\omega_1\omega_2$-plane, denoted by

$$G_{\mathbf{d}} = \left\{[\omega_1^{(j)},\omega_2^{(k)}]^T \in \mathbb{R}^2 : [\omega_1^{(j)},\omega_2^{(k)},\omega_3(\omega_1^{(j)},\omega_2^{(k)})]^T \in \Omega_\omega\right\},$$

where $\omega_1^{(j)} = \frac{\pi j}{2^d hL}$ and $\omega_2^{(k)} = \frac{\pi k}{2^d hL}$ are one dimensional uniform knots, $L = 2l+3$ is the number of used B-spline basis in the x (and y) direction, d control the density of the grid. Larger d leads to a denser grid. We usually take $d = 0$ or $d = 1$. Here two zero B-spline coefficients are introduced in x and y directions for satisfying periodic condition. Then using the 2D version of the trapezoid quadrature rule, we obtain the following approximation

$$T_{\alpha\beta\gamma} \approx 2h^3 \det(E_{\mathbf{d}}^{-1})\left(\frac{\pi}{2^d hL}\right)^2 \sum_{j=m}^{M} e^{-ih\alpha\omega_1^{(j)}} \sum_{k=m_i}^{M_i} Q(\omega_1^{(j)},\omega_2^{(k)}) R_{\beta\gamma}(\omega_1^{(j)},\omega_2^{(k)}). \tag{4.7}$$

We explain how (4.7) is used to compute $T_{\alpha\beta\gamma}$ efficiently. First consider the computation of $Q(\omega_1^{(j)},\omega_2^{(k)})$. Using the closed form of $\overline{\mathscr{F}_3(f)}(\omega_1,\omega_2,\omega_3(\omega_1,\omega_2))$, we have

$$Q(\omega_1^{(j)},\omega_2^{(k)}) = h^3 Q_1(\omega_1^{(j)},\omega_2^{(k)}) Q_2(\omega_1^{(j)},\omega_2^{(k)}) \tag{4.8}$$

with

$$Q_1(\omega_1^{(j)},\omega_2^{(k)}) = \left[\frac{sin(h\omega_1^{(j)}/2)}{h\omega_1^{(j)}/2}\frac{sin(h\omega_2^{(k)}/2)}{h\omega_2^{(k)}/2}\frac{sin(h\omega_3(\omega_1^{(j)},\omega_2^{(k)})/2)}{h\omega_3(\omega_1^{(j)},\omega_2^{(k)})/2}\right]^8, \tag{4.9}$$

$$Q_2(\omega_1^{(j)},\omega_2^{(k)}) = \sum_{\gamma} e^{ih\gamma\omega_3(\omega_1^{(j)},\omega_2^{(k)})} Q_\gamma(\omega_1^{(j)},\omega_2^{(k)}), \tag{4.10}$$

where

$$Q_\gamma(\omega_1^{(j)},\omega_2^{(k)}) = \sum_{\alpha}\sum_{\beta} f_{\alpha\beta\gamma} e^{ih\alpha\omega_1^{(j)}} e^{ih\beta\omega_2^{(k)}}. \tag{4.11}$$

It is easy to see that $Q_\gamma(\omega_1^{(j)},\omega_2^{(k)})$ has the following periodic property:

$$Q_\gamma(\omega_1^{(j+s2^{d+1}L)},\omega_2^{(k+t2^{d+1}L)}) = Q_\gamma(\omega_1^{(j)},\omega_2^{(k)}),\ \ 0 \le j \le 2^{d+1}L-1,\ \ 0 \le k \le 2^{d+1}L-1,$$

for any integers s and t. Using these formulations, $Q(\omega_1^{(j)},\omega_2^{(k)})$ can be computed as follows.

Algorithm 4.1. *Computing* $Q(\omega_1^{(j)},\omega_2^{(k)})$, $[\omega_1^{(j)},\omega_2^{(k)}]^T \in G_{\mathbf{d}}$

1. For each γ, compute $Q_\gamma(\omega_1^{(j)},\omega_2^{(k)})$ using (4.11) by 2D discrete fast Fourier transform.
2. Compute $Q_2(\omega_1^{(j)},\omega_2^{(k)})$ using (4.10).
3. Compute $Q(\omega_1^{(j)},\omega_2^{(k)})$ using (4.8) and (4.9).

The computation in the second and third steps are straightforward. Details of the first step are as follows. Let

$$G_s = \left[0,\frac{\pi}{2^d hL},\cdots,\frac{\pi(2^{d+1}L-1)}{2^d hL}\right]\times\left[0,\frac{\pi}{2^d hL},\cdots,\frac{\pi(2^{d+1}L-1)}{2^d hL}\right],$$

and $\alpha = \alpha' - (l+1)$, $\beta = \beta' - (l+1)$. Then

$$\begin{aligned} Q_\gamma(\omega_1^{(j)}, \omega_2^{(k)}) &= \sum_{\alpha=-l-1}^{l+1} \sum_{\beta=-l-1}^{l+1} f_{\alpha\beta\gamma} e^{ih\alpha\omega_1^{(j)}} e^{ih\beta\omega_2^{(k)}} \\ &= e^{-ih(l+1)(\omega_1^{(j)}+\omega_2^{(k)})} \sum_{\alpha'=0}^{L} \sum_{\beta'=0}^{L} f_{\alpha'-l-1,\beta'-l-1,\gamma} e^{ih\alpha'\omega_1^{(j)}} e^{ih\beta'\omega_2^{(k)}} \\ &= e^{-ih(l+1)(\omega_1^{(j)}+\omega_2^{(k)})} \overline{Q'_\gamma(\omega_1^{(j)}, \omega_2^{(k)})} \end{aligned} \tag{4.12}$$

with

$$Q'_\gamma(\omega_1^{(j)}, \omega_2^{(k)}) = \sum_{\alpha'=0}^{L} e^{-ih\alpha'\omega_1^{(j)}} \sum_{\beta'=0}^{L} f_{\alpha'-l-1,\beta'-l-1,\gamma} e^{-ih\beta'\omega_2^{(k)}}.$$

First compute $Q'_\gamma(\omega_1^{(j)}, \omega_2^{(k)})$ using 2D fast Fourier transform for all $[\omega_1^{(j)}, \omega_2^{(k)}]^T \in G_s$, where the involved coefficients with indices out of the range of $[0, L]$ are set to zero. Then compute $Q_\gamma(\omega_1^{(j)}, \omega_2^{(k)})$ using (4.12) for $[\omega_1^{(j)}, \omega_2^{(k)}]^T \in G_{\mathbf{d}}$. If a $[\omega_1^{(j)}, \omega_2^{(k)}]^T \in G_{\mathbf{d}} \setminus G_s$, then $Q_\gamma(\omega_1^{(j)}, \omega_2^{(k)})$ is obtained by periodic properties.

It is easy to see that the cost of computing $Q_\gamma(\omega_1^{(j)}, \omega_2^{(k)})$ in the first step is $O(TL^2 \log(L))$. The cost of the second step is $O(TL^2)$. The cost of the last step is $O(L^2)$. Taking the projections into account, the total cost is in the order of $O(pTL^2 \log(L))$. Here p, T and L stand for the number of projection directions, the number of the utilized B-spline basis $B_k(z)$ and the number of the utilized B-spline $B_i(x)$ (same as the number of $B_j(y)$), respectively. Note that $Q_1(\omega_1, \omega_2)$ does not depend on α, β and γ. Hence, it should be computed out of the α, β and γ loops in Algorithm 4.1.

Now we consider the computation of $T_{\alpha\beta\gamma}$ using (4.7). We describe the computational steps involved via the following algorithm.

Algorithm 4.2. *Computing $T_{\alpha\beta\gamma}$*

1. For each $\mathbf{d} \in \mathbb{D}$, compute $Q(\omega_1^{(j)}, \omega_2^{(k)})$ using Algorithm 4.1.
2. For each integer $\gamma \in [-t, t]$, compute $U_\gamma(\omega_1^{(j)}, \omega_2^{(k)}) := Q(\omega_1^{(j)}, \omega_2^{(k)}) e^{-ih\gamma\omega_3(\omega_1^{(j)}, \omega_2^{(k)})}$.
3. For each integer $\beta \in [-l, l]$, compute $V_{\beta\gamma}(\omega_1^{(j)}) := \sum_k U_\gamma(\omega_1^{(j)}, \omega_2^{(k)}) e^{-ih\beta\omega_2^{(k)}}$, using fast Fourier transform.
4. For each integer $\alpha \in [-l, l]$, compute $h^3 \det(E_{\mathbf{d}}^{-1}) \left(\frac{4\pi}{n}\right)^2 \sum_j e^{-ih\alpha\omega_1^{(j)}} V_{\beta\gamma}(\omega_1^{(j)})$, using fast Fourier transform.

The usage of the fast Fourier transform in the third and the fourth step is similar to that of $Q_\gamma(\omega_1^{(j)}, \omega_2^{(k)})$ in (4.12). But here it is one dimensional. Now let us analyze the complexity of this algorithm. The cost of the first step has been analyzed, which is $O(pTL^2 \log(L))$. The cost of step 2 is $O(pTL^2)$. The costs of step 3 and step 4 are $O(pTL^2 \log(L))$. The total cost of this algorithm is $O(pTL^2 \log(L))$. Because p and T are small comparing with L, the algorithm is fast.

A direct computation of $T_{\alpha\beta\gamma}$ leads to a cost $O(pTL^4)$. Our algorithms reduce the cost to $O(pTL^2 \log(L))$.

The second term of $\delta(J_1, \psi)$ The second term in (4.4) can be computed together with the first term. At first the function $\mathscr{F}_2(g_d)$ is represented as

$$\mathscr{F}_2(g_d)(\mathbf{u}) = \mathscr{F}_2\Big(\sum_i \sum_j g_{\mathbf{d}}^{(ij)} N_i N_j\Big)(\mathbf{u}) = \sum_i \sum_j g_{\mathbf{d}}^{(ij)} B_i(u) B_j(v),$$

where $g_{\mathbf{d}}^{(ij)}$ are the coefficients of the B-spline representation of $g_{\mathbf{d}}$. A fast algorithm for converting the discrete data to the B-spline representation is adopted from [45, 314]. Let

$$S_{\alpha\beta\gamma} = \int_{\mathbb{R}^2} \overline{\mathscr{F}_2(g_{\mathbf{d}})} \Phi_{\alpha\beta\gamma}\Big|_{P_{\mathbf{d}}} \mathrm{d}\mathbf{u}.$$

Then for $S_{\alpha\beta\gamma}$, we have the same form expression as (4.6) that is for $T_{\alpha\beta\gamma}$,

$$S_{\alpha\beta\gamma} = h^3 \det(E_{\mathbf{d}}^{-1}) \int_{\mathbb{R}} e^{-ih\alpha\omega_1} \mathrm{d}\omega_1 \int_{\mathbb{R}} Q(\omega_1,\omega_2) R_{\beta\gamma}(\omega_1,\omega_2) \mathrm{d}\omega_2 \tag{4.13}$$

with the same $R_{\beta\gamma}(\omega_1,\omega_2)$, but a different $Q(\omega_1,\omega_2)$, which is

$$\begin{aligned} Q(\omega_1,\omega_2) &= \overline{\mathscr{F}_2(g_{\mathbf{d}})}(\mathbf{u}(\omega_1,\omega_2)) \left[\prod_{k=1}^{2} \frac{sin(h\omega_k/2)}{h\omega_k/2}\right]^4 \left[\frac{sin(h\omega_3(\omega_1,\omega_2)/2)}{h\omega_3(\omega_1,\omega_2)/2}\right]^4 \\ &= h^2 Q_1(\omega_1,\omega_2) Q_2(\mathbf{u}(\omega_1,\omega_2)), \end{aligned} \tag{4.14}$$

where

$$\begin{aligned} Q_1(\omega_1,\omega_2) &= \left[\prod_{k=1}^{2} \frac{sin(hu_k(\omega_1,\omega_2)/2)}{hu_k(\omega_1,\omega_2)/2} \frac{sin(h\omega_k/2)}{h\omega_k/2}\right]^4 \left[\frac{sin(h\omega_3(\omega_1,\omega_2)/2)}{h\omega_3(\omega_1,\omega_2)/2}\right]^4, \\ Q_2(u,v) &= \sum_{\alpha}\sum_{\beta} g_{\mathbf{d}}^{(\alpha\beta)} e^{ih\alpha u_1} e^{ih\beta u_2}, \end{aligned}$$

and $\mathbf{u}(\omega_1,\omega_2)$ are given by (4.5). Algorithm 4.2 could be adjusted by subtracting the values $Q(\omega_1^{(j)},\omega_2^{(k)})$ of this Q from that Q in step 1. The values $Q(\omega_1^{(j)},\omega_2^{(k)})$ for this Q is computed as follows.

Algorithm 4.3. *Computing* $Q(\omega_1^{(j)},\omega_2^{(k)})$

1. Compute $Q_2(u_1^{(j)},u_2^{(k)})$ over a uniform grid G_s in the uv-plane using 2D discrete fast Fourier transform.

2. Compute $Q_2(u_1(\omega_1^{(j)},\omega_2^{(k)}),u_2(\omega_1^{(j)},\omega_2^{(k)}))$ from $Q_2(u_1^{(j)},u_2^{(k)})$ by the bilinear interpolation. If $\left[u_1(\omega_1^{(j)},\omega_2^{(k)}),u_2(\omega_1^{(j)},\omega_2^{(k)})\right]^T$ is out of the range of G_s, we use the periodical property of $Q_2(u,v)$ to generate the requited data.

3. Computing $Q(\omega_1^{(j)},\omega_2^{(k)})$ by (4.14).

The computation of $Q_2(u_1^{(j)},u_2^{(k)})$ in the first step is similar to that of $Q_\gamma(\omega_1^{(j)},\omega_2^{(k)})$ in (4.12). Hence, the cost of this step is $O(pL^2\log(L))$, the cost of the second step is $O(pL^2)$. Therefore, the computation cost of the Q in (4.13) is much smaller than that Q in (4.7).

Obviously, the computation of $\int_{\mathbb{R}^2}(X_{\mathbf{d}}f - g_{\mathbf{d}})X_{\mathbf{d}}\phi_{\alpha\beta\gamma}\mathrm{d}u\mathrm{d}v$ is our main task. This computation can also be conducted as follows basing on the central slice theorem:

1. Convert the discrete data of f to the discrete data of F in Fourier space using 3D discrete fast Fourier transform.

2. Take the slices of F and $\Phi_{\alpha\beta\gamma}$ obtaining $\mathscr{F}_2(X_{\mathbf{d}}f)$ and $\mathscr{F}_2(X_{\mathbf{d}}\phi_{\alpha\beta\gamma})$.

3. Convert the slice datum to real space by the inverse 2D FFT, obtaining the data $X_{\mathbf{d}}f$ and $X_{\mathbf{d}}\phi_{\alpha\beta\gamma}$.

4. Compute the integral $\int_{\mathbb{R}^2}(X_{\mathbf{d}}f - g_{\mathbf{d}})X_{\mathbf{d}}\phi_{\alpha\beta\gamma}\mathrm{d}u\mathrm{d}v$ in real space.

The above computational strategy is also efficient. However, our experience shows that it is not accurate enough for our purpose. Our use of the discrete FFT is for fast summation only, and there is no error introduced except for the round-off errors of arithmetic operations.

Computation of $\delta(J_2,\psi)$ The computation of $\delta(J_2,\psi)$ is straightforward. Again the computation includes two steps:

1. Compute $\delta(J_2(f),\phi_{\alpha\beta\gamma})$ for all α,β,γ.
2. Convert $\delta(J_2(f),\phi_{\alpha\beta\gamma})$ to $\delta(J_2(f),\tilde{\phi}_{\alpha\beta\gamma})$.

It is easy to see that, for a fixed function f, $\delta(J_2(f),\psi)$ is a linear function with respect to ψ. Hence the conversion from $\delta(J_2(f),\phi_{\alpha\beta\gamma})$ to $\delta(J_2(f),\tilde{\phi}_{\alpha\beta\gamma})$ is feasible. Therefore, the main task is the computation of $\delta(J_2(f),\phi_{\alpha\beta\gamma})$. Though there are $O(TL^2)$ three dimensional integrations that need to be computed, using the local support property of the B-spline basis, the computation cost is easily reduced to $O(TL^2)$.

For a uniform partitioning of Ω, the integration is computed by evaluating and summing the integrand over the grid points, and then dividing by the volume of a voxel. Since f is a C^2 continuous function, partial derivatives of f up to the second order can also be exactly computed. Using these partial derivatives, the integrands in (4.3) or (4.2) can be easily calculated. For each grid point, there are only 3^3 B-spline basis functions involved. Hence, the cost for evaluating one point but for all the $\phi_{\alpha\beta\gamma}$ is $O(1)$. For all the grid points in Ω, the cost is $O(TL^2)$. Hence the overall computation is very efficient.

Numerical and Illustrative Results

In this section, we present several numerical and illustrative results, showing that the presented method is stable, reliable and robust.

Stability Experiments In our stability experiments, we compute two steady solutions for two initial functions $f^{(0)}$ and using three different regularizers. a. $g(\mathbf{x})=0$ (means no regularizer); b. $g(\mathbf{x})=\|\nabla f(\mathbf{x})\|$; c. $g(\mathbf{x})=1$. For each case, we compute the L^2-error of the two solutions from the two initial functions. The first initial function is taken as zero. The second one is taken as

$$f^{(0)}(\mathbf{x})=\begin{cases} 0, & \|\mathbf{x}\|>10 \text{ and } \mathbf{x}\in\Omega,\\ 1, & \|\mathbf{x}\|\le 10 \text{ and } \mathbf{x}\in\Omega.\end{cases}$$

In Table 2.7, we list the L^2-errors for each case. From the table, we can see that the L^2-errors are around 10^{-4}. Hence two different initial functions lead to almost the same solution. In this experiment, 12×12 uniformly distributed projection directions are taken. The function to be projected is defined as

$$f(x,y,z)=\begin{cases} 1, & \text{if } 25.0<\sqrt{x^2+y^2+z^2}<31.0,\\ 0, & \text{otherwise.}\end{cases} \tag{4.1}$$

Table 2.7: L^2-errors for different initial functions

g	$g=0$	$g=\|\nabla f\|$	$g=1$
L^2-error	2.365e-04	9.192e-05	1.426e-04

Reliability Experiments In the experiments, given a function f on a volume Ω, we compute projection images from a set of uniformly distributed projection directions. Next we reconstruct f using the projected data and using $g=0$. On increasing the projection directions, we obtain a sequence of reconstructed functions. L^2-error between the reconstructed functions and the exact function f are computed. Table 2.8 lists these L^2-errors. The exact function to be projected is taken as

$$f(\mathbf{x})=\sum_{i=1}^{10}\exp^{-(\|\mathbf{x}-\mathbf{x}_i\|^2-r_i^2)}, \tag{4.2}$$

where $[\mathbf{x}_i,r_i]$ are taken as $[16.0,27.0,26.0,20.0]$, $[19.2,35.0,32.0,20.0]$, $[22.4,43.0,38.0,20.0]$, $[25.6,31.0,31.0,20.0]$, $[28.8,39.0,37.0,24.0]$, $[32.0,27.0,30.0,24.0]$, $[35.2,35.0,36.0,24.0]$, $[38.4,43.0,29.0,24.0]$, $[41.6,31.0,35.0,28.0]$, $[44.8,39.0,28.0,28.0]$. From the table, we can see that the errors decrease as the projection directions increase.

Table 2.8: L^2-errors between reconstructed functions and exact function

Projection #	3×3	6×6	12×12	24×24
L^2-error	0.0127	0.0072	0.0061	0.0059

Robustness Experiments These experiments are similar to the reliability experiments. However, noise is added to each of the projected images. The aim is to see how the noises affect the reconstructed results. Table 2.9 lists the L^2-errors now in the presence of noise. Where the exact function to be projected is defined by (4.1). In order to add Gaussian noise with the given signal-to-noise ratio (SNR), we use xmipp [270] package to compute the standard deviation for each given SNR value and then use the standard deviation to add Gaussian noise to each projection. For instance, for a SNR of 0.33, a total noise with a standard deviation of 15.2614 is needed. From the table, we can see that the L^2 errors are quite small and the errors decrease as SNR increases.

Table 2.9: L^2-errors between reconstructed functions from noised data and exact function

Projection #	3×3	6×6	12×12	24×24
SNR=0.33	0.056	0.0419	0.0431	0.0247
SNR=1.0	0.026	0.0179	0.0196	0.0126
SNR=10.0	0.012	0.0116	0.0089	0.0074

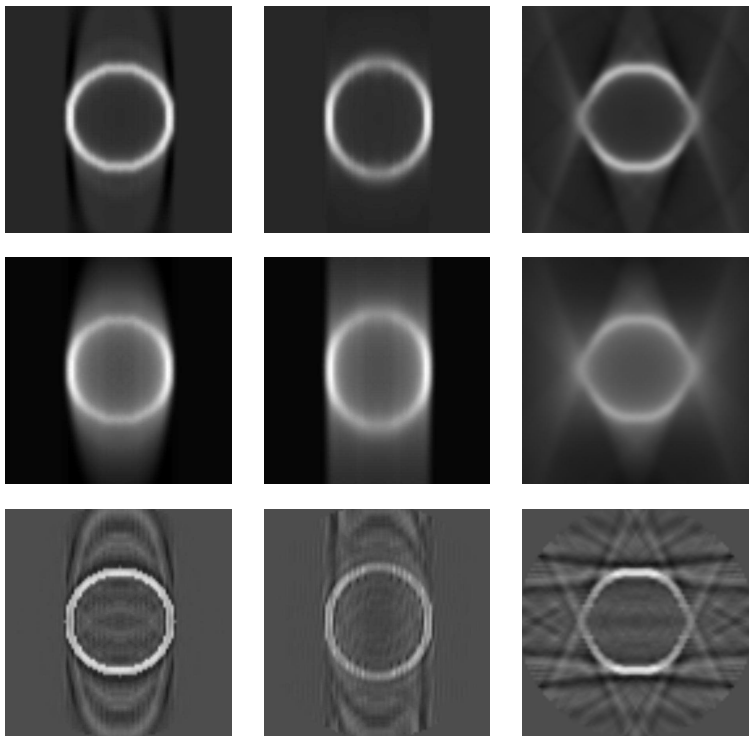
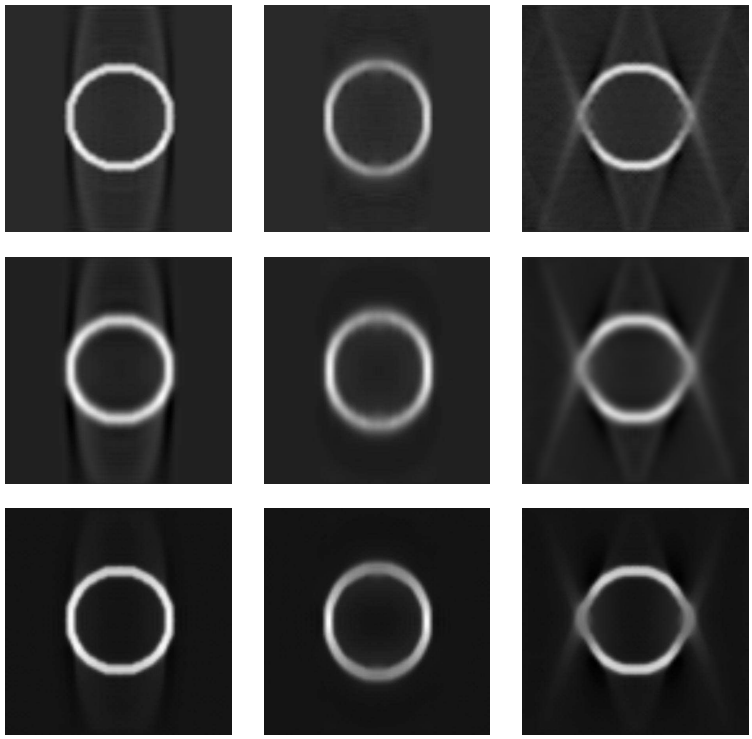


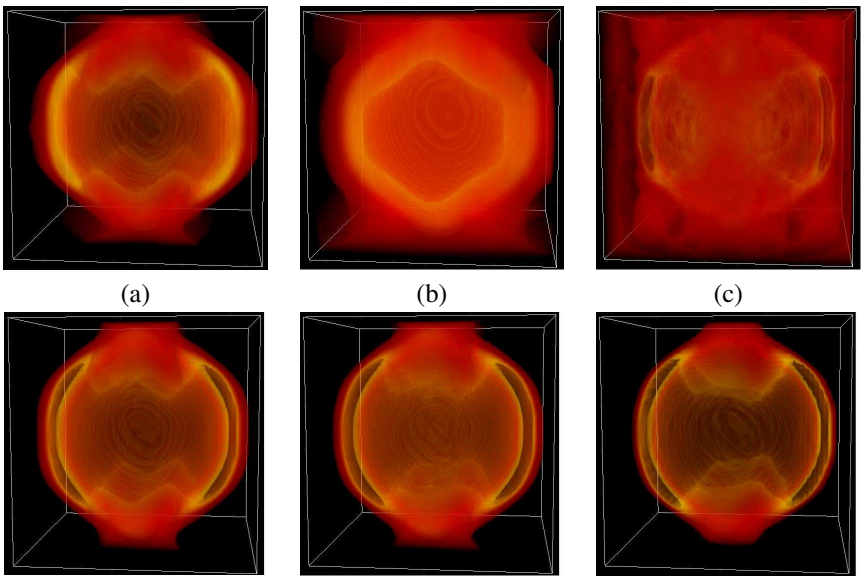
Figure 2.1: The first, second and third rows show the slices of the reconstructed functions using ART, SIRT and WBP, respectively. The first, second and third columns are the slices on XY-plane, XZ-plane and YZ-plane, respectively.

Comparative Experiments using Phantom Example To illustrate the efficiency of the proposed method, we compare the performance of our method with ART, SIRT and WBP using phantom data. The function $f(x,y,z)$ to be projected is (4.1).



(d) (e) (f)

Figure 2.2: The first, second and the third rows are extracted slices from the reconstructed functions using $g = 0$, $g = \|\nabla f\|$ and $g = 1$, respectively. The first, second and third columns are the slices on XY-plane, XZ-plane and YZ-plane, respectively.



(a) (b) (c)

(d) (e) (f)

Figure 2.3: Volume rendering: The first, second and third figures in the first row show the rendering results of the reconstructed functions using ART, SIRT and WBP. The first, second and third figures in the second row show the rendering results of the reconstructed functions using our methods with $g = 0$, $g = \|\nabla f\|$ and $g = 1$, respectively.

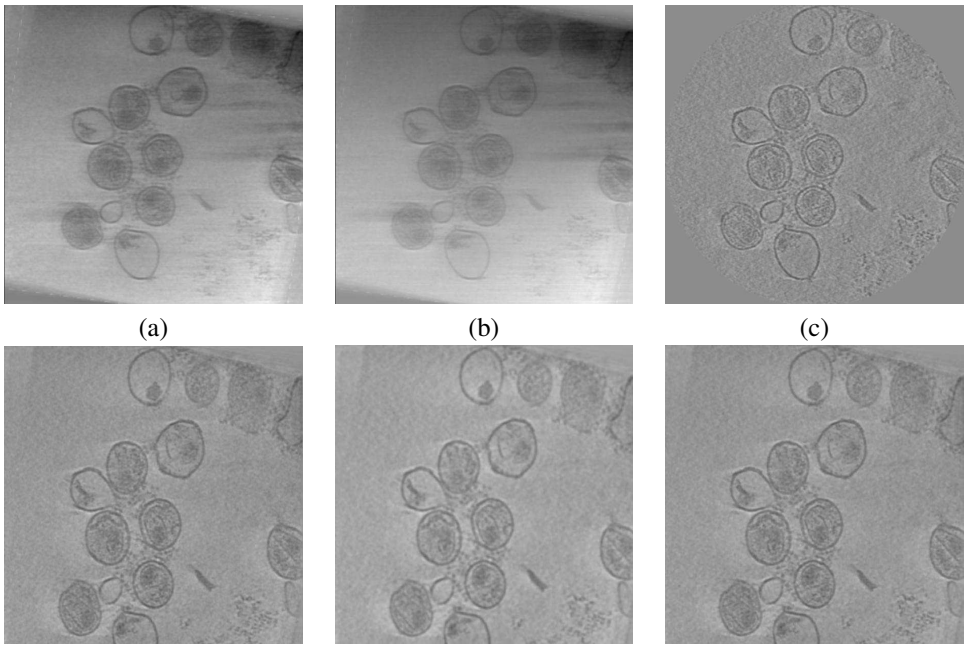


Figure 2.4: The first, second and third figures in the first row show the slices of the reconstructed functions using ART, SIRT and WBP. The first, second and third figures in the second row show the slices of the reconstructed functions using our methods with $g = 0$, $g = \|\nabla f\|$ and $g = 1$, respectively. All the slices are taken in the XY-plane.

The projected images are obtained by projecting f on 55 different directions (tilt series). The projection angles are chosen as follows: In the range $[-45°, 45°]$, 31 projections are taken with 3 degree increments. In each of the ranges $[45°, 69°]$ and $[-69°, -45°]$, 12 projections are taken with 2 degree increments. Hence, there are no projections in the ranges $(69°, 90°]$ and $[-90°, -69°)$ (missing wedge).

The three rows of the Fig. 2.1 show the slices of the reconstructed functions using ART, SIRT and WBP, where the first, second and third columns are the slices on XY-plane, XZ-plane and YZ-plane, respectively. Fig. 2.2 shows the reconstruction results of our method. The slices of the first, second and third rows are extracted from the reconstructed functions using $g = 0$, $g = \|\nabla f\|$ and $g = 1$, respectively. The first, second and third columns are the slices on XY-plane, XZ-plane and YZ-plane, respectively. The grey scale in these slices is taken as black to white for density value from zero to one. From these figures we can see that our results are better in general than the results from ART, SIRT and WBP. The best results of our method is for $g = 1$. Fig. 2.3 shows the results of volume rendering.

Application to the AIDS virus Tilt Series The input is a set of 2D projection slices at different tilt angles taken from an electron microscope of AIDS virus interacting with a neutralizing molecule D1D2-IgP [53]. Each 2D image is of size $512 * 512$ and are imaged with a defocus of $-8\mu m$. Preprocessing this tilt series includes alignment and image enhancement. The 2D projections are reconstructed into 3D volumes (also called Maps) using an L^2-gradient flow. In our data the number of projections images are 50 in tilt angular increments from $-60°$ to $+60°$, with each projection image of size 512×512. The tilt angle steps are $2° \sim 3°$.

Fig. 2.4 shows the slices of the constructed volumetric function, where the the first, second and third figures in the first row show the slices of the reconstructed functions using the methods of ART, SIRT and WBP, respectively. The first, second and third figures in the second row show the slices of the reconstructed functions using our methods with $g = 0$, $g = \|\nabla f\|$ and $g = 1$, respectively. All the slices are taken in the XY-plane. A 3D volume rendering of the reconstructed AIDS virus is shown in the figure 2.5. Reconstruction results with different kinds of regularizers and without regularizer are shown.

We perform a comparison of local contrast (LC) and signal-to-noise ratio (SNR) of the reconstructed volumes from our method with ART, SIRT and WBP. To quantify the differences of the reconstructions and compare the SNR from various reconstruction techniques, a sub-volume of the volume is extracted containing the same feature (virus spike) in all three reconstruction

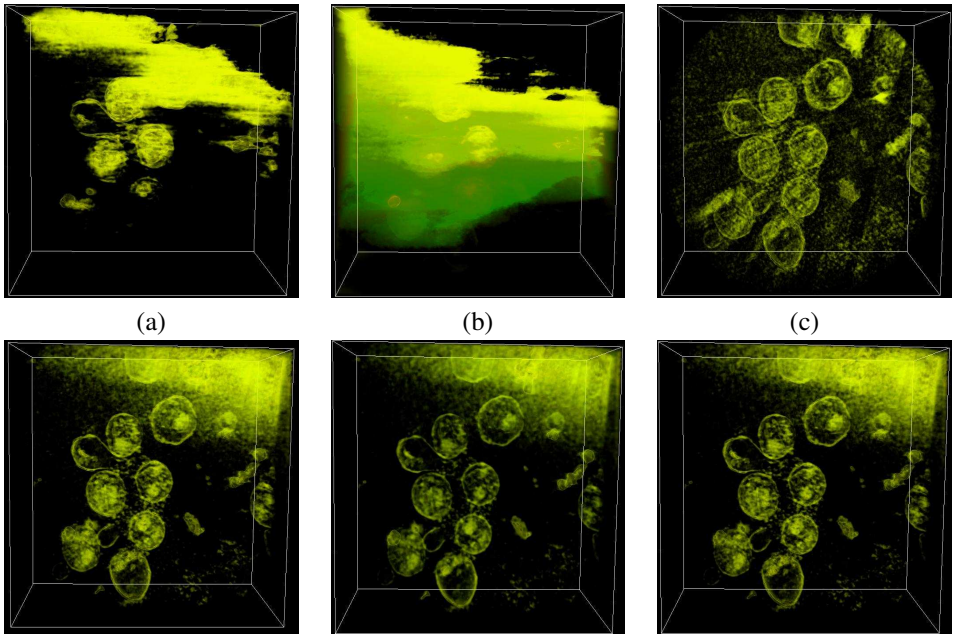


Figure 2.5: Volume rendering: The first, second and third figures in the first row show the rendering results of the reconstructed functions using ART, SIRT and WBP. The first, second and third figures in the second row show the rendering results of the reconstructed functions using our methods with $g = 0$, $g = 1$ and $g = \|\nabla f\|$, respectively.

schemes.

SNR for an image is given by:

$$SNR = \frac{\mu_{foreground} - \mu_{background}}{\sigma_{background}} \tag{4.3}$$

where $\mu_{foreground}$ is the average foreground intensity value in a window placed on the virus spike, $\mu_{background}$ is the average background intensity and $\sigma_{background}$ is the standard deviation of the background measured in a window placed in the background of the image as shown in figure 2.6.

To compare the local contrast between reconstruction techniques, the ratio of the difference between average foreground ($\mu_{foreground}$) and average background ($\mu_{background}$) to the average background ($\mu_{background}$) intensity is calculated in the window shown in figure 2.6.

$$LC = \frac{\mu_{foreground} - \mu_{background}}{\mu_{background}} \tag{4.4}$$

Reconstruction Method	SNR	LC
ART	12.512	0.0894
SIRT	24.11096	0.05932
WBP	7.8138	0.267
$g = 1$	9.144	0.3534
$g = \|\nabla f\|$	10.049	0.4275
$g = 0$	8.8968	0.3514

Table 2.10: The table shows the results of SNR and LC analysis on the reconstructed Maps.

For LC:

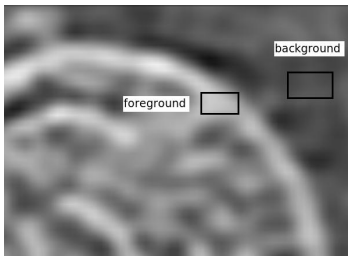


Figure 2.6: A window placed on the foreground (virus envelope) is used to calculate the average foreground intensity. A window placed on the background is used to calculate the average background intensity and the standard deviation of the background

$$(g = \|\nabla f\|) > (g = 1) > (g = 0) > WBP > ART > SIRT$$

For SNR:

$$SIRT > ART > (g = \|\nabla f\|) > (g = 1) > (g = 0) > WBP$$

LC of SIRT reconstruction is very low, but the signal to noise ratio is very good. This means that there is low contrast but the overall image has less noise. High LC ensures that segmentation and structure identification will be good and features are easy to identify from background. High SNR means that background region is fairly uniform and low in intensity and hence segmentation and structure identification will be cleaner. Ideally we would like both LC and SNR to be high. The local contrast (LC) of the reconstructed volumes from our method is superior compared to WBP, ART and SIRT.

Miscellaneous Important Facts

B-spline function. Let $\text{rect}(t)$ denote the rectangular function defined as

$$\text{rect}(t) = \begin{cases} 0 & \text{if } |t| > \frac{1}{2}, \\ \frac{1}{2} & \text{if } |t| = \frac{1}{2}, \\ 1 & \text{if } |t| < \frac{1}{2}. \end{cases}$$

Then the degree n B-spline basis $\beta_0^n(x)$ with support $[-\frac{n+1}{2}, \frac{n+1}{2}]$ is defined recursively by

$$\beta_0^n(x) = \beta_0^{n-1}(x) * \text{rect}(x),$$

where $*$ denotes the convolution of two functions, and $\beta_0^0(x) = \text{rect}(x)$. Other basis functions associate with the integer k is defined by the shifting of $\beta_0^n(x)$, i.e.,

$$\beta_k^n(x) = \beta_0^n(x-k), \;\; k = 0, \pm 1, \pm 2, \cdots.$$

By scaling the independent variable x of $\beta_k^n(x)$, we obtain the B-spline basis on equal-spaced knots

$$\cdots, \;\; -3h, \;\; -2h, \;\; -h, \;\; 0, \;\; h, \;\; 2h, \;\; 3h, \;\; \cdots$$

as

$$N_k^n(x) := \beta_k^n(h^{-1}x) = \beta_0^n(h^{-1}x - k),$$

where $a > 0$. The support of $N_k^n(x)$ is $[h(k - \frac{n+1}{2}), h(k + \frac{n+1}{2})]$. For cubic B-spline used in this paper, we use notation $N_k(x)$ to represent $N_k^n(x)$.

Fourier transform of B-spline function. Let $\mathscr{F}$ denote the one dimensional Fourier transform defined as

$$F(\omega) = (\mathscr{F} f)(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt.$$

Then it is easy to see that

$$\mathscr{F}(f(at))(\omega) = \frac{1}{|a|}F(\frac{\omega}{a}),\ \ \mathscr{F}(f(t+a))(\omega) = e^{i\omega a}F(\omega).$$

Then by

$$\mathscr{F}\beta_0^0 = \frac{sin(\omega/2)}{\omega/2},$$

we have

$$B_k^n(\omega) = (\mathscr{F}N_k^n)(\omega) = he^{-ik\omega h}\left(\frac{sin(h\omega/2)}{h\omega/2}\right)^{n+1},\ \ k = 0,\pm 1,\pm 2,\cdots. \tag{4.1}$$

Note that $B_k^n(\omega)$ is not locally supported, but it goes to zero in the rate $\frac{1}{\omega^{n+1}}$ as $|\omega| \to \infty$. Let $\mathscr{F}_k$ stand for the k-dimensional Fourier transform, i.e.,

$$F(\mathbf{X}) = (\mathscr{F}_k f)(\mathbf{X}) = \int_{\mathbb{R}^k} f(\mathbf{x})\exp\left(-i\mathbf{x}^T\mathbf{X}\right)\mathrm{d}\mathbf{x},\ \ \mathbf{x},\mathbf{X} \in \mathbb{R}^k.$$

Let $N_i(x)$, $N_j(y)$ and $N_k(z)$ be the B-spline basis in the x, y and z directions, respectively, and $\phi_{ijk}(\mathbf{x}) = N_i(x)N_j(y)N_k(z)$ their tensor product. Then

$$\begin{aligned}
\Phi_{ijk}(\mathbf{X}) &= \mathscr{F}_3(\phi_{ijk})(\mathbf{X}) \\
&= \int_{\mathbb{R}^3} N_i(x)N_j(y)N_k(z)\exp\left(-i(xX+yY+zZ)\right)\mathrm{d}x\mathrm{d}y\mathrm{d}z \\
&= \int_{\mathbb{R}} N_i(x)\exp\left(-ixX\right)\mathrm{d}x\int_{\mathbb{R}} N_j(y)\exp\left(-iyY\right)\mathrm{d}y\int_{\mathbb{R}} N_k(z)\exp\left(-izZ\right)\mathrm{d}z \\
&= \mathscr{F}(N_i)\mathscr{F}(N_j)\mathscr{F}(N_k) \\
&= B_i^{(1)}(X)B_j^{(2)}(Y)B_k^{(3)}(Z).
\end{aligned}$$

Fourier transform and discrete Fourier transform. Let $f(t)$ be a continuous function defined on the interval $[a,b]$ with the periodical condition $f(a) = f(b)$. Let the Fourier transform of f be defined as

$$F(\omega) = (\mathscr{F}f)(\omega) = \frac{1}{\sqrt{2\pi}}\int_a^b f(t)e^{-i\omega t}dt.$$

Let $t = a+(b-a)\theta/2\pi$, then we have

$$\begin{aligned}
F(\omega) &= \frac{b-a}{2\pi\sqrt{2\pi}}\int_0^{2\pi} f(a+(b-a)\theta/2\pi)e^{-i\omega(a+(b-a)\theta/2\pi)}d\theta \\
&= \frac{b-a}{2\pi\sqrt{2\pi}}e^{-i\omega a}\int_0^{2\pi} g(\theta)e^{-i\omega((b-a)\theta/2\pi)}d\theta,
\end{aligned}$$

where

$$g(\theta) = f(a+(b-a)\theta/2\pi).$$

Now let us sample $F(\omega)$ uniformly at

$$\omega_k = \frac{2k\pi}{b-a},\ \ k = -K,\cdots,-1,0,1,\cdots,K,$$

where K is a given integer. If $a = -\frac{n}{2}, b = \frac{n}{2}, W = 2\pi$, we choose

$$K = \frac{nW}{2\pi} = n.$$

We have

$$\begin{aligned} F(\omega_k) &= \frac{b-a}{2\pi\sqrt{2\pi}} e^{-i\omega_k a} \int_0^{2\pi} g(\theta) e^{-ik\theta} d\theta \\ &\approx \frac{b-a}{n\sqrt{2\pi}} e^{-i\omega_k a} \sum_{j=0}^{n-1} g\left(\frac{2j\pi}{n}\right) e^{-\frac{2ijk\pi}{n}} \\ &= \frac{b-a}{n\sqrt{2\pi}} e^{-i\omega_k a} G_k(g) \\ &= \frac{2b}{n\sqrt{2\pi}} (-1)^k G_k(g), \end{aligned}$$

where the assumptions $b > 0$ and $a = -b$ are imposed, and

$$G_k(g) = \sum_{j=0}^{n-1} g\left(\frac{2j\pi}{n}\right) e^{-\frac{2ijk\pi}{n}}, \quad k = -n, \cdots, -1, 0, 1, \cdots, n.$$

$G_0(g), \cdots, G_{n-1}(g)$ are discrete Fourier transform of g, which can be computed by discrete fast Fourier transform, $g_n = g_0$. For $k = -n, -n+1, \cdots, -1$, let $\gamma = k+n$, then it is easy to see that

$$G_k(g) = G_\gamma(g).$$

Let $g = g_r + ig_i$, where g_r and g_i are the real and imaginary parts of g. Then

$$\begin{aligned} G_k(g) &= G_k(g_r) + iG_k(g_i), \ k = 0, \cdots, n, \\ G_k(g) &= G_{-k}(g_r) + iG_{-k}(g_i), \ k = -n, \cdots, 0, \end{aligned}$$

$G_{-K}, \cdots, G_1$ are the conjugate of $G_K, \cdots, G_1$, respectively.

For two-dimensional Fourier transform

$$F(\omega, \varpi) = \frac{1}{2\pi} \int_a^b \int_c^d f(s,t) e^{-i(\omega s + \varpi t)} \mathrm{d}s\mathrm{d}t,$$

we similarly have

$$F(\omega_{k_1}, \varpi_{k_2}) \approx \frac{(b-a)(d-c)}{2\pi n_1 n_2} e^{-i(\omega_{k_1} a + \varpi_{k_2} c)} G_{k_1 k_2},$$

where

$$G_{k_1 k_2} = \sum_{j=0}^{n_1-1} \sum_{k=0}^{n_2-1} f\left(a + \frac{j(b-a)}{n_1}, c + \frac{k(d-c)}{n_2}\right) e^{-2\pi i\left(\frac{jk_1}{n_1} + \frac{kk_2}{n_2}\right)},$$

$$\omega_{k_1} = \frac{2k_1\pi}{b-a}, \ \varpi_{k_2} = \frac{2k_2\pi}{d-c}, \quad k_1, k_2 = -n, \cdots, -1, 0, 1, \cdots, n-1.$$

For $k_1, k_2 \geq 0$, $G_{k_1 k_2}$ are computed by discrete fast Fourier transform. For $k_1 < 0$ or $k_2 < 0$,

$$G_{k_1 k_2} = G_{\gamma_1 \gamma_2}, \ \text{with} \ \gamma_1 = k_1 (mod\ n_1), \ \gamma_2 = k_2 (mod\ n_2) \tag{4.2}$$

If we assume $a = -b, \ c = -d$, then

$$F(\omega_{k_1}, \varpi_{k_2}) \approx \frac{2bd}{\pi n_1 n_2} (-1)^{k_1 + k_2} G_{k_1 k_2},$$

Central Slice Theorem of Fourier Transform. Let $f(x,y,z)$ be a function defined in $\mathbb{R}^3$,

$$(X_z f)(x,y) := \int_{\mathbb{R}} f(x,y,z) \mathrm{d}z,$$

where X_z denotes the projection of f in the z-direction. Let $F(X,Y,Z) = (\mathscr{F}_3 f)(X,Y,Z)$. Then

$$\begin{aligned}(\mathscr{F}_2(X_z f))(X,Y) &= \int_{\mathbb{R}^2} (X_z f)(x,y)\exp(-i(xX+yY)\,\mathrm{d}x\mathrm{d}y \\ &= \int_{\mathbb{R}^3} f(x,y,z)\exp(-i(xX+yY)\,\mathrm{d}x\mathrm{d}y\mathrm{d}z \\ &= \int_{\mathbb{R}^3} f(x,y,z)\exp(-i(xX+yY+zZ)\,\mathrm{d}x\mathrm{d}y\mathrm{d}z\Big|_{Z=0} \\ &= (\mathscr{F}_3 f)(X,Y,Z)|_{Z=0} \\ &= F(X,Y,Z)|_{Z=0}.\end{aligned}$$

This equality says that the 2D Fourier transform of the projection in the z-direction of f is the same as the slice of the 3D Fourier transform of f at $Z=0$. This fact, named as central slice theorem, implies that if we know all the projections, then we know all the slices, and therefore, we know $F(X,Y,Z)$. By applying the inverse of the 3D Fourier transform to F, we obtain $f(x,y,z)$.

The Existence and Uniqueness of Solution

In this section, we consider the existence and uniqueness problem of the minimizing problem of (4.1). Let $\mathscr{X}$ be the function space consists of B-spline functions on the domain Ω satisfying the zero boundary conditions:

$$\frac{\partial^{i+j+k} f(x,y,z)}{\partial^i x \partial^j y \partial^k z} = 0, \ \ \forall \mathbf{x} = [x,y,z]^T \in \Gamma_\Omega, \ \ i+j+k \le 2.$$

Under the inner product

$$\langle f, g\rangle = \int_\Omega f(\mathbf{x})g(\mathbf{x})\mathrm{d}\mathbf{x},$$

$\mathscr{X}$ is a finite dimensional Hilbert space. Therefore $\mathscr{X}$ is reflective.

Since the projection directions are limited, the energy functional

$$J_1(f) = \sum_{\mathbf{d}\in\mathbb{D}} \int_{\mathbb{R}^2} (X_{\mathbf{d}} f - g_{\mathbf{d}})^2 \,\mathrm{d}u\mathrm{d}v,$$

may not be coercive. To illustrate this, let $\mathbf{d}$ be a direction in the x-direction. Then for any coefficient f_{ijk} satisfying

$$\sum_i f_{ijk} = 0, \ \ \forall j,k,$$

then we have $X_{\mathbf{d}} f = 0$. Hence, $\|f\| \to \infty$, does not imply $\int_{\mathbb{R}^2} (X_{\mathbf{d}} f - g_{\mathbf{d}})^2 \,\mathrm{d}u\mathrm{d}v \to \infty$. However, if

$$J_2(f) = \int_{\mathbb{R}^3} g(\mathbf{x})\|\nabla f(\mathbf{x})\|\mathrm{d}\mathbf{x},$$

is coercive, then $J(f) = J_1(f) + \alpha J_2(f)$ is coercive for $\alpha > 0$.

Now let us prove that $J_2(f)$ is coercive. Let $f \in \mathscr{X}$. Then if $f(\mathbf{x}) = c$ for all $\mathbf{x} \in \Omega$, then $c = 0$. This implies that if $f(\mathbf{x}) \neq 0$, then $\nabla f(\mathbf{x}) \equiv 0$ does not hold. This further implies that

$$\int_\Omega \|\nabla f(\mathbf{x})\|^2 \mathrm{d}\mathbf{x} > 0. \tag{4.1}$$

Suppose $f(\mathbf{x}) = \sum_{i=1}^N c_i \varphi_i(\mathbf{x})$, where $\{\varphi_1, \cdots, \varphi_N\}$ is a set of basis of $\mathscr{X}$. Then we have from (4.1) that the matrix

$$\mathbf{M} = \left[\int_\Omega (\nabla\varphi_i(\mathbf{x}))^T \nabla\varphi_j(\mathbf{x})\mathrm{d}\mathbf{x}\right]_{ij=1}^N$$

is nonsingular. Consider the case where $g(\mathbf{x}) = \|\nabla f(\mathbf{x})\|$. Then

$$J_2(f) = \int_\Omega \|\nabla f(\mathbf{x})\|^2 d\mathbf{x} = \mathbf{C}^T \mathbf{M} \mathbf{C},$$

where $\mathbf{C} = [c_1, \cdots, c_N]^T$. Since $\mathbf{M}$ is nonsingular, $J_2(f) \to \infty$ in the rate $O(\|\mathbf{C}\|^2)$ as $\|\mathbf{C}\| \to \infty$. Therefore, $J_2(f)$ is coercive. Now let us consider the case $g(\mathbf{x}) = 1$. Since

$$\int_\Omega \|\nabla f(\mathbf{x})\|^2 d\mathbf{x} \leq \|\nabla f\|_\infty \int_\Omega \|\nabla f(\mathbf{x})\| d\mathbf{x},$$

we have

$$J_2(f) = \int_\Omega \|\nabla f(\mathbf{x})\| d\mathbf{x} \geq \frac{1}{\|\nabla f\|_\infty} \int_\Omega \|\nabla f(\mathbf{x})\|^2 d\mathbf{x}.$$

Noticing that $\|\nabla f\|_\infty = O(\|\mathbf{C}\|)$ as $\|\mathbf{C}\| \to \infty$, we have that $J_2(f)$ is coercive. Since $\mathscr{X}$ is compact, the discussion above implies the existence of the minimization problem (4.1) for $g = 1$ and $g = \|\nabla f\|$.

On the uniqueness of the solution, it is quite easy to prove that if $g = \|\nabla f\|$, the uniqueness valid, since the the minimization problem (4.1) is quadratic, and the Euler-Lagrange equation is a nonsingular linear system.

2.5 Bio-Imaging

2.5.1 molecules — electron microscopy

2.5.2 cells with organelles —- serial section electron microscopy

2.5.3 tissues cells + extra-cellular space/filling —- confocal microscopy/micro-CT

2.5.4 organs — CT/MRI

The following algorithm is taken from [22] (bg2008).

Introduction

The importance of modern imaging techniques for capturing detailed structural information of a biological system cannot be understated. Unfortunately images do not reveal the "full functional story" and a spatially realistic computer model is often necessary for a comprehensive understanding of the complicated structural and physiological properties of the biological system's entities under investigation [311]. Deeper insights into structure-to-function relationships of different entities is achieved via finite element simulations of the modeled biomedical process. A 3D (three dimensional) finite element meshed computer model of the biological system is therefore a first step to perform such simulations.

The behavioral attributes of a biological entity or the physiological interaction between different participating components of a biological system are often modeled mathematically via a coupled set of differential and integral equations, and quite often numerically evaluated using finite element (or boundary element) simulations. To further emphasize the premise of cardiac modeling from imaging data, we state a few computational biomedical modeling and simulation examples: 3D computational modeling of the human heart for a quantitative analysis of cyclical electrical conductance on the heart membrane [203, 204, 158, 157, 312]; the biomechanical properties (stress-strain, elasticity) of the heart ventricular walls[302, 246, 248, 95, 160, 250]; 3D modeling and simulation of pulsatile blood flow through human arteries/veins for vascular by-pass surgery pre-planning on a patient specific basis [287, 286, 288, 251, 252, 319]. A finite element decomposition of the geometric domain, capturing the detailed spatial features that can be gleaned from the imaging, is therefore the essential first step toward performing the necessary numerical simulations [146, 63, 65, 100].

Modeling of human vasculature from three-dimensional (3D) Computed Tomography (CT) images of the thorax is a critical step for computer-aided diagnosis (CAD) in disease domains such as lung nodules [134], coronary artery disease [164], and pulmonary embolism (PE) [322]. Even though there are many published approaches, the problem is still unsolved. Survey of various techniques on this topic can be found in [227, 228]. Thoracic CT angiography (CTA) imaging is often performed for patients suspected of having a PE that is defined as a thrombus (or a clot of blood) [258]. Therefore, in order to detect pulmonary embolism, a suitable model of vasculature distnguishing between arterial and venous blood vessel trees is a crucial step.

In subsequent subsections we highlight the computational pipeline, the main algorithmic components and a few descriptive results of our Cardio Vascular Modeling from Imaging software.

Data Processing

The Imaging-to-Modeling software system for cardiovascular data employs both Image Processing and Geometry processing functionalities to produce a suitable linear or higher order meshed model of the anatomy. Figure 2.2 describes the data-flow layout. We describe the major algorithmic components of each of the processing modules in Sections 2.5.4 and 2.5.4. The reader must note that, the modules are selectively used depending on the nature and quality of the input imaging data.

Image Processing

[A] Contrast Enhancement:

The three dimensional intensity data often possesses a low contrast between structural features and the background, thereby making further processing all the more difficult. Image contrast enhancement is a process used to improve the

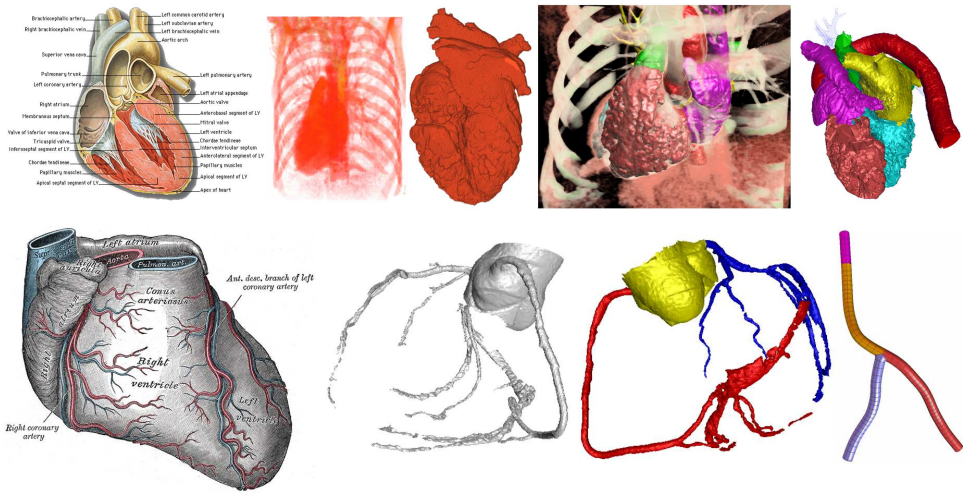


Figure 2.1: Cardiovascular Modeling from Imaging Data. Top row illustrates the modeling of patient heart from imaging data. From left to right: An illustration of internal structure of human heart (courtesy [289]), a typical Computed Tomography (CT) imaging data and the extracted geometric model of a patient heart and the annotated segmentation of the same into biologically meaningful components, namely Aorta, Pulmonary Artery, Left and Right Ventricle and Atrium (colored differently). Bottom row illustrates the modeling of coronary artery. From left to right: An illustration of annotated coronary arterial tree (courtesy http://en.wikipedia.org/wiki/Coronary_circulation), a geometric model extracted from imaging data, segmentation into left and right subtree (colored red and blue), and the NURBS model of a small portion that was used for Isogeometric analysis of blood flow in [340].

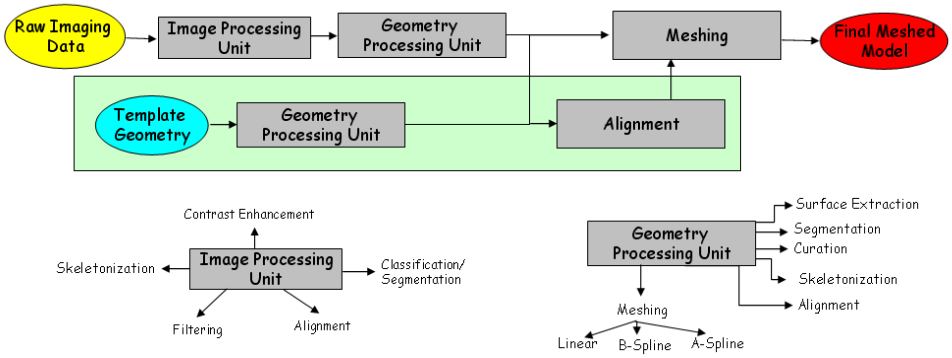


Figure 2.2: Data Flow of Cardio-Vascular Modeling from 3D Imaging Data: There are two major data processing algorithmic modules - Image Processing and Geometry Processing. The Image Processing module consists of sub-modules for *Contrast Enhancement*, *Classification/Segmentation*, *Filtering*, *Skeletonization* and *Alignment*. The Geometry Processing module consists of *Surface Extraction*, *Curation*, *Segmentation*, *Skeletonization*, *Alignment* and *Meshing*, which itself is further subdivided into linear and higher order boundary and finite element mesh generation components. The 3D CT Imaging data is first passed through the Image Processing module for improvement of image quality which is then processed by the Geometry Processing module for extraction of a clean geometry annotated with the present features. Finally the clean geometry is converted to a linear or higher-order mesh. Occasionally, to deal with incomplete or low quality imaging data, a twin data processing pipeline is employed where a template geometry is processed to extract vital geometric information which is then passed to robustly model patient specific anatomical model from low-quality imaging data.

image quality for better visual appearance for subsequent operations. The most commonly used methods utilize global contrast manipulation based on global [138, 238] or local histogram equalization [138, 238, 69, 272], *retinex model* [170, 169] and wavelet decomposition [202, 186].

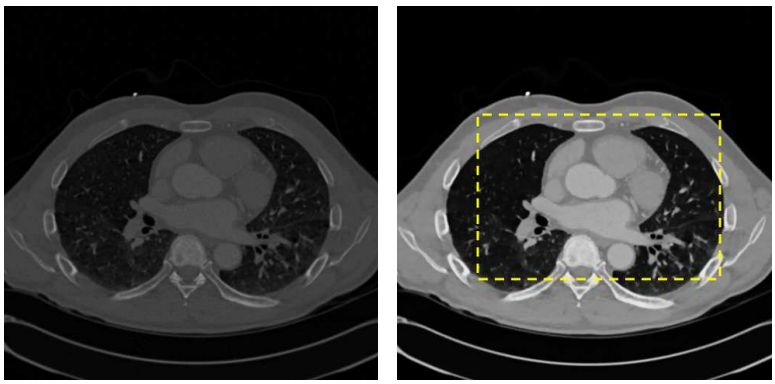


Figure 2.3: The performance of the contrast enhancement algorithm is shown on one slice of the CTA data.

We have developed a fast method for image contrast enhancement [332] based on a localized version of the classical contrast manipulation [138, 238]. In this method, we design an adaptive transfer function for each individual voxel, based on the intensities in a suitable local neighborhood. First, we compute the local statistics (local average, local minimum, and local maximum) for each voxel using a fast propagation scheme [101, 326]. Then a transfer function is designed based upon the calculated local statistics. Various linear or nonlinear functions can be used here to stretch the contrast profile. We build a transfer function which consists of two pieces: a convex curve in the dark-intensity range and a concave curve in the bright-intensity range. The overall function is C^1 continuous. Finally, we map the intensity of each voxel to a new one using the calculated transfer function. The performance of this algorithm is shown in Figure 2.3.

[B] Filtering:

The input images are often contaminated with noise and are therefore need to be filtered. Traditional image filters include Gaussian filtering, median filtering, and frequency domain filtering [138]. Compared to these, anisotropic filters are preferred as they tend to preserve the features better. Bilateral filtering [51, 117, 122, 295] is a straightforward extension of Gaussian filtering by simply multiplying an additional term in the weighting function. Partial differential equation (PDE) based techniques, known as anisotropic geometric diffusion, have also been studied [230, 306]. Another popular technique for anisotropic filtering is by wavelet transformation [112]. By carefully designing the filter, one can smooth image noise while maintaining the sharpness of the edges in an image [317]. Finally, the development of nonlinear median-based filters in recent years has also produced promising results [148, 147]. Among the aforementioned techniques, two methods for noise reduction have been suggested for tomographic data sets, namely wavelet filtering [275] and non-linear anisotropic diffusion [129].

Our approach, utilizing a bilateral pre-filtering coupled with an evolution driven anisotropic geometric diffusion PDE (partial differential equation), has shown significant results in enhancing the features of intensity maps. The PDE model is :

$$\partial_t \phi - \|\nabla\phi\| div\left(D^\sigma \frac{\nabla\phi}{\|\nabla\phi\|}\right) = 0$$

The efficacy of our method is based on a careful selection of the anisotropic diffusion tensor D^σ based on estimates of the normal and two principal curvatures and curvature directions of a feature isosurface (level-set) in three dimensions [35, 167, 333]. The diffusivities along the three independent directions of the feature isosurface are determined by the local second order variation of the intensity function at each voxel. In order to estimate continuous first and second order partial derivatives, a tricubic B-spline basis is used to locally approximate the original intensity.

[C] Classification/Segmentation:

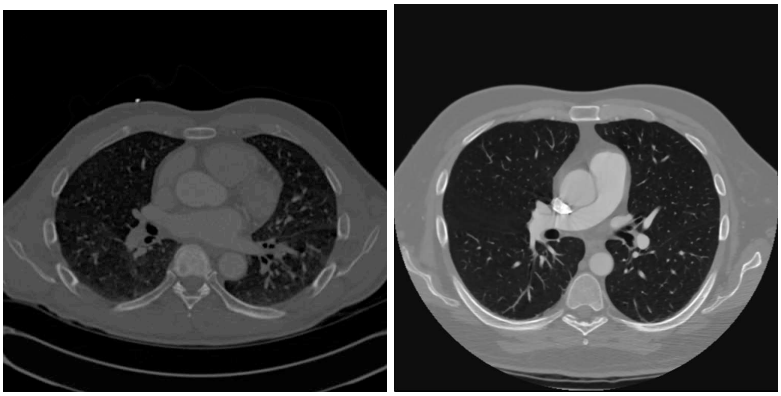
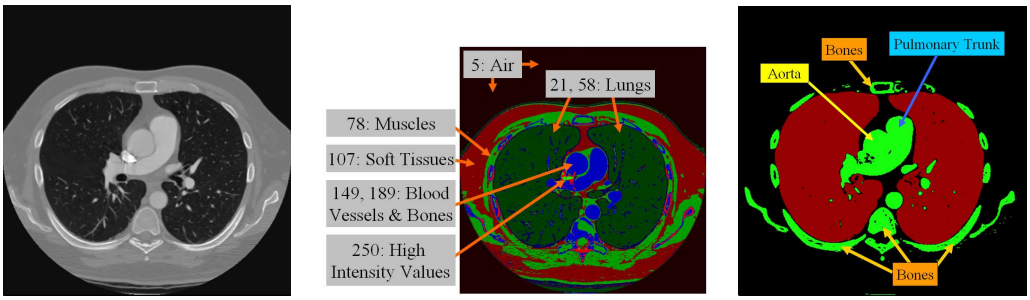


Figure 2.4: The performance of the bilaterally pre-filtered and anistropic diffusion filtered algorithm is shown on one slice of the CTA data

Voxel Classification: The Fuzzy C-Means (FCM) algorithm [56] and the Expectation Maximization (EM) algorithm [293] have been used for soft clustering in data-mining and image classification. Pham *et al.*proposed an Adaptive Fuzzy C-Means (AFCM) algorithm to classify inhomogeneous medical images and volume datasets [231, 232]. Ahmed *et al.*proposed a bias corrected FCM algorithm to compensate inhomogeneities of images of volume datasets [2, 3]. Each of these algorithms minimizes an objective function through iterative methods. Gopal *et al.*proposed a maximum likelihood estimate algorithm with a Spatially Variant Finite Mixture model (SVFMM) for image classification [139]. Laidlaw suggested a partial-volume Bayesian classification algorithm based on Bayes theorem [185].



(a) (b) (c)

Figure 2.5: Voxel Classification: A single slice of CTA image of a patient (a) is classified to tag voxels belonging to different anatomical regions (b). The intensity values are also marked (b). The final classified voxelized image is shown with most important regions (c).

Our goal of 3D map segmentation is to partition the map into a number of connected regions of interest. We have compared and implemented several classification algorithms [185, 3, 176, 184]. We first locate the seed points by gradient vector diffusion [295]. We then compute the min-max range of every seed point's neighbors and cluster the seed points to belong to the same region if the min-max ranges overlap. We then apply GVF snake [313, 331] to cluster the voxels falling into separate regions. In addition, the contour spectrum is used to identify the number of materials in the image and is also used to select critical isovalues based on volume fraction of the material [44]. We have also developed a multi-dimensional signature based voxel classification scheme [225] appropriate for medical imaging data. Figure 2.5 shows the results of the classification on a single slice of the patient scan where the voxels have been classified into the background, lungs and vasculature.

Segmentation via Fast Marching Method: Segmentation is a way to electronically dissect the significant biological components, and thereby obtain a clear view into the machinery's architectural organization[124]. Segmentation is

usually carried out either manually [156, 181, 195, 214, 149] or semi-automatically[303, 130]. Current efforts on the decomposition still largely rely on manual work with an assistance of a graphical user interface [343, 168]. Manual segmentation can be tedious and often subjective [195, 208]. Automated segmentation is still recognized as one of the hardest tasks in the field of image processing although various techniques have been proposed for automated or semi-automated segmentation. Commonly used methods include segmentation based on edge detection, region growing and/or region merging, active curve/surface motion and model based segmentation. In particular, two techniques were discussed in details in the electron tomography community. One is called *water-shed immersion method* [303] and the other is based on normalized graph cut and eigenvector analysis [130].

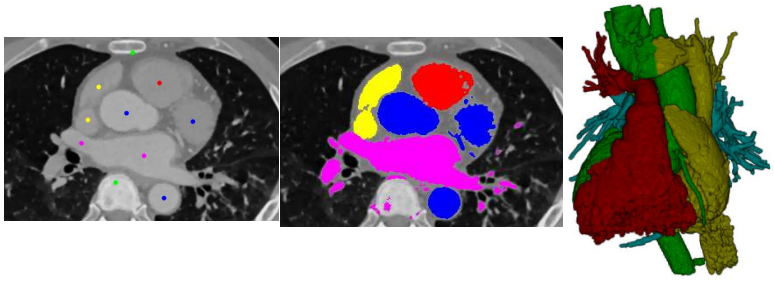


Figure 2.6: Segmentation via Fast Marching Method: Leftmost subfigure shows a single slice of the CTA image of a patient along with manually placed seeds for each of the main compartments of the human heart to be identified. The middle subfigure shows the segmentation on a single slice colored by segmented regions. The rightmost subfigure shows the three dimensional segmented model of heart via isosurface rendering. The different colors represent left and right ventricle and aorta.

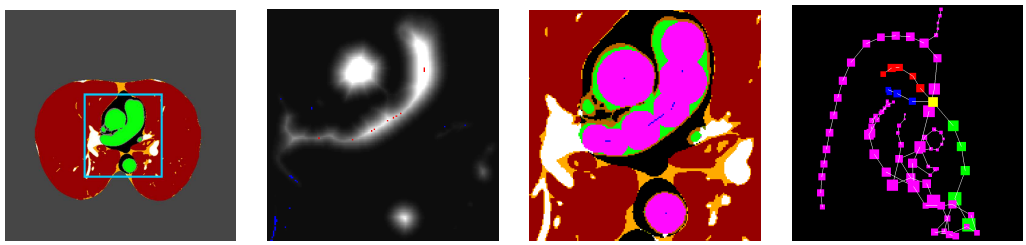
In [43, 334, 47] we adopted a variant of the fast marching method [205, 260, 259]. In this method a contour is initialized from a pre-chosen seed point, and the contour is allowed to grow until a certain stopping condition is reached. Traditionally this method is designed for a single object segmentation. We present an approach based on an idea of "re-initialization" by simply regarding and classifying the critical points as seeds. Every seed initiates one contour and all contours start to grow simultaneously and independently. We further classify the critical points into clusters and merge the growing contours which are initiated by the critical points in the same cluster. This multi-label idea has been used elsewhere (e.g., [268]), but the detection and classification of seeds are different in our approach. Figure 2.6 shows the process of segmentation, namely the seed point classification and region growing on a single slice of the image followed by the final segmented three dimensional model of human heart from patient imaging data.

[D] Skeleton Extraction:

Extraction of skeletal description often leads to a complexity-reducing better understanding of the image as it amplifies lower dimensional key features present in the data. Image based skeletonization algorithms are abound in the field of image processing. The previous efforts on this topic by other authors can be categorized into three approaches - one based on isotropic diffusion, governed by a set of linear PDEs [284, 84], one using scale-space theory [196, 235, 217] and one based on pseudo-distance map [165].

We have developed two distinct approaches to extract skeletons from imaging data which offer robustness and efficiency. First approach is based on the critical point structure of the imaging data. We first compute the gradient vector field at every voxel of the imaging data. Because of the noise, the vector field is somewhat arbitrary and do not carry much useful information. To bring out the underlying structure, we then apply gradient vector diffusion (GVF). The resulting vector field is then analyzed to detect the critical points and these critical points are joined to form a skeletal structure of the foreground of the image. The details of the process is given in [333].

The second approach for skeletonization starts with an isosurface and the distance function induced by it [227]. It then places a sequence of inner medial balls in a greedy fashion, capturing as much inner volume as possible. A neighborhood graph, based on the intersection pattern of these inner medial balls, is then constructed which provides the one dimensional skeletal structure. Figure 2.7 shows an example result of this algorithm applied on the CTA data of human heart.



A.1 A.2 A.3 A.4

Figure 2.7: Skeletonization: A slice of the imaging data, its distance map and the extracted skeletal graph using the algorithm in [227] are shown.

[E] Flexible Alignment:

Image registration is a commonly employed to flexibly match two different instances of a biological structure. In the context of cardiovascular modeling, the problem is to fit one image of the anatomy in question with its counterpart observed at a different time, or imaged from a different patient. Based on the transformation model that is to be applied on one instance (source) to fit the other (target), image resgistration is primarily of two types - Affine and Elastic/Deformable.

In case of affine registration, the relationship between the source and target image is established via a set of transformation parameters, and then those parameters are estimated by minimizing a quadratic error function. Algoithms under such approaches can be found in [74, 183, 191]. We have developed an algorithm in [10] which exploits the non-equispaced Fourier transformation techniques [123, 118] to speed up the affine image registration process.

The task is far more challenging when deformation of one image needs to be taken into account to properly fit it into the other image. For a nice survey on affine and deformable image registration algorithms, see [64]. However this survey is old, and since its publication many other algorithms have been published. Bajcsy and Kovacic modeled the elastic image registration problem by the deformation of elastic plates [46]. Christensen *et al.*considered the deforming image to be embedded in a viscous fluid whose motion is governed by Navier-Stokes equation [83]. Based on a similar viscous fluid registration scheme, Yanovsky *et al.*designed a new energy function introducing Jacobian maps [318] and this method was shown to perform better than [83] in terms of converegence and stability. There are other level-set based methods, *e.g.*by Clarenz *et al.*[86], and via edge matching technique by Mumford and Shah [219].

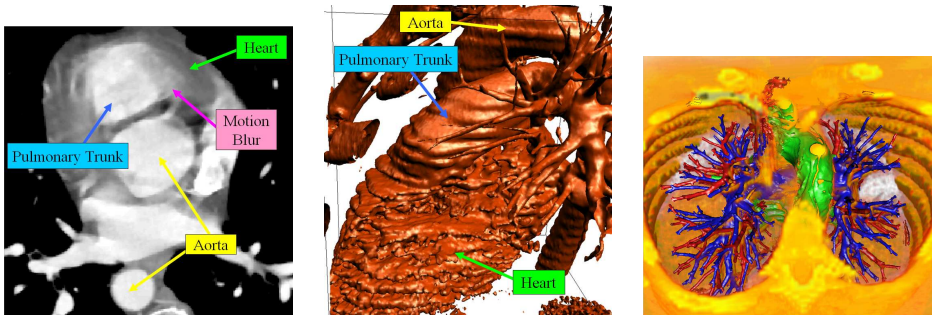
Geometry Processing

[A] Surface Extraction:

Geometry extraction from three dimensional volumetric data is a primary step toward further analysis. There are several approaches for accomplishing this task.

Contouring: Isosurfacing is a popular method to extract surface geometry from scalar volume containing intensity values of the scanned anatomy. There are typically two types of contouring method frequently used in the literature - Primal Contouring and Dual Contouring. The most widely used primal contouring technique is the Marching Cubes Method [200] which extracts the geometry in a piecewise fashion by visiting every voxel of the volume data. There are several improvements of this technique that has been reported since the first appearance of the algorithm [199]. Dual contouring technique is similar to primal contouring in a sense that it also extracts the isosurface in a piecewise fashion. However it samples every voxel, instead of every edge of the voxel, to better approximate possible sharp features in the extracted geometry. We have experimented with both techniques for extracting a geometry from Cardiac CT data and we have seen that they perform similarly without rendering any significant advantage of any technique over the other. Figure 2.8 shows sample isocontours extracted from CTA imaging data of human heart.

Other than isocontouring, one can also apply level set based methods where a seed is grown to capture the boundary of a region in the image based on the intensity values. In literature, such technique is commonly known as *snake* [313, 331]. Note, this is similar to the image segmentation technique described earlier.



(a) (b) (c)

Figure 2.8: Contouring: A single annotated slice of human cardio-vasculature and the geometry extracted from the imaging data via contouring (isosurfacing) are shown from left to right. In the rightmost subfigure, the heart of the patient is shown in green while the blood vessel tree is color coded by red and blue depending on if a branch is an artery or a vein.

Point Cloud Reconstruction: Both of these approaches are very sensitive to the noise present in the data and especially isosurfacing techniques suffer when the imaging data is inhomogeneous. To circumvent these problems we adopt a third approach which is based on scattered data interpolation. After performing image segmentation on the CT image, we obtain a set of voxels belonging to every region of interest. We then apply the point based reconstruction technique to extract the geometry from the cloud of boundary voxels.

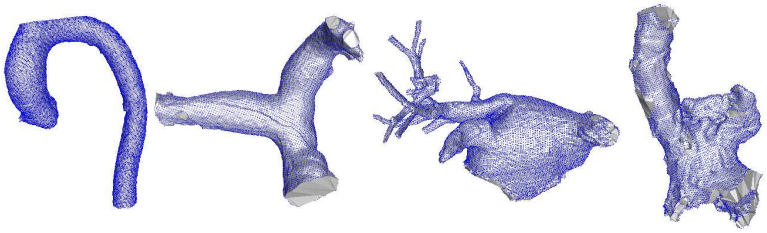


Figure 2.9: Surface Reconstruction from Point Cloud Data: Major components of the human heart are reconstructed using the voxels surrounding the boundaries of the individual regions.

The point based reconstruction technique has been researched extensively in the last decade. We refer the readers to some recent surveys for prior work in this area, *e.g.*[105]. We have adopted two recent techniques for our purpose of reconstruction - *TightCocone* and *RobustCocone*. *TightCocone* algorithm by Dey and Goswami [108] reconstructs a watertight triangulated surface from possibly undersampled input point cloud. A variant of this algorithm, called *RobustCocone*, was also developed in 2004. This algorithm is particularly suitable for noisy data. In our case, we often encounter noise in the segmented image even after applying image segmentation techniques, and to tackle such cases, we use *RobustCocone* for a reconstruction of the geometry. Figure 2.9 shows the results of surface reconstruction on the pointsets sampling compartments of human heart.

Surface reconstruction from scattered data has also been approached using variational approach. These techniques typically formulate an energy function based on the input data points and try to extract a surface that minimizes that energy. Such approach was first advocated by Zhao *et al.*in [341] who formulated the energy function as the integral of the distance function weighted by the area element of the input set of primitives for which a surface needs to be fit. Then they evolved an initial guess using a convection based approach.

We adopt an approach based on higher order level set spline (HLS) method. Given a non-negative energy function $g(\mathbf{x})$, the surface Γ is defined to be the one that minimizes the energy function $E(\Gamma) = \int_\Gamma g(\mathbf{x})d\mathbf{x} + \varepsilon \int_\Gamma h(\mathbf{x},\mathbf{n})d\mathbf{x}$. Given a input set of points, one then formulates this energy using the distance function and evolves an initial approximation to guide the evolution so that the resulting deformation minimizes the energy. Details of this method can be found in [42].

[B] Curation/Filtering:

The cardiovascular geometry extracted from imaging data typically has topological anomalies, namely small components, spurious noisy features etc. Therefore a careful investigation and subsequent removal of the spurious features present in the data is essential. Following are different scenarios.

Regularization: Geometry from volume data is often reconstructed via image segmentation. However the set of voxels segregated from the imaging data does not always conform with the true surface topology. As a result, we encounter subsets of voxels which do not sample a two dimensional manifold. Therefore it is important to recognize the dimension of the underlying space of the voxels marked by the segmentation process and remove the spurious ones. To perform this task, we use the technique described in [82]. A similar Voronoi-based approach was also reported in [107]. Following [82] we first construct a k-neighborhood graph. Then for every point we collect the neighboring points and perform Principal Component Analysis (PCA) on that subset. The eigenvalues of the covariance matrix determines the underlying dimension of the manifold. More precisely if all the eigenvalues are almost equal, the voxel is inside the segmented region and the point samples a three dimensional manifold. If two eigenvalues are almost equal and one is much smaller than the other two, the voxel lies on the boundary surface of the segmented region and is a true candidate for subsequent geometry reconstruction. Finally if two of the eigenvalues are much smaller than the third one, then the voxel samples a dangling one dimensional strand and such voxels must be removed.

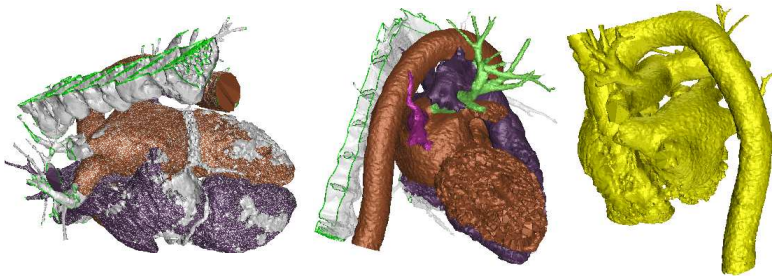


Figure 2.10: Geometry Curation: Geometry of human heart extracted from the imaging data, which is cluttered with bone and other unnecessary parts, is cleaned up using the curation process.

Volumetric Feature Quantification: Given a set of points P sampling the entire shape, possibly contaminated with topological artifacts like small connected components and thin tunnels, we synthesize a distance function h_P which assigns every point in $\mathbb{R}^3$ the distance to the nearest sample point in P. There are four types of critical points of h_P, namely maxima, index 2 saddles, index 1 saddles and minima. It was shown that these critical points can be detected efficiently using the duality of Voronoi and Delaunay diagram of the original pointset P [267]. It was further shown that the stable manifolds of the maxima interior and exterior to the shape contains useful information connecting to the primal and complementary feature space of the shape [106, 20]. Figure 2.10 shows how the bone, ribs and thin blood vessels are removed via volumetric feature quantification process since they are not essential for creating a suitable model of human heart.

[C] Segmentation:

Based on the critical points of the distance function induced by the input geometry, we perform the geometry segmentation as follows. The detail of the algorithm is given in [106].

The geometric shape is given by a set of points P sampling the shape. The *feature* of the shape is then defined in terms of the stable manifold of the maxima of the distance function h_P. The maxima are first computed by identifying the Voronoi vertices which lie inside their dual tetrahedra. Applying a Delaunay-based reconstruction technique on the pointset, one can further classify the tetrahedra holding the maxima into *inside* or *outside*. For our purpose we use only the *interior* maxima. We compute the stable manifold of these maxima using the algorithm in [106]. The adjacent stable manifolds are then merged if the generating maxima have almost same value of h_P as measured by a parameter δ. Figure 2.11

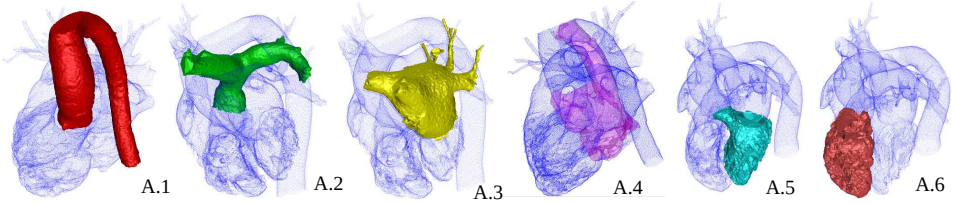


Figure 2.11: Geometry Segmentation: Different components - Aorta (A.1), Pulmonary Artery (A.2), Left Atrium (A.3), Right Atrium (A.4), Left Ventricle (A.5) and Right Ventricle (A.6) are extracted from the point sample of the whole heart.

shows the performance of this algorithm on the cardiovascular geometries. In this figure, the six main components of a human heart, namely aorta, pulmonary artery, left and right atrium and ventricle are segmented out from a set of points sampling the boundary of heart.

[D] Skeletonization:

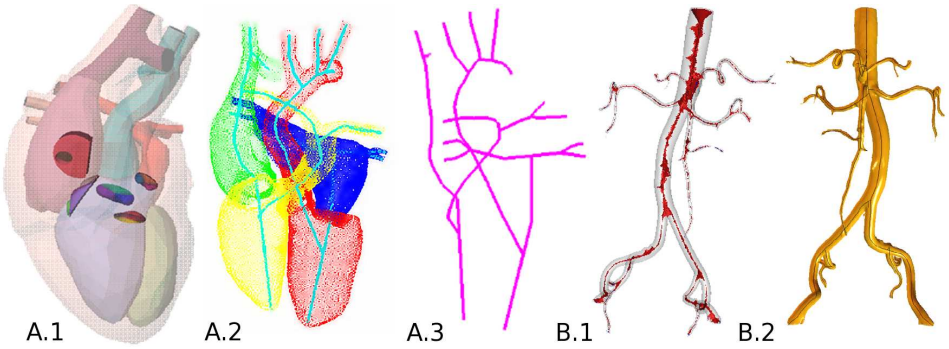


Figure 2.12: Skeletonization: A. Skeleton of the template geometry is extracted. B. The skeletal structure of the abdominal aorta is extracted. Note, compared to the medial axis (B.1), the unstable manifold of index 2 saddles is a much cleaner one dimensional skeletal structure.

Computing skeletons of a geometric shape is a research issue that has been around for a long time. Medial axis cite(Blum) is considered a standard skeletal description of a shape and there are algorithms [110, 81] and publicly available software to compute the Medial Axis Transform (MAT) of a shape from its pointsample [87]. However, medial axis is composed of planar (two dimensional) and linear (one dimensional) parts. In order to compute such dimension-dependent decomposition of the medial axis, as well as pruning away hairy branches from the medial axis one requires some extra gadgets which we describe in the next paragraph. There are some previous works which focussed on computing a linear skeleton of an arbitrary shape. Some of these are topological thinning [60], distance field based methods [162, 61, 152, 342], potential field based methods [93], thinning via medial geodesic function [109] and others [96, 222, 300]. Cornea *et al.*[92] give a comprehensive survey of these techniques.

Once a suitable description of the geometry is obtained either by reconstruction or by contouring, the distance function based approach is used to compute a skeletal structure of the shape. As for segmentation, the critical points of the function h_P is computed for a set P of input points sampling the shape. The index 1 and index 2 saddle points are then detected using the Voronoi-Delaunay duality [267]. To generate the skeletal structure, we then compute the unstable manifold of these critical points. The unstable manifold of an index 2 saddle point (U_2) is one dimensional and the unstable manifold of an index 1 saddle point (U_1) is two dimensional. Moreover, every U_1 is bounded by some U_2's. The details of the computation of U_1 and U_2 are given in [141]. Figure 2.12 shows two instances where the skeletal structures have been constructed using this method.

[E] Alignment:

Alignment of two similar but not identical geometric objects is a difficult problem. There are relatively few papers in the geometric modeling community that address this problem. Recently, an interesting technique was reported by Eckstein *et al.*[120] where generalized surface flow was used for non-rigid alignment of a template geometry into the patient data. The authors design an energy function based on pseudo-Hausdorff distance between the two geometries and evolve the template geometry to fit the patient geometry following the gradient of the energy function.

We experimented with a different approach for non-rigid fitting of the segments of the template heart into the patient data in order to inherit the correct topological structure from the template geometry as well as retrieve the missing information in the patient data. We construct a skeletal description of different parts of the template geometry as described earlier. Every segment is then described as a union of balls centered at the one dimensional skeleton following a popular approach due to [6]. A mass-spring network is then built where each ball's mass is proportional to its radius and the spring constant is taken to be constant. The normal mode analysis (NMA), which is a popular way to depict the vibrational nature of a molecule [283], is then applied to the network which produces a spectrum of possible deformed shapes. We choose one deformed shape from the spectrum that best fits the patient geometry.

[F] Quality Meshing:

The final goal of this Imaging to Modeling software system is to produce suitably discretized meshed model of the imaged biological entity. The task of meshing is primarily divided into two parts - Boundary Element and Finite Element. Each of Boundary Element and Finite Element meshing again has three sub-parts.

Boundary Elements: Boundary Element meshing refers to the meshing of the surface geometry. Depending on the smoothness and shape of every patch forming the surface mesh.

(a) Triangle/Quadrilateral Elements: Given a CT image, the task is to compute a triangulated or quadrangulated discretization of the boundary of the cardio-vascular anatomy. In a sense the task is similar to contouring the zero-set of the original intensity function or a distance function induced by the reconstructed geometry. There are numerous algorithms to accomplish such tasks as mentioned in the previous subsection. However in this step our goal is to build a boundary element mesh of superior quality than what is typically output by the contouring routines. The mesh quality metrics are described in [337].

We approach the problem in two steps. First we apply the dual contouring method proposed by Ju *et al.*[172] and apply geometric flow to improve the quality of the surface mesh. Figure 2.13 shows the performance of the boundary element meshing algorithm.

(b) B-Spline Elements: Building B-Spline model for free-form geometric objects has been an active area of research in the past. Given a triangulated or quadrilateral surface mesh, there have been numerous approaches to build a smooth network of B-Spline patches to model the underlying surface [126]. Subdivision based methods, namely Catmull-Clark [75], Doo-Sabin [113, 114] and Loop subdivision [198] for producing quadratic and cubic B-Spline models from triangulated and quadrilateral control meshes have also gained popularity due to their simplicity. However in most of the cases, it is not straight-forward how to generate the initial control mesh for free-form shapes especially when the input is a set of scattered set of points or a set of voxels representing the boundary of a segmented region inside a three dimensional volume.

Until recently there were very works that dealt with the problem of building a control quadrilateral mesh from a free-form geometry of arbitrary topology [182, 119]. Recently there have been substantial research works in the computer graphics community that tackled the problem of quadrangulating a surface mesh following the intrinsic anisotropy of the geometry [88, 111, 320]. Following these approaches, one can build a quadrilateral base surface mesh on which any of the standard subdivision scheme can be applied to build the desired B-Spline model.

(c) A-Patch Elements: The linear meshes do not always provide the adequate smoothness for them to be effectively used further. This is the reason a higher order meshed description of the geometry is often desired. Bajaj *et al.*presented a solution for this problem in 1995 where they devised a scheme that takes a triangulated surface mesh and builds a higher order (cubic) patch complex to describe the same surface. Under certain conditions, they also showed that these patches meet in a C^1 continuous manner.

They called this patched description of a two dimensional geometry an algebraic surface patch or an *A-patch* [18].

The construction of A-patches proceeds as follows. First a triangle mesh is created that linearly approximates the given geometry. A tetrahedral scaffolding is then built around it so that the triangle mesh lies inside it. By assigning a suitable weight at every node of the scaffold, a polynomial inside each tetrahedron is constructed in such a way that its zero set satisfies certain properties that makes the higher order patch inside the tetrahedron free of singularity. Moreover the pacthes are derivative-continuous across the tetrahedra. The patch complex then provides a higher order smooth description of the geometry. Details of this method can be found in [18]. This method of construction of surface A-patches has also been generalized to hexahedral elements, as well as to the multi-resolution construction of shell finite elements [77, 39]. Figure 2.13 shows a higher order geometric model of human heart.

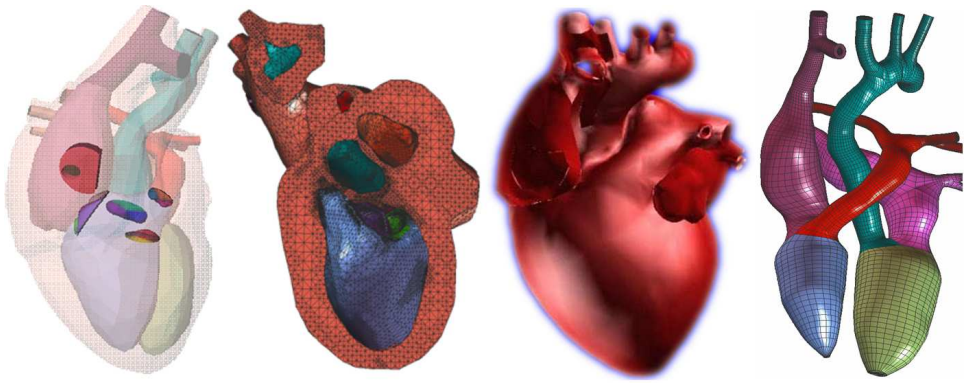


Figure 2.13: Meshing: Triangle-Tetrahedral, A-Patch, NURBS meshes.

3D Finite Elements: Finite Element meshing refers to the techniques of producing the discretization of the volume enclosed by the surface geometry.

(a) Tetrahedral/Hexahedral Elements: There have been prior works done on building volumetric (finite element) tetrahedral and hexahedral meshes from imaging data. For a detailed survey on the prior works see [142].

We have developed two algorithms for such purposes - TetMesh and HexMesh. The approach is as follows. First we extract a correct boundary using the boundary element meshing. For tetrahedral mesh generation we use the triangulated boundary and for hexahedral mesh generation we use the quadrangulated boundary meshing scheme as described above. Next, we design a series of templates to build a solid tetrahedral or hexahedral mesh from that conforms with the boundary mesh. The details about the templates are given in [338, 337]. As with the boundary element mesh, we further improve the quality of the finite element mesh by applying geometric flow [339]. Figure 2.13 shows the tetrahedral mesh cut-away for a template human heart.

(b) B-Spline Elements: As the most highly developed and widely utilized technique, NURBS (Non Uniform Rational B-Splines) [245, 233, 290] has evolved into an essential tool for a semi-analytical representation of geometric entities. Sometimes NURBS solid models are taken as input for finite element mesh generation [145]. Anderson *et al.*proposed a fast generation of NURBS surfaces from polygonal mesh models of human anatomy [9]. An enhanced algorithm was developed for NURBS evaluation and utilization in grid generation [328]. In isogeometric analysis [159], NURBS basis functions are used to construct the exact geometry, as well as the corresponding solution space.

We have developed a skeleton-based approach for building NURBS model of vasculature. Using the skeletonization approach described in the earlier paragraph, a one dimensional polylinear skeletal structure is first extracted from the tubular geometry of vasculature. Then we design a set of templates that

builds a hexahedral mesh around the skeleton. The details about the templates can be found in [340]. The hexahedral mesh is used as the control mesh for further NURBS mesh generation. Figure 2.13 shows the NURBS model of the inner blood volume of human heart.

(c) Shell A-Patch Elements: Shell structures appear frequently in biological entities. The muscle wall of heart and blood vessels are perfect examples of such surfaces. It is desired to model such *fat* surfaces with desired smoothness. Bajaj *et al.*presented algorithms that model smooth shell structures using shell A-patch finite elements [77, 39].

The algorithm takes a pair of triangle mesh as input where the correspondence between the triangle meshes is implicit. It then decimates both the meshes simultaneously to build a coarser representation and occasionally merges the adjacent triangles to form quadrilaterals wherever possible. Using the correspondence between the triangles and quadrilaterals in the two-sheeted surface, the interval volume is then filled with 3-sided (triangular) and 4-sided (quadrilateral) prisms. A C^1 piecewise trivariate function is then constructed over this collection of prisms. The range of the function varies from -1 to 1. For any scalar $\alpha \in [-1, 1]$, the 0-set thus gives the higher order approximation of the intermediate surface within the shell element. When α equals -1 or 1, the resulting patch gives a higher order approximation of the inner and outer boundaries of the shell.

Implementation Results

We exhibit the implementation results of our image and geometry processing algorithms on modeling the Heart and Vasculature (Coronary Artery, Pulmonary Arterry, Abdominal Aorta and Thoracic Aorta).

Heart We have experimented with images of patient hearts. For illustration of the performance of the algorithms, we have selected two datasets one of which is a low quality image whereas the other is of relatively better quality.

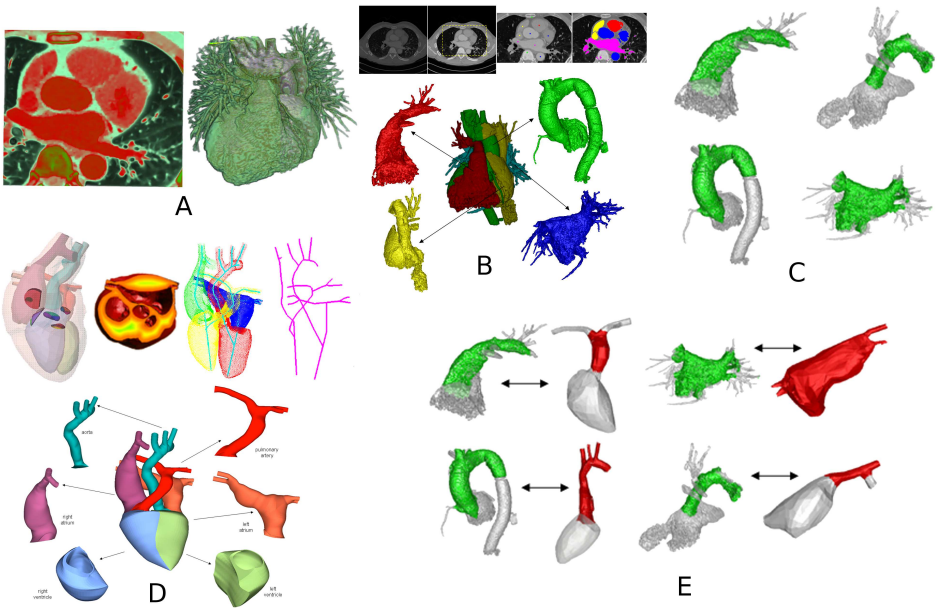


Figure 2.14: Result: Heart I. (A) One slice of the original data and isocontour-enhanced volume rendering of the input. (B) Due to relatively poor quality, the imaging data is first passed through the image processing unit that enhances the contrast and segments the volume into major components using fast marching based image segmentation. (C) The extracted geometry from the segmented image is then curated to keep only the (green) portions which can be matched with the corresponding portions from the template heart model (D,E).

The first heart dataset is of dimension $512 \times 512 \times 432$ and the spacing in x, y, z directions are respectively $0.390625\,mm, 0.390625\,mm, 0.3\,mm$. We first applied the contrast enhancement to the original image and then applied the fast marching based segmentation on the contrast-enhanced image to separate the subvolumes corresponding to the aorta, pulmonary artery, right atrium and left atrium. Because of the poor quality of the image, the ventricles could not be recovered well. The result of the image processing on this dataset is shown in Figure 2.14(B). After the four components of the patient heart are segmented from the volume, we took the boundary voxels of each of these four regions and applied surface reconstruction from scattered points to build the initial triangulated geometric models. As one can see these models have a reasonable amount of spurious parts as well as some missing information that could not be retrieved from the patient data. We analyzed each of the recovered geometries using the critical point structure of the distance function described in Section 2.5.4. Using curation and geometry segmentation, we identified the portions which most prominently correspond to the portions of a template heart model while pruning away the undesired portions. Figure 2.14(C) shows the relevant portions that have high correlation with the corresponding portions of the template geometry. Parallel to the processing of the patient heart, we also performed geometric analysis of the template heart model. We construct the one dimensional skeletal structure of all the six components of the template, as well as we segmented the template into aorta, pulmonary artery, right and left ventricle and atrium (Figure 2.14(D). From there we could draw a correspondence between the segmented portions of the patient heart with the template heart (Figure 2.14(E)).

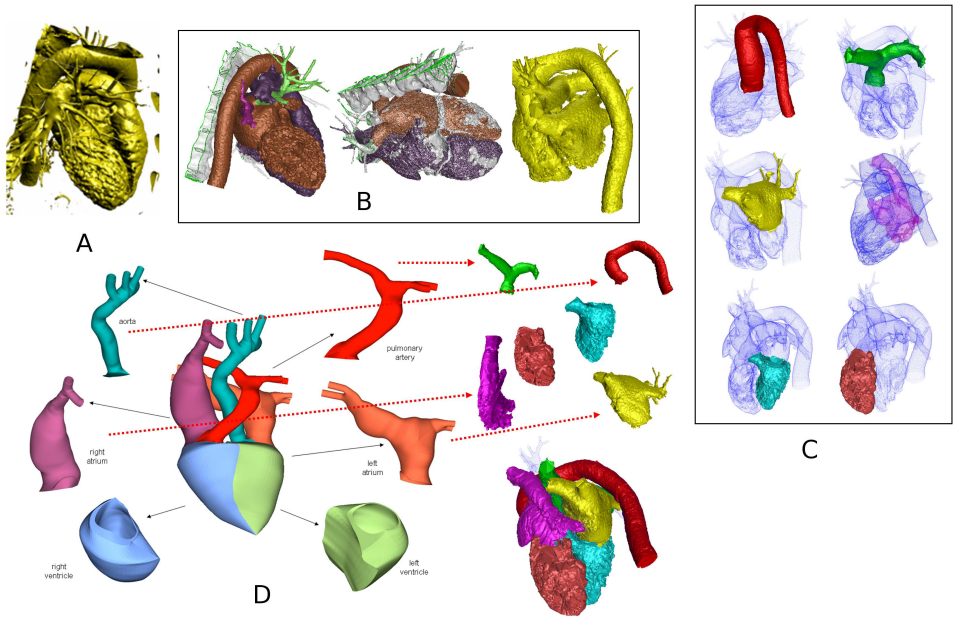


Figure 2.15: Result: Heart II. Starting with an isocontour extracted from the raw imaging data (A), the model is first curated to extract a clean geometry without the nearby bone structure or the thin blood vessels which are otherwise irrelevant for the specific task of modeling heart (B). This initial model is then geometrically segmented into six main components (C) and further a correspondence is drawn with the segmented template. The processing of the template heart model, namely, skeletonization and annotated segmentation is shown in (D).

The second dataset was of better quality in terms of contrast and noise present. We first extracted the geometry via isocontouring. However using a single isovalue could not entirely serve the purpose as it also extracts the surrounding vasculature as shown in Figure 2.15(A). We therefore resorted on curation to extract just the geometry of heart by pruning away the surrounding thin blood vessels using curation (Figure 2.15(B)). After this step we were left with the geometry of heart which we further segmented using the stable manifold of the maxima of the distance function induced by the geometry. The segmentation step was crucial as it was able to separate the six main components of the patient heart, namely aorta, pulmonary artery, right and left ventricle and atrium as shown in Figure 2.15(C). We were then able to draw the correspondence between each of the segmented parts with the corresponding ones from the template geometry as shown in Figure 2.15(D).

Vasculature

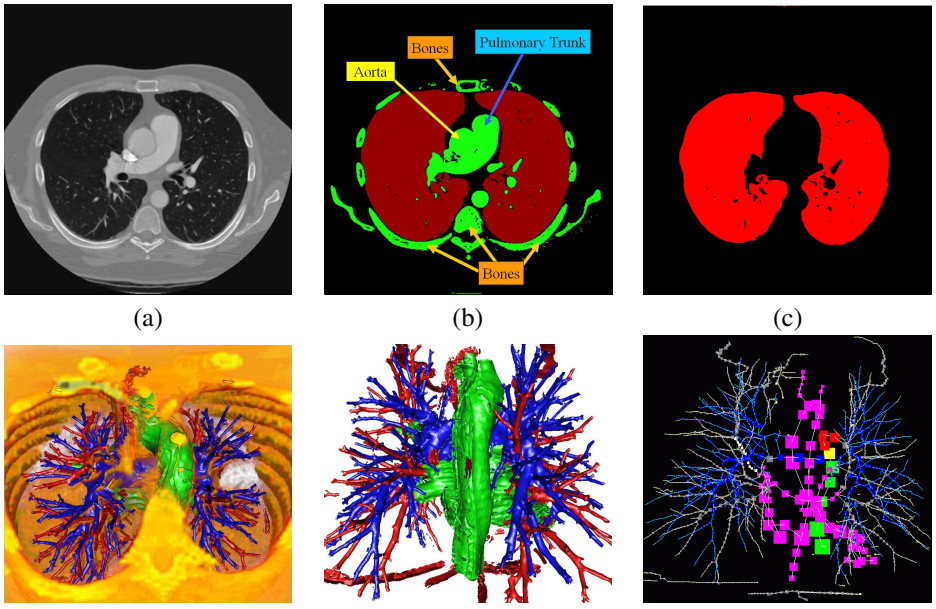


Figure 2.16: Result: Pulmonary Artery. The imaging data (one slice is shown in (a)), is first classified to identify the voxels belonging to lungs and the vasculature (b,c). Using the skeletonization technique from Section 2.5.4, the arterial (red) and venous (blue) trees are detected. The vasculature and heart (green) are shown in (d-f) superimposed with the volume rendering of the CTA image.

Pulmonary Artery The primary objective behind modeling pulmonary artery was to detect pulmonary embolism (PE) automatically [228]. This required an initial artery-vein separation from the CT scans of the vasculature which was performed directly from the input imaging data by the image skeletonization technique described in Section 2.5.4. The skeletons are traced from the end of the branches and traversed upward toward the heart. Once the trace reached the patient heart through a series of disambiguation, as needed because of the poor image quality, one of the branches was tagged arterial while the other is tagged venous. At the same time, the rest of the volume was classified using the voxel classification technique as described in Section 2.5.4 (Figure 2.16(b,c)). This led to a complete characterization of the CT data into the major components along with the artery and vein separated (Figure 2.16(d-f)).

Abdominal Aorta Starting with the CT scan of the abdominal section of the patient, we first extracted the geometry using an isovalue that best captures the geometry of the abdominal aorta along with the surrounding bone structure and other anatomical parts which are however irrelevant for the modeling of the aorta itself. To separate the aorta, we performed geometry segmentation, using the stable manifold of the maxima of the distance function. The segmented aorta is shown in green in Figure 2.17 (a,b). We then performed geometry skeletonization on this segmented geometry and as a result a one-dimensional skeletal structure was produced (Figure 2.17 (c,d)). This skeleton was then used for building a swept hexahedral volume that best represented the geometry of the aorta, as well as served the purpose of a control mesh which was then used to build a solid NURBS model of the abdominal aorta (Figure 2.17 (e,f)).

Thoracic Aorta Starting with the scan of the patient heart, we first segmented out and extracted the geometry of the thoracic aorta via isocontouring. We then performed the skeletonization of this geometry in order to obtain an one-dimensional polylinear skeletal (Figure 2.18 (A.1)). An external pathway is added to the skeletal structure in order to simulate the LVAD used in time of open-heart surgery. Using the skeleton as a sweeping path, we then built a hexahedral control mesh that best

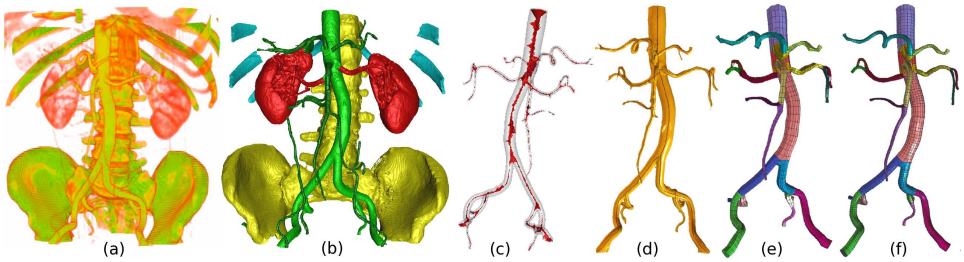


Figure 2.17: Result: Abdominal Aorta. Volume rendering of the original imaging data is shown in (a). An isocontour is extracted from the imaging data from which the abdominal aorta (green) is segmented (b). The medial axis and the linear skeleton extracted from it are shown in (c,d) from which an initial control polyhedron (e) and the final NURBS model (f) are created.

represented the aorta along with its inner volume (Figure 2.18 (A.2)). This control mesh was then used to construct a solid NURBS mesh of the thoracic aorta as shown in Figure 2.18 (A.3).

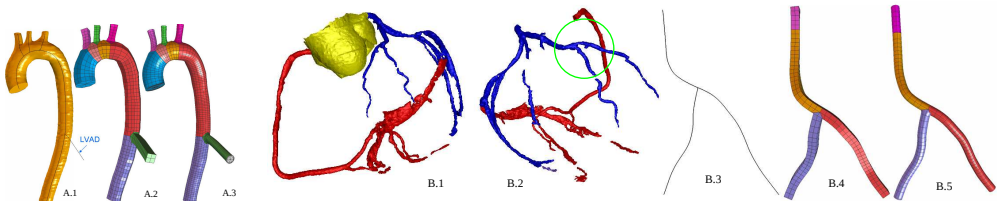


Figure 2.18: Result: A: Thoracic Aorta. Skeleton is extracted from the geometric model (A.1). An artificial pathway is added to simulate the LVAD. The hexahedral control mesh and the resulting NURBS model are shown in (A.2,3). B: Coronary Artery. The coronary artery is segmented into left and right subtrees (B.1,2). NURBS mesh of a small portion of the tree is then built using the extracted skeleton (B.3-5)

Coronary Artery Starting with a CT scan of the patient heart, first the coronary artery was extracted as shown in Figure 2.18 (B.1). The coronary artery has two main branches - right and left, which were then geometrically segmented from the whole vascular structure. Figure 2.18 (B.2) shows both the vasculature trees colored in red and blue. We then computed the skeletal structure for each of these substructures using the skeletonization method described earlier (Figure 2.18 (B.3)) and subsequently a NURBS mesh model was constructed using the swept volume technique as described in the meshing subsection earlier (Figure 2.18 (B.4, B.5)).

2.5.5 systems — live cell imaging, phase-contrast and tagged MRI

2.6 Bibliographic Notes

2.6.1 Pixel/Voxel Operations

Filtering

The following was taken from [36] (Bajaj2001).

For PDE based surface fairing or smoothing, there are several methods that have been proposed (see [37, 85, 104, 103]) recently. Desbrun et al in [104, 103] also use Laplacian, which is discretized as the umbrella operator in the spatial direction. In the time direction discretization, they propose to use the semi-implicit Euler method to obtain a stable numerical scheme. Clarenz et al in [85] generalize the Laplacian to the Laplace-Beltrami operator , and use linear finite elements to discretize the equation. In [37], the problem is reformulated for 2-dimensional Riemannian manifold embedded in aiming at smooth geometric surfaces and functions on surfaces simultaneously. The higher-order finite element space used is defined by the Loop's subdivision (box spline). One of the shortcomings of all these proposed methods that we address here is their non-adaptivity. All of them use uniform timesteps. Hence they quite often suffer from under-fairing or over-fairing problems.

The following was taken from [34] (BajajWu_noyear / aniso02).

The use of the Partial Differential Equation (PDE) and geometry-driven flows in image processing has become an interesting research topic in the past few years. We would like to refer [12, 255, 306] as good sources to the reader. Here we review only a few works related mostly to the method we employ in this paper. The simplest PDE-based image smoothing method is to apply a linear diffusion process described by the heat equation:

$$\partial_t \varphi - \mathrm{div}\nabla\varphi = 0 \tag{6.2}$$

In essence, this isotropic diffusion process is equivalent to Gaussian filtering, namely, the solution is

$$\varphi(t) = \begin{cases} \varphi_0 & (t=0) \\ K_{\sqrt{2t}} * \varphi_0 & (t>0) \end{cases} \tag{6.3}$$

(2) where $K_\sigma(.)$ denotes the Gaussian filter of width σ >0 (standard deviation) The image is considered as initial data of a suitable evolution problem. Time represents the scale parameter that leads from noisy fine scale to smoothed and enhanced coarse scale representations of the data. Linear diffusion does not only smooth noise, but also simultaneously blurs features such as edges. To avoid this drawback, Perona and Malik[230] propose a nonlinear diffusion method, which reduces the diffusivity at those locations having a larger likelihood to be edges.

This likelihood is measured by $|\nabla\varphi|$. The Perona-Malik filter is based on equation

$$\partial_t \varphi - \mathrm{div}(g(|\nabla\varphi|^2)\nabla\varphi) = 0 \tag{6.4}$$

and it uses diffusivity such as

$$g(s^2) = \frac{1}{1+s^2/\lambda^2} \tag{6.5}$$

or

$$g(s^2) = \exp(-\frac{s^2}{k^2}) \tag{6.6}$$

However, this nonlinear diffusion process is ill-posed in the sense that images close to each other are likely to diverge during the diffusion process [323]. To design a more robust model, Catté, Lions, Morel and Coll [76] replace the diffusivity $g(|\nabla\varphi|)$

of the Perona-Malik model by $g(|\nabla\varphi_\sigma|)$ with $\varphi_\sigma := K_\sigma * \varphi$. It should be noted that Catté's method still utilizes a scalar-valued diffusivity instead of a diffusion tensor. Weickert proposes a nonlinear anisotropic diffusion model as follows:

$$\partial_t \varphi - \mathrm{div}(D(|\nabla\varphi_\sigma|)\nabla\varphi) = 0 \tag{6.7}$$

The diffusion tensor D is chosen to gradient structure tensor, and the diffusion tensor is to inhibit diffusion across edges and encourage diffusion along edge [306].

The work directly related to our work is Preuβer and Rumpf's level set method for anisotropic geometric diffusion in 3D image processing [239].The core of their method is an evolution driven by anisotropic geometric diffusion of level surfaces. They construct an anisotropic diffusion tensor based on pre-filtered principal curvatures and principal directions of curvature of the level set. The diffusion tensor is constructed from a surface and expanded into 3D space by mapping normal space to zero.Â [243] give a connection between level set of 3D image and gradient structure tensor in (6.7), so this model is a special case of (6.7), and it nullifies the diffusion along the gradient direction. In addition, this model are not applicable for the local region where isosurface doesn't exist, for example, line structure or noisy regions are two special examples in 3D volume.

Contrast Enhancement

The following was taken from [332].

There have already been many techniques for enhancing image contrast. The most commonly seen methods include various contrast manipulations and histogram equalization [138, 238, 249]. Classic contrast manipulation [138, 238] is usually based on a globally defined stretching function (or called transfer function in the following) that is either linear or nonlinear. Histogram clipping might be needed before the pixel-by-pixel stretching. Traditionally histogram equalization is also a global technique in the sense that the enhancement is based on the equalization of the global histogram of the entire image. However, it is well recognized that using only global information is quite often not enough to achieve good contrast enhancement (for example, global approaches often cause an effect of intensity saturation).

To remedy this problem, some authors proposed localized (or adaptive) histogram equalization [138, 238, 234, 69, 272], which considers a local window for each individual pixel and computes the new intensity value based on the local histogram defined on the local window. This adaptive technique can usually improve the enhancement results but it is computationally intensive even though there are some fast implementations for updating the local histograms [161]. Furthermore, adaptive histogram equalization is a uniform local operator in the sense that all the pixels within the local window play an equally important role in determining the new value of the center pixel being considered. Sometimes, like Gaussian filter *vs*. uniformly averaging filter for image smoothing, a weighted contribution of all the pixels within the local window to the center pixel is more desired. In other words, to determine the new value for a pixel P, it is more desirable that pixels close to P should contribute more than other pixels that are farther from P.

A more recently developed technique is called *retinex model* [169, 170], in which the contribution of each pixel within the local window is weighted by computing the local average based on Gaussian function. As we will see in section 3, retinex model does not work well in some cases. A later version, called *multiscale retinex model* [169], gives better results but it is computationally more intensive. Another technique for contrast enhancement is based on wavelet decomposition and reconstruction and has been largely used for medical image enhancement especially for mammography image enhancement [202, 186].

Classification/Segmentation

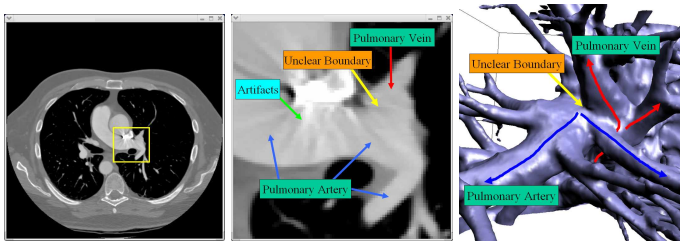
The following was taken from [226] (park2006artery).

Human vasculature reconstruction from three-dimensional (3D) computed tomography (CT) images of the thorax is a critical step for computer-aided diagnosis (CAD) in disease domains such as lung nodules [134], coronary artery disease [164], and pulmonary embolism (PE) [322]. It remains a challenging problem even though there have been many published approaches [177] [175].

Thoracic CT angiography (CTA) imaging is often performed for patients suspected of having pulmonary embolus (PE) that is defined as a thrombus (or a clot of blood) [258]. However, blood vessel extraction is often not enough for PE detection from

CTA, since the position of thrombi is always in the pulmonary trunk and/or the subsegmental arteries [151]. Differentiation of pulmonary arterial from venous trees significantly reduces false positive PE detection in pulmonary veins. While several authors have addressed the issue of artery-vein separation (AV separation) from magnetic-resonance angiography (MRA) [94] [273] [292] [216], Sluimer *et al.* mentions that AV separation from CTA is one of their primary future challenges [163].

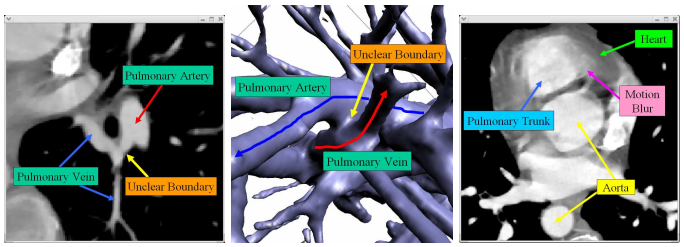
For AV separation from MRA, Bemmel *et al.* suggests a level-set framework [94]. In their method, the central arterial and venous axes are determined by using a supervised learning (SL) procedure. Their SL method computes a minimum-cost path between two (or more) user-defined points in the arterial and venous parts of the vasculature. The high probability voxels which become part of the center of a vascular structure are enhanced using a multiscale filter. Stefancik and Sonka show that a graph search algorithm coupled to a knowledge-based approach could be applied to AV separation [273]. Lei *et al.* utilized the principles and algorithms of fuzzy connected object delineation, for AV separation [292]. Bock *et al.* proposed correlation analysis between manually selected regions of interest and the acquired 3D MRA data sets, viewed as a time series [216]. Since diverse medical imaging modalities require different approaches [177], in this paper we focus on robust AV separation approach from 3D thoracic CTA images, knowing that the features of CTA images are quite different from MRA.



(a) CTA slice (b) Enlarged image (c) Iso-surfaces

Figure 2.19: Slice images of CT-angiography and Iso-surfaces: The yellow box of (a) is enlarged in (b) that shows the unclear boundary between the pulmonary artery and vein and the same region is visualized in (c) with iso-surfaces.

Fig. 2.19 (a,b) (image slices) and (c) (3D iso-surface) show some of the main problems that make it difficult to differentiate pulmonary arterial and venous trees from 3D thoracic CTA images. First, since the resolution of CTA images is limited and the artery and vein are very close to each other, there is an unclear boundary between them. The same problem is present in many other places even in a single data set. Fig. 2.20 shows another region that contains the similar problem. Second, the bright area in Fig. 2.19(b), which represents the vena cava, possesses many neighboring CTA artifacts. Since the artifacts are more distinct than the unclear boundary in Fig. 2.19(b), they are difficult to remove as unnecessary features, while identifying the unclear boundaries.



(a) CTA slice (b) Iso-surface (c) Motion Blur

Figure 2.20: There are many unclear boundaries in 3D CTA images. It makes it hard to separate pulmonary arteries and veins.

The unclear boundary problem is addressed by Bemmel *et al.* in [94]. In their method, voxels are labeled arterial or venous based on the arrival time of the respective level-set fronts. The front evolution is depend on three external forces based on the information of intensity, gradient, and vesselness. However, the labeling approach does not work properly for the problem region of Fig. 2.19(b), since the pulmonary artery is thicker than the vein, and the evolution of level-set fronts can be hampered by high gradients at the artifacts. Furthermore, since there are many unclear boundaries between pulmonary arteries and veins in 3D thoracic CTA images, all the problem regions should be skeletonized before using the level-set framework, which requires heavy user interactions. Kiraly *et al.* used branching angles to remove false branches for only arterial subtrees distal to a site

of PE [11]. Finally, Bülow *et al.* proposed an algorithm for pulmonary arterial tree extraction. It is based on the fact that pulmonary arteries accompany the bronchial tree [294]. The mislabeling of the algorithm increases as vessel radius decreases and it is not specifically evaluated.

Fig. 2.20(c) shows another problem in 3D CTA images. Since the heart keeps beating, when the images are taken, it causes motion blur and the heart boundaries are ambiguous. In this paper, we suggest AV separation methods that solve the problems of distinct artifacts, unclear boundaries between arteries and veins, motion blur effects.

Symmetry Detection

The following is taken from [334]. (YuBajaj2005)

An animal, plant, or bacterial virus, is a parasite on its host cells. Mature viruses, called virions, cannot "grow" in isolation, as they do not posses the requisite protein synthesis machinery. All virions must depend on their host cells to accomplish their biological processes and reproduce themselves, by which they "consume" the cells' nutritients, and disrupt the metabolism of the host cells to eventually kill the host cells [66]. Viruses or virions (we henceforth use this interchangeably) in their life cycle span can significantly influence and impact their local and global environment. For example, an infestation of the rice dwarf virus, a major pathogen of the rice plants in southeast Asia, would result in severe global economic consequences [343]. The herpes simplex virus type 1 (HSV-1), a human pathogen, is responsible for various diseases including cold sores, blindness, and encephalitis [308]. Understanding the molecular structure of virions and its inherent geometric complexity (ultrastructure) is the first step towards an understanding of the molecular processes of how these pathogens infect the host cells, and often necessary for devising means to suppress the consequences of viral infections.

Many studies have revealed that most if not all functions of biological units including viruses can be traced back to their structures and composition. Despite the large range of variations, most viruses consist of two fundamental components: the genomic nucleic acids (DNA/RNA) and the coat proteins. The coat proteins form a shell (capsid) and play two major roles: to protect the fragile DNA/RNA and to help the viruses recognize the host cells. The coat proteins are very important for a virus to live and reproduce itself, and much attention has been drawn to study the three dimensional (3D) structures of these proteins. X-ray diffraction has been so far the most powerful technique to determine the structures of virions (and many other individual proteins) at an atomic level. However this method requires crystalline forms of the purified virus and many viruses are resistant to the formation of crystals, large enough for the high-resolution image reconstruction through x-ray diffraction. Nuclear magnetic resonance (NMR) is another increasingly popular method used to determine the atomic structure of molecules. However, this method is limited to only relatively small molecules ($<$ 50K Dalton), which makes it difficult to study even the smallest virus particles.

To overcome the limitations of both x-ray crystallography and NMR methods, a non-crystallography reconstruction technique using cryo-electron microscopy (cryo-EM) has been providing a powerful tool in revealing the "full picture" of large macromolecular complexes at sub-nanometer resolutions (5 - 10Å) [49, 131, 297]. In the cryo-EM reconstruction pipeline, biological particles (viruses, ribosomes, etc.) are rapidly frozen in water to liquid nitrogen temperatures (about -195^0C). In this process the frozen water does not form ice crystals but a thin layer containing the virus particles to absorb and protect the native structure of particles from the high intensity electron radiation beams. However, the particle images taken with transmission electron microscopes have a large amount of noise and very low contrast between the vitreous ice and the biological particles, so that thousands (or more) of identical particles have to be purified, imaged and averaged in order to improve the signal-to-noise ratio for a reliable and reasonably high-resolution 3D reconstruction [153]. This technique is commonly known as *single particle reconstruction* [49, 131, 297]. The fundamental theory of 3D reconstruction from 2D projections is basically the same as the well-known CT technique as seen in medical imaging. However, in *single particle reconstruction*, the major difficulty arises from the unknown orientations of the particles that randomly "float" in the vitreous ice. Although there are ways to estimate the orientations of the particles, it is always beneficial if the particles under investigation demonstrate certain symmetries. For example, one projection of an icosahedral virus structure can readily reproduce identical projections in 59 other directions. Fortunately biological studies have continually revealed that most virions do adopt two basic types of symmetries: helical and icosahedral [66]. One representative of the helical viruses family is the Tobacco mosaic virus (TMV) [68]. The most frequent virus capsid shell symmetry that is observed, is that of an icosahedron [168, 343, 308, 49]. Therefore, there is sufficient demand to develop enhanced computational tools for structure determination and analysis of icosahedral virus maps.

The reconstruction of the 3D density map of a virus particle, though still a challenging problem, has already been studied for several years [49, 131, 297] and a number of reconstructed maps have been made accessible in public databases. With the increasing availability of 3D virus maps at increased resolutions, we are now facing new challenges to efficiently and accurately

explore and interpret the detailed structure. To this end, the segmentation of each individual subunit on the virus capsids, also plays a crucial role in the structure interpretation of a map, and shall be the focus of the present paper. Specifically, the segmentation of an icosahedral map includes two major steps: the detection of symmetry axes, and the segmentation of each asymmetric unit. In many cases, the segmented asymmetric units represent individual proteins and can be further analyzed (e.g., secondary structure identification) to reveal their quasi-atomic structures. Although the structural modelling is part of the full pipeline of virus map interpretation, it is outside the scope of the present paper.

Current efforts on the decomposition of an icosahedral map largely rely on manual work with assistance of graphical user interfaces [343, 168]. Manual segmentation is apparently tedious, time-consuming, and quite often subjective. Automatic segmentation in general is one of the fundamental but yet the hardest problems in image processing. A few papers, including the water-shed immersion method [303], the normalized graph cut and eigenvector analysis [128, 130], and the fast marching method [43], have attempted to automatically segment imaged specimens at the cell and molecular levels. However, to the best of our knowledge, no one has reported an automatic/semi-automatic method to completely segment any of the icosahedral virus maps based on their symmetries. To achieve this goal, we present in the following a variant of the traditional fast marching method [260]. We also present an efficient way to detect the symmetry axes of an icosahedral map, including the global (icosahedral) symmetry and the local symmetry axes. An automatic method for seed detection is discussed as well, and based on anisotropic vector diffusion.

Compression

The following is taken from [16] (Bajaj05).

Bio-molecular Complexes form the machinery responsible for most biological processes and are relevant to understanding many diseases such as cancer and metabolic disorders. Today, hybrid experimental approaches utilizing X-ray crystallography, cryo-electron microscopy (Cryo-EM), and molecular tomographic imaging, together with structural proteomics have led to a rapid increase in the number of three-dimensional (3D) structures and 3D electron density map data of bio molecules, in the Protein Data Bank (PDB) and at the European Bioinformatics Institute (EBI). Supercomputers across the world are also now routinely deployed for computation of associated properties of these bio-molecules such as volumetric force fields and interaction potentials, including terms derived from volumetric Poisson-Boltzmann electrostatics potentials, or Generalized Born solvation energy estimations.

While molecular visualization software has developed over the years, few tools today, if any, are capable of interactive visual exploration of bio-molecules and their associated properties. An important challenge in achieving enhanced visualization of large datasets with minimal interactive performance degradation is effective data processing and management, so that one can manipulate the large volumetric data in real time. Many large datasets cannot all fit into the main memory of most desktops. To do large data analysis and visualization efficiently and interactively, new algorithms need to be developed that process compressed data "in situ", using minimal additional memory space and time. Furthermore, interactive performance is dependent on rapid decompression of sub volumes, especially at random locations within the dataset.

Our approach to enabling real time interactive exploration of molecular maps and associated properties is to visualize directly from compressed representations of data with fast and random local decompression of this data. To achieve this goal, we introduce a new wavelet compressed volumetric representation of bio-molecules and associated properties, and address the tradeoff of excellent data compression, while maintaining rapid random and local decompression. These wavelet function spaces are custom built for rectilinear volumetric data, and are the span of a hierarchical basis (HB). Our HB wavelet representations of bio-molecules and associated properties preserves biological features at varying resolutions, while achieving a very high degree of compression, and yet allowing for efficient local decompression.

An important solution technique used in conjunction large data visualization algorithms is data compression ([23, 26, 31, 25, 24, 30]). Compression yields maximal bandwidth for transmission of data from disk to compute servers, and from servers to display clients. Additionally, data compression allows for maximal memory utilization, wherein algorithms work directly from compressed data in-core, and use minimal incremental decompression for processing. Another obvious use of data compression is the archival storage of input volumetric density maps, as well as output iso-surfaces and/or volume renderings. Among the many compression techniques developed, wavelets ([99, 276]) have become an integral part for modern archival image storage and transmission as exemplified by the JPEG 2000 standard ([285]). Moreover, extensions have been developed to the compression of geometric data and operator compression and stabilization ([71, 72, 327]).

Applications of wavelets to volume rendering and geometric manipulation at interactive speeds have been quite limited. A visualization-specific compression scheme is presented in ([25]), in which weights are assigned to voxels and combined with

compression algorithms such that information about important features is preserved. In all of these data compression schemes, the objective is to allow rapid random access to compressed data while keeping a very high compression ratio (25:1 to 50:1), resulting in real-time visualization of extremely large datasets. However, due to the discontinuity of the basis, this scheme leads to biologically incorrect artifacts. A new representation is sought. In this paper, we deal with efficient representation and manipulation of tensor product representations of volumes over a grid using a hierarchical basis.

Numerous molecular representations have been proposed for the computer visualization of the structure of proteins, nucleic acids, and their properties ([187]). All these are different visual representation of an underlying hierarchical skeletal model of the positions of atoms, bonds, chains, and residues in the molecule. Hence, structural models aim mainly at representing the primary, secondary, tertiary and quaternary geometric structures of the molecule. A basic representation for molecular shape is the union of balls ([90]), also known as the CPK model, where each ball represents an atom of the molecule. In the presence of a solvent, the radius of the solvent atom increases the radii of the balls. This is called *solvent accessible surface* (SAS) ([89]) since it reflects the accessibility for a solvent.

The *solvent contact surface* (SCS), also known as the Connolly and Lee-Richards surface ([89, 90, 188]), is computed by "rolling" a solvent atom probe over the union of balls. Formally, a point p belongs to the *solvent contact surface* $\mathscr{S}_c$ of a molecule M with respect to a solvent with atoms of radius r if there exists a ball B_1 of radius r that contains p in its boundary and whose interior does not intersect M, and there is no ball B_2 of radius r whose interior contains p but does not intersect M. The solvent contact surface can be triangulated ([253, 298]) for the purpose of visualization and other computations. Sanner and Olson ([253]) define a reduced surface for a molecule and give an algorithm to compute its triangulation in $n\log(n)$ time. ([298]) introduces the concept of *interface surface* and develops an algorithm for simplification of the polygonal mesh representation of molecular surfaces. The simplified mesh has the properties: (a) every point is within a user-specified distance ε from the original mesh, (b) it is topologically consistent (has the same genus) with the original mesh, (c) its vertices are a subset of the vertices of the input mesh.

Representations for molecular surface models defined above often use spheres or piecewise linear approximation as their boundaries ([4, 89, 212, 304, 254, 299]). Explicit representation of the boundary of the molecular surface using exact smooth trimmed NURBS (non-uniform rational B-splines) is presented in ([27]. In [28]), *Bajaj et al* introduces techniques for computing a molecular surface boundary representation as a set of NURBS patches and discusses maintaining such a NURBS representation under dynamic settings.

Other groups ([115, 193, 213, 244, 98]) represent molecular surfaces with expansions of spherical harmonic functions. This surface definition has fewer discontinuities than the hard sphere model and offers distinct advantages: it is realistic, and is computationally easier to manipulate a suitably defined electron density function ([116, 178]).

2.6.2 Alignment

2.6.3 Projections

Forward Radon Transform

Fourier Transform

2.6.4 Reconstruction

Inverse Radon Transform

The following is taken from [194] (Li2009Reconstruction).

In the past few years, cryo-electron microscope imaging techniques have established themselves as indispensable tools for determining the three dimensional(3D) structures of large macro- molecules and biological machineries. Different from X-ray crystallographic methods, Electron Microscopy (EM) does not require crystallization. Specimens are applied to a grid covered with holey carbon, and then is rapidly plunged into a cryogen-usually liquid ethane cooled by liquid nitrogen [132]. This data acquirement process can be very fast. Hence, these methods provide good opportunities to determine the 3D structures of large macromolecules that are either too large or too heterogeneous to be investigated by X-ray methods [297].

Cryo-electron microscope imaging techniques can be separated into several imaging modali- ties depending on the symmetry of the specimen and the data collection geometry. In this way, we have single particle analysis (multiple copies of a low symmetry unit), electron tomogra- phy (a single copy of a low symmetry unit), electron crystallography (multiple copies of a low symmetry unit symmetrically arranged in a lattice), helical particle analysis (it is a crystal in which the lattice is follows a

helix), and icosahedral particle analysis (multiple copies of a highly symmetrical object) [269]. In this paper, we will concentrate on single-particle analysis in which multiple copies of assumedly identical particles are imaged in different orientations. The orien- tations of the particles within the microscope can be completely random, or follow some special pattern (non-random or random conical tilt [291]). Of particular interest for this work is random conical tilt. It is an experimental technique in which the particles have a preferential interaction with the supporting grid. After tilting the grid within the microscope beam, this translates into a regular distribution of the projection directions along a circle in the projection sphere. In Fourier space this distribution is characterized by the lack of information within a cone aligned with the microscope beam. In this paper, we show that the newly developed algorithm has an advantage over existing algorithms in the case of a random distribution of projection directions as well as in the case of Random Conical Tilt [269].

As stated before, single-particle techniques make use of multiple occurrences of the single molecules in random orientations. These single molecules in solution have six degrees of free- doms: three translational ones X, Y, Z, and three rotational Euler Angles α, β; and γ. The image created in the electron microscope is a projection along the Z direction. Five parameters need to be determined for every particle image: the two in-plane translational parameters X and Y; the in-plane rotation γ and two out-of-plane rotations α and β. The proces of determining these five parameters is called three-dimensional alignment. The two main three-dimensional alignment techniques are projection matching [229, 240, 271, 171] and the common-line method [127], [296]. In this paper we will assume that this problem has already been solved by using any of the aforementioned methods [229, 240, 271, 171].

The next step is the 3D reconstruction process which is the problem addressed by the al- gorithm introduced in this paper. The usually used reconstruction algorithms are Altered back projection(FBP) [221], direct fourier reconstruction [221], and iterative methods including algebraic reconstruction technique(ART) [140], simultaneous iterative reconstructive technique(SIRT) [136], and simultaneous algebraic reconstruction technique(SART) [7]. In order to accelerate conver- gence of these algorithms block iterative technique have been proposed [207, 140].

The 3D reconstruction from the 2D projection images is a classical inverse problem. Theoret- ically, an object can be reconstructed uniquely from its projections when all of its projections are known. However, in practice, only limited number of projection images can be obtained. Hence, in most cases, the inverse problem is ill-posed. Some constraint conditions should be imposed to stabilize the algorithms. Projection onto convex sets(POCS) is such a technique which assume that the object f belongs to the intersection of some closed convex sets [325, 261]. Regularization is another technique to find a well-posed approximation solution. Classical Tikhonov regular- ization has been employed in tomography reconstruction [220]. In this paper, we propose a volume constraint algorithm which assumes the volume of the object would be a prior knowledge and keep unchanged during the imaging process.

Besides, we propose a new variational reconstruction method which employs the level set and L2 gradient flow technique with volume-preserving regularization. Our method is a series expansion method. Assume the object f has a mathematical expression. Instead of using pixel basis functions, we use cubic B spline basis functions. f can be expressed as a linear combination of these B spline basis functions. One of the advantages of using B spline basis functions is that we can obtain a C2 smooth object. And the local support property of B spline can accelerate the reconstruction process. We use finite element method (FEM) to solve the variational problem. To avoid solving large linear equations, orthogonalization of B spline basis functions will be employed.

Fourier Based Tomography

2.6.5 Bio-Imaging

molecules — electron microscopy

The following is taken from [21] (bajaj2006secondary).

Prior Work. There are relatively few published algorithms that detect α-helices and β-sheets of proteins from 3D Maps at coarse resolution (> 6 Å). Wen et al have devised an algorithm called *Helix Hunter* [166] for detecting α-helices in a low resolution map where they modeled the helices as cylinders and the density function of the cylinder is convolved with the original map to detect the peaks of the cross-correlation. The main disadvantage of this technique is that it searches exhaustively over the space of all rigid-body transformations and therefore is very slow. A similar approach for detection of β-sheets was adopted by Kong and Ma [179, 180] who modeled it as a disk like primitive and searched through the input map to find the

possible positions of the disk that yielded high cross-correlation. This work, due to its exhaustive search paradigm, is also extremely compute intensive. Recently, Yu and Bajaj [335] have developed a secondary structure elucidation algorithm based on the relative magnitudes of the eigenvalues of the structure tensor computed at various select groups of voxels.

Given a compact surface Σ smoothly embedded in $\mathbb{R}^3$, a distance function h_Σ can be assigned over $\mathbb{R}^3$ that assigns to each point its distance to Σ. $h_\Sigma : \mathbb{R}^3 \to \mathbb{R},\ x \mapsto \inf_{p\in\Sigma} \|x-p\|$ In applications, Σ is often known via a finite set of sample points P of Σ. Therefore it is quite natural to approximate the function h_Σ by the function $h_P : \mathbb{R}^3 \to \mathbb{R},\ x \mapsto \min_{p\in P} \|x-p\|$ which assigns to each point in $\mathbb{R}^3$ the distance to the nearest sample point in P. Distance functions have found use in surface reconstruction [15, 121, 78, 135], Medial axis approximation [80, 266], shape segmentation and feature analysis [106]. Recently, Goswami et al [141] have presented an algorithm to compute the unstable manifolds of the index 1 and index 2 saddle points of this distance function and demonstrated its use in detecting flat and tubular features of any shape.

cells with organelles —- serial section electron microscopy

tissues cells + extra-cellular space/filling —- confocal microscopy/micro-CT

organs — CT/MRI

systems — live cell imaging, phase-contrast and tagged MRI

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphics, and Mathematical Tables*. America Dover Publications, Inc., 1972. book.

[2] M. N. Ahmed, S. M. Yamany, A. A. Farag, and T. Moriarty. A bias field estimation and adaptive segmentation of MRI data using a modified Fuzzy C-Means algorithm. In *Proc. of 13th Int. Conf. on Computer Assisted Radiology and Surgery*, 1999.

[3] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, and T. Moriarty. A modified fuzzy c-means algorithm for bias field estimation and segmentation of mri data. *IEEE Transactions on Medical Imaging*, 21(3), 2002.

[4] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Applied Mathematics*, 71:5–22, 1996.

[5] K. Amaratunga and J. E. Castrillón-Candás. Surface wavelets: A multiresolution signal processing tool for 3d computational modeling. *International Journal of Numerical Methods in Engineering*, 52:239–271, 2001.

[6] N. Amenta, S. Choi, and R. Kolluri. The power crust, unions of balls, and the medial axis transform. *Computational Geometry: Theory and Applications*, 19(2-3):127–153, 2001.

[7] A. H. Andersen and A. C. Kak. *Simultaneous algebraic reconstruction technique (sart): A superior implementation of the art algorithm*. Ultrason Imaging., 6(1):81-94, 1984.

[8] C. Anderson and M. D. Dahleh. Rapid computation of the discrete fourier transform. *SIAM J. Sci. Computing*, 17:913–919, Mar. 1996.

[9] C. W. Anderson and S. Crawford-Hines. Fast Generation of NURBS Surfaces from Polygonal Mesh Models of Human Anatomy. In *Technical Report CS-99-101, Colorado State University*, 2000.

[10] R. Araiza, M. Averill, G. Keller, S. Starks, and C. Bajaj. 3D image registration using Fast Fourier Transform, with potential applications to geoinformatics and bioinformatics. In *Proc. Int. Conf. on Inform. Proc. and Management of Uncertainty in Knowledge-Based Systems IPMU06*, pages 817–824, 2006.

[11] D. P. N. Atilla P. Kiraly, Eric Pichon and C. L. Novak. Analysis of arterial sub-trees affected by pulmonary emboli. In *Proceedings of SPIE (Medical Imaging)*, volume 5370, pages 1720–1729. The International Society for Optical Engineering, 2004.

[12] G. Aubert and P. Kornprobst. Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations, volume 147 of Applied Mathematical Sciences. *Springer-Verlag, New York*, 137:138, 2002.

[13] B. Baxter and G. Roussos. A New Error Estimate of the Fast Gauss Transform. *SIAM J. Sci. Comput.*, 24(1):257–259, 2002.

[14] C. Bajaj. Volrover: An interactive volume visualization tool.

[15] C. Bajaj, F. Bernardini, and G. Xu. Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans. *Proc. of the ACM Siggraph Computer Graphics, SIGGRAPH 95*, 29(3):109–118, 1995.

[16] C. Bajaj, J. Castrillon-Candas, V. Siddavanahalli, and Z. Xu. Compressed representations of macromolecular structures and properties. *Structure*, 13:463–471, 2005.

[17] C. Bajaj, J. Castrillon-Candas, V. Siddavanahalli, and Z. Xu. Compressed Representations of Macromolecular Structures and Properties Structure. *Discrete Applied Mathematics*, 13(3):463–471, 2005.

[18] C. Bajaj, J. Chen, and G. Xu. Modeling with cubic A-patches. *ACM Transactions on Graphics*, 14:103–133, 1995.

[19] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Interactive visual exploration of large flexible multi-component molecular complexes. In *Proc. of the Annual IEEE Visualization Conference*, pages 243–250, Oct. 2004.

[20] C. Bajaj, A. Gillette, and S. Goswami. Topology based selection and curation of level sets. In *TopoInVis 2007*, 2007.

[21] C. Bajaj and S. Goswami. Secondary and Tertiary Structural Fold Elucidation from 3D EM Maps of Macromolecules. *Computer Vision, Graphics and Image Processing*, pages 264–275, 2006.

[22] C. Bajaj and S. Goswami. Status report of modeling cardiac anatomy from patient-specific imaging data. Technical report, Institute of Computational Engineering and Sciences, University of Texas at Austin, 2008.

[23] C. Bajaj, I. Ihm, and S. Park. Compression-based 3D texture mapping for real-time rendering. *Graphical Models*, 62(6):391–410, 2000.

[24] C. Bajaj, I. Ihm, and S. Park. 3D RGB image compression for interactive applications. *ACM Transactions on Graphics*, 20(1):10–38, 2001.

[25] C. Bajaj, I. Ihm, and S. Park. Visualization-specific compression of large volume data. In *Proc. of Pacific Graphics*, pages 212–222, Tokyo, Japan, Oct. 2001.

[26] C. Bajaj, I. Ihm, S. Park, and D. Song. Compression-based Ray Casting of Very Large Volume Data in Distributed Environments. In *HPC-Asia 2000*, pages 720–725, Beijing, China, May 2000.

[27] C. Bajaj, H. Lee, R. Merkert, and V. Pascucci. Nurbs based b-rep models for macromolecules and their properties. *Proceedings of the 4th Symposium on Solid Modeling and Applications (ACM Press)*, pages 217–228, 1997.

[28] C. Bajaj, V. Pascucci, R. Holt, and A. Netravali. Dynamic Maintenance and Visualization of Molecular Surfaces. *Fourth issue in the special series of Discrete Applied Mathematics on Computational Molecular Biology*, 127(1):23–51, 2001.

[29] C. Bajaj, V. Pascucci, and D. Schikore. Fast Isocontouring for Improved Interactivity. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 39–46, San Francisco, CA, 1996.

[30] C. Bajaj, V. Pascucci, and G. Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Proceedings of IEEE Visualization Conference*, pages 307–316, San Francisco, CA, Oct. 1999.

[31] C. Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *IEEE Data Compression Conference*, pages 247–256, 1999.

[32] C. Bajaj and V. Siddavanahalli. Fast error-bounded surfaces and derivatives computation for volumetric particle data. ICES Technical Report TR-06-06, 2006.

[33] C. Bajaj, B. Sohn, and A. Shamir. *Isocontour Tracking in Time-Varying Data*. Technical report, Department of Computer Sciences, 2001.

[34] C. Bajaj and Q. Wu. Volmetric anisotropic diffusion. Technical report, Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin.

[35] C. Bajaj, Q. Wu, and G. Xu. Level set based volumetric anisotropic diffusion. In *ICES Technical Report 301, The Univ. of Texas at Austin*, 2003.

[36] C. Bajaj and G. Xu. Adaptive surface fairing by geometric diffusion. In *Information Visualisation, International Symposium of Computer Aided Geometric Design, IV2001-CAGD, England*, pages 731–737. IEEE Computer Society, 2001. inproceedings.

[37] C. Bajaj and G. Xu. Anisotropic diffusion of noisy surfaces and noisy functions on surfaces. In *Proceedings of Information Visualization 2001 Cmputer Aided Geometric Design*, 2001.

[38] C. Bajaj and G. Xu. Anisotropic diffusion of surfaces and functions on surfaces. *ACM Transaction on Graphics*, 22(1):4–32, 2003. article.

[39] C. Bajaj, G. Xu, R. Holt, and A. Netravali. Hierarchical multiresolution reconstruction of shell surfaces. 19:89–112, 2002.

[40] C. Bajaj, G. Xu, and Q. Zhang. A Higher Order Level Set Method with Applications to Smooth Surface Constructions. ICES Report 06-18. *Institute for Computational Engineering and Sciences, The University of Texas at Austin*, 2006.

[41] C. Bajaj, G. Xu, and Q. Zhang. A fast variational method for the construction of resolution adaptive C2-smooth molecular surfaces. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1684–1690, 2009.

[42] C. Bajaj, G. Xu, and X. Zhang. Smooth surface constructions via a higher-order level-set method. In *Pro. CAD/GRAPHICS 2007*, page to appear, 2007.

[43] C. Bajaj, Z. Yu, and M. Auer. Volumetric feature extraction and visualization of tomographic molecular imaging. *Journal of Structural Biology*, 144(1-2):132–143, Oct. 2003.

[44] C. L. Bajaj, V. Pascucci, and D. R. Schikore. The contour spectrum. In R. Yagel and H. Hagen, editors, *Proceeding Visualization '97*, pages 167–173, Phoenix, AZ, Oct. 1997. IEEE Computer Society & ACM SIGGRAPH.

[45] C. L. Bajaj, G. Xu, and Q. Zhang. High-order level-set method and its application in biomolecular surface construction. *J. Comput. Sci & Technol.*, 23(6):1026–1036, 2008.

[46] R. Bajcsy and S. Kovacic. Multiresolution elastic matching. *Computer Vision Graphics and Image Processing*, 46(1-21):1–21, 1989.

[47] M. Baker, Z. Yu, W. Chiu, and C. Bajaj. Automated segmentation of molecular subunits in electron cryomicroscopy density maps. *Journal of Structural Biology, accepted*, 2006. TY - JOUR.

[48] N. Baker, D. Sept, S. Joseph, M. Holst, and J. McCammon. Electrostatics of Nanosystems: Application to Microtubules and The Ribosome. *Proc. Natl. Acad. Sci.*, 98:10037–10041, 2001.

[49] T. Baker, N. Olson, and S. Fuller. Adding the third dimension to virus life cycles: three-dimensional reconstruction of icosahedral viruses from cryo-electron micrographs. *Microbio. and Mol. Bio. Reviews*, 63(4):862–922, 1999. TY - JOUR.

[50] D. Barash. Bilateral filtering and anisotropic diffusion: Towards a unified viewpoint, Hewlett-Packard Laboratories Technical Report. Technical report, HPL-2000-18, 2000.

[51] D. Barash. A fundamental relationship between bilateral filtering, adaptive smoothing and the nonlinear diffusion equation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(6):844–847, 2002.

[52] R. Beatson and L. Greengard. A short course on fast multipole methods. In M. Ainsworth, J. Levesly, W. Light, and M. Marietta, editors, *Wavelets, Multilevel Methods, and Elliptic PDE's*. Oxford Univ Press, 1997.

[53] A. Bennett, J. Liu, D. V. Ryk, D. Bliss, J. Arthos, R. M. Henderson, and S. Subramaniam. Cryoelectron tomographic analysis of an hiv-neutralizing protein and its complex with native viral gp120. *The Journal of Biological Chemistry*, 282, 2007.

[54] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res.*, 28:235 – 242, 2000.

[55] G. Beylkin. On the fast fourier transform of functions with singularities. *Appl. Comput. Harmon. Anal*, 2:363–381, Mar. 1995.

[56] J. Bezdek. A convergence theorem for the fuzzy ISODATA clustering algorithm. *IEEE Trans. Pattern Anal. Machine Intell*, 2(1):1–8, Jan. 1980.

[57] G. E. Blelloch. Prefix sums and their applications. Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, Nov. 1990.

[58] J. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[59] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.

[60] G. Borgefors, I. Nystrom, and G. D. Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7), 1999.

[61] S. Bouix, K. Siddiqi, and A. Tannenbaum. Flux driven fly throughs. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 449–454, 2003.

[62] S. Boys. Electronic wave functions. i, a general method of calculation for the stationary states of any molecular system. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, 200(1063):542–554, 1950.

[63] D. Braess. Towards algebraic multigrid for elliptic problems of second order. *Computing*, 55:379–393, 1995.

[64] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, 1992.

[65] P. Brown, G. Byrne, and A. Hindmarsh. VODE: A variable-coefficient ode solver. *SIAM Journal on Scientific Computation*, 10:1038–1057, 1989.

[66] A. Cann. *Principles of Molecular Virology (2nd edition)*. Academic Press, ISBN: 0121585328, 1997.

[67] J. Carazo, G. Herman, C. Sorzano, and R. Marabini. *Algorithms for three dimensional reconstruction from imperfect projection data provided by electron microscopy*. 2006.

[68] B. Carragher, D. Fellmann, F. Guerra, R. Milligan, F. Mouche, J. Pulokas, B. Sheehan, J. Quispe, C. Suloway, Y. Zhu, and C. Potter. Rapid, routine structure determination of macromolecular assemblies using electron microscopy: current progress and further challenges. *J. Synchrotron Rad.*, 11:83–85, 2004.

[69] V. Caselles, J. Lisani, J. Morel, and G. Sapiro. Shape preserving local histogram modification. *IEEE Trans. Image Processing*, 8(2):220–230, 1998. TY - JOUR.

[70] D. L. D. Caspar and A. Klug. Physical principles in the construction of regular viruses. *Cold Spring Harbor Symp.*

Quant. Biol., 27:1, 1962.

[71] J. Castrillon-Candas. *Spatially Adaptive Multiwavelet Representations on Unstructured Grids with Applications to Multidimensional Computational Modeling*. Ph.D. thesis, MIT, 2001. TY - BOOK.

[72] J. Castrillon-Candas and K. Amaratunga. Spatially adapted multiwavelets and sparse representation of integral operators on general geometries. *SIAM Journal on Scientific Computing*, 24(5):1530–1566, 2003. TY - JOUR.

[73] J. Castrillon-Candas, C. Bajaj, and V. Siddavanahalli. An adaptive irregularly spaced fourier method for protein-protein docking. Technical Report TR-05-34, The University of Texas at Austin, Department of Computer Sciences, July 2005.

[74] E. D. Castro and C. Morandi. Registartion of translated and rotated images using finite fourier transforms. *IEEE Transactions on Patten Analysis and Machine Intelligence*, 9(5):700–703, 1986.

[75] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological surfaces. *Computer-Aided Design*, 10(6):350–355, 1978.

[76] F. Catté, P. Lions, J. Morel, and T. Coll. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM J. Numer. Anal.*, 29(1):182–193, 1992. article.

[77] C.Bajaj and G. Xu. Smooth shell construction with mixed prism fat surfaces. *Geometric Modeling, Springer Verlag, Computing Supplementum*, 14:19–36, 2001.

[78] R. Chaine. A geometric convection approach of 3-d reconstruction. In *Proc. Eurographics Sympos. on Geometry Processing*, pages 218–229, 2003.

[79] T. F. Chan and L. A. Vese. A level set algorithm for minimizing the Mumford-Shah functional in image processing. In *IEEE Workshop on Variational and Level Set Methods*, pages 161–168, 2001.

[80] F. Chazal and A. Lieutier. Stability and homotopy of a subset of the medial axis. In *Proc. 9th ACM Sympos. Solid Modeling and Applications*, pages 243–248, 2004.

[81] F. Chazal and A. Lieutier. The λ-medial axis. *Graphical Models*, 67(4):304–331, 2005.

[82] S.-W. Cheng, Y. Wang, and Z. Wu. Provable dimension detection using principal component analysis. In *Proc. Sympos. on Comput. Geom.*, pages 208–217, 2005.

[83] G. E. Christensen, R. D. Rabbitt, and M. I. Miller. Deformable templates using large deformation kinematics. *IEEE Transaction on Image Processing*, 5(10):1435–1447, 1996.

[84] D. Chung and G. Sapiro. Segmentation-free skeletonization of gray-scale images via pde's. In *Proc. Int'l Conf. Image Processing*, pages 927–930, 2000.

[85] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *VIS '00: Proceedings of the Conference on Visualization*, pages 397–405, Los Alamitos, CA, USA, 2000. inproceedings.

[86] U. Clarenz, M. Droske, and M. Rumpf. Towards fast non-rigid registration. *Proceedings of the AMS*, 313:67–84, 2002.

[87] Cocone. Tight Cocone Software for surface reconstruction and medial axis approximation. *http://www.cse.ohio-state.edu/∼tamaldey/cocone.html*.

[88] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. In *Proc. SIGGRAPH*, pages 905–914, 2004.

[89] M. L. Connolly. Analytical molecular surface calculation. 16:548–558, 1983.

[90] M. L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221:709 – 713, 1983.

[91] T. H. Cormen, L. C. E., R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw-Hill, Second Edition, 2001.

[92] N. Cornea, D. Silver, and P. Min. Curve skeleton applications. In *IEEE Visualization*, pages 95–102, 2005.

[93] N. Cornea, D. Silver, X. Yuan, and R. Balasubramaniam. Computing hierarchical curveskeletons of 3D objects. *The Visual Computer*, 21(11):945–955, 2005.

[94] M. A. V. Cornelis M. van Bemmel, Luuk J. Spreeuwers and W. J. Niessen. Level-set-based arter-vein separation in blood pool agent ce-mr angiograms. *IEEE Transactions on Medical Imaging*, 22(10):1224–1234, 2003.

[95] K. D. Costa, P. J. Hunter, J. S. Wayne, L. K. Waldmann, J. M. Guccione, and A. D. McCulloch. A three-dimensional finite element method for large elastic deformations of ventricular myocardium: Ii - prolate spheroidal coordinates. *J. Biomedical Engineering*, 118(4):464–472, Nov. 1996.

[96] L. Costa. Multidimensional scale space shape analysis. In *IWSNHC3DI*, pages 214–217, 1999.

[97] R. A. Crowther, D. J. DeRosier, and A. Klug. The reconstruction of a three-dimensional structure from projections and its application to electron microscopy. In *Proceedings of the Royal Society of London. Series B, Biological Sciences*, volume 182, pages 89–102, 1972.

[98] G. K. D Ritchie. Protein docking using spherical polar fourier correlations. *Proteins: Struct. Funct. Genet*, 39:178–194, 2000. [PubMed: 10737939].

[99] Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, 41:909–996, 1988.

[100] J. de Munck. A linear discretization of the volume conductor boundary integral equation using analytically integrated elements. *IEEE Trans Biomed. Eng.*, 39(9):986–990, Sept. 1992.

[101] R. Deriche. Fast algorithm for low-level vision. *IEEE Trans. on Pattern Recognition and Machine Intelligence*, 12(1):78–87, 1990. TY - JOUR.

[102] S. Derrode and F. Ghorbel. Shape analysis and symmetry detection in gray-level objects using the analytical Fourier-Mellin representation. *Signal Processing*, 84(1):25–39, 2004.

[103] M. Desbrun, M. Meyer, P. Schr
"oder, and A. Barr. Discrete differential-geometry operators in nD. *preprint, the Caltech Multi-Res Modeling Group*. http://www.multires.caltech.edu/pubs/.

[104] M. Desbrun, M. Meyer, P. Schr
"oder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, page 324. ACM Press/Addison-Wesley Publishing Co., 1999.

[105] T. K. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge Monographs on Applied and Computational Mathematics, 2006.

[106] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In F. Dehne, J.-R. Sack, and M. Smid, editors, *Proc. Workshop Algorithms Data Strucutres (WADS 03)*, LNCS 2748, pages 25–36, Berlin, Germany, 2003.

[107] T. K. Dey, J. Giesen, S. Goswami, and W. Zhao. Shape dimension and approximation from samples. *Discrete and Computaional Geometry*, 29:419–434, 2003.

[108] T. K. Dey and S. Goswami. Tight cocone: A water-tight surface reconstructor. In *Proc. 8th ACM Sympos. Solid Modeling and Applications*, pages 127–134, 2003.

[109] T. K. Dey and J. Sun. Defining and computing curve-skeletons with medial geodesic functions. In *Sympos. Geom. Processing*, pages 143–152, 2006.

[110] T. K. Dey and W. Zhao. Approximate medial axis as a Voronoi subcomplex. *Computer Aided Design*, 36(2):195–202, 2003.

[111] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, and J. Hart. Spectral surface quadrangulation. *ACM Transactions on Graphics*, 25(3):1057–1066, 2006.

[112] D. Donoho and I. Johnson. Ideal spatial adaptation via wavelet shrinkage. *Biometrika*, 81:425–455, 1994.

[113] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *Proceedings on Interactive Techniques in Computer Aided Design*, pages 157–165, 1978.

[114] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[115] B. Duncan and A. Olson. Approximation and characterization of molecular surfaces. *Biopolymers*, 33:219–229, 1993.

[116] B. S. Duncan and A. J. Olson. Shape analysis of molecular surfaces. *Biopolymers*, 33:231–238, 1993.

[117] F. Durand and J. Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM Conference on Computer Graphics (SIGGRAPH)*, pages 257–266, 2002.

[118] A. Dutt and V. Rokhlin. Fast fourier transform for nonequispaced data ii. *Appl. Comput. Harmon. Anal.*, 2:85–100, 1995.

[119] M. Eck and H. Hoppe. Automatic reconstruction of B-Spline surfaces of arbitrary topological type. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 325–334. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[120] I. Eckstein, A. A. Joshi, C. J. Kuo, R. Leahy, and M. Desbrun. Generalized surface flows for deformable registration and cortical matching. In *MICCAI*, page to appear, 2007.

[121] H. Edelsbrunner. Surface reconstruction by wrapping finite point sets in space. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Ricky Pollack and Eli Goodman Festschrift*, pages 379–404. Springer-Verlag, 2002.

[122] M. Elad. On the bilateral filter and ways to improve it. *IEEE Transactions On Image Processing*, 11(10):1141–1151, 2002.

[123] B. Elbel. Fast fourier transform for non equispaced data. *Approximation theory, C.K. Chui and L.L. Schumaker (eds.), Vanderbilt University Press*, 1998.

[124] R. Ellis. Macromolecular crowding: obvious but underappreciated. *Trends Biochem. Sci.*, 26(10):597–604, 2001. TY - JOUR.

[125] L. C. Evans and R. F. Gariepy. *Measure Theory and Fine Properties of Functions*. CRC Press, 1992.

[126] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide (5th ed.)*. Morgan-Kaufmann, 2002.
[127] R. A. C. P. for three-dimensional recnostruction of spherical viruses by fourier sythesis from electron micrographs. Phil. Trans. R. *Soc*. Lond., B(261):221-230, 1971.
[128] A. Frangakis and F. Forster. Computational exploration of structural information from cryo-electron tomograms. *Current Opinion in Structural Biology*, 14(3):325–331, 2004.
[129] A. Frangakis and R. Hegerl. Noise reduction in electron tomographic reconstructions using nonlinear anisotropic diffusion. *J. Struct. Biol.*, 135:239–250, 2001.
[130] A. Frangakis and R. Hegerl. Segmentation of two- and three-dimensional data from electron microscopy using eigenvector analysis. *Journal of Structural Biology*, 138(1-2):105–113, 2002. TY - JOUR.
[131] J. Frank. *Three-Dimensional Electron Microscope of Macromolecular Assemblies*. San Diego: Academic Press, 1996. TY - BOOK.
[132] J. Frank. *Three-Dimensional Electron Microscopy of Macromolecular Assemblies*. Oxford, New York, 2006.
[133] J. Frank, M. Radermacher, P. Penczek, J. Zhu, Y. Li, M. Ladjadj, and A. Leith. SPIDER and WEB: processing and visualization of images in 3D electron microscopy and related fields. *Journal of Structural Biology*, 116:190–199, 1996.
[134] I. . C. W. Gady Agam, Samuel G. Armato. Vessel tree reconstruction in thoracic ct scans with application to nodule detection. *IEEE Transaction on Medical Imaging*, 24(4):486–499, 2005.
[135] J. Giesen and M. John. The flow complex: a data structure for geometric modeling, 2003. TY - CONF The flow complex: a data structure for geometric modeling.
[136] P. Gilbert. *Iterative methods for the three-dimensional reconstruction of an object from projections*. Journal of Theoretical Biology, 36(1):105-117, 1972.
[137] G. Gloub and C. Van Loan. Matrix Computations, Baltimore, MD: John Hopkins Univl, 1989.
[138] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, Third Edition, 2007.
[139] S. S. Gopal and T. J. Hebert. Maximum likelihood pixel labeling using a spatially variant finite. *IEEE Transaction on Nuclear Science*, 44(4):1578–1582, Aug. 1999.
[140] R. Gordon, R. Bender, G. T. H. A. reconstruction techniques (art) for three-dimensional electron microscopy, and x-ray photography. J. *Theoretical Biology*, 29:471–481, 1970.
[141] S. Goswami, T. Dey, and C. Bajaj. Identifying flat and tubular regions of a shape by unstable manifolds. *Proceedings of the 2006 ACM symposium on Solid and Physical Modeling*, 2006.
[142] S. Goswami, A. Gillette, and C. Bajaj. Efficient Delaunay mesh generation from sampled scalar function. In *Proc. 16th Int. Meshing Roundtable*, page to appear, 2007.
[143] J. Grant and B. Pickup. A gaussian description of molecular shapes. *J. Phys. Chem.*, 99:3503–3510, 1995.
[144] J. M. Grimes, J. N. Burroughs, P. Gouet, J. M. Diprose, and R. Malby. The atomic structure of the bluetongue virus core. *Nature*, 395:470, 1998.
[145] H. N. Gursoy. Tetrahedral Finite Element Mesh Generation from NURBS Solid Models. *Engineering with Computers*, 12(19):211–223, 1996.
[146] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer Verlag, Berlin, Heidelberg, New York, Tokyo, 1985.
[147] A. B. Hamza and H. Krim. Image denoising: A nonlinear robust statistical approach. *IEEE Transactions on Signal Processing*, 49(12):3045–3054, 2001.
[148] A. B. Hamza, P. Luque, J. Martinez, and R. Roman. Removing noise and preserving details with relaxed median filters. *Journal of Mathematical Imaging and Vision*, 11(2):161–177, 1999.
[149] M. Harlow, D. Ress, A. Stoschek, R. Marshall, and U. McMahan. The architecture of active zone material at the frog's neuromuscular junction. *Nature*, 409:479 – 484, 2001. TY - JOUR.
[150] M. Harris, S. Sengupta, and J. Owens. Parallel prefix sum (scan) with CUDA. *GPU Gems*, 3, 2007.
[151] I. Hartmann and M. Prokop. Spiral ct in the diagnosis of acute pulmonary embolism. *Medica Mundi*, 46(3):2–11, 2002.
[152] M. S. Hassouna and A. A. Farag. Robust centerline extraction framework using level sets. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 458–465, 2005.
[153] R. Henderson. The potential and limitations of neutrons, electrons, and x-rays for atomic resolution microscopy of unstained biological macromolecules. *Q. Rev. Biophys.*, 28:171–193, 1995.
[154] G. Herman. *Image Reconstructions from Projections: The Fundamentals of Computerized Tomography*. Computer Science and Applied Mathematics, 1980.
[155] G. Herman and R. Davidi. Image reconstruction from a small number of projections. *Inverse Problems*, 24, 2008.
[156] D. Hessler, S. Young, and M. Ellisman. A flexible environment for the visualization of three-dimensional biological structures. *Journal of Structural Biology*, 116(1):113–119, 1996. TY - JOUR.

[157] B. Hille. *Ionic Channels of Excitable Membranes.* Sinauer Associates, Inc., Sunderland, MA, Second edition, 1992.

[158] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117:500–544, 1952.

[159] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. 194:4135–4195, 2005.

[160] P. Hunter, A. McCulloch, P. Nielsen, and B. Smaill. A finite element model of passive ventricular mechanics. *ASMS BED*, 9:387–397, 1988.

[161] H. Huttunen and O. Yli-Harja. Fast algorithm for updating the local histogram of multidimensional signals. In *Proc. 1998 IEEE International Conference on Image Processing*, 1998.

[162] A. E. K. Ingmar Bitter and M. Sato. Penalized-distance volumetric skeleton algorithm. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):195–206, 2001.

[163] M. P. Ingrid Sluimer, Arnold Schilham and B. van Ginneken. Comptuer analysis of computed tomography scans of the lung: Survey. *IEEE Transactions on Medical Imaging*, 25(4):385–405, 2006.

[164] F. W. X. Y. L.-M. A. M. S. Jamshid Dehmeshki, Xujiong Ye and S. Qanadli. An accurate and reproducible scheme for quantification of coronary artery calcification in ct scans. In *Proceedings of the 26th Annual International Conference of IEEE EMBS*, pages 1918–1921. IEEE, The International Society for Optical Engineering, Sept. 2004.

[165] J. Jang and K. Hong. A pseudo-distance map for the segmentation-free skeletonization of gray-scale images. In *Proc. Int'l Conf. Computer Vision*, pages 18–23, 2001.

[166] W. Jiang, M. Baker, S. Ludtke, and W. Chiu. Bridging the information gap: computational tools for intermediate resolution structure interpretation. *J. of Mol. Bio.*, 308:1033–1044, 2001. TY - JOUR.

[167] W. Jiang, M. Baker, Q. Wu, C. Bajaj, and W. Chiu. Applications of bilateral denoising filter in biological electron microscopy. *Journal of Structural Biology*, 144(1-2):114–122, Oct. 2003.

[168] W. Jiang, Z. Li, M. L. Baker, P. E. Prevelige, and W. Chiu. Coat protein fold and maturation transition of bacteriophage P22 seen at subnanometer resolution. *Nature Structural Biology*, 10(2):131–135, 2003.

[169] D. Jobson, Z. Rahman, and G. Woodell. A multiscale retinex for bridging the gap between color images and the human observation of scenes. *IEEE Trans. Image Processing*, 6(7):965–976, 1997. TY - JOUR.

[170] D. Jobson, Z. Rahman, and G. Woodell. Properties and performance of a center/surround retinex. *IEEE Trans. Image Processing*, 6(3):451–462, 1997. TY - JOUR.

[171] S. Jonic, C. O. S. Sorzano, P. Thevenaz, E.-B. C. S. D. Carlo, and M. Unser. *Spline-based image-to-volume registration for three-dimensional electron*. Ultramicroscopy, 103/104:303-317, 2005.

[172] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. In *SIGGRAPH*, pages 339–346, 2002.

[173] J. Kacur and K. Mikula. Solution of nonlinear diffusion appearing in image smoothing and edge detection. *Appl. Numer.Math.*, 58:603–611, 1991.

[174] A. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988.

[175] P. F. Katja Bühler and A. L. Cruz. Geometric methods for vessel visualization and quantification- a survey. In H. M. G. Brunnett, B. Hamann, editor, *Geometric Modelling for Scientific Visualization*, pages 1–24. Springer, 2003.

[176] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Proceedings of 1998 Symposium on Volume Visualization*, pages 79–86. 1998.

[177] C. Kirbas and F. Quek. A review of vessel extraction techniques and algorithms. *ACM Computing Surveys (CSUR)*, 36(2):81–121, 2004.

[178] A. Koide, A. Doi, and K. Kajioka. Polyhedral approximation approach to molecular orbital graphics. *Journal of Molecular Graphics*, 4:149–155, 1986.

[179] Y. Kong and J. Ma. A structural-informatics approach for mining β- sheets: locating sheets in intermediate-resolution density maps. *J. of Mol. Bio*, 332:399–413, 2003.

[180] Y. Kong, X.Zhang, T.S.Baker, and J. Ma. A structural-informatics approach for tracing β-sheets: building pseudo-Ca traces for β-strands in intermediate-resolution density maps. *J. of Mol. Bio.*, 339:117–130, 2004.

[181] J. Kremer, D. Mastronarde, and J. McIntosh. Computer visualization of three-dimensional image data using imod. *J Struct Biol*, 116:71–76, 1996. TY - JOUR.

[182] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *Proceedings of SIGGRAPH*, pages 313–324, 1996.

[183] C. D. Kuglin and D. C. Hines. The phase correlation image alignment method. In *Proc. IEEE International Conference on Cybrnetics and Society*, pages 163–165, 1975.

[184] D. Laidlaw. *Geometric Model Extraction from Magnetic Resonance Volume Data (PhD thesis)*. PhD thesis, CalTech

Univ., 1995.

[185] D. H. Laidlaw, K. W. Fleischer, and A. H. Barr. Bayesian mixture classification of mri data for geometric modeling and visualization. *a poster presented at the First International Workshop on Statistical Mixture Modeling, Aussois, France*, 1995.

[186] A. Laine, S. Schuler, J. Fan, and W. Huda. Mammographic feature enhancement by multiscale analysis. *IEEE Trans. Medical Imaging*, 13(4):725–738, 1994. TY - JOUR.

[187] A. Leach. *Molecular Modelling: Principles and Applications*. Prentice Hall, Harlow, UK, 2 edition, 2001.

[188] B. Lee and F. Richards. The Interpretation of Protein Structures: Estimation of Static Accessibility. *Journal Molecular Biology*, 55:379–400, 1971.

[189] S. Lee, M. E. Sowa, Y. Watanabe, P. B. Sigler, W. Chiu, M. Yoshida, and F. T. Tsai. The structure of clpb, a molecular chaperone that rescues proteins from an aggregated state. *Cell*, 115(2):229–40, 2003. 22929951 0092-8674 Journal Article.

[190] A. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. A streaming narrow-band algorithm: interactive computation and visualization of level sets. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):422–433, 2004.

[191] T. M. Lehmann. A two stage algorithm for model-based registration of medical images. In *Proceedings of the International Conference on Pattern Recognition ICPR'98*, pages 344–352, 1998.

[192] J. Lei, S. Liu, Z. Li, H. I. Schlaberg, and M. Sun. An image reconstruction algorithm based on the regularized total least squares method for electrical capacitance tomography. *Flow Measurement and Instrumentation*, 2008.

[193] S. Leicester, J. Finney, and R. Bywater. Description of molecular surface shape using fourier descriptors. *J. Mol. Graphics*, 6:104–108, 1988.

[194] M. Li, G. Xu, C. O. S. Sorzano, and C. Bajaj. Electric-potential reconstructions of single particles using l2-gradient flows. Technical report, 2009.

[195] Y. Li, A. Leith, and J. Frank. Tinkerbell-a tool for interactive segmentation of 3D data. *Journal of Structural Bioloby*, 120(3):266–275, 1997. TY - JOUR.

[196] T. Lindeberg. *Scale-space Theory in Computer Vision*. The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Netherlands, 1994.

[197] Y. Livnat, H.-W. Shen, and C. Johnson. A near optimal isosurface extraction algorithm using the span s pace. In *IEEE Transactions on Visualization and Computer Graphics*, pages 73–84, 1996.

[198] C. Loop. A g^1 triangular spline surface of arbitrary topological type. *Computer Aided Geometric Design*, 11(3):303–330, 1994.

[199] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. *IEEE Transactions on Visualization and Computer Graphics*.

[200] Lorensen, W., and Cline, H. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21:163–169, 1987.

[201] G. Loy and A. Zelinsky. Fast radial symmetry for detecting points of interest. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(8):959–973, 2003.

[202] J. Lu, D. Healy, and J. Weaver. Contrast enhancement of medical images using multiscale edge representation. *Optical Engineering*, 33(7):2151–2161, 1994. TY - JOUR.

[203] C. Luo and Y. Rudy. A model of the ventricular cardiac action potential: Depolarization, repolarization and their interaction. *Circulation Research*, 68(6):1501–1526, 1991.

[204] C. Luo and Y. Rudy. A dynamic model of the cardiac ventricular action potential: I. simulations of ionic currents and concentration changes. *Circulation Research*, 74(6):1071–1096, 1994.

[205] R. Malladi and J. Sethian. A real-time algorithm for medical shape recovery, 1998. TY - CONF A real-time algorithm for medical shape recovery.

[206] E. J. Mancini, M. Clarke, B. Gowen, T. Rutten, and S. D. Fuller. Cryo-electron microscopy reveals the functional organization of an enveloped virus, semliki forest virus. *Molecular Cell*, 5(255-266), (2000).

[207] R. Marabini, G. T. Herman, and J. M. Carazo. 3d reconstruction in electron microscopy using art with smooth spherically symmetric volume elements (blobs). *Ultramicroscopy*, 72:53–65, 1998.

[208] M. Marko and A. Leith. Sterecon - three-dimensional reconstructions from stereoscopic contouring. *Journal of Structural Biology*, 116(1):93–98, 1996. TY - JOUR.

[209] I. M. B. Martin, D. C. Marinescu, R. E. Lynch, and T. S. Baker. Identification of spherical virus particles in digitized images of entire electron micrographs. *Journal of Structural Biology*, 120:146–157, 1997.

[210] T. Masuda, K. Yamamoto, and H. Yamada. Detection of partial symmetry using correlation with rotated-reflected images.

Pattern Recognition, 26(8):1245–1253, 1993.

[211] S. Matej and R. M. Lewitt. Practical considerations for 3-d reconstruction using spherical symmetric volume elements. *IEEE Transactions on Medical Imaging*, 15, 1996.

[212] N. Max. Computer representation of molecular surfaces. *IEEE Computer Graphics and Applications*, pages 21–29, Aug. 1983.

[213] N. Max and E. Getzoff. Spherical harmonic molecular surfaces. *IEEE Computer Graphics & Applications*, 8:42–50, 1988.

[214] B. McEwen and M. Marko. Three-dimensional electron micros-copy and its application to mitosis research. *Methods Cell Biol*, 61:81–111, 1999. TY - JOUR.

[215] J. Mestres, D. Rohrer, and G. Maggiora. Mimic: A molecular -field matching program. exploiting applicability if molecular similarity approaches. *Journal of Computational Chemistry*, 18(7):934–, 1997.

[216] F. F. a. L. R. S. Michael Bock, Stefan O. Schoenberg. Separation of arteries and veins in 3d mr angiography using correlation analysis. *IEEE Transactions on Medical Imaging*, 43:481–487, 2000.

[217] B. Morse, S. Pizer, D. Puff, and C. Gu. Zoom-invariant vision of figural shape: effects on cores of images disturbances. *Computer Vision and Image Understanding*, 69(1):72–86, 1998.

[218] MPI. Message passing interface forum, MPI: A message-passing interface standard. *manual*, 1995.

[219] D. Mumford and A. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math*, XLII:577–685, 1989.

[220] F. Naterer. *The mathematics of computerized tomography*. John Wiley & Sons, New York, 1986.

[221] F. Naterer and F. Wubbeling. *Mathematical Methods in Image Reconstruction*. SIAM, Philadelphia, 2001.

[222] R. L. Ogniewicz and O. Kubler. Hierachic voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[223] S. Osher and R. P. Fedkiw. *Level set methods and dynamic implicit surfaces*. Springer, 2003.

[224] G. Papandreou and P. Maragos. Multigrid geometric active contour models. *IEEE Transactions on Image Processing*, 16(1):229, 2007.

[225] S. Park and C. Bajaj. Feature selection of 3d volume data through multi-dimensional transfer functions. *Pattern Recognition Letters*, 28(3):367–374, 2007.

[226] S. Park, C. Bajaj, and G. Gladish. Artery-vein separation of human vasculature from 3d thoracic ct angio scans. *CompImage 2006*, pages 23–30, 2006.

[227] S. M. Park, G. W. Gladish, and C. L. Bajaj. Artery-vein separation from thoracic CTA scans. *IEEE Transactions on Medical Imaging*.

[228] S. M. Park, G. W. Gladish, and C. L. Bajaj. Automatic pulmonary embolism detection from thoracic CTA scans. *IEEE Transactions on Medical Imaging*.

[229] P. A. Penczek, R. A. Grasucci, and J. Frank. The ribosome at improved resolution: New techniques for merging and orientation reÂ¯nement in 3d cryo-electron microscopy of biological particles. *Ultramicroscopy*, 53:251–270, 1994.

[230] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell*, 12:629–639, 1990. TY - JOUR.

[231] D. L. Pham and J. L. Prince. Adaptive fuzzy segmentation of magnetic resonance images. *IEEE Transactions on Medical Imaging*, 18(9):737–752, Aug. 1998.

[232] D. L. Pham and J. L. Prince. An adaptive fuzzy c-means algorithm for image segmentation in the presence of intensity inhomogeneities. *Pattern Recognition Letters*, 20(1):57–68, 1999.

[233] L. Piegl and W. Tiller. *The NURBS Book (Monographs in Visual Communication), 2nd ed.* Springer-Verlag, New York, 1997.

[234] S. Pizer, E. Ambum, J. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. Romeny, and J. Zimmerman. Adaptive histogram equalization and its variations. *Computer Vision, Graphics and Image Processing*, 39(3):355–368, 1987.

[235] S. Pizer, D. Eberly, D. Fritsch, and B. Morse. Zoom-invariant vision of figural shape: the mathematics of cores. *Computer Vision and Image Understanding*, 69(1):55–71, 1998.

[236] D. Potts, G. Steidl, and A. Nieslony. Fast convolution with radial kernels at nonequispaced knots. *Numer. Math.*, 98(2):329–351, 2004.

[237] D. Potts, G. Steidl, and M. Tasche. Fast fourier transforms for nonequispaced data: A tutorial. In I. J. J. Benedetto and P. J. S. G. Ferreira, editors, *Modern Sampling Theory: Mathematics and Applications*, pages 249–274. Birkhaser, Boston, Massachusetts, 2001.

[238] W. Pratt. *Digital Image Processing (2nd Ed.)*. A Wiley-Interscience Publication, 1991. TY - BOOK.

[239] T. Preuber and M. Rumpf. A level set method for anisotropic geometric diffusion in 3d image processing. *SIAM J. Appl.*

Math., page (to appear), 2002.

[240] M. Radermacher. Three-dimensional reconstruction from random projections: orientational alignment via radon transforms. *Ultramicroscopy*, 53:121–136, 1993.

[241] M. Radermacher. *Weighted Back-projection Methods*. Springer, 2006.

[242] V. Reddy, P. Natarajan, B. Okerberg, K. Li, K. Damodaran, R. Morton, C. I. Brooks, and J. Johnson. VIrus Particle Explorer (VIPER), a website for virus capsid structures and their computational analyses. *Journal of Virology*, 75:11943–11947, 2001.

[243] B. Rieger, F. Timmermans, L. Van Vliet, and P. Verbeek. On curvature estimation of ISO surfaces in 3D gray-value images and the computation of shape descriptors. *IEEE transactions on pattern analysis and machine intelligence*, pages 1088–1094, 2004.

[244] D. W. Ritchie and G. J. L. Kemp. Fast computation, rotation, and comparison of low resolution spherical harmonic molecular surfaces. *Journal of Computational Chemistry*, 20(4):383–395, Feb. 1999.

[245] D. F. Rogers. *An Introduction to NURBS With Historical Perspective*. Academic Press, San Diego, CA, 2001.

[246] J. M. Rogers and A. D. McCulloch. A collocation-galerkin finite element model of cardiac action potential propagation. *IEEE Trans Biomed. Eng.*, 41:743–757, 1994.

[247] L. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.

[248] Y. Rudy and R. Plonsey. A comparison of volume conductor and source geometry effects on body surface and epicardial potentials. *Circ. Res.*, 46:283–291, 1980.

[249] J. Russ. *The Image Processing Handbook (2nd edition)*. CRC Press, 2000.

[250] F. B. Sachse. *Computational Cardiology: Modeling of Anatomy, Electrophysiology, and Mechanics.* LNCS 2966. Springer, Berlin, Heidelberg, New York, 2004.

[251] O. Sahni. *Adaptive procedure for efficient blood-flow simulations*. PhD thesis, RPI, 2005.

[252] O. Sahni, J. Mueller, K. E. Jansen, M. S. Shephard, and C. A. Taylor. Efficient anisotropic adaptive discretization of the cardiovascular system. Technical report, RPI, 2005.

[253] M. Sanner, A. Olsen, and J.-C. Spehner. Fast and robust computation of molecular surfaces. In *Proceedings of the 11th ACM Symposium on Computational Geometry*, pages C6–C7, New York, 1995. ACM.

[254] M. F. Sanner and A. J. Olson. Reduced surface: An efficient way to compute molecule surfaces. *Biopolymers*, 38:305–320, 1996.

[255] G. Sapiro. *Geometric partial differential equations and image analysis*. Cambridge Univ Pr, 2001.

[256] C. G. Schiek. Terrain change detection using aster optical satellite imagery along the kunlun fault, tibet. Master's thesis, Department of Geological Sciences, University of Texas at El Paso, Mar. 2004.

[257] C. G. Schiek, R. Araiza, J. M. Hurtado, A. A. Velasco, V. Kreinovich, and V. Sinyansky. Images with uncertainty: Efficient algorithms for shift, rotation, scaling, and registration, and their applications to geosciences. In I. M. Nachtegael, D. V. der Weken, and E. E. Kerre, editors, *Soft Computing in Image Processing: Recent Advances*. Springer Verlag, Berlin (to appear.

[258] U. J. Schoepf and P. Costello. Ct angiography for diagnosis of pulmonary embolism: State of the art. *Radiology*, 230(2):329–337, 2004.

[259] J. Sethian. *Level Set Methods and Fast Marching Methods, 2nd edition*. Cambridge University pPress, 1999.

[260] J. A. Sethian. A marching level-set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci*, 93(4), 1996.

[261] M. Sezan and H. Stark. *Image restroation by the method of convex projections: part 2-applications and numerical results*. IEEE Trans. Med. Imaging, MI-1(2):95-100, 1982.

[262] O. Sharma and F. Anton. CUDA based Level Set Method for 3D Reconstruction of Fishes from Large Acoustic Data. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 17*, 2009.

[263] O. Sharma, Q. Zhang, F. Anton, and C. Bajaj. Multi-domain, Higher Order Level Set Scheme for 3D Image Segmentation on the GPU. In *Proc of The Twenty-Third IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010.

[264] D. Shen, H. S. Ip, K. T. Cheung, and E. K. Teoh. Symmetry detection by generalized complex moments: a close form solution. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(5):466–476, 1999.

[265] H.-W. Shen. Isosurface extraction in time-varying fields using a temporal hierarchical index tree. In *IEEE Proceedings of Visualization '98*, pages 159–166, 1998.

[266] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock graphs and shape matching. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 222–229, 1998.

[267] D. Siersma. Voronoi diagrams and morse theory of the distance function. *Geometry in Present Day Science*, 1999.

[268] E. Sifakis and G. Tziritas. Moving object localization using a multi-label fast marching algorithm. *Signal Processing: Image Communication*, 16(10):963–976, 2001.

[269] C. Sorzano, S. Jonic, M. Cottevieille, E. Larquet, N. Boisset, and S. Marco. 3d electron microscopy of biological nanomachines: principles and applications. *European Biophysics Journal*, 36:995–1013, 2007.

[270] C. Sorzano, R. Marabini, J. Vela¡äzquez-Muriel, J. Bilbao-Castroa, S. Scheresa, J. Carazo, and A. Pascual-Montano. Xmipp: a new generation of an open-source image processing package for electron microscopy. *Journal of Structural Biology*, 148:194–204, 2004.

[271] C. O. S. Sorzano, S. Jonic, C. El-Bez, J. M. Carazo, S. D. Carlo, P. Thevennaz, and M. Unser. A multiresolution approach to pose assignment in 3-d electron microscopy of single particles. *J. Structural Biology*, 146:381–392, 2004.

[272] J. Stark. Adaptive contrast enhancement using generalization of histogram equalization. *IEEE Trans. Image Processing*, 9(5):889–906, 2000. TY - JOUR.

[273] R. M. Stefancik and M. Sonka. Highly automated segmentation of arterial and venous trees from three-dimensional magnetic resonance angiography (mra). *The International Journal of Cardiovascular Imaging*, 17:37–47, 2001.

[274] G. Steidl. A note on fast fourier transforms for nonequispaced grids. *Advances in Computational Mathematics*, 9:337–352, 1998.

[275] A. Stoschek and R. Hegerl. Denoising of electron tomographic reconstructions using multiscale transformations. *J. Struct Biol*, 120:257–265, 1997.

[276] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[277] R. Strzodka and M. Rumpf. Level set segmentation in graphics hardware. In *Proc. IEEE International Conference on Image Processing (ICIP-2001)*, pages 1103–1106.

[278] E. Suli and A. Ware. A spectral method of characteristics for hyperbolic problems. *SIAM J. Numer. Anal*, 28:423–445, Mar. 1991.

[279] C. Sun and J. Sherrah. 3D symmetry detection using the extended Gaussian image. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):164–168, 1997.

[280] P. M. Sutton and C. D. Hansen. Accelerated isosurface extraction in time-varying fields. *IEEE Transactions on Visualization and Computer Graphics*, 6(2):98–107, 2000.

[281] W. Sweldens. The lifting scheme: A custom-design construction of biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[282] W. Sweldens. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal.*, 29(2):511–546, 1998. TY - JOUR.

[283] F. Tama, O. Miyashita, and C. R. Brooks. Flexible multi-scale fitting of atomic structures into low-resolution electron density maps with elastic network normal mode analysis. *Journal of molecular biology*, 337(4):985–99, 2004.

[284] S. Tari, J. Shah, and H. Pien. Extraction of shape skeletons from gray-scale images. *Computer Vision and Image Understanding*, 66(2):133–146, 1997.

[285] D. S. Taubman and M. W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and P*. Norwell, MA: Kluwer Academic Publishers, 2001.

[286] C. Taylor, T. Hughes, and C. Zarins. Finite element modeling of 3-dimensional pulsatile flow in the abdominal aorta: Relevance to atherosclerosis. *Annals of Biomedical Engineering*, 26(6):1–13, 1998.

[287] C. Taylor, T. Hughes, and C. Zarins. Finite element modeling of blood flow in arteries. *Computer Methods in Applied Mechanics and Engineering*, 158(1-2):155–196, 1998.

[288] C. Taylor, T. Hughes, and C. Zarins. Effect of exercise on hemodynamic conditions in the abdominal aorta. *Journal of Vascular Surgery*, 29:1077–89, 1999.

[289] P. E. Team. *Essential Atlas of Anatomy*. Parramon Ediciones, Barcelona, Spain, English edition, 2001.

[290] J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Grid Generation*. CRC Press LLC, 1999.

[291] M. R. T. three-dimensional reconstruction of single particles from random and non-random tilt series. J. Electron microsc. *Tech.*, 9:359–394, 1988.

[292] P. K. S. Tianhu Lei, Jayaram K. Udupa and D. Odhner. Artery-vein separation via mra - an image processing approach. *IEEE Transactions on Medical Imaging*, 20(8):689–703, 2001.

[293] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical Analysis of Finite Mixture Distrubutions*. John Wiley and Sons, Chichester, UK, 1985.

[294] T. B. C. L. . S. R. Tomas Bülow, Fafael Wiemker. Automatic extraction of the pulmonary artery tree from multi-slice ct data. *Medical Imaging 2005: Physiology, Function, and Structure from Medical Images*, 6(23):730–740, 2005.

[295] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the 1998 IEEE International Conference on Computer Vision*, pages 839–846, New Delhi, India, 1998.

[296] M. van Heel. Angular reconstruction: a posteriori assignment of projection directions for 3d reconstruction. *Ultramicroscopy*, 21:111–124, 1987.

[297] M. van Heel, B. Gowen, R. Matadeen, E. V. Orlova, R. Finn, T. Pape, D. Cohen, H. Stark, R. Schmidt, M. Schatz, and A. Patwardhan. Single-particle electron cryo-microscopy: towards atomic resolution. *Cambridge Quarterly Review of Biophysics*, 33:307–369, Mar. 2000.

[298] A. Varshney. Hierarchical geometric approximations. *PhD thesis, Dept. of CS, U. of North Carolina, Chapel Hill, TR-050*, 1994.

[299] A. Varshney and J. Brooks. Fast analytical computation of richard's smooth molecular surface. *Proceedings of the Visualization Conference, G. M. Nielson and D. Bergeron (ed.)*, pages 300–307, 1993.

[300] A. Verroust and F. Lazarus. Extracting skeletal curves from 3d scattered data. *The Visual Computer*, 16:15–25, 2000.

[301] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the Mumford and Shah model. *International Journal of Computer Vision*, 50(3):271–293, 2002.

[302] F. Vetter, A. McCulloch, and J. Rogers. A finite element model of passive mechanics and electrical propagation in the rabbit ventricles. *Computers in Cardiology*, pages 705–708, 1998.

[303] N. Volkmann. A novel three-dimensional variant of the watershed transform for segmentation of electron density maps. *Journal of Structural Biology*, 138(1-2):123–129, 2002. TY - JOUR.

[304] R. Voorintholt, M. T. Kosters, G. Vegter, G. Vriend, and W. G. Hol. A very fast program for visualizing protein surfaces, channels and cavities. *Journal of Molecular Graphics*, 7(4):243–245, Dec. 1989.

[305] A. F. Ware. Fast approximate fourier transforms for irregularly spaced data. *SIAM Rev*, 40(4):838–856.

[306] J. Weickert. *Anisotropic Diffusion In Image Processing*. ECMI Series, Teubner, Stuttgart, ISBN 3-519-02606-6, 1998. TY - BOOK.

[307] J. Weickert. Fast segmentation methods based on partial differential equations and the watershed transformation. In *P. Levi, R.-J.Ahlers, F. May, M. Schanz (Eds.), Mustererkennung 1998, Springer, Berlin*, pages 93–100, 1998.

[308] R. J. Whitley. Herpes simplex viruses. *Fields Virology (B.N.Fields, D.M. Knipe, P.M. Howley, R.M. Chanock, J.L. Melnick, T.P. Monath, B. Riozman, and S.E. Strauss, Eds), 3rd edit*, 2:2297–2342, 1996.

[309] W. Wikoff, L. Liljas, R. Duda, H. Tsuruta, R. Hendrix, and J. Johnson. Topologically linked protein rings in the bacteriophage HK97 capsid. *Science*, 289:2129–2133, 2000.

[310] J. Wilhelms and A. V. Gelder. Octrees for Faster Isosurface Generation. *Computer Graphics*, 24:57–62, 1990.

[311] R. L. Winslow, D. F. Scollan, J. L. Greenstein, C. K. Yung, W. B. Jr., G. Bhanot, D. L. Gresh, and B. E. Rogowitz. Mapping, modeling, and visual exploration of structure-function relationships in the heart. *Deep computing for the life sciences*, 40(2), 2001.

[312] S. D. H. A. Y. C. Z. J. J. M. Winslow, R.L. Electrophysiological modeling of cardiac ventricular function: from cell to organ. *Annu. Rev. Biomed. Eng.*, 2:119–155, 2000.

[313] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. Image Processing*, 7(3):359–369, 1998.

[314] G. Xu. *Geometric Partial Differential Equation Methods in Computational Geometry*. Science Press, Beijing, China, 2008.

[315] G. Xu, M. Li, A. Gopinath, and C. Bajaj. Computational inversion of electron tomography images using l2-gradient flows. Technical report, 2009.

[316] G. Xu and Y. Shi. Progressive computation and numerical tables of generalized Gaussian quadrature formulas. *Journal on Numerical Methods and the Computer Application*, 27(1):9–23, 2006.

[317] Y. Xu, J. B. Weaver, D. M. Healy, and J. Lu. Wavelet transform domain filters: A spatially selective noise filtration technique. *IEEE Trans. Image Processing*, 3(6):747–758, 1994.

[318] I. Yanovsky, P. Thompson, S. Osher, and A. Leow. Large deformation unbiased diffeomorphic nonlinear image registration: Theory and implementation. Technical report, UCLA CAM, 2006.

[319] L. Yin, X. Luo, and M. S. Shephard. Identifying and meshing thin sections of 3-d curved domains. Technical report, RPI, 2005.

[320] L. Ying and D. Zorin. A simple manifold-based construction of surfaces of arbitrary smoothness. *ACM Trans. Graph.*, 23(3):271–275, 2004.

[321] R. Yip, W. Lam, P. Tam, and D. Leung. A Hough transform technique for the detection of rotational symmetry. *Pattern Recognition Letters*, 15:919–928, 1994.

[322] H. M. . J. D. Yoshitaka Masutani. Computerized detection of pulmonary embolism in spiral ct angiography based on

volumetric image analysis. *IEEE Transaction on Medical Imaging*, 21(12):1517–1523, 2002.

[323] Y. You, W. Xu, A. Tannenbaum, and M. Kaveh. Behavioral analysis of anisotropic diffusion in image processing. *IEEE Transactions on Image Processing*, 5(11):1539–1553, 1996.

[324] Y. You, W. Xu, A. Tannenbaum, and M. Kaveh. Behavioral analysis of anisotropic diffusion in image processing. *Image Processing, IEEE Transactions on*, 5(11):1539–1553, 2002.

[325] D. Youla and H. Webb. *Image restoration by the method of convex projections: part 1-theory*. IEEE Trans. Med. Imaging, MI-1(2):81-94, 1982.

[326] I. Young and L. Vliet. Recursive implementation of the gaussian filter. *Signal Processing*, 44:139–151, 1995. TY - JOUR.

[327] H. Yserentant. Hierarchical bases. *ICAIM 91, R.E. Ed., SIAM Philadelphia 49*, pages 329–412, 1992.

[328] T.-Y. Yu and B. K. Soni. NURBS Evaluation and Utilization for Grid Generation. In *5th International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 323–332, 1996.

[329] Z. Yu and C. Bajaj. Anisotropic vector diffusion in image smoothing. In *Proceeding of the 9th IEEE ICIP*, volume 1, pages 828–831, 2002.

[330] Z. Yu and C. Bajaj. Image segmentation using gradient vector diffusion and region merging. In *International Conference on Pattern Recognition*, pages 941–944, 2002.

[331] Z. Yu and C. Bajaj. Normalized gradient vector diffusion and image segmentation. In *European Conference on Computer Vision*, pages 517–530, 2002.

[332] Z. Yu and C. Bajaj. A fast and adaptive algorithm for image contrast enhancement. In *Proc of IEEE International Conference on Image Processing*, pages 1001–1004, 2004.

[333] Z. Yu and C. Bajaj. A segmentation-free approach for skeletonization of gray-scale images via anisotropic vector diffusion. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'04)*, volume 1, pages 415–420, 2004.

[334] Z. Yu and C. Bajaj. Automatic ultrastructure segmentation of reconstructed cryoem maps of icosahedral viruses. *IEEE Transactions on Image Processing: Special Issue on Molecular and Cellular Bioimaging*, 14(9):1324–37, Sept. 2005.

[335] Z. Yu and C. Bajaj. Computational approaches for automatic structural analysis of large biomolecular complexes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2007.

[336] K. S. Y. Yuen and W. W. Chan. Two methods for detecting symmetries. *Pattern Recognit. Lett.*, 15:279, 1994.

[337] Y. Zhang and C. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. *Computer Methods in Applied Mechanics and Engineering (CMAME)*, 195(9-12):942–960, 2006.

[338] Y. Zhang, C. Bajaj, and B.-S. Sohn. 3D finite element meshing from imaging data. *The special issue of Computer Methods in Applied Mechanics and Engineering (CMAME) on Unstructured Mesh Generation*, 194(48-49):5083–5106, 2005.

[339] Y. Zhang, C. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. In *Proceedings of 14th International Meshing Roundtable*, pages 449–468, 2005.

[340] Y. Zhang, Y. Bazilevs, S. Goswami, C. L. Bajaj, and T. J. R. Hughes. Patient-specific vascular nurbs modeling for isogeometric analysis of blood flow. *Computer Methods in Applied Mechanics and Engineering (CMAME)*, 196(29-30):2943–2959, 2007.

[341] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *1st IEEE Workshop on Variational and Level Set Methods*, pages 194–202, 2001.

[342] Y. Zhou and A. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5:196–209, 1999.

[343] Z. H. Zhou, M. L. Baker, W. Jiang, M. Dougherty, J. Jakana, G. Dong, G. Lu, and W. Chiu. Electron cryomicroscopy and bioinformatics suggest protein fold models for rice dwarf virus. *Nature Structural Biology*, 2001.

Exercises

Chapter 3

Elucidating Geometry from Images

chap 3 image to geometry extraction: boundary segmentation, variational methods, surfaces, matching

3.1 Digital Topology Filtering and Geometric Sampling

from point clouds to boundary meshes
linear and non-linear (PDE) approach

3.2 Segmentation

linear and non-linear

3.3 Skeletonization

linear and non-linear

The following algorithm is taken from [74] (YB2004).

Image Skeletonization via Gradient Vector Diffusion

The skeleton of a scalar/binary map is widely recognized as one of the most important shape descriptors in image processing and pattern recognition. In the context of automatic structure analysis of large bio-molecular complexes (LBCs), we use the skeletons to extract the secondary structures of a protein density map and to fit an atomic (PDB) structure or an animo acid sequence into a cryo-EM density map, as will be shown in the following chapters. Besides the skeletons, the critical points of a density map are frequently used in structure analysis of LBCs (e.g., symmetry detection, segmentation, alignment, secondary structure identification). Critical points of a scalar map include local maxima, local minima, and saddle points. We are particularly interested in the local maximal critical points, which are usually treated as zero-dimensional skeletons of a cryo-EM density map. For this reason, the critical point detection can be thought of as a special case of skeletonization.

Gradient vector diffusion (GVD) has proved to be useful in active snake models for image segmentation [69]. In this chapter, we will demonstrate that the gradient vector diffusion is also very useful in extracting skeletons (including critical points) of a scalar image/map. In the rest of this chapter, we first give a brief review on skeletonization. Then we introduce various GVD methods, followed by the applications of GVD in critical point detection and boundary-free skeletonization. Finally we conclude this chapter by showing some results and comparing our approach with others.

Gradient Vector Diffusion

Gradient vector diffusion is commonly used for two purposes. First, it can smooth the noise seen in the given vector field. This property is similar to the various techniques of image smoothing [53, 66]. Second, vector diffusion can make the non-zero vectors propagate toward the areas of zero-vectors. This property is extremely useful for many real-world images, which contain "flat" regions and hence zero gradients.

Linear Vector Diffusion In [69], the authors described a diffusion technique to smooth gradient vector fields. The gradient vectors are represented by Cartesian coordinates and similar partial differential equations (PDEs) are separately applied to each component of the vectors:

$$\begin{cases} \frac{du}{dt} = \mu\nabla^2 u - (u - f_x)(f_x^2 + f_y^2 + f_z^2) \\ \frac{dv}{dt} = \mu\nabla^2 v - (v - f_y)(f_x^2 + f_y^2 + f_z^2) \\ \frac{dw}{dt} = \mu\nabla^2 w - (w - f_z)(f_x^2 + f_y^2 + f_z^2) \end{cases} \tag{3.1}$$

where (u,v,w) is initialized with $\nabla f(x,y,z)$, and $f(x,y,z)$ is an edge strength map of the original image; e.g., $f(x,y,z) = \|\nabla G_\sigma(x,y,z) * I(x,y,z)\|^2$, where $G_\sigma(x,y,z)$ stands for a Gaussian kernel. These diffusion equations are originally used for image segmentation [69]. In the case of Dey:06, the initialization of gradient vector fields may vary with different situations, as described later in this chapter.

The above equations obviously are linear or isotropic. It therefore inherits the drawbacks of most linear systems (e.g., blurred maps and biased skeletons).

Nonlinear Vector Diffusion Another way to diffuse a gradient vector field is based on the polar-coordinate representation of a vector [71, 70]. In other words, we apply the diffusions on r (the magnitude of a vector) and θ (the orientation of a vector). In the diffusion scheme based on r and θ, a very weak vector can largely affect a strong vector, both on its magnitude and on its orientation. The vector magnitude r and orientation θ are diffused separately in two different partial differential equations. This type of vector diffusion has proved to be more robust in case of boundary concavities and/or gaps in image segmentation [71]. The drawback of this method is its computational burden due to the efforts that have to be made to deal with the periodicity of orientations. More details can be found in [71, 70].

We present here another type of partial differential equations for gradient vector diffusion and attempt to address the aforementioned problems. The new PDEs are similar to Equation 3.1 except that they are now based on anisotropic diffusion and that the second terms on the right sides of the equations are removed because they were originally used for better segmentation [69], not for skeletonization.

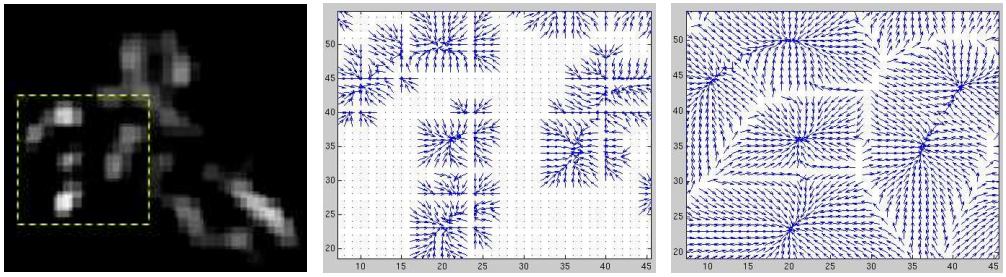
$$\begin{cases} \frac{du}{dt} = \mu \cdot div(g(\alpha) \cdot \nabla u) \\ \frac{dv}{dt} = \mu \cdot div(g(\alpha) \cdot \nabla v) \\ \frac{dw}{dt} = \mu \cdot div(g(\alpha) \cdot \nabla w) \end{cases} \tag{3.2}$$

where $g(\cdot)$ is a decreasing function and α is the angle between the central vector and the surrounding vectors. For faster implementation, the calculation of the angle between two vectors is usually approximated by the inner-product of two vectors divided by their magnitudes. For instance, we can define $g(\alpha)$ as follows:

$$g(\vec{c},\vec{s}) = \begin{cases} e^{\kappa\cdot(\frac{\vec{c}\cdot\vec{s}}{\|\vec{c}\|\|\vec{s}\|}-1)} & if\ \vec{c}\neq 0\ and\ \vec{s}\neq 0 \\ 0 & if\ \vec{c}=0\ or\ \vec{s}=0 \end{cases} \tag{3.3}$$

where κ is a positive constant; $\vec{c}$ and $\vec{s}$ stand for the central vector and one of the surrounding vectors, respectively. In our implementation of 2D images, we consider a 4-neighborhood for each pixel.

A study of Equation 3.2 would suggest the following: First, its implementation is similar to that of Equation 3.1. Hence, the time complexity is comparable to the conventional scheme [69]. Secondly, the weighting function $g(\cdot)$ is designed such that the problem of skeleton bias can be easily avoided. This is because the weighting function $g(\cdot)$ decreases as the angle between two



(a) (b) (c)

Figure 3.1: Illustration of critical point extraction using gradient vector diffusion. (a) one slice of herpesvirus capsid protein, vp5. (b) gradient vector field without diffusion corresponding to the boxed out area in (a). (c) gradient vector field after diffusion (10 iterations) improves the detection of critical points.

vectors (the center and the neighbor) increases from 0 to π. If $g(\cdot)$ goes to zero as the angle approaches π, then the vectors from both sides of curvilinear structures would stop on the central lines of the structures regardless of how different the contrasts would be on either side of the structures. We shall demonstrate such an example in Section 3.3. Thirdly, Equation 3.2 is very similar to the anisotropic diffusion equation of image smoothing [53]. Therefore, we would expect to see something that shows the uniqueness of the anisotropic property of our system, similar to the edge-preserving property as we saw in Perona-Malik model [53]. In fact, the skeleton strength map, as shown later, does demonstrate this unique property.

Critical Point Detection

The critical points of a scalar map can be classified as one of three types: local maxima, local minima, and saddle points of the given scalar function. However, in the context of structure identification, the maximal critical points are of great interest, due to the fact that, in a molecular density map, higher densities imply the existence of more atoms. These critical points can be easily computed from the local maxima of a given scalar map. Since noise is always present in the original maps, a pre-filtering process should be applied. A scalar map pre-filter can be either linear or nonlinear. A linear filter (e.g., Gaussian filtering) may destroy some weak features and hence eliminate some critical points. A nonlinear pre-filter [53, 66], however, tends to "deform" a sub-region, yielding many unwanted critical points.

A good alternative is a vector field filtering technique that is based on the gradient vectors of the scalar 3D map, from which the afore-mentioned critical points can be easily extracted. The gradient vector field generated directly from the original map suffers from noise and cannot be used to detect the critical points. In addition, some regions containing critical points may be "flat" such that the gradient vectors vanish and hence the critical points vanish too. For those reasons, we apply the gradient vector diffusion as shown in Equation 3.2 before we extract the critical points.

Once the gradient vector field is generated and diffused, we can detect the *source* points, as defined below:
Definition: A point is called *source* if none of its neighbors points to it. In other words, a point A is a *source* if and only if, for any neighbor B of A:

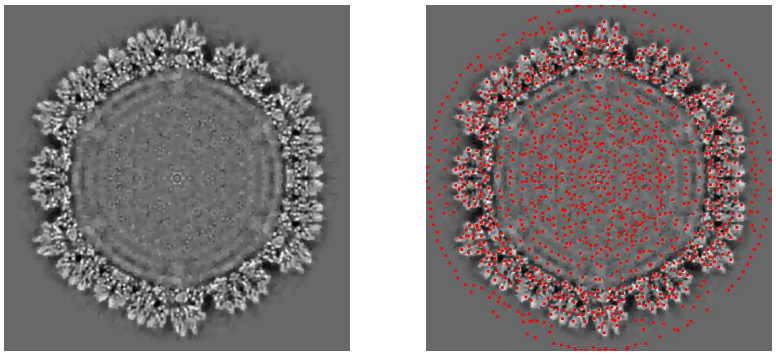
$$\vec{v}_B \cdot \vec{BA} \leq 0 \tag{3.4}$$

where $\vec{v}_B$ is the diffused gradient vector at B and $\vec{BA}$ is the vector from B to A. All the *source* points are regarded as the critical points.

To better illustrate the application of the anisotropic gradient vector diffusion technique to accurately extract critical points from a given 3D map, we show cross-sectional two-dimensional (2D) slices (Figure 3.1). The images are from a slice of the herpes virus capsid protein vp5 [78]. For better illustration of vector fields, we only consider a small area as boxed out in Figure 3.1(a). The vector field in Figure 3.1(b) is computed before the vector diffusion. Figure 3.1(c) demonstrates the power of the anisotropic vector diffusion, from which one can easily extract the critical points. Another example with greater detail, is illustrated in Figure 3.2, where one can see that the critical points can be correctly detected by anisotropic vector diffusion.

Boundary-Free Skeletonization

We present here a boundary-free approach for skeletonization of a scalar image (either 2D or 3D). In other words, our method can extract the skeletons directly from the input images without detecting the feature boundaries. Our skeletonization algorithm



(a) (b)

Figure 3.2: Illustration of critical point extraction using gradient vector diffusion. (a) One slice (noise reduced) of rice dwarf virus (RDV). The regions with high intensities correspond to molecular features. (b) About 1200 critical points were found after 10 iterations of gradient vector diffusions. The number of critical points can be reduced by removing those whose density values are less than a certain threshold.

includes two major steps. First, the regions containing features are thinned such that the skeletons are "enhanced" (much sharper than original features). Second, we extract the skeletons by tracing the skeleton strength map using either canny's method or the tensor-based approach. The first step is further decomposed into three sub-steps: generating initial vector field, diffusing the vector field, and calculating the skeleton strength map. In the following, we will describe the algorithm step-by-step (the skeleton-tracing is treated as one of the "sub-steps").

Initializing Generation Vector Field We first consider images where the features are brighter than the background (this is usually the case we see in most molecular density maps). The initial gradient vector field is generated in the following way:

$$gvf(\vec{r}) = (I(\vec{r}) - I(\vec{r'})) \times \frac{\vec{r'} - \vec{r}}{\|\vec{r'} - \vec{r}\|} \tag{3.5}$$

where $I(\vec{r})$ is the intensity value at $\vec{r}$. $\vec{r'}$ is one of the (eight) immediate neighbors of $\vec{r}$, with the lowest intensity among the neighbors. Note that this definition of gradient vector is different from other conventional definitions (e.g., the one defined by central difference). We shall see later how Equation 3.5 performs better than the conventional definitions.

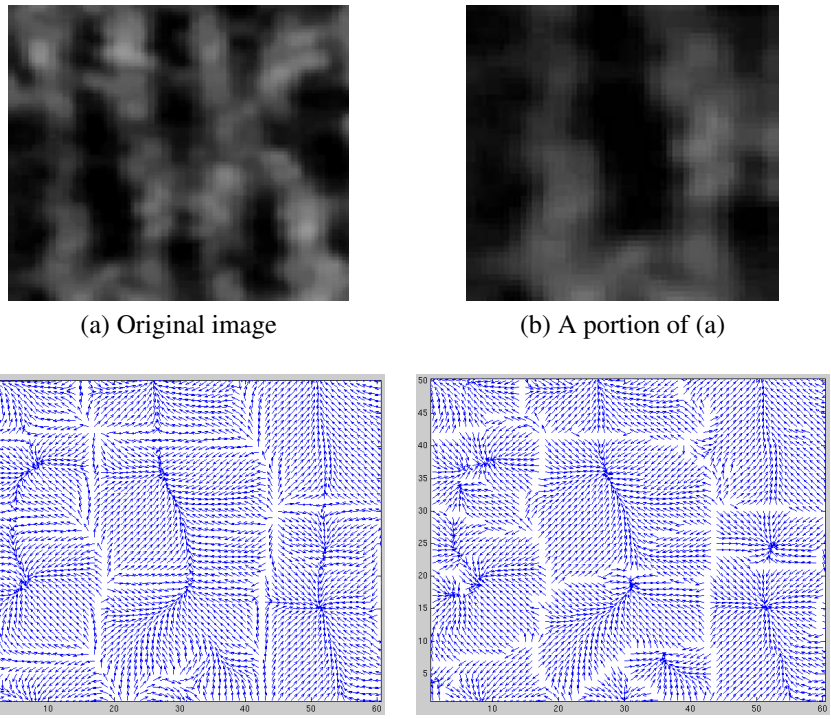
In case that the features have lower intensities than the background, the gradient vectors are defined by:

$$gvf(\vec{r}) = (I(\vec{r'}) - I(\vec{r})) \times \frac{\vec{r'} - \vec{r}}{\|\vec{r'} - \vec{r}\|} \tag{3.6}$$

where $\vec{r'}$ is one of the (eight) immediate neighbors of $\vec{r}$, with the highest intensity among the neighbors.

Equation 3.5 and Equation 3.6 deal with the situations where the objects are always either brighter or darker than the background. This is the cases discussed in [15]. In some cases [63, 29], one may have to first compute the edge strength map (e.g., $\|\nabla G_\sigma(x,y) * I(x,y)\|$), and then Equation 3.6 is applied to the edge strength map to generate the initial vector field.

Diffusing Gradient Vector Field The initial vector field computed above is then diffused using either the isotropic scheme (Equation 3.1) or anisotropic scheme (Equation 3.2). The PDE's are iteratively solved by finite difference technique. In Figure 3.3(a), we show an example of a synthesized molecular image. In this example, we use Equation 3.5 to generate the initial vector field, as the molecular structure is brighter than the background. The diffused vector fields by both isotropic and anisotropic schemes are shown in Figure 3.3(c) and Figure 3.3(d), respectively. For better illustration of vectors, Figure 3.3(c-d) correspond to only a portion of the original image, as seen in Figure 3.3(b). We can see that the major difference between these two schemes is that the anisotropic diffusion preserves the "sharp" features (the "empty" regions), where most of the surrounding vectors point away from the central point. Apparently those features correspond to the skeletons of the original gray-scale image. The advantages of anisotropic diffusion (compared to the isotropic diffusion) will be further demonstrated later.



(a) Original image (b) A portion of (a)

(c) Isotropic diffusion (d) Anisotropic diffusion

Figure 3.3: The demonstration of isotropic and anisotropic vector diffusion schemes on a synthesized molecular image (shown in (a)). For better illustration of vectors, (c-d) are restricted to the left-bottom portion of the original image, as shown in (b), and all vectors in (c-d) are normalized to the same magnitude.

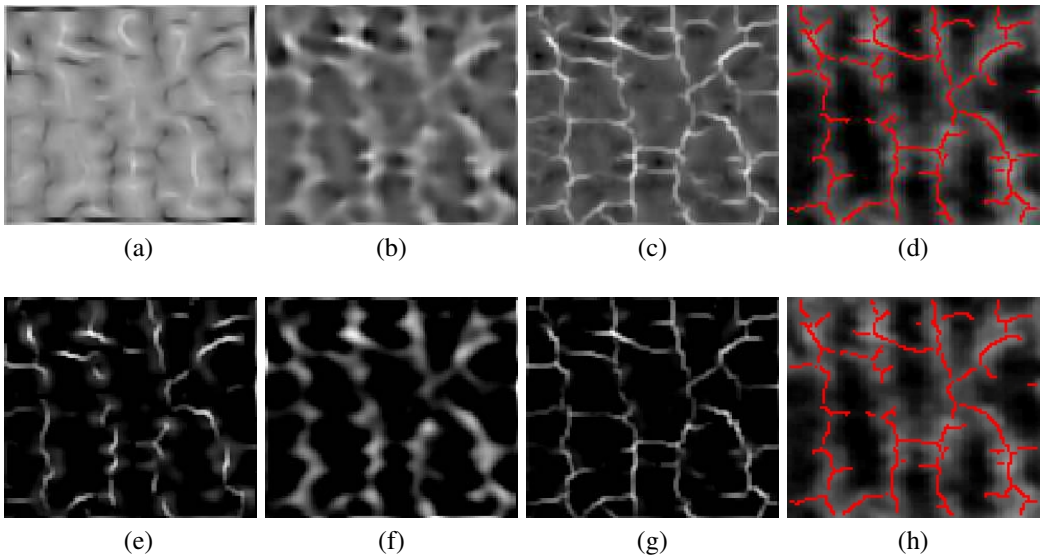


Figure 3.4: The skeleton strength maps and skeleton-tracing results. The original image is shown in Figure 3.3(a). (a) SSM by anisotropic vector diffusion, initialized by the classical central difference scheme. (b) SSM by isotropic vector diffusion, initialized by Equation 3.5 and calculated by Equation 3.7. (c) SSM by anisotropic vector diffusion, initialized by Equation 3.5 and calculated by Equation 3.7. (d) Skeletons traced from (c). (e) Same as (c) except that the vectors are normalized before calculating SSM. (f) Skeletons traced from (e). (g) SSM by anisotropic vector diffusion, initialized by Equation 3.6 and calculated by Equation 3.8. (h) Critical points detected from (g).

Computing Skeleton Strength Map To locate the "empty" regions of the diffused gradient vector field, we compute what we call *skeleton strength map* (SSM) by:

$$SSM(\vec{r}) = max(0, \sum_{\vec{r'} \in N(\vec{r})} \frac{gvf(\vec{r'}) \cdot (\vec{r'} - \vec{r})}{\|\vec{r'} - \vec{r}\|}) \tag{3.7}$$

where $N(\vec{r})$ is the set of the eight immediate neighbors of $\vec{r}$. The skeleton strength map, similar to the edge strength map, is a scalar map defined on every pixel and indicates the likelihood of each pixel being on the skeletons. With the original image seen in Figure 3.3(a), we show in Figure 3.4(a) the skeleton strength map, generated by anisotropic vector diffusion but initialized by the classical central difference scheme. We can see that this map does not give much clear information on the skeletons of the molecular structures. Figure 3.4(b) shows the SSM, generated by the isotropic vector diffusion but initialized by Equation 3.5. In contrast to the isotropic diffusion technique, Figure 3.4(c) shows the SSM, generated by anisotropic diffusion and initialized by Equation 3.5. From Figure 3.4(b) and Figure 3.4(c), we can see that the isotropic vector diffusion normally tends to blur the skeletons while the anisotropic vector diffusion can preserve the "sharp" skeletons very well. This is analogous to the isotropic/anisotropic diffusion commonly seen in image smoothing [53, 66].

There are two variants of Equation 3.7. First, we can normalize the vectors before calculating the skeleton strength map (SSM). This may be risky if there are a lot of noise in the image. However, sometimes it may be helpful. Figure 3.4(e) shows the skeleton strength map based on the normalized vectors. The skeletons look more sharper and clearer, although some "weak" features are over-emphasized. Another variant of Equation 3.7 is by considering the total amount of "incoming" vectors at a point instead of the total amount of "outgoing" vectors. The new skeleton strength map is defined as:

$$SSM^*(\vec{r}) = max(0, \sum_{\vec{r'} \in N(\vec{r})} \frac{gvf(\vec{r'}) \cdot (\vec{r} - \vec{r'})}{\|\vec{r} - \vec{r'}\|}) \tag{3.8}$$

where $N(\vec{r})$ is the set of the eight immediate neighbors of $\vec{r}$. Figure 3.4(g) shows the skeleton strength map, initialized by Equation 3.6 and calculated by Equation 3.8. It is interesting that this map highlights the critical points (zero-dimensional skeletons) instead of the one-dimensional skeletons.

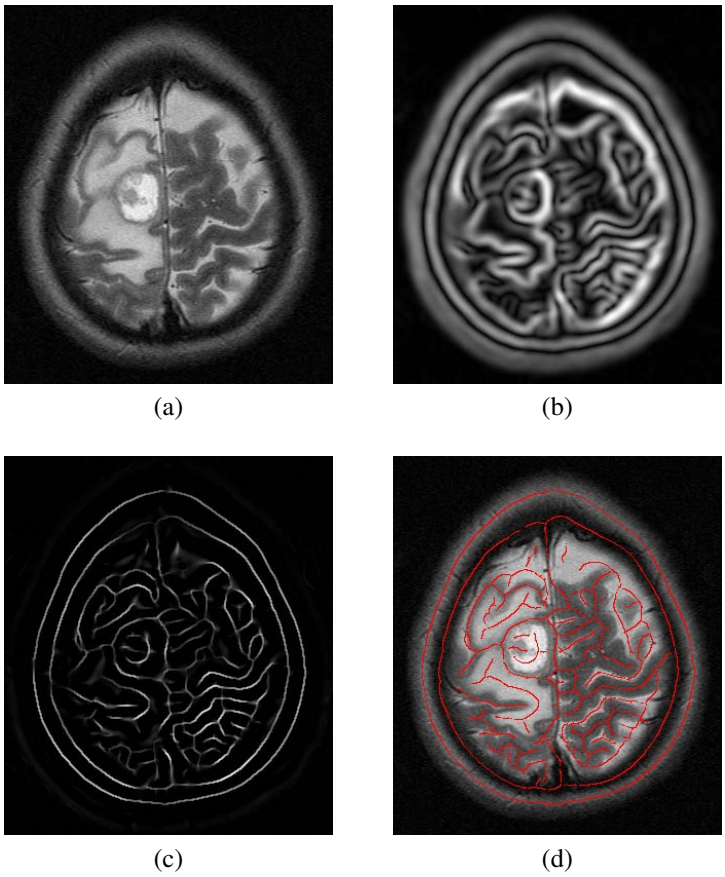


Figure 3.5: The demonstration of skeletonization on a brain MRI data. (a) Original image. (b) Edge strength map. (c) Skeleton strength map. (d) Skeletons traced (in red color).

Tracing Skeletons To compute the final skeletons, we need to find a way to trace the skeletons from the skeleton strength map. A good technique for this purpose is by Canny's method [13], which was originally designed for image edge detection. The most promising strategies used in this method are non-maximal suppression and double-threshold. The non-maximal suppression is first applied to the gradient magnitude map in order to obtain "thin" edges and extract candidate edges. Two thresholds are assumed such that candidate edges above the higher threshold are always recognized as true edges and candidate edges that are connected to the true edges by a path of pixels with gradient magnitudes higher than the lower threshold are also recognized as true edges. This idea can be readily applied to our skeleton extraction by simply treating the skeletons as the edges. Figure 3.4(d), (f), and (h) show the skeletons traced by this method from the skeleton strength maps shown in Figure 3.4(c), (e), and (g), respectively.

There is a more sophisticated method, based on the local structure tensor [22, 66], for tracing the skeletons.

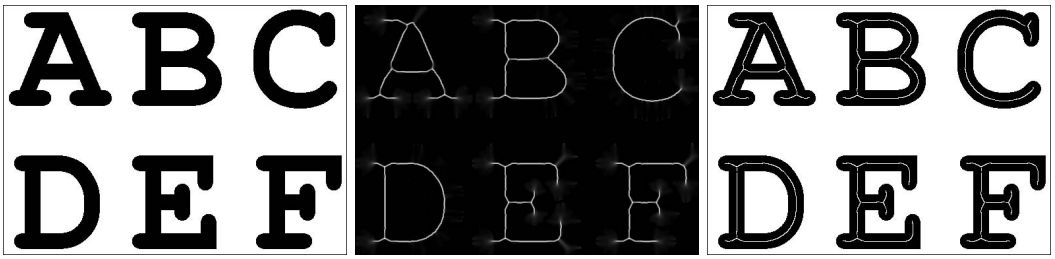
Results and Discussions

In this section we show several examples and their skeletonization results extracted by our algorithm. All vectors are diffused by the anisotropic vector diffusion (Equation 3.2) using the finite difference scheme. The *skeleton strength maps* are calculated according to Equation 3.7 without vector normalization, unless otherwise specified.

Figure 3.5(a) shows a brain image. Its edge strength map (or gradient magnitude map) is first computed using the derivative of Gaussian kernel function (Figure 3.5(b)). Equation 3.6 is applied to this map to generate an initial vector field. Ten iterations are applied to solve Equation 3.2. The *skeleton strength map*, shown in Figure 3.5(c), is calculated from the diffused vector field using Equation 3.7. Finally, the skeletons (Figure 3.5(c)) are traced from the skeleton strength map using Canny's method. The skeletons here correspond to both the bright features and dark features.

(a) (b) (c)

Figure 3.6: The demonstration of skeletonization on a brain MRI image. (a) Original image. (b) Skeleton strength map. (c) Skeletons traced (in red color).



(a) (b) (c)

Figure 3.7: The demonstration of skeletonization on an image of several English characters. (a) Original image. (b) Skeleton strength map. (c) Skeletons traced.

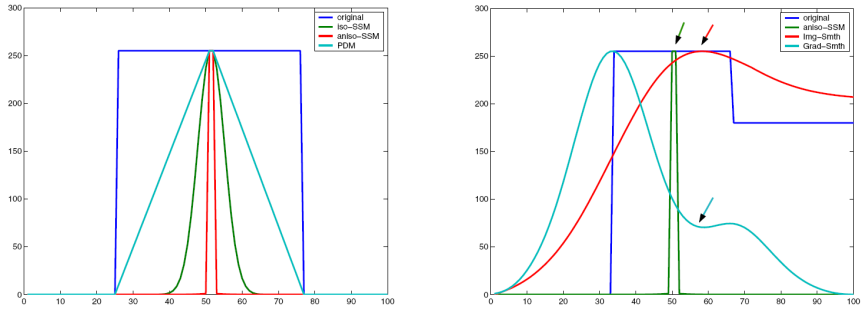
Figure 3.6(a) shows another brain image. By the fact that the brain structures have higher intensities than the background, we skip the calculation of the edge strength map, and directly apply Equation 3.5 to the original image in order to generate the initial vector field. Our anisotropic vector diffusion scheme (Equation 3.2) is then applied with twenty iterations repeated to solve the PDE. The *skeleton strength map*, shown in Figure 3.6(b), is calculated from the diffused vector field using Equation 3.7, and the final skeletons are shown in Figure 3.6(c).

We also demonstrate our skeletonization on a binary image consisting of several characters. Although our algorithm is designed for gray-scale images, it can certainly be applied to binary images. Figure 3.7(a) shows an image of characters. The *skeleton strength map* and the final skeletons are shown in Figure 3.7(b) and (c), respectively. Note that, since the objects (characters) are darker than the background, we use Equation 3.6 to generate the initial vector field. Twenty iterations are applied to solve the PDE using the finite difference scheme.

The skeletonization algorithm consists of several parts: gradient vector initialization and diffusion, skeleton strength map calculation, and skeleton tracing. The time complexity of the first part is $O(N \times M \times K)$, where N is the size of the given image (or the number of pixels), M is the size of neighborhood (usually 3×3), and K is the number of iterations during the diffusion process. The time complexity of computing skeleton strength maps is $O(N \times M)$, where N and M are defined above. The skeleton tracing, if Canny's method is used, has a time complexity of $O(N)$, where N is the size of the given image.

We mentioned before that the pseudo-distance map (PDM) [29] was most related to our algorithm. In order to see the difference between the PDM and our skeleton strength map (SSM), we consider a simple 1D function, as colored by blue in Figure 3.8(a). This is a box function with same contrast on both sides. The pseudo-distance map of this function is given by light blue color in Figure 3.8(a). The skeleton strength maps, generated by isotropic vector diffusion (Equation 3.1) and anisotropic vector diffusion (Equation 3.2), are given by green and red curves, respectively, in Figure 3.8(a). We can see from this example that our skeleton strength maps give "sharper" ridges. We can also observe that the anisotropic diffusion performs better than the isotropic diffusion.

Finally let us address the bias problem, as we mentioned earlier. For this purpose, we consider a (1D) box function with different contrasts on both sides, as colored by blue in Figure 3.8(b). The green curve shows the results of our skeleton strength



(a) (b)

Figure 3.8: Two tests: (a) Comparison between PDM and SSM. (b) Removal of bias problem by our method.

map (anisotropic), from which we can correctly locate the skeleton at the center of the box, as indicated by green arrow in the figure. In contrast, we consider two classical linear methods. One is the direct convolution of a Gaussian kernel with the original data (e.g., see [39, 54, 62]). The result of the convolution is shown by red curve in Figure 3.8(b). The other linear system, as used in [63], is more general, where the edge strength map is first generated and diffused. The skeletons are then extracted from certain critical points of the diffused edge strength map (e.g., see [63]). The result of the diffused edge strength map is shown by light blue color in Figure 3.8(b). From this figure, we can conclude that both linear methods yield biased skeletons, as indicated by red and light blue arrows. The skeletons "shift" to the side with lower contrast. The bias problem was discussed in details in [62], where a lot of efforts have to be made to remove the bias from the extracted skeletons. From the above description, we can see that this problem can be easily solved using our anisotropic vector diffusion.

Conclusion

In this section, we described various types of anisotropic vector diffusions, from which both critical points and skeletons can be extracted. The skeletons of a gray-scale image are detected in our approach without any boundary information. We introduced a concept, called *skeleton strength map*, which is analogous to the edge strength map as seen in [63]. Our skeleton strength map gives much "sharper" ridges of the original images than the pseudo-distance map [29]. We also addressed the removal of bias as commonly seen in many linear skeletonization approaches. Our approach for skeleton extraction is treated in a similar way to the edge detection.

The following algorithm is taken from [76] (Yu:07).

Introduction

X-ray crystallography [67, 1] and nuclear magnetic resonance (NMR) [21, 27] are two widely used techniques that are used to reveal the structure of individual proteins. These structural models are then deposited in the Protein Data Bank (PDB) [10]. While individual proteins provide important structural information about our bodies' fundamental blocks, the structural determination of large biological complexes (e.g., viruses, ion channels, and the ribosome) offer a more complete description of the protein machinery of life. Detailed structural knowledge of these complexes not only provide the mechanistic descriptions of how macromolecules interact in an assembly, but also yield clues for developing therapeutic interventions related to disease. While x-ray crystallography and NMR spectroscopy are quite often restricted to relatively small biological structures, cryo-electron microscopy (cryo-EM) of single particles has emerged as a powerful technique in revealing the ultra-structure of large bio-molecular complexes [9, 64]. The rapid development of improved image acquisition and sophisticated computational signal and image processing methods have now made it possible to resolve these large biological complexes at sub-nanometer resolutions (6Å-10Å) [79, 32, 78, 46].

Computational signal and image processing have been extensively used in 3D biological structure reconstruction and at multiple scales (tissue, cell, molecular) [refer to IEEE image proc. special issue]. In particular, most modern signal/image processing algorithms including 2D image restoration, noise reduction, contrast enhancement, feature detection, alignment, classification, 3D reconstruction, boundary segmentation, and skeletonization. Although several 2D algorithms (and briefly reviewed in Section II), have been utilized in single particle cryo-EM processing software packages (e.g., EMAN [43], SPIDER [24], IMAGIC [28]), quantitative and automatic analysis/interpretation techniques operating on the reconstructed 3D electron density maps of bio-molecular assemblies, remains largely undeveloped. Current methods for interpreting reconstructed 3D maps depend primarily upon manual selection and visual inspection, and with the help of interactive graphic tools. Due to the large physical size and structural complexity of the bio-molecular assemblies, however, mechanical interactive processing is tedious, and subjective. Automatic structural interpretation and analysis of 3D maps of large bio-molecular complexes has thus become the preferred avenue of research and development.

In prior work [75], we presented an automatic approach to segmenting the reconstructed 3D maps of bio-molecular complexes into dozens to thousands of individual structure units. This automatic segmentation without doubt makes it considerably easier to determine the structural interactions of the bio-molecular assembly. Furthermore, with segmentation we are able to isolate the various grouped protein structure units (also called conformers), and interpret the ultra-structure of each of the proteins individually. For symmetric structures, such as most protein capsid shells of viruses [Baker=ref], the segmentation of the entire capsid into asymmetric protein structure units also helps eliminate structural representation redundancy. In this paper, we present two additional automatic algorithms of protein structure interpretation from reconstructed 3D Maps. The first algorithm is for automatic spatial alignment, and computation of structural similarity score, between two 3D map segmented structure units. The second algorithm automatically identifies and locates secondary structure units (for e.g., protein α-helices, β-sheets) within a segmented protein structure unit. The accuracy and hence the need for the identification of protein α-helices and β-sheets increases correspondingly with the increased rise in resolution and availability of reconstructed 3D maps at sub-nanometer resolution (e.g., better than 10Å) [79]. Clearly, the better the understanding of the macromolecular ultra-structure the better the ability to determine its structure-function relationships.

The rest of this paper is organized as follows. Section II provides a brief introduction to 3D electron microscopy imaging and 3D map reconstruction. In Section III we our automatic 3D structure unit alignment and similarity scoring algorithm, along with with applications to related problems. We present our algorithm of automatic protein secondary structure identification and localization in Section IV, along with several example results of our implementation on both reconstructed and simulated 3D Maps.

3D Cryo-Electron Microscopy Imaging and Reconstruction

Electron microscopy (EM) imaging has been extensively used in structural biology to study the activities of cells and organelles. Three-dimensional electron microscopy (3D-EM) imaging plays a unique role in structural biology, thanks to its remarkable capability to reveal the three dimensional structure of biological entities. The mathematical principles of 3D-EM reconstruction

from projection data (experimental EM images) is basically the same as that used in 3D Computed Tomography (CT) medical imaging. The major difficulty with EM images is the extremely low signal-to-noise ratio (SNR). This is true partly because the electron dose used in EM imaging has to be kept to an extremely low level (approximately 0.5$\sim$4 e/Å^2) in order to reduce the radiation damage to the specimen. The flash cooling technique, known as cryo-EM, quickly cools the samples under study, to liquid nitrogen temperature (about 77 Kelvin or less) such that the surrounding water does not form crystalline ice, but remains in a vitreous state. Thus cryo-EM has proved to be quite successful in preserving the native in-vivo structure of biological specimens while reducing the damage caused to the specimen, and hence its growing popularity in structural Biology[9, 14].

Using different sample preparations and data collection methods, 3D-EM encompasses three major techniques: electron crystallography, electron tomography, and the single particle cryo-EM method. Electron crystallography [2], is similar to x-ray crystallography, and can reveal the bio-molecular structures at near atomic resolution. However, the weakness of this technique is that a two-dimensional crystal has to be grown for cryo-EM imaging, which in many cases is difficult to do, especially for macromolecules. Electron tomography [47, 37], is also similar to CT, and is used to study 3D ultra-structures of cell organelles or whole cells at relatively low resolutions. Mathematically, the 3D structure of a cell specimen can be reconstructed from a series of 2D projections generated from different tilt angles. There are several methods to collect the projection data: single-axis tilt, double-axis tilt, and uniform conical tilt, depending on how the specimen is rotated under the fixed camera. However, the tilt cannot exceed certain angles (usually $\pm 70^0$) due to the relative orientation of the tilt-stage and the specimen thickness limitations [52, 23]. For this reason, the reconstructed density maps always have significant distortions in certain regions, commonly known as the missing wedge (pyramid, or cone) problem. Partly due to this problem, and partly because of the limited electron dose used on a single specimen with consideration to the radiation damage, the resolution of this type of reconstructions is often limited in the range of 20Å$\sim$ 200Å [47, 37].

The above resolution problems could be resolved if we had a number of structurally identical particles at multiple random orientations and found a way to align and average them together to yield a single 3D structural image. On the one hand, the average of particles in the same orientation (after proper alignment) can improve the signal-to-noise ratio (SNR). On the other hand, particles in different orientations are very likely to complement each other such that the missing wedge (pyramid, or cone) of projection orientations of one particle could be filled by the other particles. This technique, known as *single particle cryo-EM reconstruction*, has been used to resolve 3D structures at about 6Å [79, 44]. To reduce the radiation damage, each particle is imaged only from one tilt angle and thousands (and more) particles are used to reconstruct a single 3D structure. Fig. 3.9(a) shows the overall pipeline of single particle structure reconstruction and analysis. Starting from 2D digitized microscopy images, the 3D structure Map reconstruction includes several major steps:

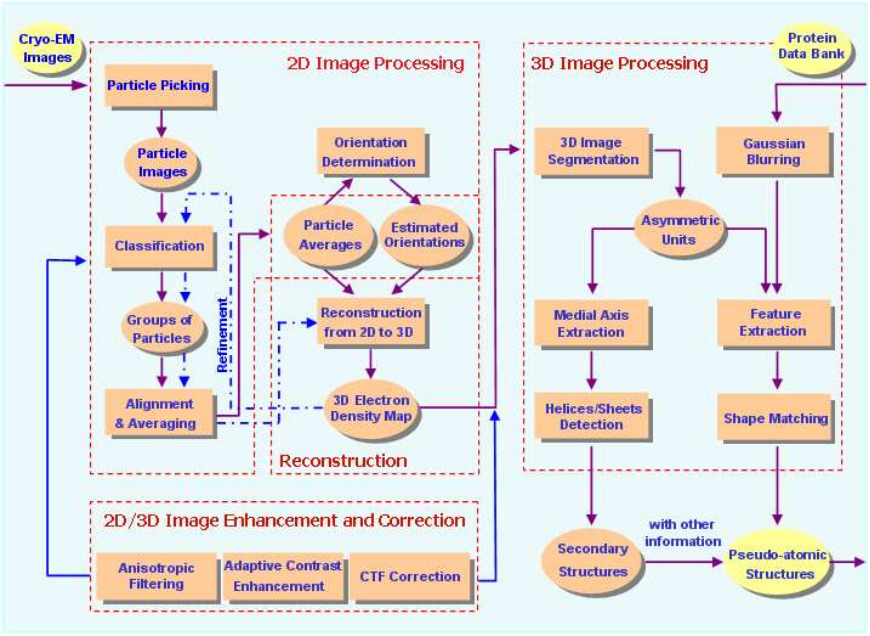
- *Particle Picking*. The goal of particle picking is to box out all particles that look reasonably good as projection of particles, rather than noise, in both size and shape [55, 72, 50]. Fig. 3.9(b) shows a small portion of the electron micrograph of the rice dwarf virus [79], from which we can see how noisy a typical particle image looks.

- *Particle Classification and Alignment*. Particle classification is to group all the particle images that have the same appearance but do not have to be in the same orientation. The classified particles can be aligned and averaged such that the class averages have significantly improved signal-to-noise ratios (SNR) [23, 60, 61].

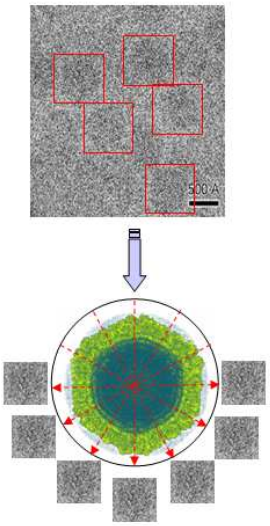
- *Orientation Assignment*. Since in most cases the particles appear in "in-vivo" random orientations, we do not know the orientation of each particle image. To assign the orientations, there are a direct methods based on the *common line theorem* [64] and an iterative method based on an initial 3D model [43].

- *Reconstruction and Refinement*. The mathematical theorem is well established for 3D reconstruction from 2D projections, given that the orientation of each projection is known. The most popular methods include the direct Fourier space reconstruction [43] and the real-space filtered back projection method [64]. Fig. 3.9(b) illustrates the 3D reconstruction from a series of projections at different angles. The reconstructed 3D map can be used as an initial model to refine the particle classification, alignment, and orientation assignments.

The reconstructed 3D maps do not convey meaningful information unless they are correctly interpreted. The ultra-structure of the 3D maps can be interpreted in two ways as shown in Fig. 3.9(a). For maps at intermediate resolutions (6Å-10Å), the secondary structures are visually identifiable and computationally auto-detectable. A pseudo-atomic model can be built based on the detected secondary structure elements and their topological connections [79]. When the resolution of the reconstructed 3D maps degrade beyond 10Å, however, we cannot discern the secondary structure elements with high confidence but we still can attempt to construct a pseudo-atomic model by matching and fitting a high-resolution x-ray structure model (from the PDB)



(a) Pipeline of the single particle reconstruction and analysis



(b) An example

Figure 3.9: Illustrations of single particle cryo-EM technique. (a) The overall pipeline, including particle picking, particle classification/alignment, orientation assignment, 3D reconstruction, and map interpretation. In addition, EM contrast transfer function (CTF) correction [30], anisotropic filtering [66], and adaptive contrast enhancement [73] may also be applied before or after 3D reconstructions, to improve the signal-to-noise ratio. (b) An example of electron microscopy (EM) images showing particle picking (top) and 3D reconstruction from 2D projections (bottom).

into the 3D map based on a density distribution and correlation [68]. In either case, the prior segmentation of the 3D map into individual protein structure units is both meaningful and necessary for fast and accurate ultra-structure interpretation [75].

In the subsequent sections, we present computational approaches for further automatic structural interpretation of 3D Maps, namely, (a) an automatic 3D structure unit alignment coupled to structural similarity scoring and (b) automatic protein secondary structure identification and localization.

3D Structure Alignment

The goal of 3D structure alignment is to find the transformation matrix from one 3D structure unit to another, such that the two 3D maps are best matched according to a similarity scoring function. In the following we first define the similarity scoring we use between two 3D maps and then present a fast algorithm for computing the transformation matrix aligning two 3D maps of structure units. We additionally present a couple of further applications of this structural alignment.

Similarity Scoring Function A traditional similarity scoring function between two 3D maps, denoted by f and g, is defined by cross-correlation as follows:

$$S^1_{f,g}(T) = \sum_{i,j,k} f(i,j,k) \times g(T(i,j,k)), \tag{3.9}$$

where T is the transformation matrix from f to g. The 3D structural alignment problem is then reduced to determining the best T, given two maps f and g, such that the scoring function $S^1_{f,g}(T)$ achieves its maximum. It is easy to see that maximizing $S^1_{f,g}(T)$ is equivalent to minimizing the square difference between f and g:

$$S^2_{f,g}(T) = \sum_{i,j,k} (f(i,j,k) - g(T(i,j,k)))^2. \tag{3.10}$$

The advantage of the cross-correlation method is that the fast Fourier transform (FFT) can be deployed to speed up the search in the relative translational space (three degrees of freedom). However, the search in rotational space has to been performed in a conventional way. As we shall see, our 3D alignment of two structure units involves both translation and rotation, however each of these is restricted to a single degree of freedom search. Furthermore, we define the similarity scoring function in real space, and to speed up the search, we compute the similarity score on a set of critical points of the 3D maps, instead of the entire collection of voxels that make up the 3D maps of the structure units. The critical points are those that best capture the topological features of a molecular density 3D map. In most cases, the critical points can be defined by the local maxima, local minima, and saddle points of the scalar 3D map. If the data is noisy, one initially preprocesses the 3D map using gradient vector diffusion as discussed in [75].

Our similarity scoring function is thus defined in the following way:

$$S^3_{f,g}(T) = 1 - \frac{\sum_{m=1}^{M} |f(c_m) - g(T(c_m))| + \sum_{n=1}^{N} |f(T^{-1}(d_n)) - g(d_n)|}{\sum_{m=1}^{M} max\{f(c_m),\ g(T(c_m))\} + \sum_{n=1}^{N} max\{f(T^{-1}(d_n)),\ g(d_n)\}}, \tag{3.11}$$

where $c_m, m = 1,2,\cdots,M$, are critical points of f and $d_n, n = 1,2,\cdots,N$, are critical points of g. Fig.3.10 illustrates the idea of this similarity scoring function. It is worth noting that there is no direct relationship between the number of critical points and the size of the map − a big map with slowly-varying densities may have a small number of critical points, while a small map with sharp density variations can have a large number of critical points. There are two major differences between $S^3_{f,g}(T)$ and $S^2_{f,g}(T)$. First, the new similarity scoring function is based only on the critical points. Therefore, the search for the best T using $S^3_{f,g}(T)$ is much faster than the search using $S^2_{f,g}(T)$, as the latter is based on all the voxels of f and g. Secondly, the scoring function of $S^3_{f,g}(T)$ is normalized such that the similarity scores are always scaled to the range of [0, 1], where 0 means no similarity and 1 signifies the highest similarity.

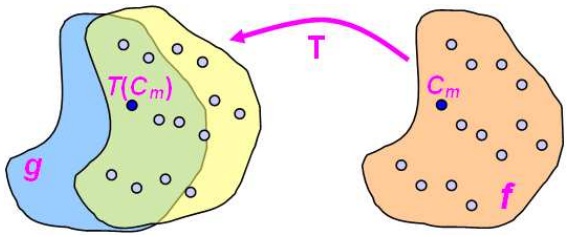


Figure 3.10: Illustration of similarity calculation based on critical points. Every critical point c_m, $(m = 1,2,\cdots,M,)$ in f is transformed to g according to T and the difference between the densities at c_m and its transformed position $T(c_m)$ is calculated and summed. If the two maps are identical and the matching is perfect, the total difference should return zero. Accordingly, we calculate the total difference between the densities at $d_n, n = 1,2,\cdots,N$, in map g and their transformed positions in f. The final normalized similarity score is given in Equation 3.11.

3D Alignment Algorithm In our previous paper [75] we discussed how to automatically detect the local symmetry axes of protein conformers in a reconstructed 3D density map of a macromolecule (eg virus capsid), and based on which we segment out, each of the individual locally symmetric structure units, that comprise the building blocks of that macromolecule. Given two such individual, segmented, structure units A and B, (as shown in blue and magenta respectively in Fig. 3.11), our problem is to structurally align A to B. Given the segmented structure units A and B and the symmetry axes computed in [75], we transform A to B in four steps:

[A] Translate A by t_0.

[B] Rotate A by r_0.

[C] Translate A by t.

[D] Rotate A by r.

Since the symmetry axes of both A and B are given, the first two of the above transformations, the translation t_0 and the rotation r_0, are uniquely determined by the relative position/orientation of the symmetry axes of A and B. More specifically, since each symmetry axis is given by two end points, we translate A from S_A to S_B, followed by the rotation between the two dashed blue arrows in Fig. 3.11. However, the translation t and the rotation r have to be decided based on the similarity scores between the density maps of A and B as discussed in Section 3.3. Therefore, the transformation from one structure unit to another has two degrees of freedom: one translation and one rotation, as illustrated in Fig. 3.11.

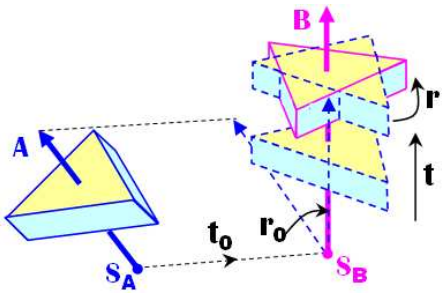


Figure 3.11: The alignment between two structure units include translations t_0 and t, and rotations r_0 and r. But only t and r are unknown and need to be determined based on the similarity scoring function.

Putting the four transformation matrices together, we have the following matrix that transforms subunit A to subunit B:

$$T(t,r) = M_4(r) \times M_3(t) \times M_2(r_0) \times M_1(t_0) \tag{3.12}$$

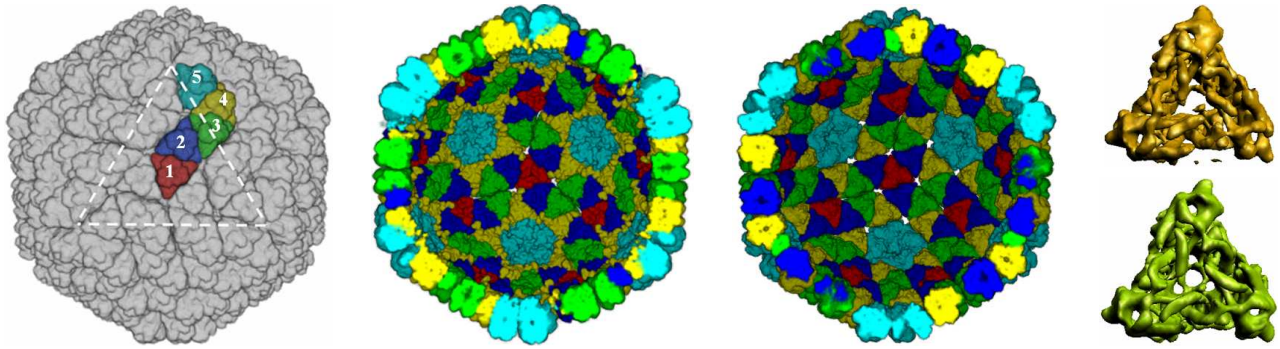
The matrices M_1, M_2, M_3, M_4 are conventional transformation matrices for translations or rotations. One can easily derive the exact expressions for these matrices based on the given information (e.g., symmetry axes with start/end points). Substituting equation 3.12 into equation 3.11, we have the following maximization equation for the 3D alignment between structure units A and B, with density functions f and g respectively:

$$\max_{\{t,r\}}\{S^3_{f,g}(t,r) = 1 - \frac{\sum_{m=1}^{M} |f(c_m) - g(T_{t,r}(c_m))| + \sum_{n=1}^{N} |f(T_{t,r}^{-1}(d_n)) - g(d_n)|}{\sum_{m=1}^{M} max\{f(c_m),\ g(T_{t,r}(c_m))\} + \sum_{n=1}^{N} max\{f(T_{t,r}^{-1}(d_n)),\ g(d_n)\}}\}, \tag{3.13}$$

where $T_{t,r}$ is the transformation matrix from A to B as defined in equation 3.12. Since the scoring function defined in equation 3.13 indicates the similarity between two structure units, the transformation matrix that maximizes equation 3.13 should give the best alignment between the two structure units. The range of t is user-defined. In our experiments we assumed that $t \in [-10, 10]$ in pixel unit. The range of r is from 0^0 to 360^0 for general objects. However many of the reconstructed 3D maps from single particle cryo-EM are virus capsid structures, which have typically icosahedral or helical symmetries. Therefore, the segmented structure units usually have n-fold symmetry and the range of r should be from 0^0 to $\left(\frac{360}{n}\right)^0$.

There are a number of optimization techniques to find the $\{t,r\}$ that maximize the similarity score between A and B as defined in Equation 3.13. In order to find the best $\{t,r\}$, we need to calculate the similarity score for every sample taken from the appropriate ranges for t and r as discussed above. Dense samplings could yield more accurate t and r but require more computational time. In contrast, coarse samplings are faster but less accurate. We adopt a simple two-level hierarchical method for our search. On the coarse level, the translation variable t is sampled by every one pixel unit from -10 to 10 and the rotation variable r is sampled by every 5^0 from 0^0 to $\left(\frac{360}{n}\right)^0$ where n is the fold number of the symmetry. For each combination of these t and r, we calculate the similarity score and the one that maximizes the scoring function is the solution of equation 3.13 on the coarse level, denoted by $\{t^{(1)}, r^{(1)}\}$. On the fine level, the sampling is taken within a small range around $t^{(1)}$ and $r^{(1)}$. For the translation, the sampling is taken on the interval $[t^{(1)} - 0.9, t^{(1)} + 0.9]$ by every 0.1 pixel unit. The sampling for rotation is on the interval $[r^{(1)} - 4^0, r^{(1)} + 4^0]$ by every 1^0. Again, the similarity score for each of these samples is calculated and the best one, denoted by $\{t^{(2)}, r^{(2)}\}$, gives the final solution of equation 3.13. By substituting $\{t^{(2)}, r^{(2)}\}$ into Equation 3.12, we have the transform matrix $T(t^{(2)}, r^{(2)})$, which gives the 3D alignment from subunit A to B. A couple of applications of this 3D alignment are presented next.

Applications We consider two examples in this section. The first one is the rice dwarf virus (RDV), which has icosahedral symmetry at a resolution of 6.8Å[79]. There are five independent structure units with 3-fold local symmetry, also known as trimers, as shown in Fig.3.12(a). The second example is the bacteriophage $\phi 29$ at a resolution of 15Å[48]. This map has a 5-fold global symmetry imposed along the vertical axis. As shown in Fig. 3.13(a), there are ten independent structure units including one tail on the bottom (structure unit #3), three 5-fold structure units or pentons (#0, #1, #2), and six 6-fold structure units or hexons (#4 $\sim$ #9).



(a) Segmented trimers (b) Old segmentation (c) New segmentation (d) Trimers

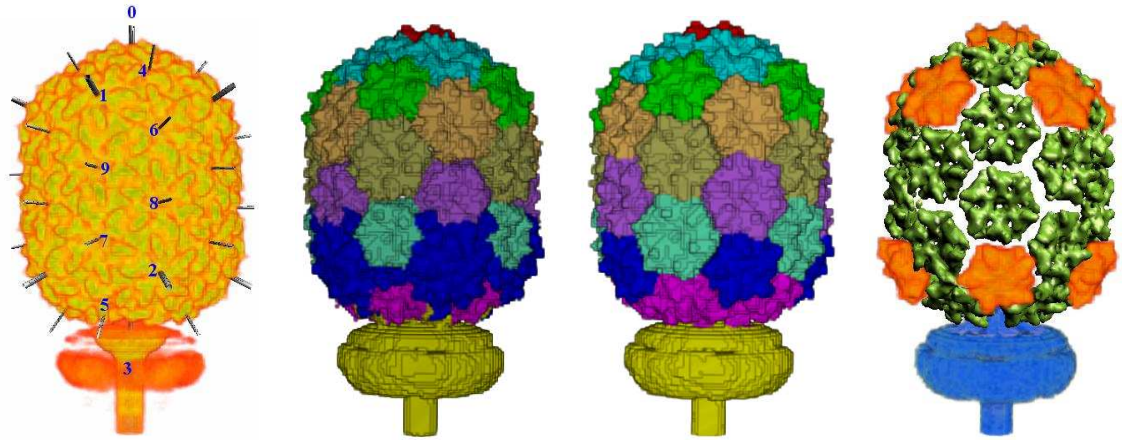
Figure 3.12: Illustrations of segmentation and averaging on rice dwarf virus (RDV). (a) The five independent trimers arranged on the virus capsid (viewed from outside). (b) The segmentation results without structure unit alignment (viewed from inside). (c) The segmentation results with structure unit alignment showing better consistency between structure units. (d) The averaged trimer (in green) looks less noisy and more symmetric than the original trimer (in golden).

Improvement of Segmentation

Our previous segmentation algorithm [75] is based on the multi-seeded fast marching method [59]. In this method each subunit is assigned with an initial contour which keeps growing according to a pre-defined speed function and eventually stops on the boundaries between structure units. When the structure is symmetric, some of the structure units are dependent (identical) and hence the contours corresponding to those structure units must grow simultaneously by the same amount in the same way. With this constraint, our previous segmentation [75] incorporates global symmetry (for example, icosahedral for RDV and 5-fold for $\phi 29$) and the local n-fold symmetry of each subunit into the segmentation. However, we did not consider the structural similarity between the independent structure units. In other words, the contours corresponding to independent structure units grew independently. This can sometimes yield noticeable errors especially when the boundaries between structure units are indistinguishable. An example is shown in Fig.3.12(b). This is the segmentation of the outer layer of the rice dwarf virus (RDV) and the five types of trimers are colored differently (see Fig.3.12(a) for their relatively locations). We can see that the structure units in different colors are not consistent to each other. For example, the yellow structure units occupy more space than the others. To remedy this problem, we utilize the 3D structure unit alignment as discussed in Section 3.3, such that the contours within all the structure units grow simultaneously by the same amounts in the properly-aligned directions. The results with this additional constraint are shown in Fig. 3.12(c). We can see that the structure units in the new results look much more uniform and consistent than those seen in Fig.3.12(b). The segmentations of $\phi 29$ are given in Fig. 3.13(b) without alignment and in Fig. 3.13(c) with alignment. We can also see that the alignment-based segmentation demonstrates much better consistency between the segmented structure units. However, it should be pointed out that the structure units must be segmented before we can align them. Therefore, our previous segmentation algorithm [75] is still fundamental in providing us with the initial segmentation from which the structure units can be aligned.

Quantifying 3D Structural Similarity

Another application of our 3D alignment algorithm is a similarity measure between structure units. This is functionally very important because it provides us with the structural similarity quantitatively between any two independent structure units. It is computationally straightforward to obtain a similarity score. In fact, it is given by the maximum of the similarity scoring



(a) Density map (b) Old segmentation (c) New segmentation (d) Averaged map

Figure 3.13: Illustrations of segmentation and averaging on $\phi 29$. (a) The reconstructed cryo-EM density map of $\phi 29$. The symmetry axes are detected using our automatic method [75] and a total of ten independent structure units are labeled. (b) The structure units are segmented based on our previous segmentation method, without 3D alignment between structure units. One can clearly see some inconsistency between either pentons or hexons. For example, the structure units in blue are much larger than those in green although both are pentons. (c) The structure units are segmented based on our previous segmentation method, combined with the 3D alignment approach as presented in Section 3.3. (d) The segmented pentons and hexons can be averaged and the whole structure can be reconstructed from the averaged structure units based on the 3D alignment matrices.

function $S^3_{f,g}(t,r)$ when Equation 3.13 is optimized. In Table 3.1 we give the similarity scores between the five independent trimers as shown and indexed in Fig. 3.12(a). The scores in bold on the diagonal indicate how symmetric each individual trimer is, and they are calculated by equation 3.11 where T is the rotation along the related symmetry axis by an amount of $\frac{2\pi}{3}$ (in general, $\frac{2\pi}{n}$, where n is the fold number). Table 3.2 shows the similarity scores between the four 5-fold structure units of $\phi 29$, where the indexing numbers are given in Fig. 3.13(a). Since the tail structure unit (#3) is obviously different from the other three 5-fold structure units, the similarity scores between the tail and the others are very low. However, the tail still shows a high 5-fold symmetry since it locates exactly on the global 5-fold symmetry axis. In Table 3.3 we show the similarity score between the six hexons. The subunit #5 seems less similar to the others because it is close to and very likely "disturbed" by the tail subunit.

Table 3.1: Similarity scores between the five trimers of RDV

	S_1	S_2	S_3	S_4	S_5
S_1	**0.955**	0.911	0.848	0.900	0.894
S_2		**0.926**	0.854	0.889	0.880
S_3			**0.872**	0.848	0.845
S_4				**0.934**	0.885
S_5					**0.856**

Table 3.2: Similarity scores between the four pentons of $\phi 29$

	S_0	S_1	S_2	S_3
S_0	**0.991**	0.948	0.949	0.344
S_1		**0.960**	0.958	0.306
S_2			**0.961**	0.314
S_3				**0.991**

Table 3.3: Similarity scores between the six hexons of $\phi 29$

	S_4	S_5	S_6	S_7	S_8	S_9
S_4	**0.973**	0.793	0.947	0.940	0.869	0.880
S_5		**0.741**	0.793	0.785	0.772	0.787
S_6			**0.971**	0.963	0.884	0.884
S_7				**0.965**	0.888	0.884
S_8					**0.829**	0.938
S_9						**0.842**

Averaged 3D Map of Structure units

If the segmented structure units of a large bio-molecular complex (such as viruses) have high similarities, they can be averaged such that the structural analysis of the entire complex can be simplified to the analysis of a single averaged structure unit. Apparently averaging several structure units is straightforward if the alignment between any two of them is known. In general, the averaged map has a higher signal-to-noise ratio than each individual structure unit, which makes it easier to further analyze the structures (e.g., secondary structure identification, as will be discussed). Fig. 3.12(d) shows one segmented trimer of RDV (in golden) and the averaged map of the five independent trimers (in green). We can see that the averaged trimer has higher 3-fold symmetry and better signal-to-noise ratio. Fig. 3.13(d) shows the map constructed from the tail (blue), the averaged penton (orange) and the averaged hexon (green).

Secondary Structure Identification

Proteins typically contains two types of secondary structure elements: α-helices and β-sheets. There have recently been a few papers published on secondary structure identification. An approach for detecting α-helices has been described in [31], where the α-helix is modeled as a cylinder (length and thickness) with a Gaussian distribution density function for each cylinder. The cylinders are then cross-correlated with the segmented protein map, in an exhaustive search and scoring procedure. The exhaustive search occurs in both translation space (three degrees of freedom) and orientation space (two degrees of freedom), and necessarily is computationally expensive [at least O($n^6 * N * K$) where n is the discretization of translational and rotational space and N is the size of the 3D protein density map, for an O(K) scoring computation time]. A related approach, designed for β- sheet detection, was recently presented in [35]. This method uses a planar disk model for beta sheets, and performs exhaustive cross-correlation search with the protein density function. This method [35] also inherits a similar disadvantage of high computational time complexity as of [31], due to the similar exhaustive search, and cross-correlation scoring, in six dimensional translation and orientation space.

In the following sub-sections, we present a fast skeleton-based approach for protein secondary structure identification from 3D maps. We capture the location of α-helices and β-sheets by their skeletons (curves and surfaces, respectively). Our skeletonization method is based on a prior seed selection from the 3D density map, and a tracing procedure guided by the eigenvectors of the local 3x3 structure tensor at select points of the protein density map. With the help of eigen-analysis of local structure 3x3 matrices, we can extract the entire "linear" α helical and "planar" β-sheet skeletons without exhaustively searching the relative six dimensional translation and rotation space of a geometric template and the 3D protein density map. Compared to the above prior methods, our approach is extremely fast (hundreds of times faster for traditional protein 3D maps, as it doesn't involve the O(n^6) time complexity of the prior methods) and additionally yields high accuracy identification of secondary structure elements3.3.

Local Structure Tensor Local structure tensor have been used in image processing for solving a number of problems such as anisotropic filtering [22, 66] and motion detection [33]. Given a 3D map $f(x,y,z)$, the gradient tensor is defined as:

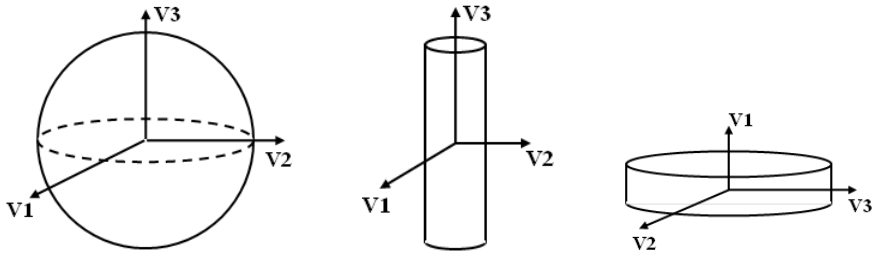
$$G = \begin{pmatrix} f_x^2 & f_x f_y & f_x f_z \\ f_x f_y & f_y^2 & f_y f_z \\ f_x f_z & f_y f_z & f_z^2 \end{pmatrix} \tag{3.14}$$

This matrix has only one non-zero eigenvalue: $f_x^2 + f_y^2 + f_z^2$. The corresponding eigenvector of this eigenvalue is exactly the gradient (f_x, f_y, f_z). Therefore, this matrix alone does not give more information than the gradient vector. To make the gradient tensor useful, a spatial average (over the image domain) is computed for each of the entries of the gradient tensor, yielding what is called the *local structure tensor*. The averaging is usually based on a Gaussian filter:

$$T_\alpha = \begin{pmatrix} f_x^2 * g_\alpha & f_x f_y * g_\alpha & f_x f_z * g_\alpha \\ f_x f_y * g_\alpha & f_y^2 * g_\alpha & f_y f_z * g_\alpha \\ f_x f_z * g_\alpha & f_y f_z * g_\alpha & f_z^2 * g_\alpha \end{pmatrix} \tag{3.15}$$

Here g_α is a Gaussian function with standard deviation α. The eigenvalues and eigenvectors of the structure tensor T_α indicate the overall distribution of the gradient vectors within the local window or 3D sub-domain, and is similar to the well-known principal component analysis (PCA). Three typical structures can be characterized based on the eigenvalues of this structure tensor [22]. Let the eigenvalues be $\lambda_1, \lambda_2, \lambda_3$ and $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Then we have the following cases (see Fig.3.14):

- Spheres: $\lambda_1 \approx \lambda_2 \approx \lambda_3 > 0$.
- Lines: $\lambda_1 \approx \lambda_2 >> \lambda_3 \approx 0$.
- Planes: $\lambda_1 >> \lambda_2 \approx \lambda_3 \approx 0$.



(a) Spheres (b) Lines (c) Planes

Figure 3.14: Three typical cases for a local structure tensor.

Skeletonization There are a number of prior skeletonization approaches that have been published, including boundary-based methods [38, 51] and boundary-free methods [29, 74]. While a pre-segmentation is required in boundary-based methods, boundary-free methods do not have this requirement and so the skeletons are in general extracted by a volume thinning or a skeleton-tracing step. In the following, we present our method to trace the skeletons, based on a selection and classification of skeleton-seeds, and guided by eigenvectors of local structure tensors of the 3D density map.

Selecting and Classifying Seeds

As discussed in Section 3.3, the local structure tensor can be used to distinguish between spherical, linear, and planar structures. The spherical case is usually of little interest in skeletonization. Therefore, we focus on the other two cases in the following. As we shall see in Section 3.3, these two types of local structures exactly correspond to the alpha-helices and beta-sheets of a protein density map.

Seeds are the starting points for tracing the skeletons. There are multiple ways to define/compute the seed points, with the common condition that all the seed points must be on the skeleton. As we shall see, the graph of the 3D protein density maps are similar to 3D mountain ranges − with high densities (features) corresponding to the peaks and ridges, with the skeleton extraction akin to ridge tracing. For this particular reason, we select the maximal critical points of the 3D density maps as the seed points of our skeleton-tracers. The stable computation of these critical (seed) points has been detailed in our previous work [75].

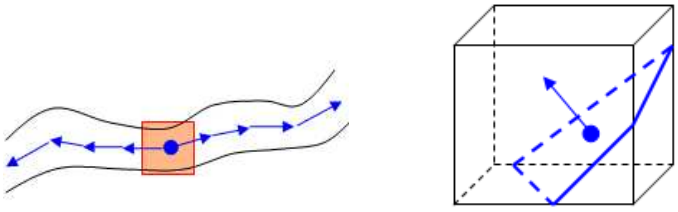
Since we are dealing with two types of skeletons: linear and planar, for which the skeleton-tracings are different, we need to classify each maximal critical point into either "linear" or "planar" seeds before tracing the skeletons. In [22], the authors used the three real eigenvalues of the real symmetric structure tensor to distinguish lines from the planes. However, this criterion does not work well for protein secondary structures because some parts of proteins (e.g., coils, turns) look like helices except that they are thinner. Therefore, a better way to classify them is to use the thickness of the secondary structure elements along three principal axes. The thickness along any direction is defined by the width of the region above a pre-chosen threshold in that direction. Since we know the typical thickness of a helix, the threshold values can actually be determined automatically from the seed points that correspond to the helices based on the initial linear classification using the eigenvalue criterion in [22]. Once we know the thickness information for each seed point, we classify the seeds into "linear" and "planar" according to a range criterion. Let t_1, t_2, t_3 be the thicknesses corresponding to the eigenvectors as shown in Fig.3.14. The following ranges are then used to classify the seeds:

- lines: $\frac{t_1+t_2}{2} > helix_{min}$ and $\frac{t_1+t_2}{2} < helix_{max}$ and $min(\frac{t_1}{t_2}, \frac{t_2}{t_1}) > max(\frac{t_1}{t_3}, \frac{t_2}{t_3})$,
- planes: $t_1 > sheet_{min}$ and $t_1 < sheet_{max}$ and $min(\frac{t_2}{t_3}, \frac{t_3}{t_2}) > max(\frac{t_1}{t_2}, \frac{t_1}{t_3})$,

where $helix_{min}$ and $helix_{max}$ are the minimal and maximal thicknesses of helices, and $sheet_{min}$ and $sheet_{max}$ are the minimal and maximal thicknesses of sheets. While we have chosen default ranges in our implementation, each of these parameters are modifiable by the biological users.

Tracing Skeletons

As mentioned earlier, the α-helices and β-sheets are respectively captured by curves and surfaces in our method. The seeds discussed above provide good starting points for the individual skeletons. To avoid exhaustive search, we extract the skeletons by tracing the eigenvectors of the local structure tensors. The line-tracer (used for α-helices) is one-dimensional and hence is much easier than the plane-tracer (used for β-sheets). To trace a line structure, we start from the seed in two opposite directions, and follow the principal axis, defined by the eigenvector corresponding to the minimum eigenvalue of the local structure tensor (see Fig.3.15(a)). To trace a planar structure, we utilize the popular isocontouring technique[41]. We start from the seed point and compute the plane that is perpendicular to the eigenvector corresponding to the maximal eigenvalue (v_1 in Fig.3.14(c)). The plane partitions all eight vertices of the cell containing the seed point into two classes: positive and negative. In addition, the plane intersects with some of the twelve edges of the cell. Both the classification of the vertices and the intersection information with the edges can be used in the isocontouring lookup table, yielding a polygon (a list of triangles) representing the skeletons within the current cell (see Fig.3.15(b)). Next, we move to the neighboring cells with which the detected skeleton (polygon) intersects. For the example in Fig.3.15(b), we need to check four neighboring cells (back, front, right, and bottom). For each of those new cells, the “checking” point (similar to the seeds) is calculated as the center of the existing intersecting points between the already-detected skeletons and the new cell. The new polygon within the new cell is extracted using the idea as explained above, based on the “checked” point and the new structure tensor around it. This process is repeated until a certain stopping criterion is reached. The plane-tracer outputs a triangular mesh of skeletons. The stopping criteria for both line-tracer and plane-tracer are similar: the tracing process terminates whenever no new cells satisfy the criteria as discussed in Section 3.3.

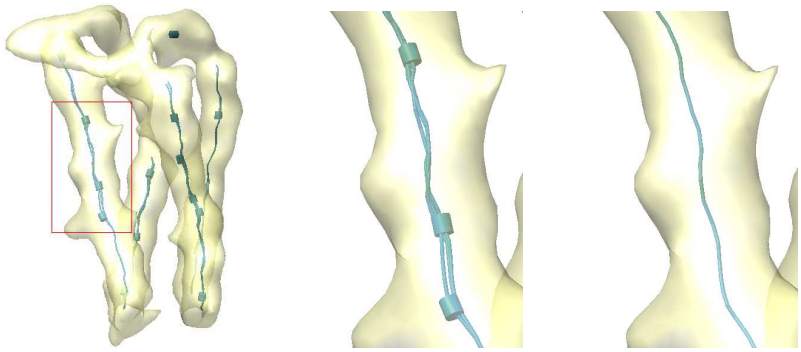


(a) Line-tracer (b) Plane-tracer

Figure 3.15: Skeleton-tracers. (a) Tracing 1D skeletons. (b) Tracing 2D skeletons (the marching cube method [42]).

Merging Skeleton

For each of the seed points, we extract a curve or a surface, corresponding to a helix or sheet, respectively. However, quite often we see more than one seed point corresponding to the same secondary structure. One example is illustrated in Fig.3.16(a) and a closer view is shown in Fig.3.16(b). In order to have only one curve/surface for each helix/sheet, we merge the superfluous skeletons corresponding to the same helix/sheet. Besides eliminating the redundancy, the merging process also yield balanced skeletons because the new skeletons are the average of previously redundant ones. Fig.3.16(c) shows the skeleton after merging. It is worth pointing out that the topological ambiguity problem seen in the original marching cube method [42] can be resolved by an improved algorithm as discussed in [41].



(a) Traced skeleton (b) Closer view (c) Merged skeleton

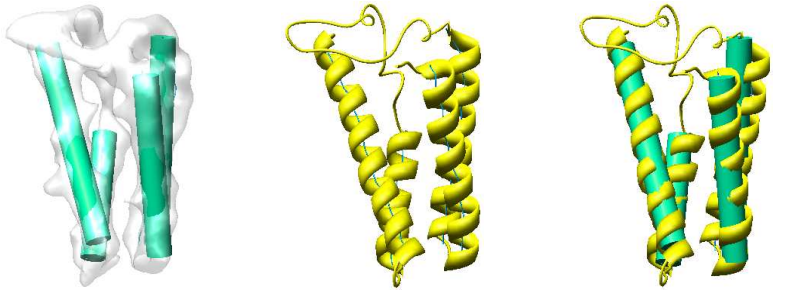
Figure 3.16: Skeleton extraction. (a) The initial skeletons traced by the line-tracer. (b) A zoomed-in view of the rectangular region as shown in (a). The thick "dots" are seed points. (c) The skeletons after merging

Application: Protein Secondary Structure Detection Once the skeletons of a protein density map are extracted, it is quite straightforward to locate the protein secondary structure elements. It is generally enough to represent beta-sheets using the extracted triangular mesh. However, if the resolution of the given maps is higher than 5Å, it is possible to see the individual beta-strands. In this case, we apply the line-tracer to extract the strands and get a more refined model of the beta-sheets. As for alpha-helices, we construct a cylinder model for each helix based on the extracted skeletons.

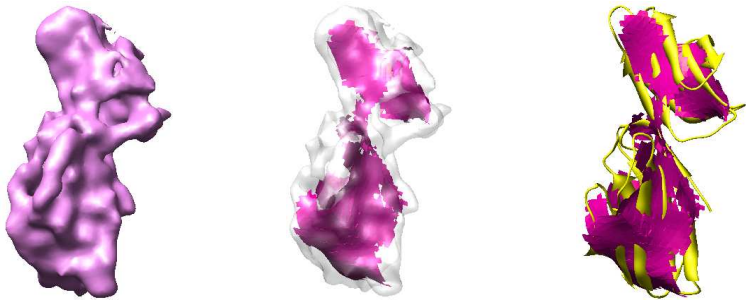
We have tested our skeleton-based protein secondary structure identification approach on a large number of simulated and experimentally-reconstructed protein density 3D maps. We only provide a few typical examples here. Fig. 3.17 illustrates the performance of our approach on a Gaussian blurred map of two X-ray atomic structures. The first example is cytochrome c' (PDBID = 1bbh). The blurred map at 8Å and detected skeletons are shown in Fig.3.16. From the skeletons, four alpha helices are detected as shown in Fig.3.17(a). To demonstrate the accuracy of our approach, the skeletons and detected helices are compared with the PDB structure in Fig.3.17(b) and (c). Another example is the blurred map of rat CD4 (PDBID = 1cid) as seen in Fig.3.17(d). The detected skeletons (sheets) are shown in Fig.3.17(e) and compared with PDB structures in Fig.3.17(f).

We also tested our approach on two simulated maps containing both alpha-helices and beta-sheets. Fig.3.18(a) shows the blurred map of the triose phosphate isomerase from chicken muscle (PDBID = 1tim) at 8Å. The detected alpha-helices and beta-sheets are shown in Fig.3.18(b) with density map and in Fig.3.18(c) with PDB structure. We can see that most helices/sheets agree very well with the PDB structure except that one small alpha-helix that is immediately adjacent to a long helix is missed (indicated by a red arrow). This result also agrees with that seen in [31]. Fig.3.18(d) and (e) give another view of (b) and (c), respectively. In Fig.3.19(a) we show a more complicated simulated map at 8Å, blurred from the PDB structure of the bluetongue virus VP7 (PDBID = 1bvp). This map contains a total of 27 alpha-helices in the lower domain and a few beta-sheets in the upper domain. The detected alpha-helices and beta-sheets are shown in Fig.3.19(b) with density map and in Fig.3.19(c) with PDB structure. All 27 alpha-helices and major portions of beta-sheets are correctly identified and agree very well with the PDB structure. Although two small alpha-helices (indicated by red arrows) are misidentified due to a couple of turns getting very close to each other, our results show better performance than the method proposed in [31]. Fig.3.19(d) shows another view of (c). A closer view of the skeletons together with the PDB structure is shown in Fig.3.19(e).

We also demonstrate our approach on a 3D map, reconstructed from experimental cryo-EM images of the rice dwarf virus (RDV) [79]. Fig.3.20(a) shows the reconstructed 3D map, where the other half sphere in the front is cut out. There are two



(a) Detected helices (b) Skeletons (c) Helices with PDB

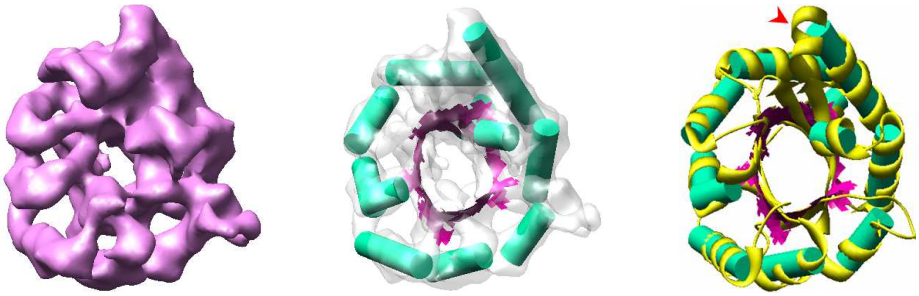


(d) Blurred map (e) Skeletons (f) Sheets

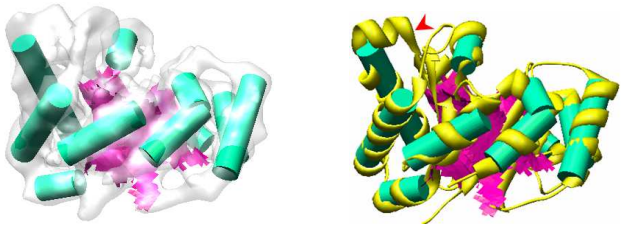
Figure 3.17: Examples on two blurred maps of x-ray crystal structures. (a) The alpha-helices detected from the blurred map of cytochrome c' (PDBID = 1bbh). The blurred map at 8Å and extracted skeletons were shown in Fig.3.16. (b) The skeletons detected from the blurred map and compared with the PDB structure. (c) The detected helices are compared with the PDB structures. (d) The blurred map of rat CD4 (PDBID = 1cid) at 8Å. (e) The skeletons (corresponding to beta-sheets) detected from the blurred map. (f) The sheets are compared with the PDB structures.

capsid layers as shown in Fig.3.20(b): the outer layer shown on the top and the inner layer shown on the bottom. For better illustrations, only a quarter of the whole layers are seen here. The segmented P3 protein (from inner layer) and P8 protein (from outer layer) are shown in Fig.3.20(c). The detected alpha-helices and beta-sheets for both proteins are shown in Fig.3.20(d).

In addition to the high accuracy, the speed of our approach is another advantage compared to the previous methods [31, 35]. The helix-hunter used in [31] may take up to one hour for the map such as P8 or P3 proteins of RDV, to detect only the alpha-helices. Our method takes only 3 seconds for P8 protein and 10 seconds for P3 proteins on a typical Linux workstation with single processor, to detect both alpha-helices and beta-sheets. We do not have the timings for sheet-miner as presented in [35]. However, we believe that the sheet-miner of [35] should be as slow as, if not slower than, the helix-hunter of [31] because both methods used exhaustive searching schemes in the translational and orientational spaces.

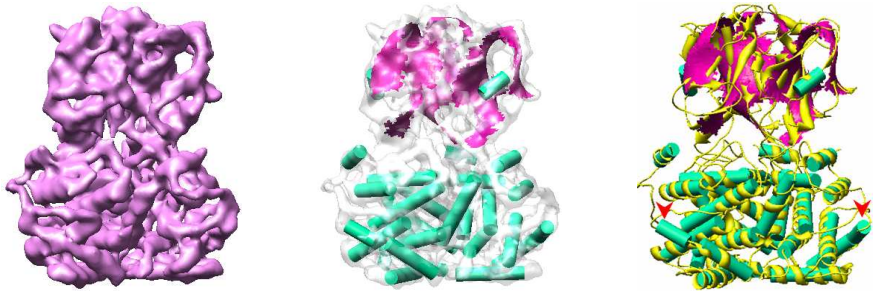


(a) Blurred map (b) Helix-sheet (map) (c) Helix-sheet (PDB)

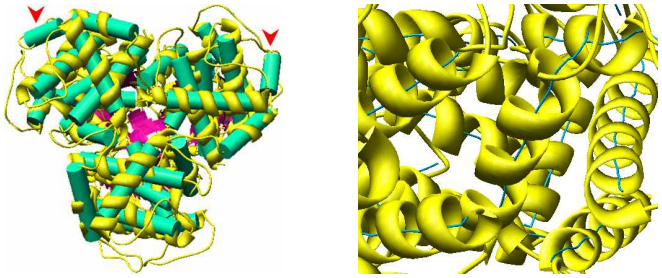


(d) Another view of (b) (e) Another view of (c)

Figure 3.18: Secondary structure identification on the triose phosphate isomerase from chicken muscle (PDBID = 1tim). (a) The blurred maps at 8Å from the x-ray crystal structure. (b) The alpha-helices (green) and beta-sheets (pink) detected using our method. (c) The detected helices/sheets are compared with the PDB structures. We can see that most helices/sheets agree very well with the PDB structure except that one small alpha-helix that is immediately adjacent to a long helix is missed (indicated by red arrow). This result also agrees with that seen in [31]. (d) Another view of (b). (e) Another view of (c).

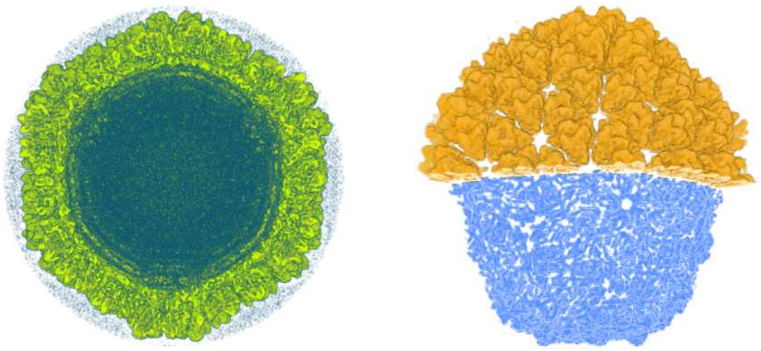


(a) Blurred map (b) Helix-sheet (map) (c) Helix-sheet (PDB)

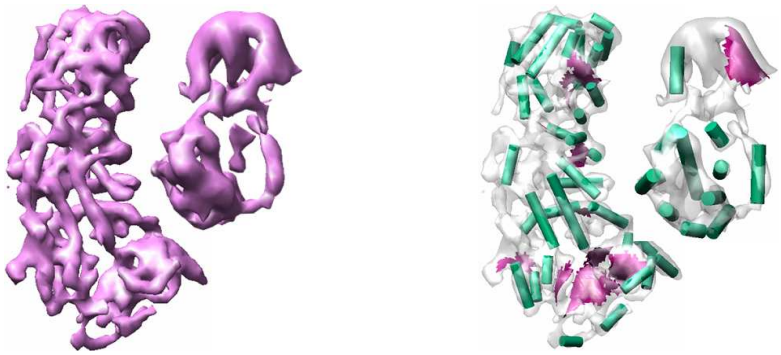


(d) Another view of (c) (e) Zoomed-in skeletons

Figure 3.19: Secondary structure identification on the bluetongue virus VP7 (PDBID = 1bvp). (a) The blurred maps at 8Å from the x-ray crystal structure. (b) The alpha-helices (green) and beta-sheets (pink) detected using our method. (c) The detected helices/sheets are compared with the PDB structures. All alpha-helices and major portions of beta-sheets are correctly identified and agree very well with the PDB structure. Although two small alpha-helices (indicated by red arrows) are misidentified due to a couple of turns running into each other, our results show better performance than the method proposed in [31]. (d) Another view of (c). (e) A closer view of the detected skeletons together with the PDB structure. The chosen region is roughly in the rectangular area right below the center of (c).



(a) Reconstructed map (b) Segmented layers



(c) Segmented proteins (d) Detected helices/sheets

Figure 3.20: The example of a real cryo-EM reconstructed map. (a) The reconstructed map of the rice dwarf virus. (b) The segmented outer layer (top) and inner layer (bottom). Shown here is only a quarter of the whole layers. (c) The segmented P3 protein (left) from inner layer and P8 protein (top-right) from outer layer. (d) The detected alpha-helices (green) and beta-sheets (pink) from the P3 and P8 proteins.

The following algorithm is taken from [6] (bajaj2006secondary).

Introduction

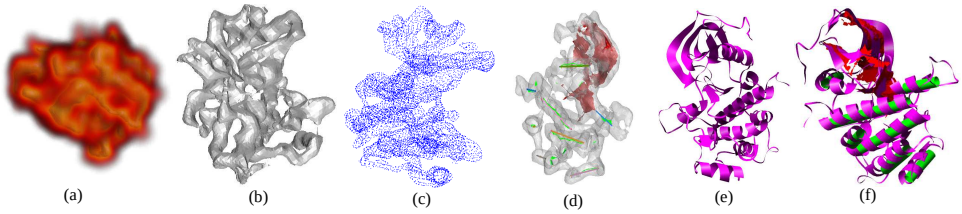


Figure 3.21: Secondary structure elucidation algorithm for Insulin Receptor Tyrosine Kinase Domain with pdbid: 1IRK. (a) volume rendering of its blurred 3D Map at 8Å resolution (b) surface rendering of the protein's molecular surface (c) pointset sampling of the molecular surface (d) The red patch inside the transparent surface depicts the β-sheet while the straight lines designate the axes of the cylinders which correspond to the α-helices (e) The secondary structural motifs, documented in the Protein Data Bank, where the helices are shown as ribbon coils and the sheets are sets of ribbon strands. (f) combined display of (d) and (e)

Three dimensional Electron Microscopy reconstruction(3D EM) and in particular single particle cryo-EM reconstruction [23], has advanced rapidly over recent years, such that several macromolecules (complexes of proteins and ribo-nucleic acids or RNA) can be resolved routinely at low resolution (10-20 Å) and in some cases at sub-nanometer (intermediate) resolution (7-10 Å) [9]. The ultra-structure of these complexes once elucidated from the 3D EM (henceforth 3D Maps), provide not only insights into individual protein and RNA folds and structural motifs, but even more importantly provide information about how the various structural components interact. In addition, with the increasing capability of determining multiple structural folds and conformers (secondary structures and tertiary arrangements) of a complex [79], there is the promise of studying the dynamics of such interacting systems.

Proteins are polypeptide chains of amino acids, (and nucleic acids are sugar-phosphate chains of acidic bases). The secondary structure of proteins are made up of a set of helical (or crudely, cylindrical) arrangement of sub-chains called α-helices and mostly planar arrangement of sub-chains called β-sheets. The various structural conformations of sub-groups of helices and sheets, yield the various different tertiary folds [12]. Relatively similar configuration of secondary and tertiary folds arise also in RNA [56].

In this paper, we provide a solution to the problem of automatically elucidating the structural secondary and tertiary folds of proteins (and nucleic acids) from 3D Maps of macromolecular complexes. A 3D Map is akin to a 3D spatial matrix of electron density values. With the improved reconstruction resolution of 3D Maps of macromolecules via 3D EM, the secondary and tertiary structural folds of proteins and RNA can be fully elucidated. Often (as evidenced by structures in the PDB), the atomic resolution structures of individual proteins or RNA, that make up the macromolecule are also discerned via X-ray diffraction and/or Nuclear Magnetic Resonance techniques, allowing us to validate our 3D EM secondary/tertiary structure elucidation algorithms [8]. Depending on the resolution of the 3D Map (in Å), our goals here are to either detect the secondary structural motifs, or segment the molecule into significant components that can be associated with different tertiary structural folds.

Our processing pipeline has three macro steps, namely, (i) segmentation of the macromolecular 3D Map into individual protein (or RNA), subvolumes using the techniques of [75] and implemented in the publicly available VolRover tool [18]. (ii) computation of a "distance function" to a suitable molecular surface approximation of the individual protein (or RNA) using the methods detailed in [4, 7] (iii) medial axis computation, classification, and construction of stable and unstable manifolds of the critical points of different indices. We skip a description of the first step, as the most recent developments are summarized in [75]. Step (ii) is sketched in Section 3.3, for completeness. The main contributions of this paper are: (a) Elucidation of secondary structural motifs (α-helices and β-sheets) from relatively higher resolution (5-10Å) 3D Map of individual protein or RNA molecules, (b) Decomposition of the molecular surface of individual proteins or RNA, into its tertiary structural motifs, for

3D Maps at relatively coarser resolution (10-15Å). Figure 3.21 shows the steps of the secondary structure elucidation process on the 3D Map of protein 1IRK at a resolution 8Å.

Preliminaries

Voronoi-Delaunay Diagram: We do not to go over the detail about this well-known datastructure due to space limitation and encourage reader to consult a standard computational geometry textbook [19]. In this work, we primarily use the duality of Voronoi and Delaunay diagram which states that every $k \leq 3$ dimensional Voronoi element is dual to a $3-k$ dimensional Delaunay simplex.

Critical Points of h_P: The critical points of h_P are the points in $\mathbb{R}^3$ which lie within the convex hull of its closest points from P. It turns out that the critical points of h_P are the intersection points of the Voronoi objects with their dual Delaunay objects [58].

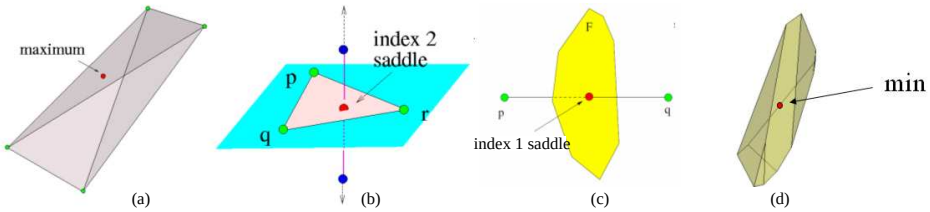


Figure 3.22: The relative position of Voronoi and their dual Delaunay objects that results in the generation of critical points.

- *Maxima* are the Voronoi vertices contained in their dual tetrahedra,
- *Index 2 saddles* lie at the intersection of Voronoi edges with their dual Delaunay triangles,
- *Index 1 saddles* lie at the intersection of Voronoi facets with their dual Delaunay edges, and
- *Minima* are the sample points themselves as they are always contained in their Voronoi cells.

An illustration of the four types of critical points and the relative position of the Voronoi/Delaunay objects resulting these four types is shown in Figure 3.22.

At any point $x \in \mathbb{R}^3$, one can assign a vector field which is the direction of the steepest ascent of the distance function. The critical points are assigned zero vectors. This vector field induces a *flow*. If a point is allowed to move following the vector field, it traces an *orbit* and ends at a critical point. The set of points whose orbits end at a critical point c is called the *stable manifold* of c. Similarly, a point traces an *inverted orbit* when it follows the steepest descent of the distance function, and ends at a critical point c'. The set of points whose inverted orbits end at c' is called *unstable manifold* of c'.

The stable manifold of a maximum is a three dimensional polytope which is bounded by the stable manifold of critical points of lower indices. Similarly, the unstable manifold of a minimum is a three dimensional polytope which is bounded by the unstable manifold of critical points of higher indices. In this paper, our focus is on stable manifold of maxima and unstable manifold of the index 1 and 2 saddle points.

Secondary Structure Identification

The processing pipeline of α-helix and β-sheet identification from 3D Maps, consists of the following steps:

(a) Molecular Surface Extraction and Sampling: Starting with a 3D EM map of a protein (or RNA), a molecular surface is extracted via contouring [42]. For robust contouring, we use the implementation in the publicly available software TexMol [5]. Although several possible isosurfaces can be computed from the 3D Map, we select an isovalue using contour trees [65] (also implemented in TexMol), and topological curation wherein the isosurface is a single connected component, after removal of smaller completely nested surface components (i.e. voids). We call this extracted and curated surface component the molecular

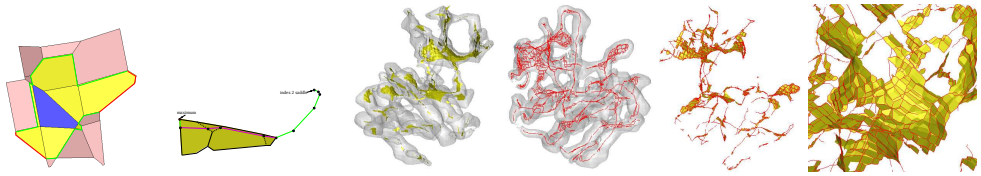
surface S of the protein (or RNA). Further, a sufficiently dense set of points are sampled from S. We call this poinset P which becomes the input to the latter stages of our algorithm.

(b) Detection of index 1 and index 2 critical points of h_P (Sec. 3.3).
(c) Computing unstable manifold of a subset of those critical points (Sec. 3.3).
(d) Detection of α-helix and β-sheets from the unstable manifolds (Sec. 3.3).

Computing U_1 and U_2 from P Structure and computation of the unstable manifold of an index 1 saddle point (U_1) and an index 2 saddle point (U_2) have been described in detail in [25]. For completeness, we describe it briefly here.

U_1**:** Unstable Manifolds of index 1 saddle points are two dimensional. An index 1 saddle point, c lies at the intersection of a Voronoi facet F and a Delaunay edge. For any point $x \in F \setminus c$, h_P increases radially outward from c. Therefore the orbit of one such x hits the Voronoi edges bounding F. Thus F is in $U(c)$. Once the flow hits a Voronoi edge, if the dual Delaunay triangle is acute angled, the flow is along the Voronoi edge, and otherwise, the flow enters the Voronoi facet dual the Delaunay edge opposite to the largest angle of the dual Delaunay triangle. This iterative process computes the unstable manifold of c. The exact computation and its approximation have been described in [25]. Figure 3.23(a) illustrates an intermediate stage of this computation where the blue facet contains c, yellow facets are currently in $U(c)$ and pink facets are to be included in the subsequent iterations.

U_2**:** An index 2 saddle point is generated by the intersection of a Voronoi edge and a Delaunay triangle. The unstable manifold of an index 2 saddle point is one dimensional. It is a polyline with one endpoint at the saddle point and the other endpoint at a local maximum. The polyline consists of segments that are either subsets of Voronoi edges or lie in the Voronoi facets. Due to the later case, the polyline may not be a subcomplex of Vor P. Again, the exact computation and its approximation have been described in [25]. Figure 3.23(b) illustrates an example. Figure 3.23(c) shows the unstable manifolds of index 1 and index 2 saddle points on the interior medial axis of the 3D map of molecule 1IRK.



(a) (b) (c)

Figure 3.23: General structure of U_1 and U_2 is shown in (a,b). (c) A collage of U_1 (yellow) and U_2 (red) of the molecule 1IRK. Note U_2 bounds U_1.

α-helix and β-sheet Selection The unstable manifold of index 1 and index 2 saddle points restricted to the medial axis of the molecular surface decomposes the medial axis into linear (U_2) and planar (U_1) portions. We call the linear subset M_L and the planar subset M_F. The next task is to select a subset of M_L and M_F that gives the α-helices and β-sheets of the protein. Typically, an α-helix is a cylindrical subvolume of the molecule which is of width approximately 2.5 Å. Also the subvolume does not deviate much from a straight cylinder for proteins [12, 31]. These two conditions, dictate the following computational steps.

The unstable manifold of every index 2 saddle on the medial axis is a polyline with Voronoi vertices at the joints. Every Voronoi vertex has h_P value which can be computed by the circumradius of the dual Delaunay tetrahedron. Locally this gives the radius of the cylinder that best fits the molecular surface in the cylindrical regions. We first populate the set H with the Voronoi vertices whose h_P values fall within 2 Åand 3 Å. A 3 Å neighborhood graph is then computed over H that clusters the points. The choice of 3 Å is dictated by the fact that pitch of the helices is 1.5 Å and usually there is more than 2 turns in every helix. The diametrical point pair in every cluster is then computed. The maximum deviation of any intermediate point from the straight line joining the point pair decided how straight the fitted cylinder to the cluster is. This way we select those clusters from H and the cylinders fitted to these clusters produce the detected α helices. The process is shown in 3.24(b).

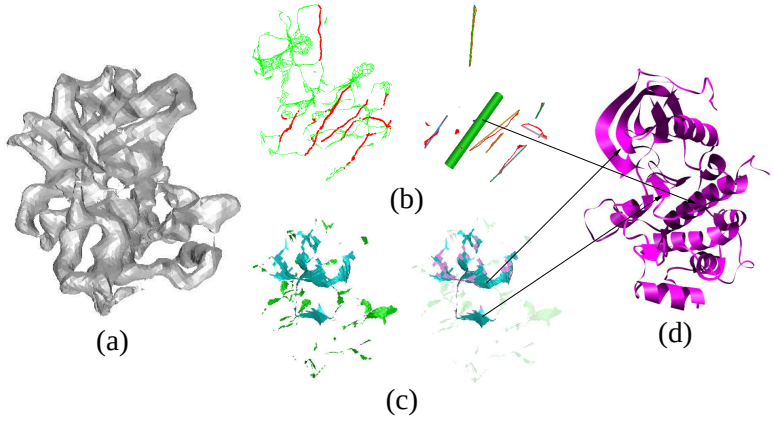


Figure 3.24: (a) The molecular surface of 1IRK. (b) The selected Voronoi vertices on U_2 and the fitted cylinder. (c) Filtering out subsets of U_1 which are small (green) or do not satisfy the width test (magenta). (d) shows the secondary structures obtained from the PDB and its correspondence with the computed structure (b,c).

The selection of β-sheets is similar. U_1 gives the possible candidates for β-sheets. First we notice that there are some tiny components that are created due to sampling artifacts, and they do not correspond to real planar substructures of the molecule. We first filter these small clusters out (green patches in Figure 3.24(c)). After this first stage of filtering, we are left with the planar subsets of the medial axis (cyan in Figure 3.24(c)). At this point we apply the knowledge, that β-sheets are of width roughly 1.5 Å [12, 36] and we filter out the planar patches which do not satisfy this "thickness" criterion. The width of a facet in U_1 is easy to check as they are the Voronoi facets and therefore have an 1-1 correspondence with their dual Delaunay edges which cross the medial axis. Therefore, we select only those Voronoi facets from U_1 whose dual Delaunay edges are of length between 1 Å and 2 Å. The portions of U_1 which are filtered out by this test are shown in magenta in Figure 3.24(c). The rest of U_1, which qualify for β-sheets, are shown in cyan. The sheets detected in are shown to correspond well with the β-sheets of the molecule 1IRK obtained from Protein Data Bank.

Tertiary Fold Elucidation

Tertiary structural folds (or motifs) provide useful information about the conformational and packing arrangement of a protein molecule. Such tertiary folds can be discerned when a coarser resolution 3D Map of the molecule is available. In this section, we show how such information can be gleaned again by looking at the distance function h_P induced by the set of points P sampled on an approximation of the molecular surface S extracted from a coarse resolution 3D Map. Our main focus is on the maxima of h_P. We have already seen that the maxima are the intersection of the Delaunay tetrahedra with their dual Voronoi vertices. In other words, these are the circumcenters of only those Delaunay tetrahedra whose circumcenters lie inside them. As a first step of the tertiary fold detection, we collect the maxima which are the circumcenters of the interior Delaunay tetrahedra.

Once the set of interior maxima is populated, we use their stable manifolds to decompose the volume bounded by S into a set of bio-chemically meaningful segments. These stable manifolds are three dimensional solid subsets of the interior of the molecular surface S. Such techniques have been proved useful earlier in segmentation and matching of free-form objects [20]. We follow the algorithm for computing the stable manifold of a maximum approximately, as described in [20].

Given two Delaunay tetrahedra σ and σ' which share a common triangle t, we say $\sigma < \sigma'$, if the circumcenter of σ lies in the half-space defined by t that does not contain the fourth vertex of σ. Figure 3.25(a) describes this case. Note that σ' is not always unique (Figure 3.25(b)). However, this can be proved that, a Delaunay tetrahedron σ can have at most two neighbors σ_1 and σ_2, for which $\sigma < \sigma_1$ and $\sigma < \sigma_2$. Also, it is to be noted, that the Delaunay tetrahedron σ_{max} whose dual Voronoi vertex is a maximum, has none of its neighbors σ' for which $\sigma_{max} < \sigma'$.

Following the above observation the stable manifold of the maxima are approximated. The set of maxima is sorted according to h_P. Starting from the biggest maximum, the algorithm collects all σ which falls under the transitive closure of the relation '$<$'. A subtle problem remains. The stable manifolds of the maxima are often numerous, and therefore they need to be clubbed carefully to bring out the underlying features of the molecule. To this goal, we apply a merging step, which adjoins two stable

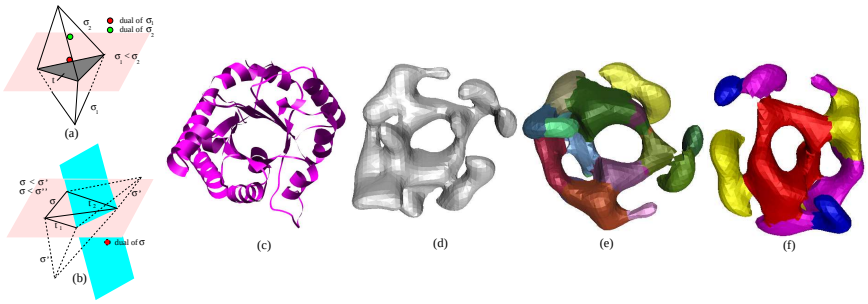


Figure 3.25: (a) $\sigma_1 < \sigma_2$. (b) The situation when σ has two neighbors σ' and σ'' for both of which $\sigma < \sigma'$ and $\sigma < \sigma''$. (c) 1TIM: Helices surround the sheets to form the tertiary structure called α/β-*barrel*. (d) Molecular surface of 1TIM at 15 Å resolution. (e,f) The initial segmentation and further refinement to bring out the β-fold of the *barrel* from the surrounding helices (yellow, magenta and blue).

PDBID	Protein	PDBID	Protein
1IRK	Insulin Receptor Tyrosin Kinase Domain	1TIM	Triose Phosphate Isomerase
1PLQ	Proliferating Cell Nuclear Antigen	1RIE	Rieske Iron-sulphur Protein
1CID	T Cell Surface Glycoprotein CD4	1MBN	Myoglobin
1VDF	Cartilage Oligomeric Matrix Protein	1JPC	Agglutinin
1AOR	Aldehyde Ferrodoxin Oxydoreductase	1BBH	Cytochrome C'

Table 3.4: Name and PDBID of the proteins used in the calibration process.

manifolds of two maxima sharing a common boundary when the h_P function values at the maxima as well as at a point on the common boundary are comparable to each other.

Figure 3.25(c-f) shows an example of the stable manifolds and the tertiary folds that they correspond to. It is worth mentioning that, in our experience, tertiary motifs are not always readily decomposable using this algorithm. Nevertheless, we observe that the decomposition algorithm successfully detects the helical region and separates them from the beta regions. The beta regions are sometimes decomposed into more than one component which need to be associated separately to reflect a single fold.

Implementation and Results

For calibrating our structure elucidation algorithms, we downloaded atomic level descriptions of proteins from the Protein Data Bank [10]. For each protein, we first "blurred" them into a 3D map (to correspond to a reconstructed 3D EM map) at varying resolutions (5 to 15 Å), using publicly available software EMAN [43]. From these volumetric maps, we extracted the proteins molecular surface using TEXMOL [5]. Next we collected a pointset sampling of the molecular surface, and used the COCONE software for surface reconstruction and medial axis computation [16]. The next step was to detect the critical points and compute the stable/unstable manifolds of a subset of the critical points. These computations rely on the Voronoi Diagram - Delaunay triangulation of the pointset, and were done using the CGAL library [17].

The calibration process is essential before we apply them to 3D EM maps of unknown atomic descriptions. The datasets used in the calibration process are summarized in Table 3.4. Details of the molecules are available from the Protein Data Bank (PDB) via the pdbid. Figure 3.26 shows the snapshots of the key steps of the algorithm. Figure 3.26(a) shows the U_1 (green) and U_2 (red) of 1BBH. The clusters of points lying on U_2, selected by the width criterion of α-helix, and the axis of the cylinders to be fitted to the clusters are shown in Figure 3.26(b). Figure 3.26(c) shows the secondary structure of 1BBH documented in the PDB. It has total 8 helices and all of them are detected correctly (green cylinders) by our algorithm (Figure 3.26(d)). The second row in Figure 3.26 shows similar set of pictures for 1RIE. It has three β-sheets which are identified correctly (red, yellow and blue patches). Third and fourth row show the performance of the algorithm on six more molecules at varying resolutions.

Figure 3.27 shows the performance of our tertiary fold detection algorithm. Top row shows the secondary structure of

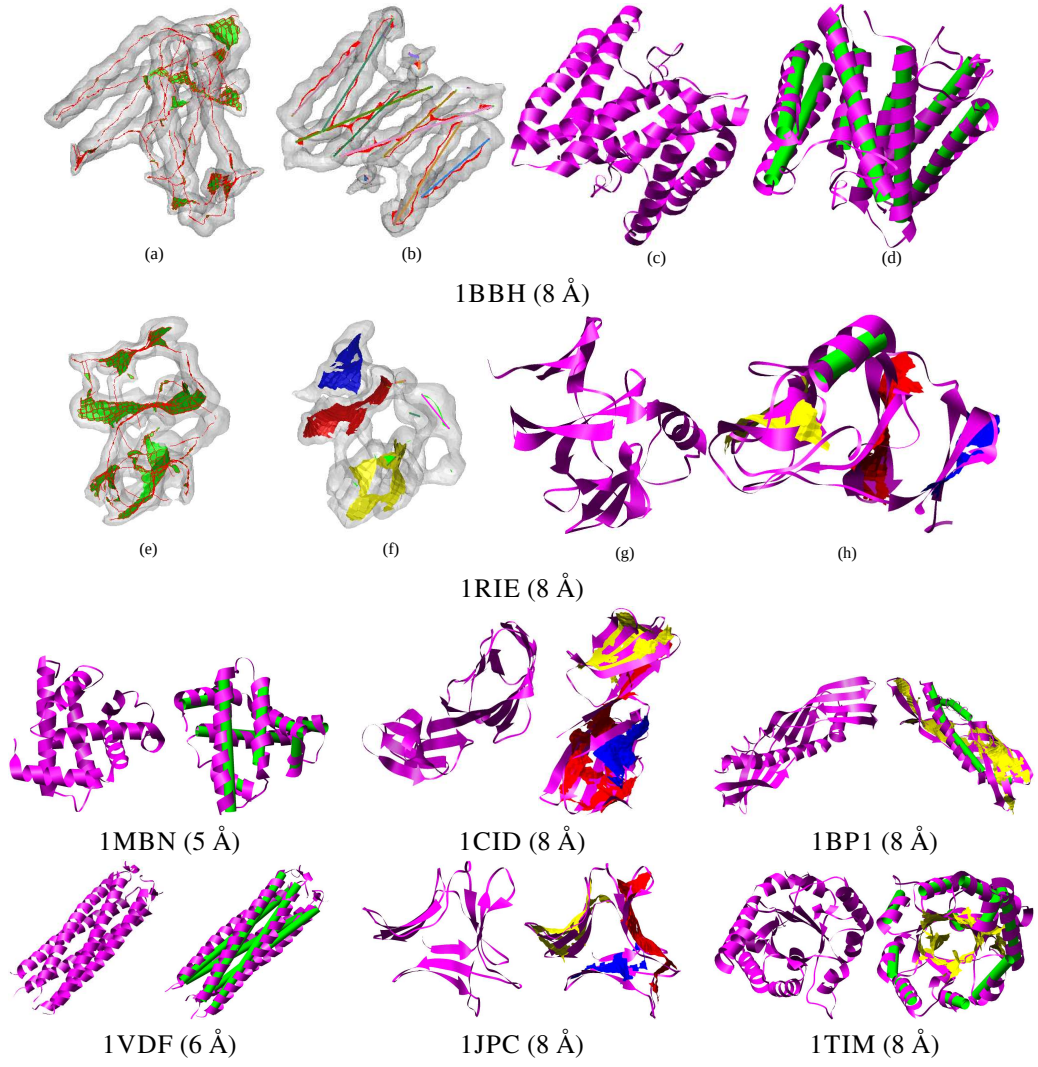


Figure 3.26: Performance of our α-helix and β-sheet detection algorithm.

1AOR as provided by PDB along with the ensemble of the segments of the protein at 12 Å. The tertiary structure of 1AOR is named as β *sandwich*. The two red segments (in c) correspond to the two β-sheets. The bottom row shows similar results for 1PLQ (at 12 Å). The tertiary fold elucidation can successfully separate the three sheets (red, yellow, green) from the helices (blue).

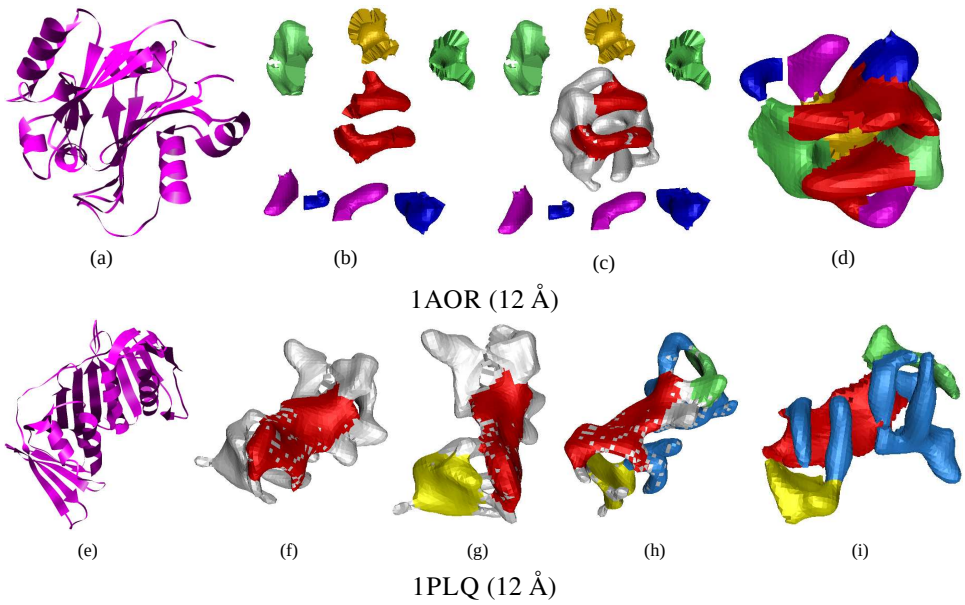


Figure 3.27: Performance of the tertiary fold elucidation algorithm.

3.4 Biological Structure Elucidation

3.4.1 molecules — ultrastructure elucidation of viruses, ribosomes from EM

3.4.2 cells with organelles —- reconstruction of neuronal circuits from ssTEM

3.4.3 tissues cells + extra-cellular space/filling —- reconstruction of T-tubules from, and bone tissue from micro-CT

3.4.4 organs — heart reconstruction from CT/ brain reconstruction from MRI

3.4.5 systems — developmental cell reconstruction, beating heart parameters from phase-contrast and tagged MRI

3.5 Bibliographic Notes

3.5.1 Skeletonization

The following is taken from [74] (YB2004).

Since the first study by Blum [11], the skeletonization of shapes has attracted attentions from many researchers in various fields. Commonly used computational methods for skeleton extraction include topological thinning [38], approaches based on distance maps [3, 34, 45], hierarchical methods based on Voronoi diagrams [51], voxel-coding based methods [77], and some approaches based on physical simulations [26] or curve evolution [57]. However, all of these techniques compute the skeletons from the object's boundaries. In other words, it is a prerequisite that we should first segment the original images into meaningful regions with well-defined boundaries before the above-mentioned approaches could be applied. Unfortunately, efficient and effective segmentation of an arbitrary real-world image still remains an open problem especially with the existence of much noise. In addition, as also mentioned in [29], region segmentation (or edge detection, if applicable) often involves a loss of information, which, as a result, leads to an inaccurate skeletonization of the original images.

Our goal is to compute the skeletons directly from a gray-scale image, without the segmentation (or edge detection) as an intermediate step. The previous efforts on this topic by other authors can be categorized into three approaches. The first one is based on isotropic diffusion, governed by a set of linear PDE's. In [63], Tari *et al.* proposed a method, which extracts the skeletons from a set of level-set curves of the edge strength function of the original image. The edge strength function is calculated by means of a linear diffusion equation. A later paper, by Chung and Sapiro [15], deals with situations where the features (objects) are always brighter or darker than the background. The original image is treated as a 2D function defined over the image domain, and is diffused according to a family of linear erosion/dilation equations. A post-processing step is employed to extract the crest lines of the deformed image, which are taken as the final skeletons.

The second approach is based on scale-space theory [39]. In [54, 49], the authors proposed a method, which extracts the "cores" from the ridges of a medialness function in scale-space. Lindeberg [40] treated the skeletonization (ridge detection) in a similar way to edge detection in scale-space, with automatic scale selection. It has been pointed out that the Gaussian kernels used in scale-space are closely related to the linear diffusion equations when they are applied to the images. Both the isotropic diffusion and the scale-space techniques have the following major drawbacks. First, linear systems often cause feature-blurring and even a loss of information. Second, linear systems may extract the biased skeletons when curvilinear structures are considered and when the contrasts on both sides of the structures are different. The second problem was addressed in detail in [62], where special efforts were made to remove the bias of the extracted skeletons.

The third approach dealing with the skeletonization of gray-scale images without segmentation was proposed more recently by Jang and Hong [29]. In their method, a pseudo-distance map is calculated from the original image using a nonlinear governing equation. Although a nonlinear equation was utilized, the results of their method look quite similar to those of linear systems, e.g., Gaussian smoothing, as pointed out by the authors themselves [29]. In addition, it is not clear in [29] whether this method can handle the problem of biased skeletons, as the authors made use of Steger's algorithm [62] to extract the skeletons from a pseudo-distance map. Regardless of these drawbacks, this approach is most related to our algorithm described below. Hence we shall give additional words in Section 4 to compare this method with ours.

In the following we compute the skeletons based on *gradient vector diffusion* [69]. We consider a set of anisotropic diffusion equations applied to initial vector fields that are obtained by various ways. A *skeleton strength map*, analogous to

the *edge strength function* [63], is calculated from the diffused vector field, and the final skeletons are extracted by tracing the ridges of the skeleton strength map. The *skeleton strength map* gives a measure of how possible each pixel could be on the skeletons. As demonstrated later, the skeleton strength map gives much higher contrast than most methods described above. In addition, the problem of biased skeletons is easily avoided in our approach, due to the anisotropic property of the diffusions.

Bibliography

[1] Online course on protein crystallography. *http://www-structmed.cimr. cam.ac.uk/course.html*.

[2] L. Amos, R.Henderson, and P.N.Unwin. Three-dimensional structure determination by electron microscopy of two-dimensional crystals. *Prog. Biophys. Mol. Biol*, 39:183–231, 1982.

[3] C. Arcelli and G. Baja. Ridge points in euclidean distance maps. *Pattern Recognition Letters*, 13:237–243, 1992.

[4] C. Bajaj, F. Bernardini, and G. Xu. Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans. *Proc. of the ACM Siggraph Computer Graphics, SIGGRAPH 95*, 29(3):109–118, 1995.

[5] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Interactive visual exploration of large flexible multi-component molecular complexes. In *Proc. of the Annual IEEE Visualization Conference*, pages 243–250, Oct. 2004.

[6] C. Bajaj and S. Goswami. Secondary and Tertiary Structural Fold Elucidation from 3D EM Maps of Macromolecules. *Computer Vision, Graphics and Image Processing*, pages 264–275, 2006.

[7] C. Bajaj, H. Lee, R. Merkert, and V. Pascucci. NURBS based B-rep models from macromolecules and their properties. In *Proceedings of the Fourth Symposium on Solid Modeling and Applications*, pages 217–228, 1997.

[8] C. Bajaj and Z. Yu. Geometric processing of reconstructed 3D maps of macromolecular complexes. In S. Aluru, editor, *Handbook of Computational Molecular Biology*, page Chap 45. Chapman and Hall, CRC Press, 2005.

[9] T. Baker, N. Olson, and S. Fuller. Adding the third dimension to virus life cycles: three-dimensional reconstruction of icosahedral viruses from cryo-electron micrographs. *Microbio. and Mol. Bio. Reviews*, 63(4):862–922, 1999. TY - JOUR.

[10] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Res.*, 28:235 – 242, 2000.

[11] H. Blum. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form (W. Wathen-Dunn, Ed.), MIT Press*, pages 363–380, 1967.

[12] C. Branden and J. Tooze. *An Introduction to Protein Structure*. Garland Publishing Inc., New York, 1999. TY - BOOK.

[13] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:679–698, 1986.

[14] W. Chiu. What does electron cryomicroscopy provide that x-ray crystallography and nmr spectroscopy cannot? *Ann. Rev. Biophys. Biomol. Struct.*, 22:233–255, 1993.

[15] D. Chung and G. Sapiro. Segmentation-free skeletonization of gray-scale images via pde's. In *Proc. Int'l Conf. Image Processing*, pages 927–930, 2000.

[16] Cocone. Tight Cocone Software for surface reconstruction and medial axis approximation. *http://www.cse.ohio-state.edu/∼tamaldey/cocone.html*.

[17] C. Consortium. CGAL: Computational Geometry Algorithms Library. *http://www.cgal.org*.

[18] U. CVC. Volrover. *http://cvcweb.ices.utexas.edu/software/guides.php*, 2005. TY - JOUR.

[19] M. de Berg, van Krefeld M., M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Second Edition, 2000.

[20] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In F. Dehne, J.-R. Sack, and M. Smid, editors, *Proc. Workshop Algorithms Data Strucutres (WADS 03)*, LNCS 2748, pages 25–36, Berlin, Germany, 2003.

[21] A. E. Ferentz and G. Wagner. NMR spectroscopy : a multifaceted approach to macromolecular structure. *Quarterly Reviews of Biophysics*, 33:29–65, 2000.

[22] J.-J. Fernandez and S. Li. An improved algorithm for anisotropic nonlinear diffusion for denoising cryo-tomograms. *Journal of Structural Biology*, 144:152–161, 2003. TY - JOUR.

[23] J. Frank. *Three-Dimensional Electron Microscope of Macromolecular Assemblies*. San Diego: Academic Press, 1996. TY - BOOK.

[24] J. Frank, M. Radermacher, P. Penczek, J. Zhu, Y. Li, M. Ladjadj, and A. Leith. SPIDER and WEB: processing and visualization of images in 3D electron microscopy and related fields. *Journal of Structural Biology*, 116:190–199, 1996.

[25] S. Goswami, T. Dey, and C. Bajaj. Identifying flat and tubular regions of a shape by unstable manifolds. *Proceedings of the 2006 ACM symposium on Solid and Physical Modeling*, 2006.

[26] T. Grogorishin, G. Abdel-Hamid, and Y. Yang. Skeletonization: an electrostatic field-based approach. *Pattern Analysis and Application*, 1(3):163–177, 1996.

[27] P. Guntert. Structure calculation of biological macromolecules from NMR data. *Quarterly Reviews of Biophysics*, 31:145–237, 1998.

[28] M. V. Heel, G. Harauz, E. Orlova, R. Schmidt, and M. Schatz. A new generation of the IMAGIC image processing system. *Journal of Structural Biology*, 116:17–24, 1996.

[29] J. Jang and K. Hong. A pseudo-distance map for the segmentation-free skeletonization of gray-scale images. In *Proc. Int'l Conf. Computer Vision*, pages 18–23, 2001.

[30] J.Fernandez, J.R.Sanjurjo, and J.Carazo. A spectral estimation approach to contrast transfer function detection in electron microscopy. *Ultramicroscopy*, 68:267–295, 1997.

[31] W. Jiang, M. Baker, S. Ludtke, and W. Chiu. Bridging the information gap: computational tools for intermediate resolution structure interpretation. *J. of Mol. Bio.*, 308:1033–1044, 2001. TY - JOUR.

[32] W. Jiang, Z. Li, M. L. Baker, P. E. Prevelige, and W. Chiu. Coat protein fold and maturation transition of bacteriophage P22 seen at subnanometer resolution. *Nature Structural Biology*, 10(2):131–135, 2003.

[33] G. Kaphne, J. Weickert, O. Schuster, and S. Richter. A tensor-driven active contour model for moving object segmentation, 2001. TY - CONF A tensor-driven active contour model for moving object segmentation.

[34] R. Kimmel, D. Shaked, N. Kiryati, and A. Bruckstein. Skeletonization via distance maps and level sets. *Computer Vision and Image Understanding*, 62(3):382–391, 1995.

[35] Y. Kong and J. Ma. A structural-informatics approach for mining b- sheets: locating sheets in intermediate-resolution density maps. *J. of Mol. Bio*, 332:399–413, 2003. TY - JOUR.

[36] Y. Kong and J. Ma. A structural-informatics approach for mining β- sheets: locating sheets in intermediate-resolution density maps. *J. of Mol. Bio*, 332:399–413, 2003.

[37] A. Koster, R. Grimm, D. Typke, R. Hegerl, A. Stoschek, J. Walz, and W. Baumeister. Perspectives of molecular and cellular electron tomography. *Journal of Structural Biology*, 120:276–308, 1997.

[38] L. Lam, S. Lee, and C. Suen. Thinning methodologies - a comprehensive survey. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(9):869–885, 1992.

[39] T. Lindeberg. *Scale-space Theory in Computer Vision*. The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Netherlands, 1994.

[40] T. Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–154, 1998.

[41] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cube algorithm for isosurfacing. *IEEE Trans. Visualization and Computer Graphics*, 9(1):16–29, 2003.

[42] Lorensen, W., and Cline, H. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21:163–169, 1987.

[43] S. Ludtke, P. Baldwin, and W. Chiu. EMAN: semiautomated software for high-resolution single-particle reconstructions. *J. of Struct. Bio.*, 128:82–97.

[44] S. Ludtke, D. Chen, J. Song, D. Chuang, and W. Chiu. Seeing groel at 6-angstrom resolution by single particle electron cryomicroscopy. *Structure*, 12:1129–1136, 2004.

[45] G. Malandain and S. Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317–327, 1998.

[46] R. Matadeen, A. Patwardhan, B. G. E. Orlova, T. Pape, M. Cuff, F. Mueller, and R. B. M. van Heel. The e. coli large ribosomal subunit at 7.5 angstrom resolution. *Structure*, 7:1575–1583, 1999.

[47] B. McEwen and M. Marko. The emergence of electron tomography as an important tool for investigating cellular ultrastructure. *The Journal of Histochemistry & Cytochemistry*, 49(5):553–563, 2001.

[48] M. Morais, K. Choi, J. Koti, P. Chipman, D. Anderson, and M. Rossmann. Conservation of the capsid structure in tailed dsdna phage the pseudoatomic structure of phi29. *Molecular Cell*, 18:149–159, 2005.

[49] B. Morse, S. Pizer, D. Puff, and C. Gu. Zoom-invariant vision of figural shape: effects on cores of images disturbances. *Computer Vision and Image Understanding*, 69(1):72–86, 1998.

[50] W. Nicholson and R. Glaeser. Review: automatic particle detection in electron microscopy. *Journal of Structural Biology*, 133(2-3):90–101, 2001.

[51] R. Ogniewicz and O. Kubler. Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343–359, 1995.

[52] P. Penczek, M. Marko, K. Buttle, and J. Frank. Double tilt electron tomography. *Ultramicroscopy*, 60, 1995.

[53] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell*, 12:629–639, 1990. TY - JOUR.

[54] S. Pizer, D. Eberly, D. Fritsch, and B. Morse. Zoom-invariant vision of figural shape: the mathematics of cores. *Computer Vision and Image Understanding*, 69(1):55–71, 1998.

[55] A. Roseman. Particle finding in electron micrographs using a fast local correlation algorithm. *Ultramicroscopy*, 94:225–236, 2003.

[56] B. Schneider, Z. Morveki, and H. Berman. Rna conformational classes. *Nucleic Acids Research*, 32, 5:1666–1677, 2004. TY - JOUR.

[57] K. Siddiqi, S. Bouix, A. Tannenbaum, and S. Zucker. The hamilton-jacobi skeleton. In *Proc. Int'l. Conf. Computer Vision*, pages 828–834, 1999.

[58] D. Siersma. Voronoi diagrams and morse theory of the distance function. *Geometry in Present Day Science*, 1999.

[59] E. Sifakis and G. Tziritas. Moving object localization using a multi-label fast marching algorithm. *Signal Processing: Image Communication*, 16(10):963–976, 2001.

[60] F. Sigworth. A maximum-likelihood approach to single-particle image refinement. *Journal of Structural Biology*, 122:328–339, 1998.

[61] M. Sjors, M. Valle, R. Nuñez, C. Sorzano, R. Marabini, G. Herman, and J. Carazo. Maximun likelihood multi-reference refinement for eletron microscopy images. *Journal Molecular Biology*, 348:139–149, 2005.

[62] C. Steger. An unbiased detector of curvilinear structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(2):113–125, 1998.

[63] S. Tari, J. Shah, and H. Pien. Extraction of shape skeletons from gray-scale images. *Computer Vision and Image Understanding*, 66(2):133–146, 1997.

[64] M. van Heel, B. Gowen, R. Matadeen, E. V. Orlova, R. Finn, T. Pape, D. Cohen, H. Stark, R. Schmidt, M. Schatz, and A. Patwardhan. Single-particle electron cryo-microscopy: towards atomic resolution. *Cambridge Quarterly Review of Biophysics*, 33:307–369, Mar. 2000.

[65] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface generation. In S. Rana, editor, *Topological Data Structures for Surfaces*, pages Chap 5, pg 71 –86. John Wiley & Sons, 2004.

[66] J. Weickert. *Anisotropic Diffusion In Image Processing*. ECMI Series, Teubner, Stuttgart, ISBN 3-519-02606-6, 1998. TY - BOOK.

[67] M. Woolfson. *An Introduction to X-ray Crystallography*. Cambridge University Press, Jan. 1997.

[68] W. Wriggers and P. Chacon. Modeling tricks and fitting techniques for multiresolution structures. *Structure*, (9):771–778, 2001.

[69] C. Xu and J. L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Trans. Image Processing*, 7(3):359–369, 1998.

[70] Z. Yu and C. Bajaj. Anisotropic vector diffusion in image smoothing. In *Proceeding of the 9th IEEE ICIP*, volume 1, pages 828–831, 2002.

[71] Z. Yu and C. Bajaj. Image segmentation using gradient vector diffusion and region merging. In *International Conference on Pattern Recognition*, pages 941–944, 2002.

[72] Z. Yu and C. Bajaj. Detecting circular and rectangular particles based on geometric feature detection in electron micrographs. *Journal of Structural Biology*, 145(1-2):168–180, 2004.

[73] Z. Yu and C. Bajaj. A fast and adaptive algorithm for image contrast enhancement. In *Proc of IEEE International Conference on Image Processing*, pages 1001–1004, 2004.

[74] Z. Yu and C. Bajaj. A segmentation-free approach for skeletonization of gray-scale images via anisotropic vector diffusion. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'04)*, volume 1, pages 415–420, 2004.

[75] Z. Yu and C. Bajaj. Automatic ultrastructure segmentation of reconstructed cryoem maps of icosahedral viruses. *IEEE Transactions on Image Processing: Special Issue on Molecular and Cellular Bioimaging*, 14(9):1324–37, Sept. 2005.

[76] Z. Yu and C. Bajaj. Computational approaches for automatic structural analysis of large biomolecular complexes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2007.

[77] Y. Zhou and A. Toga. Efficient skeletonization of volumetric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5:196–209, 1999.

[78] Z. Zhou, M. Dougherty, J. Jakana, J. He, F. Rixon, and W. Chiu. Seeing the herpes virus capsid at 8.5 angstrom. *Science*, 288:877–880, 2000.

[79] Z. H. Zhou, M. L. Baker, W. Jiang, M. Dougherty, J. Jakana, G. Dong, G. Lu, and W. Chiu. Electron cryomicroscopy and

bioinformatics suggest protein fold models for rice dwarf virus. *Nature Structural Biology*, 2001.

Exercises

Chapter 4

Geometry of Finite Elements

chap4 describes geometry to geometric features and finite element meshing: triangular, tetrahedral, quadrilateral and hexahedral meshes, A-patch and B-Spline models,

4.1 Reconstruction

from point clouds to boundary meshes

The following algorithm is taken from [29] (Bajaj2010HLS).

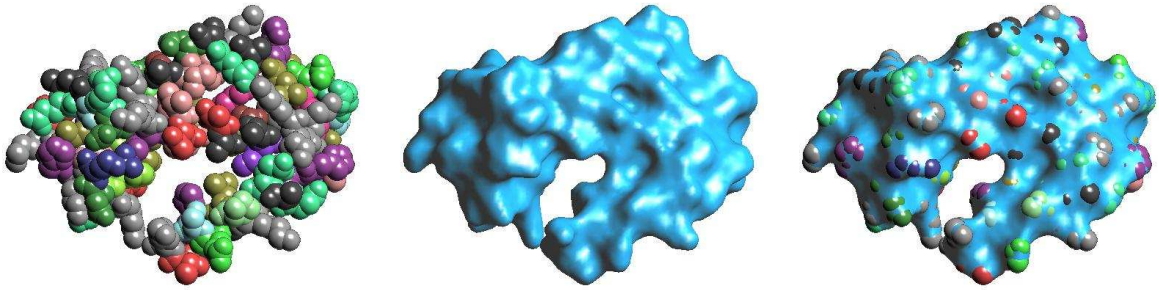
Given a non-negative function $g(\mathbf{x})$ over a domain $\Omega \in \mathbb{R}^3$. Find a surface Γ in Ω, such that the energy functional

$$E(\Gamma) = \int_{\Gamma} g(\mathbf{x}) d\mathbf{x} + \varepsilon \int_{\Gamma} h(\mathbf{x}, \mathbf{n}) d\mathbf{x} \tag{1.1}$$

is minimal, where $\mathbf{x}$ and $\mathbf{n}$ is a surface point and its surface normal, respectively. Further, $h(\mathbf{x}, \mathbf{n})$ is another given non-negative function defined over $\mathbb{R}^3 \times \mathbb{R}^3$ which is used for regularizing the constructed smooth surface. Finally, $\varepsilon \geq 0$ is a constant. Many solid and physical modeling problems, such as surface (solid boundary) reconstruction, and physically based simulation of deformable interfaces could be formulated as minimizing an energy in the form of (1.1). By minimizing the energy functional (1.1), a partial differential equation (PDE) in the level-set formulation can be generated. The PDE is solved using the higher-order spline level-set (HLS) method that we present here. Fig. 4.1 and 4.2 show two examples of smooth surface construction, an interface (surface) that separates a molecule's atoms from the solvent atoms (typically water), and a smooth surface reconstruction fit to a point set generated by a 3D surface scanner.

Why Use a Level-set Method? In shape deformation simulations, topology changes may occur. This topology change makes parametric form surface tracking difficult. However, implicit form surface deformation could overcome this difficulty. Implicit surface splines, such as tetrahedral A-patches and prism A-patches, have been successfully used in computer graphics and surface modeling in the past decades (see [8, 248] for references), however mostly used in static surface modeling. The level-set method described here allows one to dynamically deform and track an implicit surface spline using a governing PDE, which describes various laws of motion depending on geometry, external forces, or a desired energy minimization (see [174, 175, 182, 272, 273, 274]). Furthermore, the underlining data structure is simple and topological changes are handled easily, with the computation being restricted to a thin shell (traditionally called a narrow band for curve evolution) surrounding the level-set.

Why Use a Higher-order Spline Method? The level-set surfaces obtained from classical level-set methods are generally bumpy due to the use of piecewise tri-linear interpolation from the discrete function data computed on a rectilinear grid. To produce a better quality surface, a denser grid needs to be used. However, the increased grid resolution substantially increases the computation costs. Another drawback of using discrete data over grids is the non-trivial requirement of estimating derivatives for smooth interpolation. In many surface construction problems, such as the construction of molecular surface,



(a) (b) (c)

Figure 4.1: (a) shows the van der Waals surface of a molecule. (b) shows the corresponding solvent excluded molecular surface constructed using our C^2 tri-cubic spline level-set method. (c) illustrates that the smooth solvent excluded surface constructed tightly encloses the van der Waals surface (a).

(a) (b) (c)

Figure 4.2: (a) shows the point set data produced by a 3D surface scanner. (b) shows the the reconstruction result using our C^2 tri-cubic spline level-set method. (c) shows the reconstruction result for a C^0 tri-linear level-set method (also generated from our implementation).

the underlining surface is at least C^1 smooth. Therefore, a smooth level-set function is highly desirable. In this paper, we are solving second-order geometric partial differential equations. In the solutions of these PDE's we utilize accurate estimates of mean surface curvature Therefore, we use C^2 tri-cubic spline as the level-set function basis. Note that tri-cubic is the lowest order B-spline that could achieve C^2 continuity in 3D. The advantages using C^2 spline function bases include:

[A] Since the level-set function is C^2 smooth, the level-set surface is G^2 smooth. There do exist a finite number of critical level-set values where the level-set may have a finite number of isolated singular points (i.e the gradient of the level-set function vanishes). However working in a finite precision numerical domain one automatically avoids these finite set of critical level-set values.

[B] Derivatives up to the second order and curvatures, which appear in the governing geometric partial differential equations, are easily and exactly computed from the C^2 level-set function.

[C] Using smooth level-set functions implies that larger and higher-order spline iso-surface patches could be directly generated.

Main Contributions. We derive and describe an efficient solution of higher-order local level-set methods. Using a first order energy functional, a second order geometric partial differential equations (PDE) in the level-set formulation is derived. To achieve higher efficiency in using tri-cubic spline functions for solving PDEs, a fast and exact cubic spline interpolation algorithm is presented. Furthermore, the local property of cubic spline function are analyzed. Based on this analysis, a local spline level-set method is realized. Though we use tri-cubic spline functions here, we could easily use any other order splines as well. We apply our method to the smooth construction of biomolecular solvation interfaces, and smooth surface reconstruction from scanned 3D point-set data. Our construction method is simple, efficient and error bounded.

Algorithm Outline

From minimizing the energy (1.1), we obtain the following evolution equation for the detail derivation and [255] for more geometric PDEs)

$$\begin{aligned}\frac{\partial \phi}{\partial t} &= (g+\varepsilon h)\mathrm{d}iv\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\| + \varepsilon\ \mathrm{d}iv(\mathrm{P}\nabla_n h)\|\nabla\phi\| \\ &+ 2[\nabla(g+\varepsilon h)]^T\nabla\phi = L(\phi)+H(\nabla\phi), \end{aligned} \tag{1.1}$$

where

$$\begin{aligned} L(\phi) &= (g+\varepsilon h)\mathrm{d}iv\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\|, \\ H(\nabla\phi) &= \varepsilon\ \mathrm{d}iv(\mathrm{P}\nabla_n h)\|\nabla\phi\| + 2[\nabla(g+\varepsilon h)]^T\nabla\phi, \end{aligned}$$

$\mathrm{P} = \mathrm{I} - \frac{\nabla\phi}{\|\nabla\phi\|}\otimes\frac{\nabla\phi}{\|\nabla\phi\|}$ is a projection operator onto the tangent space and I indicates the identity mapping. ∇ and ∇_n denote the usual gradient operator with respect to $\mathbf{x}$ and $\mathbf{n}$, respectively. Note that $L(\phi)$ is a parabolic term and $H(\nabla\phi)$ is a hyperbolic term. Hence, in solving equation (1.1) in the following, the first order term $H(\nabla\phi)$ is computed using an upwind scheme (see [177] for the reason of using an upwind scheme) over a finer grid, the higher order term $L(\phi)$ is computed using a spline presentation defined on a coarser grid.

Consider the solution of equation (1.1) over the domain $\Omega = [a,b]\times[c,d]\times[e,f]\in\mathbb{R}^3$. For simplicity, we assume $b-a = d-c = f-e > 0$. We suppose that the domain Ω is uniformly partitioned with vertices $G_0 = \{\mathbf{x}_{ijk}\}_{ijk=0}^n := \{x_i\}_{i=0}^n\times\{y_j\}_{j=0}^n\times\{z_k\}_{k=0}^n$, where

$$x_i = a+i\Delta x,\ y_j = c+j\Delta y,\ z_k = e+k\Delta z,$$

and $\Delta x = \Delta y = \Delta z = (b-a)/n$. Let G_l be the set of vertices of the grid which is generated by binary subdividing G_0 uniformly l times. Let ϕ be a piecewise tri-linear level-set function over the grid G_l, Φ be a tri-cubic spline approximation of ϕ over the grid G_0. In general, l is chosen as 0 or 1 or 2. In our implementation, we take $l = 1$. If $l = 0$, Φ and ϕ are defined on the same grid G_0. This is the simplest case. The aim of the following algorithm is to compute the spline level set function Φ.

Algorithm Steps.

[A] **Initialization**. Given a initial Γ, construct a piecewise tri-linear level-set function ϕ over the grid G_l. If necessary, apply a re-initialization step to set ϕ to be a signed distance function to Γ (see section 4.1 for detail). Convert ϕ to Φ (see section 4.1).

[B] **Evolution**. Resampling Φ to obtain a new ϕ over the grid G_l. Compute $L(\Phi)$ and $H(\nabla\phi)$ in the thin shell $N = \{(x_i,y_j,z_k)\in G_l : |\phi(x_i,y_j,z_k)| < \mathscr{H}\}$. Update ϕ in N for one time step to get $\tilde{\phi}$ by an ODE time stepping method (see section 4.1).

[C] **Re-initialization**. Applying re-initialization step to $\tilde{\phi}$ in the shell

$$N = \{(x_i,y_j,z_k)\in G_l : \min_{-1\le\mu,\nu,\lambda\le 1}|\phi_{i+\mu,j+\nu,k+\lambda}| \le \mathscr{H}\}.$$

to get a new ϕ (see section 4.1). Convert ϕ to Φ (see section 4.1). Go back to step 2 if the termination condition is not satisfied.

[D] **Iso-contouring**. Extract 3-sided or 4-sided iso-surface patches (vertices with normals) of $\Phi = c$, where c is a given iso-value. G^1 approximate these patches by parametric surfaces.

Remark 2.1. For the problem of molecular surface construction, the grid size G_0 should be less than the radii of atoms so that atoms are distinguishable from the level set surface. In our implementation, the grid size is chosen to be one-half of the minimal value of the atom radii.

Remark 2.2. The aim of using $l > 0$ is to make ϕ a more accurate approximation of the signed distance function. The larger the value of l we use, the better approximation of the signed distance function we have. Since the scanned data to be approximated in general suffers from noise, we use the approximation Φ over a coarse grid G_0 for denoising. Furthermore, for generating larger level-set surface patches, again a coarse grid needs to be used.

Fast Cubic Spline Interpolation

Algorithm Let

$$\beta^3(x) = \begin{cases} \frac{2}{3} - x^2 + \frac{1}{2}|x|^3, & 0 \le |x| < 1, \\ \frac{1}{6}(2-|x|)^3, & 1 \le |x| < 2, \\ 0, & 2 \le |x|. \end{cases}$$

be the cubic B-spline base function over the knots $\{-2,-1,0,1,2\}$, and let

$$s(x) = \sum_{i=1}^{n-1} c_i \beta^3(x-i)$$

be a cubic spline function. Then the spline interpolation problem is to determine the coefficients $\mathbf{c} = \{c_i\}_{i=1}^{n-1}$ such that the following interpolation conditions

$$s(k) = \sum_{i=1}^{n-1} c_i \beta^3(k-i), \;\; k = 1, \cdots, n-1 \tag{1.1}$$

are satisfied for any given function values $\{s(k)\}_{k=1}^{n-1}$. Such a problem could be easily solved by solving the linear system (1.1) directly for the unknowns $c_1, \cdots, c_{n-1}$. The algorithm derived here is in $O(n)$ complexity. Given two function values $s(0)$ and $s(n)$, system (1.1) could be augmented as

$$s(k) = \sum_{i=1}^{n-1} c_i \beta^3(k-i), \;\; k = 0, \cdots, n. \tag{1.2}$$

Recursive Algorithm. *Using the initial values*

$$c_0^+ = -\frac{1}{1-z_1^{2n}} \sum_{k=1}^{n-1} s(k) z_1^k (1 - z_1^{2(n-k)}), \quad c_n^- = 0, \tag{1.3}$$

the recursive processes

$$\begin{aligned} c_k^+ &= s(k) + z_1 c_{k-1}^+, \quad k = 1, 2, \cdots, n-1, \\ c_k^- &= z_1 (c_{k+1}^- - c_k^+), \quad k = n-1, n-2, \cdots, 1, \end{aligned}$$

yields exact B-spline coefficients $c_i = 6c_i^-$ *of the interpolation problem, where* $z_1 = \sqrt{3} - 2$.

Remark 4.1.1. The computation of c_0^+ can be accelerated by neglecting small terms. Given a error control tolerance ε, we compute

$$K = \left\lceil \frac{\log(\varepsilon)}{\log(|z_1|)} \right\rceil.$$

If $n > K$, we replace c_0^+ in (1.3) with

$$c_0^+ = -\frac{1}{1-z_1^{2n}} \sum_{k=1}^{n-1} s(k) z_1^k, \tag{1.4}$$

which can be computed using the Horner scheme for evaluating a polynomial.

Locality of Spline Interpolation Since the basis function $\beta^3(x)$ is locally supported, the ith coefficient c_k the spline function of

$$s(x) = \sum_{i=1}^{n-1} c_i \beta^3(x-i)$$

has influence on $s(x)$ only on the interval $(k-2, k+2)$. The inverse is not true. A set of function values $\{s(i)\}$

$$s(i) = \begin{cases} 0, & i \neq k, \\ 1, & i = k, \end{cases}$$

define a set of coefficients $\{c_i\}$, each of which is nonzero. However, the coefficients $c_{k\pm l}$ is decreasing with respect to l. To see this, let us compute c_k^-. Suppose we consider the interpolation problem over $(-\infty,+\infty)$. From the definition of c_k^-, we have

$$c_k^- = -z_1(1+z_1^2+z_1^4+\cdots) = -\frac{z_1}{1-z_1^2} = \frac{\sqrt{3}}{6}.$$

Similarly,

$$c_{k\pm l}^- = -z_1^{l+1}(1+z_1^2+z_1^4+\cdots) = -\frac{z_1^{l+1}}{1-z_1^2} = z_1^l\frac{\sqrt{3}}{6},$$

for $l = 0,1,2,\cdots$. Therefore,

$$c_{k\pm l} = z_1^l\sqrt{3},\ \ l = 0,1,2,\cdots.$$

Note that $z_1 = \sqrt{3}-2 = -0.2679491924$. $c_{k\pm l}$ will decrease to zero rapidly. The first ten values are 1.732, -0.464, 0.124, -0.0333, 0.00893, -0.00239, 0.000641, -0.000172, 0.0000460, 0.0000123. Given a threshold values ε, the coefficients $c_{k\pm l}$ with $|c_{k\pm l}| < \varepsilon$ could be ignored. We take $\varepsilon = 0.005$. Therefore, we keep the term

$$c_{k\pm l},\ \ l = 0,1,2,3,4.$$

If we require that the second order derivatives have accuracy ε, we need to determine l, such that

$$|z_1|^l\sqrt{3} < \Delta x^2\varepsilon, \tag{1.5}$$

where Δx is the spacing of the knots.

Remark 4.1.2. High dimension spline interpolation is recursively computed by using lower dimensional spline computations. We do not mention the details of this here.

Conversion of Piecewise Tri-linear Functions to Tri-cubic Splines Let ϕ_l be a piecewise tri-linear function over the grid G_l. Now we want to convert it approximately to a trivariate tri-cubic spline function Φ over the grid G_0. If $l = 0$, the conversion algorithm is described in section 4.1. Now we assume $l > 0$. The conversion could be done in several ways. One simple way is to use Φ to interpolate ϕ_l over the grid G_l in the least square sense. This requires us to solve a linear system with size n^3, which is the number of grid in G_0. Another algorithm is firstly convert ϕ_l to a spline function Φ_l over the grid G_l, and then use repeatedly the knots removal algorithm of spline function to obtain Φ_{l-1}, Φ_{l-2}, $\cdots$, Φ_0 . Since Φ_k may not be represented exactly by Φ_{k-1}, a least square approximation has to be used. This also leads to solving a linear system.

The method we use in this paper is to convert ϕ_l to ϕ_{l-1}, ϕ_{l-2}, $\cdots$, ϕ_0, where ϕ_k is a piecewise tri-linear function over the grid G_k. Finally, we convert ϕ_0 to Φ using the conversion algorithm described in section 4.1. For converting ϕ_k to ϕ_{k-1}, we use a local tri-linear function at each grid point of G_{k-1} to interpolate ϕ_k at 27 neighbor grid points of G_k in the least square sense. This leads to a 8×8 linear system of equations for each of the grid point of G_{k-1}. Note that all these systems have the same coefficient matrix. Hence, we only need to inverse the matrix once. Furthermore, this conversion are conducted only in a neighborhood of the level-set surface (see section 4.1). Hence it is very efficient.

Solving the PDE by a Local Level-set Method

In this section, we describe a local level-set method, using a tri-cubic spline level-set function Φ over the grid G_0 and a piecewise tri-linear level-set function ϕ over the grid G_l, which represents a signed distance function. We first define a thin shell around the interface in which the level-set function is updated. Then we determine the initial level-set function followed by the evolution of the level-set function. Finally we explain the re-initialization step.

Thin Shell of a Tri-cubic Spline During the evolution process, we confine the movement of the level-set surface to no more than one grid size Δx of G_l for each time step. Since $\|\nabla\phi\| = 1$ before the movement, we require that function values, and up to the second order partial derivatives of ϕ, are accurately computed (within the given error tolerance) in the $c+\Delta x$ neighborhood of the interface, where c is the iso-value used for extraction the level-set (see the last step of main algorithm in section 4.1). From (1.5), we define the thin shell as

$$N = \{\mathbf{x}\in G_l : |\phi(\mathbf{x})| \le \mathscr{H}\}.$$

where

$$\mathscr{H} = \left[\frac{\log(\Delta x)^2 \varepsilon - \log\sqrt{3}}{\log|z_1|} + 2 \right] \Delta x + |c|. \tag{1.6}$$

As mentioned before, we take $\varepsilon = 0.005$. Function values that are beyond $\mathscr{H}$ are truncated by $\mathscr{H}$, meaning if $\phi(\mathbf{x}) > \mathscr{H}$, then set $\phi(\mathbf{x})$ as $\mathscr{H}$, if $\phi(\mathbf{x}) < -\mathscr{H}$, then set $\phi(\mathbf{x})$ as $-\mathscr{H}$. The evolution of ϕ is performed in this thin shell. The time step is so determined that the function values of ϕ are not changed more than Δx. By using an explicit Euler method, this is easy to achieve. After the evolution step, the function ϕ is truncated by $\mathscr{H}$.

In the re-initialization step, the thin shell is updated by introducing new vertices that are the neighbor vertices of the previous thin shell, and furthermore by removing the vertices whose values are greater than $\mathscr{H}$ in magnitude.

Initialization Let Γ^0 be a given initial closed surface with interior $\mathscr{D} \subset \mathbb{R}^\ni$. We define an initial level-set function $\phi^0(\mathbf{x})$ in $\mathbb{R}^3$ satisfying

$$\begin{cases} \phi^0(\mathbf{x}) < 0 & \text{in } \mathscr{D}, \\ \phi^0(\mathbf{x}) = 0 & \text{on } \Gamma^0, \\ \phi^0(\mathbf{x}) > 0 & \text{elsewhere.} \end{cases} \tag{1.7}$$

Γ^0 may be given as a close triangular surface mesh, or a level-set of a given function or scattered data set. For each of the three cases, the initial ϕ^0 needs to be computed differently.

(a) Γ^0 is a surface mesh. If Γ^0 is a surface mesh, we first compute the Euclidean distance from the grid points to the surface mesh using a local fast method (see Algorithm 4.1.1), and then we specify negative/positive sign to the distance values for the grids inside/outside the surface (see Algorithm 4.1.2). Since the function values need to be exact only in the thin shell, we compute distance values for the grid points locally around each triangle within the distance $\mathscr{H}$. This is done by the following algorithm.

Algorithm 4.1.1. *Distance Computation*

[A] Set initial values for each grid point to be $\mathscr{H}$.

[B] For each triangle of the mesh, find a containing cube such that the distance of the cube boundary to the triangle is no less than $\mathscr{H}$. For each grid point in the cube, compute the distance from the grid point to the triangle. If this distance is less than the existing value previously computed, then replace it with the newly computed one.

Algorithm 4.1.2. *Sign Designation*

[A] Perturb the vertices of the given triangular surface mesh such that (i) the x-components of all the face normals are not zero; (ii) The projections of all the edges of the mesh onto the yz-plane do not pass through any grid point on this plane.

[B] Compute distance from the grid points to the (perturbed) surface mesh using Algorithm 4.1.

[C] First we associate each grid point $\mathbf{q} \in \{\mathbf{q}_{jk}\}_{jk=0}^n := \{y_j\}_{j=0}^n \times \{z_k\}_{k=0}^n$ on the yz-plane with a set of numbers, called $\mathbf{q}$-pool. This pool is empty at beginning. Let $\triangle_x$ be the projection of the triangle $\triangle$ of the surface mesh onto the yz-plane. For each grid point $\mathbf{q} \in \triangle_x$, compute the intersection of the line segment $[x, \mathbf{q}]^T \in \mathbb{R}^3$, $x \in [a,b]$ with the triangle $\triangle$. Put the x-component of the intersection point into $\mathbf{q}$-pool.

[D] After the computation above, we have a pool for each grid points in the yz-plane. Then we sorting the numbers in each of the pool in the increasing order. Let $t_1 < t_2 < \cdots < t_k$ be the sequence in the $\mathbf{q}$-pool. Then if $a + j\Delta x$ is in the interval $[t_i, t_{i+1}]$, the sign to the grid point $[a + j\Delta x, \mathbf{q}]^T$ is set to $(-1)^i$, where $t_0 = a$, $t_{k+1} = b$.

Remark 4.1.1. The aim of the perturbation in the first step of Algorithm 4.1.1 is to make the intersection of the line $[x, \mathbf{q}]^T$, $x \in [a,b]$ with $\triangle$ to be unique and there is no duplicated numbers in each of $\mathbf{q}$-pool. This perturbation is not harmful to the surface construction algorithm, since Γ^0 is served as an initial surface.

Remark 4.1.2. The time complexities of both Algorithm 4.1.1 and 4.1.2 are linear with respect to the number of triangles of the given mesh.

(b) Γ^0 is a level-set. If Γ^0 is a level-set $\{\mathbf{x} : d(\mathbf{x}) = c\}$ of a smooth function $d(\mathbf{x})$, a naive method is to first extract the level-set and then compute a signed distance function using the method described earlier. Since extracting iso-surface is not a trivial

task, we seek a more efficient method. Suppose $\nabla d(\mathbf{x}) \neq \mathbf{0}$, $\mathbf{x} \in \Gamma^0$, then a simple way is to use $d(\mathbf{x}) - c$ or $c - d(\mathbf{x})$ as the initial ϕ^0. For example, and c could be chosen as 2γ (see section in the problem of solvent excluded surface construction, $d(\mathbf{x})$ is a Gaussian map, $c = 1$ (see section 4.1).

(c) Γ^0 is a scattered data set. Suppose Γ^0 is given as a set of scattered data $\{\mathbf{x}_i\}_{i=1}^m$ obtained from sampling a sold with sampling density γ. First we present an algorithm to compute the distances from grid points to the scattered data in the neighborhood of the data set.

Algorithm 4.1.3. *Distance Computation*

[A] Set initial values for each grid point to be $\mathscr{H}$, where $\mathscr{H} > 0$ is a given constant representing the neighborhood size.

[B] For each point $\mathbf{x}_i$ of the given scattered data, find a containing cube such that the distance of the cube boundary to $\mathbf{x}_i$ is not less than $\mathscr{H}$. For each grid point in the cube, compute the distance from the grid point to $\mathbf{x}_i$. If this distance is less than the existing value previously computed, then replace it with the newly computed one.

Remark 4.3. The complexity of Algorithm 4.1.3 is linear with respect to the number of points of the scattered data.

Algorithm 4.1.4. *Computation of ϕ^0*

[A] Compute the distances from the grid points to the scattered data using Algorithm 4.1.3 with $\mathscr{H} = \max\{\gamma, \Delta x\}$.

[B] A grid point is labeled as boundary point if its distance value is $\mathscr{H}$ and at least one of its neighbors has distance value less than $\mathscr{H}$. To label all the boundary points, we first search in x-direction starting from $\mathbf{x}_{0jk}$ for all fixed j and k. Once a boundary point is met, then labeled this point and stop this searching. This search need to be conducted for the six directions, i.e., x-direction, $-x$-direction, y-direction, $\cdots$. After the six direction searchings, put all the labeled boundary points in a list, then search in this list repeatedly, if a point in the list has a neighbor that has not been labeled, then label this point and put it into the list. This list searching process terminates if there is no point that has un-labeled neighbors.

[C] Compute the distances d_{ijk} from grid points $\mathbf{x}_{ijk}$ to the boundary grid points determined above using Algorithm 4.1.3 with $\mathscr{H}$ defined by (1.6) with $c = 0$.

[D] If a grid point $\mathbf{x}_{ijk}$ is not a exterior and boundary point, change the distance value d_{ijk} to $-d_{ijk}$. The signed distance function computed above is served as ϕ^0.

Remark 4.1.4. The algorithm is very efficient and gives quite close initial surface to the scattered data even for high genus solid models (see Fig. 4.6(b)). Since the boundary points are defined at the grid points, the iso-surface of ϕ^0 is bumpy.

Evolution Suppose we have an initial ϕ^0 satisfying (1.7). First we apply the local re-initialization step, if necessary, to set ϕ^0 to be the signed distance function of Γ^0 (see section 4.1). Then we define a thin shell around Γ^0 by

$$N^0 = \{\mathbf{x} \in G_l : |\phi^0(\mathbf{x})| < \mathscr{H}\}.$$

To prevent numerical oscillations at the boundary of the thin shell, we introduce a cubic cut-off function $c(x)$ in (1.8), which is defined by C^1 Hermite interpolating the boundary data on $[\mathscr{H}_0, \mathscr{H}]$

$$c(x) = \begin{cases} 1 & \text{if } |x| \leq \mathscr{H}_0, \\ \frac{(|x|-\mathscr{H})^2(2|x|+\mathscr{H}-3\mathscr{H}_0)}{(\mathscr{H}-\mathscr{H}_0)^3} & \text{if } \mathscr{H}_0 < |x| \leq \mathscr{H}, \\ 0 & \text{if } |x| > \mathscr{H}, \end{cases}$$

where $\mathscr{H}_0$ is chosen as $0.5\mathscr{H}$. Our experiment show that linear cut-off function works equally well. We update ϕ^0 by solving the following equation,

$$\begin{cases} \frac{\partial \phi}{\partial t} = c(\phi)[L(\phi) + H(\nabla\phi)], \\ \phi(\mathbf{x}, 0) = \phi^0(\mathbf{x}), \end{cases} \tag{1.8}$$

on N^0 for one time step and get $\phi^1(\mathbf{x})$. The time step is chosen such that the interface moves less than one grid size Δx. At each grid point $\mathbf{x}_{ijk}$ in the thin shell N^0, compute $v^0(\mathbf{x}_{ijk}) = c(\phi^0(\mathbf{x}_{ijk}))[L(\Phi^0(\mathbf{x}_{ijk})) + H(\nabla\phi^0(\mathbf{x}_{ijk})]$. Let

$$\tau = \min\left\{\Delta x, \Delta x / \max_{\mathbf{x}_{ijk}\in N^0} |v^0(\mathbf{x}_{ijk})|\right\}.$$

Then update ϕ^0 by the explicit Euler scheme

$$\phi^1(\mathbf{x}_{ijk}) = \phi^0(\mathbf{x}_{ijk}) + \tau v^0(\mathbf{x}_{ijk}), \;\; \mathbf{x}_{ijk} \in N^0.$$

Since ϕ^1 is no longer a signed distance function, a re-initialization step is required (see section 4.1) to get a new ϕ^1 and a new thin shell N^1. The process from ϕ^0 to ϕ^1 described above is repeated to get a sequence $\{\phi^m\}_{m\geq 0}$ of ϕ, and a sequence of thin shells $\{N^m\}_{m\geq 0}$, until the following termination conditions

$$\max_{\mathbf{x}_{ijk}\in N^m} |v^m(\mathbf{x}_{ijk})| < \varepsilon \;\text{ and }\; m < M \tag{1.9}$$

are satisfied. We choose $\varepsilon = 0.001$. M is a given upper bound of the iteration number, we choose $M = n$, where n^3 is the number of grid points in G_l.

Remark 4.1.5. The solution to equation (1.1) may have discontinuous derivatives, even if the initial data is smooth (see [177]). Using simple central difference to approximate the first order term is not appropriate. Hence, we use monotone and upwind scheme for Hamilton-Jacobi equations as developed in [177] for computing $H(\nabla\phi^m)$. The higher order term $L(\phi^m)$ is computed from the spline function Φ^m directly. Four monotone discretization schemes (Lax-Friedrichs scheme, Godunov type scheme, Local Lax-Friedrichs scheme and Roe with LLF entropy correction) have been reviewed in [177]. We use Godunov's scheme because it is also upwinding.

Adaptive Re-initialization To simplify the notation, let $\phi^m(\mathbf{x})$ be denoted as $\phi(\mathbf{x})$. In general, it is impossible to prevent $\phi(\mathbf{x})$ deviating away from a signed distance function. Flat and/or steep regions will develop around the interface, making further computation and contour plotting highly inaccurate. Hence a re-initialization step to reset the level-set function $\phi(\mathbf{x})$ to be a signed distance function to the interface $\{\mathbf{x} : \phi(\mathbf{x}) = 0\}$ is necessary. Keeping ϕ to be a signed distance function is desirable for numerically stability and necessary for certain applications (see next section) as well.

Re-initialization here is a process of replacing $\phi(\mathbf{x})$ by another function $\tilde{\phi}(\mathbf{x})$ that is the signed distance function to the zero level-set of ϕ and then taking this new function $\tilde{\phi}(\mathbf{x})$ as the initial data to use until the next round of re-initialization. To achieving this goal, people usually solve the following equation (see [182]) for the unknown function $d(\mathbf{x}, \tau)$

$$\begin{cases} \frac{\partial d}{\partial \tau} + S(d)(\|\nabla d\| - 1) = 0, \\ d(x, 0) = \phi(\mathbf{x}). \end{cases} \tag{1.10}$$

We propose to solve the following Hamilton-Jacobi equation,

$$\begin{cases} \frac{\partial d}{\partial \tau} + S(d)(\nabla d^T \nabla d - 1) = 0, \\ d(x, 0) = \phi(\mathbf{x}). \end{cases} \tag{1.11}$$

for its steady state solution, with $S(d)$ approximated by

$$s = \frac{d}{\sqrt{d^2 + |Dd|^2 \Delta x^2}}, \tag{1.12}$$

where Dd is a discrete approximation of ∇d. The advantages of using (1.11) instead of (1.10) is that the right-handed side of (1.11) is a smooth function of d, and it also facilitates the usage of semi-implicit temporal discretization scheme.

Discretization. Again, we use the Godunov scheme (see [177]) for the spatial discretization because it is monotone and upwinding. The scheme is described abstractly in [177] for a 2D problem. Here we present the details for our 3D problem (1.11)

$$\begin{aligned} d_{ijk}^{n+1} = d_{ijk}^n \; & -\Delta\tau s_{ijk}^+ \left(\max[(a^+)^2, (b^-)^2] + \max[(c^+)^2, (d^-)^2]\right. \\ & \left. + \max[(e^+)^2, (f^-)^2] - 1\right) \\ & -\Delta\tau s_{ijk}^- \left(\min[(a^-)^2, (b^+)^2] \; + \min[(c^-)^2, (d^+)^2]\right. \\ & \left. + \min[(e^-)^2, (f^+)^2] - 1\right), \end{aligned} \tag{1.13}$$

where s_{ijk} is the approximation to $S(s_{ijk})$ with (1.12), $(x)^+ = \max(x,0)$, $(x)^- = \min(x,0)$; a, b, c, d are defined by

$$a = D_x^- d_{ijk}^n, \;\; c = D_y^- d_{ijk}^n, \;\; e = D_z^- d_{ijk}^n,$$
$$b = D_x^+ d_{ijk}^n, \;\; d = D_y^+ d_{ijk}^n, \;\; f = D_z^+ d_{ijk}^n,$$

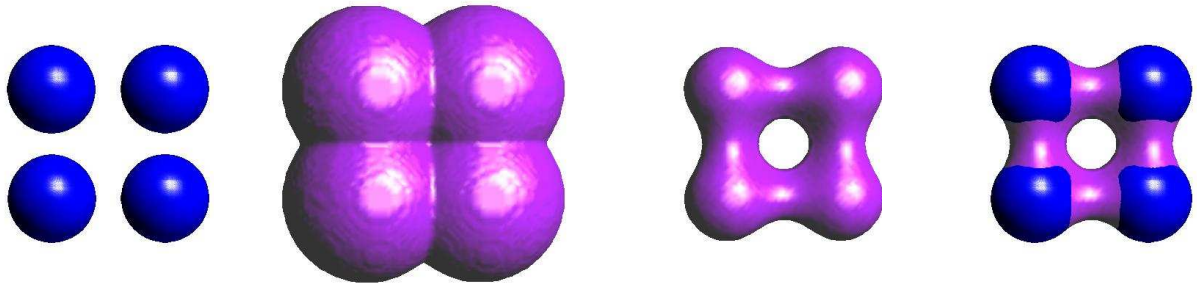
where $D_x^{\pm} d_{ijk}$, $D_y^{\pm} d_{ijk}$ and $D_z^{\pm} d_{ijk}$ denote the one-sided divided differences at the grid point $\mathbf{x}_{ijk}$ in x, y and z directions, respectively. These one-sided differences are computed with ENO (essentially non-oscillatory) interpolation (see [177]). Our experiment shows that the second order ENO interpolation usually performs the best among the first, second and third order interpolations. Scheme (1.13) is easily seen to be monotone; i.e., the right-handed side of (1.13) is a nondecreasing function of all the d_{ijk}^n if

$$\frac{\Delta\tau}{\Delta x}|s_{ijk}| < \frac{1}{2}. \tag{1.14}$$

This is also upwind, which means that away from sonic points (for those with either $S(d)d_x = 0$, or $S(d)d_y = 0$, or $S(d)d_z = 0$) the scheme has a domain of dependence which is in the same direction as that of the characteristics.

Adaptive Updating the Thin Shell. During the process of re-initialization, function values around the level-set change, and so the thin shell defined by $\{\mathbf{x} \in G_l : |\phi(\mathbf{x})| < \mathscr{H}\}$ should be updated correspondingly. This is automatically achieved by including new grids with at least one of their one-ring neighboring grids with value less than $\mathscr{H}$. Additionally we throw away the old grids in the current thin shell where the function values are greater than $\mathscr{H}$, and function values of all their one-ring neighbor grids are also greater than $\mathscr{H}$.

Smooth Biomolecular Surface Construction



(a) (b) (c) (d)

Figure 4.1: Schematic illustration of the construction algorithm. Figure (a) shows the van der Waals surface for a simple molecule of four atoms. (b) shows the solvent accessible surface defined by $\phi = 0$. (c) shows the solvent excluded surface defined by $\phi = -1.4$. The pink strips in figure (d) show the solvent contact surface where the solvent is taken to be a sphere with radius 1.4.

In this section, we use our higher order level-set method with C^2 tri-cubic spline functions to construct biomolecular interfaces. As is well known, there are typically three types of molecular surfaces [194], [20]: the van der Waals surface (VWS), the solvent-accessible surface (SAS), and the solvent-excluded surface (SES) or Connolly surface [71]. The van der Waals surface is defined from the van der Waals radii of the atoms, which is the boundary of the region formed by the union of all the spheres (atoms). The SAS introduced by Lee-Richards is defined to be the locus of the center of the (rolling) spherical water molecule which makes contact with the VWS surface [191]. Hence the SAS is an inflated VWS with a probe radius of 1.4 Angstroms. The solvent-excluded surface is the solvent (sphere) surface inside of which the probe never intrudes. In another word, SES is the offset surface of SAS in the inward direction with the solvent probe radius as the offset radius. This kind of surface can be represented by alpha shapes [143, 54, 55]. The alpha shape technique has been extensively used for molecular surface modeling [3], area and volume computation and cavity and pocket recognition [143], [144]. This technique requires a a complex geometrical data structure.

Gaussian blurred maps $\sum_{i=1}^m e^{-d(\|\mathbf{x}-\mathbf{x}_i\|^2 - r_i^2)/r^2}$ have been frequently used in the molecular surface construction (see [44], [85], [117], [21], [271]). Both VWS and SAS can be easily approximated by the level-set of the Gaussian map within any given tolerance by properly choosing the parameter d. However, the Gaussian map method cannot generate an accurate approximation

to the SES (see Fig. 4.3(a)). It is well known that SES is an important molecular surface (dielectric interface) for many applications, such as electrostatic energy computation (see [35]), various simulations (see [121]). Hence, we focus our attention in this section on the computation of SES using our HLS method.

Structural models of molecules M consisting of a collection of atoms with centers $\{\mathbf{x}_i\}_{i=1}^m$ and radii $\{r_i\}_{i=1}^m$ (see Fig. 4.1(a)) are retreivable from the Protein Data Bank (PDB). To construct the SES for M, we minimize the energy functional

$$E(\Gamma) = \int_\Gamma g(\mathbf{x})^2 d\mathbf{x} + \varepsilon \int_\Gamma d\mathbf{x}, \tag{1.15}$$

where

$$g(\mathbf{x}) = 1 - \sum_{i=1}^m e^{-C(\|\mathbf{x}-\mathbf{x}_i\|^2 - \tilde{r}_i^2)}, \quad \tilde{r}_i = r_i + r_b,$$

where r_b is the probe radius, which usually is 1.4. The constant $C > 0$ is determined so that $g(\mathbf{x}) = 0$ is an approximation of the solvent accessible surface within a given tolerance (see Fig. 4.1(b) and section 4.1). The second term of (1.15) is used to regularize the constructed surface, where ε is a small number. In the examples provided in the following, we choose ε as 0.1. The corresponding level-set formulation of the evolution equation for the energy (1.15) is

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= (g^2 + \varepsilon)\mathrm{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\| + 4g(\nabla g)^T\nabla\phi \\ &= L(\phi) + H(\nabla\phi), \end{aligned} \tag{1.16}$$

where

$$L(\phi) = (g^2+\varepsilon)\mathrm{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\|, \quad H(\nabla\phi) = 4g(\nabla g)^T\nabla\phi.$$

As before, the first order term $H(\nabla\phi)$ is computed using an upwind scheme over a finer grid, the higher order term $L(\phi)$ is computed using a spline presentation defined on a coarser grid. If ϕ is a signed distance function and a steady solution of equation of (1.16), then the iso-surface $\phi = -r_b$ is an approximation of SES (see Fig. 4.1(c)). The algorithm in section 2 could be specialized as follows.

Algorithm 4.1.1. *Solvent excluded surface construction*

[A] Compute $g(\mathbf{x})$ over the grid G_l (see section 4.1 for the fast computation of $g(\mathbf{x})$).

[B] Compute an initial ϕ by taking $\phi(\mathbf{x}) = g(\mathbf{x})$ and then update it using a re-initialization step, such that $\|\nabla\phi\| = 1$ (see section 4.1).

[C] Updating ϕ by solving equation (1.16) for one time step using a divided difference method (see section 4.1).

[D] Re-initialize ϕ, and then go back to the previous step if the stopping criterion (1.9) is not satisfied.

[E] Generate iso-surface $\{\mathbf{x} : \Phi(\mathbf{x}) = -r_b\}$, which is the required approximation of SES.

Remark 4.1.1. In step 2, $\phi(\mathbf{x})$ may change rapidly, we may slow down the changes of ϕ by dividing it with a constant before taking the re-initialization step. Since the gradient of $e^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]}$ is $2Ce^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]}[x-x_i, y-y_i, z-z_i]^T$, the length of the gradient is approximately $2C\tilde{r}_i$ at the sphere surface $\{\mathbf{x} : \|\mathbf{x}-\mathbf{x}_i\| = \tilde{r}_i\}$. Hence we normalize the function ϕ by dividing it with $2C * \max_i \tilde{r}_i$.

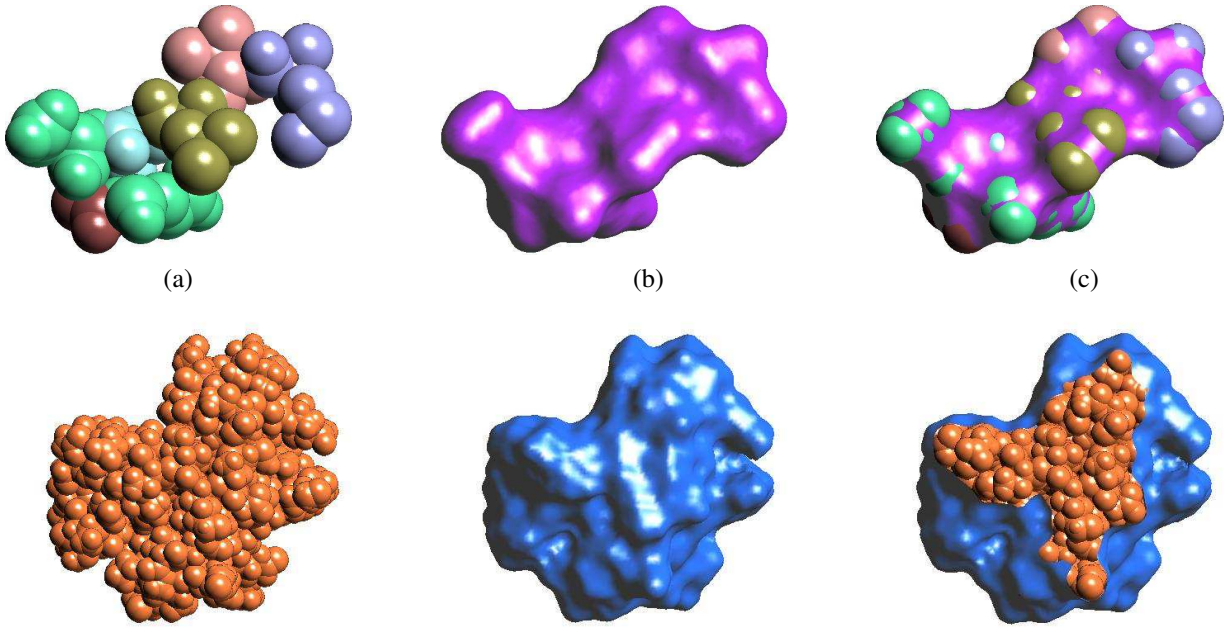
Error Bounded Approximation Let ε be a given an error tolerance for the constructed SES. We determine the constant C, that satisfies this error constraint. Assume that at each surface point of SAS, at most m atomic spheres intersect. Each of these spheres thus have a maximum contribution of $1/m$ to the Gauss map $g(\mathbf{x})$ at such surface points of the SAS . That is

$$e^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]} = e^{-C[(\tilde{r}_i+\varepsilon)^2-\tilde{r}_i^2]} = \frac{1}{m}.$$

From this we cobtain

$$C \approx \frac{\ln m}{2\tilde{r}_i\varepsilon}.$$

In general, if m is 4, we choose $\varepsilon = 0.01$, and C is chosen as $\frac{50\ln 4}{r_b + \min_i r_i}$.



(a) (b) (c)

(d) (e) (f)

Figure 4.2: (a) and (d) show the van der Waals surface (VWS). (b) and (e) are the corresponding solvent excluded surface (SES) constructed. (c) and (f) illustrate the tight enclosure of the smooth SES to the VWS.

Fast Computation of Gaussian Map Since each term $e^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]}$ in $g(\mathbf{x})$ decreases rapidly as $||\mathbf{x}-\mathbf{x}_i||$ increases, we evaluate $e^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]}$ locally around $\mathbf{x}_i$. Suppose we evaluate it for $\mathbf{x}$ such that

$$e^{-C[\|\mathbf{x}-\mathbf{x}_i\|^2-\tilde{r}_i^2]} > \varepsilon,$$

where ε is a given threshold value (for e.g. we take $\varepsilon = 10^{-5}$), we have

$$\|\mathbf{x}-\mathbf{x}_i\|^2 < \tilde{r}_i^2 - \frac{\ln \varepsilon}{C}. \tag{1.17}$$

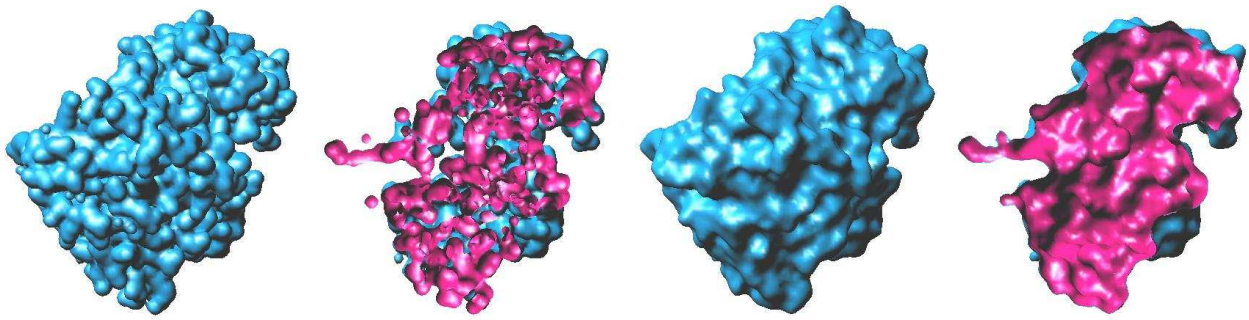
Hence, we only evaluate the term over the grids within the spherical range defined by (1.17). For the simplicity of implementation, we determine a minimal bounding cube containing the sphere defined by (1.17), and evaluate the function within the cube. The time complexity of this algorithm is obviously linear with respect to the number of atoms.

Illustrative Examples We have implemented our higher order level set algorithm in the molecular visualization software package TexMol [13]. We present a few examples to demonstrate the quality of the SES for molecules from the PDB. In these examples, the solvent probe (water molecule) radius is chosen to be 1.4. Fig. 4.2 shows two molecular surface constructions, where figures (a) and (d) show the VWS, respectively, (b) and (e) are the corresponding SES. (c) and (f) illustrate how the smooth SES tightly encloses the VWS and how the SES smoothly transits between the atoms (spheres).

Fig. 4.3 shows the difference between our method with Gaussian blurring (for e.g. [271]) for an enzyme with 1UDI as its PDB ID. The surface of Gaussian blurring is defined by $\{\mathbf{x} \in \mathbb{R}^3 : g(\mathbf{x}) = 1\}$, where

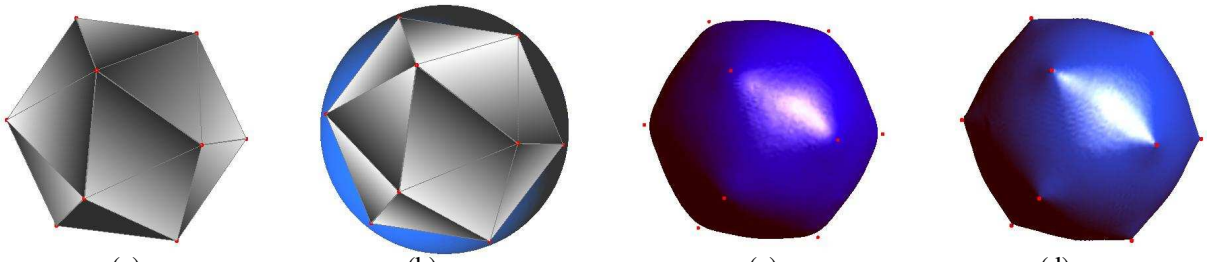
$$g(\mathbf{x}) = \sum_{i=1}^{n} e^{-\frac{d}{r_i^2}\left[\|\mathbf{x}-\mathbf{x}_i\|^2-(r_i)^2\right]}.$$

The surface generated by Gaussian blurring reveals creases (see Fig. 4.3(a)). These creases should be covered by the rolling solvent probe. The cutoff results further show that a lot of redundant interior structures exist (see Fig. 4.3(b)). Our method gives more accurate results to the final SES (see Fig. 4.3(c) and (d)).



(a) (b) (c) (d)

Figure 4.3: Comparing the results of Gaussian blurring and our new method. (a) and (b) show the result of Gaussian blurring with $d = -2.3$. (c) and (d) are the level-set surfaces of $\phi = -1.4$ of the level set method. (b) and (d) show cross-section views of (a) and (c), respectively, by a cross-section plane perpendicular to the view direction. The inner side of the surface is rendered in pink to differentiate with the outer side.



(a) (b) (c) (d)

Figure 4.4: The effect of regularization term. Figure (a) shows the input spherical data. (b) show the initial surface constructed. (c) shows the reconstruction result using $\varepsilon = 0.01\gamma^2$. (d) shows the reconstruction result using zero ε. Both (c) and (d) are the results of 350 iterations with $\tau = 0.0006$.

Surface Reconstruction from Scanned 3D Point-Set Data

In this section, we use our HLS method to construct surfaces from scanned 3D point-set data. Such a problem has been studied by many authors, such as Chen et al's group using optimization technique (see [233], [258], [257], [259]), Zhao et al's work (see [182], [273], [274]) using tri-linear level-set method. Our goal is to construct G^2 smooth surface using tri-cubic spline level-set function. Suppose we are given a set of scattered data $\{\mathbf{x}_i\}_{i=1}^m$ sampled from a closed surface with sampling density γ. Let $d(\mathbf{x})$ be a distance function to the scattered data. To construct a surface that fit the scattered data, we minimize the energy functional

$$E(\Gamma) = \int_\Gamma d(\mathbf{x})^2 d\mathbf{x} + \varepsilon \int_\Gamma d\mathbf{x}. \tag{1.18}$$

Again, the second term of (1.18) is used to smooth the constructed surface, where ε is a small number. In the examples provided in the following, we choose ε as $0.1\gamma^2$. The corresponding level-set formulation of the evolution equation for the energy (1.15) is

$$\begin{aligned} \frac{\partial \phi}{\partial t} &= (d^2+\varepsilon)\mathrm{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\| + 2(\nabla d^2)^T\nabla\phi \\ &= L(\phi) + H(\nabla\phi), \end{aligned} \tag{1.19}$$

where

$$L(\phi) = (d^2+\varepsilon)\mathrm{div}\left(\frac{\nabla\phi}{\|\nabla\phi\|}\right)\|\nabla\phi\|, \quad H(\nabla\phi) = 2(\nabla d^2)^T\nabla\phi.$$

As before, the first order term $H(\nabla\phi)$ is computed using upwind scheme over a finer grid, the higher order term $L(\phi)$ is computed using a spline presentation defined on a coarser grid. If ϕ is steady solution of equation (1.19), then the iso-surface

$\phi = 0$ is the approximation surface. The algorithm in section 2 could be specialized as follows.

Algorithm 4.1.1. *Surface Reconstruction from Scattered Data*

[A] Compute $d(\mathbf{x})$ over the grid G_l using Algorithm 4.1.3 with $\mathscr{H}$ defined by (1.6) and $c = 0$.

[B] Compute an initial $\phi(\mathbf{x})$ using Algorithm 4.1.4 and then updating it using a re-initialization step, such that $\|\nabla\phi\| = 1$ (see section 4.1).

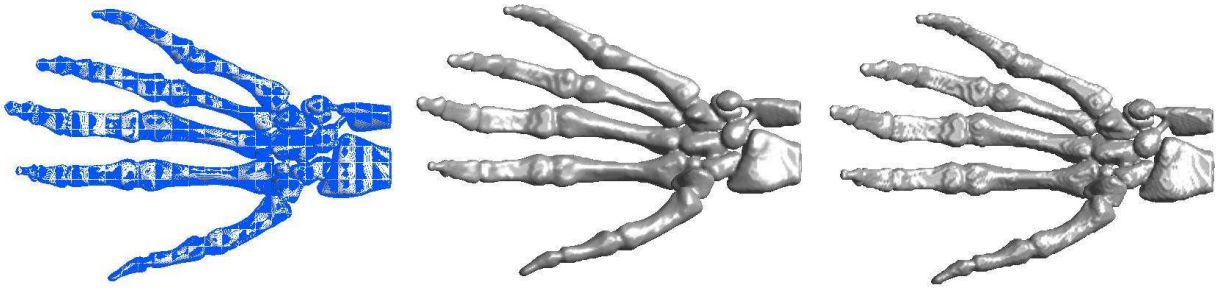
[C] Update ϕ by solving equation (1.19) for one time step using divided difference method (see section 4.1).

[D] Re-initialize ϕ, and then go back to the previous step if the stopping criterion (1.9) is not satisfied.

[E] Generate iso-surface $\{\mathbf{x} : \Phi(\mathbf{x}) = 0\}$, which is the required approximation of the scattered data.

Remark 6.1. The energy functional (1.18) has been used in the previous works (see [273], [274]) without the regularization term. Since the solution $\phi(\mathbf{x})$ of the equation will obviously approach to $d(\mathbf{x})$ as $t \to \infty$, while the level-set of $d(\mathbf{x})$ consists of the input isolated points. Hence the surface reconstructed will suffer from the first order discontinuous. Spinal points around the input points may be generated. With a properly chosen ε, the regularization term can prevent the level-set from generating spiny points. Fig. 4.4 shows a simple example, where the different effects of having and not having the regularization term are presented.

Fig. 4.5 and 4.6 show two reconstruction examples from input scattered data. Fig. 4.5(a) and 4.6(a) show the input scattered data points. Fig. 4.5(b) and 4.6(c) show the the reconstruction result using our tri-cubic C^2 spline level-set method. Fig. 4.5(c) and 4.6(d) show the reconstruction result using a tri-linear level-set method (our own level set method with a tri-linear basis). It could be observed that the results from the tri-cubic level-set method is smoother than that from tri-linear level-set method. Fig. 4.6(b) shows the initial surface construction using algorithm 4.4. It is easy to see that the constructed initial surface is very close the scanned surface point-set data.



(a) (b) (c)

Figure 4.5: (a) shows the input scanned point-set data in 3D. (b) shows the the reconstruction result using our tri-cubic C^2 spline level-set method. (c) shows the reconstruction result using a tri-linear version of our level-set method.

Conclusions

We have proposed a general framework of higher-order spline level-set (HLS) method for surface construction that could be used to solve smooth surface construction problems. We applied our method to the construction of C^2 smooth solvent excluded surfaces of biomolecules. Compared with Gaussian blurring techniques, our method yields even smoother solvent excluded surfaces with a tighter enclosure of the atomic sphere models. Next we applied our method to the c^2 smooth surface reconstruction from a scanned point-set in 3D data, leading to equally desirable results. Our HLS generalization of the linear order level set method, is of course applicable to several other smooth surface construction applications [174] where C^1 and higher-order smoothness is both necessary and/or desirable.

The following algorithm is taken from [10] (Bajaj1996Arbitrary).

(a) (b) (c) (d)

Figure 4.6: (a) shows the input scanned point-set data in 3D. (b) show the constructed initial surface using algorithm 4.4. (c) shows the the reconstruction result using our tri-cubic C^2 spline level-set method. (d) shows the reconstruction result using a tri-linear level-set method.

Introduction

Technologies such as magnetic resonance imaging (MRI), computed tomography (CT), and ultrasound imaging allow measurements of internal properties of objects to be obtained in a nondestructive fashion. These measurements are usually obtained one slice at a time, where each slice is a 2D array of scalar values corresponding to measurements distributed over a plane passing through the object. The set of planes generating the slices are usually parallel to each other and equispaced along some axis through the object.

Once these measurement slices have been obtained, the goal is to enable a human to easily visualize, in 3D, this large collection of data. Many algorithms have been developed for this purpose, but they can all be classified into two categories [89]: *volume rendering methods* and *surface reconstruction methods*.

This paper concentrates on surface reconstruction methods, all of which proceed by extracting the isosurfaces corresponding to a specified image intensity. Each isosurface is represented as an assembly of simple surface primitives, such as triangles or other polygons. Once these surface primitives are calculated, they can be quickly rendered from different viewpoints using widely available graphics hardware. This allows the user to quickly examine many different viewing spaces.

This paper presents a surface-based algorithm which achieves both faster rendering and lower likelihood of reconstruction error than previous surface reconstruction algorithms. These improvements are obtained by taking a unified approach to the three problems inherent in all surface-based approaches [162]: the correspondence problem; the tiling problem; and, the contour branching problem. These problems and their previous solutions are discussed in the following subsections.

Overview of Previous Approaches

The three fundamental problems in surface-based reconstruction – the correspondence problem, the tiling problem, and the branching problem – have motivated many research efforts.

The correspondence problem The correspondence problem involves finding the correct connections between the contours of adjacent slices. Fig. 4.7 shows an example with four different joint topologies (b)-(e) resulting from the same cross sections as in (a). If the distance between slices is large, *a priori* knowledge or global information is required to determine the correct correspondence. Bresler et al. [49] uses domain knowledge to constrain the problem. Meyers et al. [162] and Soroka [219]

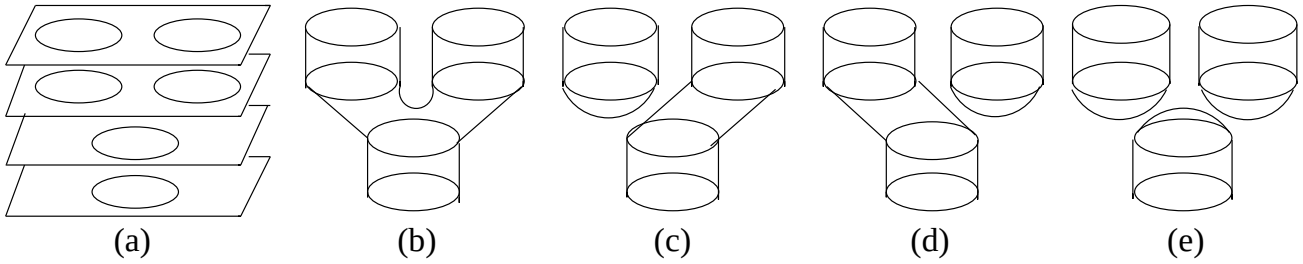


Figure 4.7: The correspondence problem: (a) cross section contours; (b)-(e) different topologies with the same cross sections as in (a).

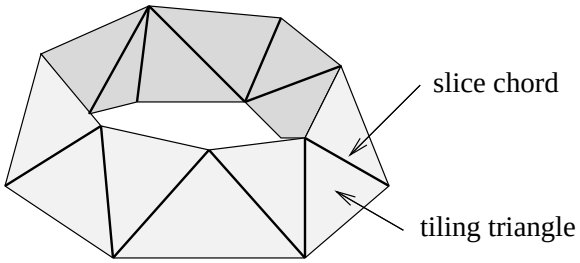


Figure 4.8: an example of tiling.

approximate the contours by ellipses and then assemble them into cylinders to determine the correspondence. Wang et al. [235] checks the overlapping area as the criterion for the correspondence.

The tiling problem Tiling means using slice chords to triangulate the strip lying between contours of two adjacent slices into tiling triangles (Fig. 4.8). A slice chord connects a vertex of a given contour to a vertex of contour in an adjacent slice. Each tiling triangle consists of exactly two slice chords and one contour segment. There are two related issues. One is how to accomplish optimal tiling in terms of certain metrics such as surface area, enclosed volume, etc. The other is the topological correctness of the tiling.

The problem of mating points between contours into triangles is formalized by Keppel [130] into a graph search problem. Fuchs et al. [99] provides an efficient algorithm based on an Euler tour of a toroidal graph to obtain an optimal solution. Their algorithm has a time complexity of $O(n^2 \log n)$, where n is the total number of vertices on the contours bounding the

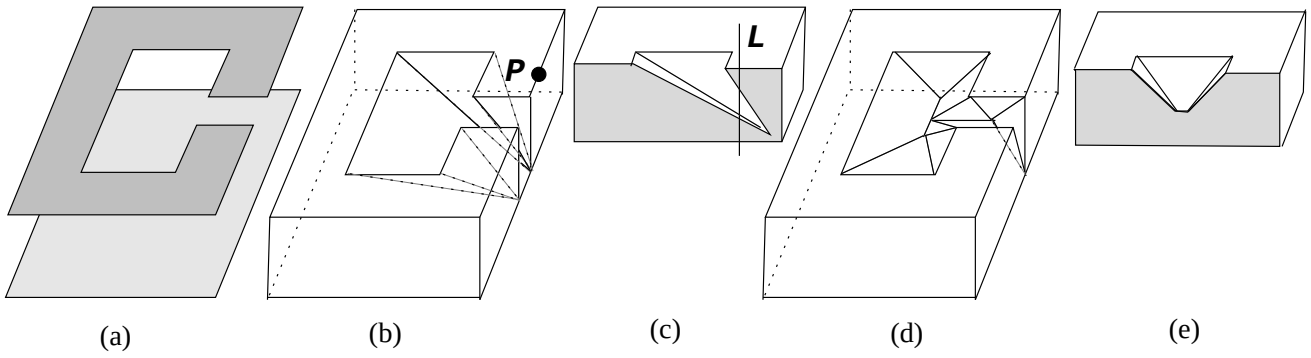


Figure 4.9: Tiling of dissimilar contours: (a) the contours of two adjacent slices. (b) a tiling in which all vertices of the top contours tile to vertices of the bottom contour, (c) the vertical section passing through the point P of the solid in (b), (d) a tiling in which the dissimilar portion of the top contour tiles to its medial axis located at between two slices. (e) a vertical section through the solid shown in (d).

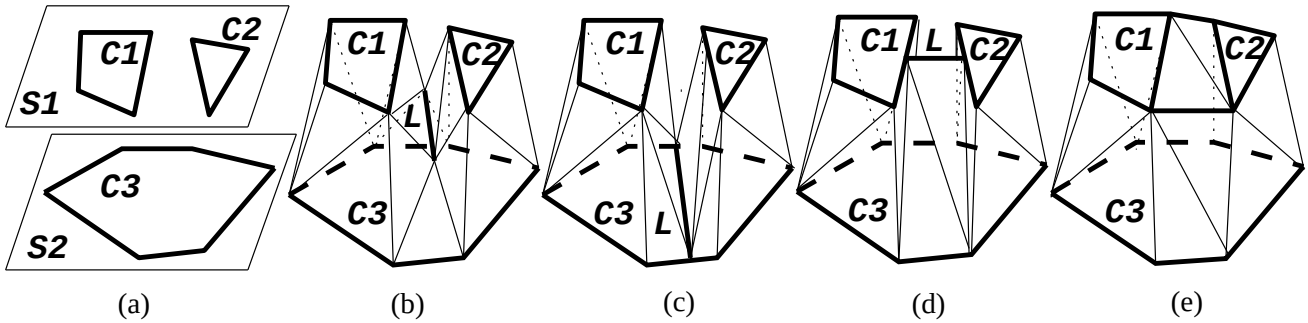


Figure 4.10: Different reconstructions for branching contours: (a) branching contours on adjacent slices, (b)-(e) different surface reconstructions.

triangles. Sloan et al. [217] locate bottlenecks and improve the speed of Fuchs' algorithm by a constant factor. Shinagawa et al. [212] generalizes the discrete toroidal graph into a continuous one. Homotopy is used for reconstructing smooth surfaces from the toroidal graph. Homotopy is similar to metamorphosis (morphing) in which one contour is gradually changed into another contour. Kehtarnavaz et al. [129] represent the search problem as a Levenshtein graph and use dynamic programming to find its minimum cost path. Wang et al. [235] present a method which first assigns an initial merit to each triangle. It then uses relaxation to iteratively refine these weights, and finishes by utilizing the A^* search algorithm to find a triangulation with minimum weight.

Some fast heuristic methods have also been developed for tiling. The strategy of Christiansen et al. [66] is based on selection of shortest slice chords. Ganapathy et al. [101] use the concept of least tension as a heuristic guideline to tiling. These heuristic methods [66, 101] usually work quickly and work well when the contours being matched have similar shapes. Ekoule et al. [88] develop an approach to tile two dissimilar contours which have similar convex hulls. The two convex hulls of corresponding contours are first heuristically tiled using shortest slice chord metric. Thereafter, their method maps the concave portions of a contour onto its convex hull to look up the tiling pair from the convex hull tiling. This method avoids some abnormalities produced by Christiansen's algorithm. After comparing different algorithms, Meyers [161] points out that the minimum surface area optimization approach produces fewer abnormalities.

When two corresponding contours are very different, it is difficult to obtain a topologically correct and natural tiling. Gitlin et al. [112] show one example in which two polygons cannot be tiled to form a polyhedron. Their example is a pair of extremely different contours. Even in a moderately dissimilar contour pair in which a polyhedron can be formed, the tiling algorithm may result in a non-polyhedron. For the example in Fig. 4.23(a), the minimum surface optimizing algorithm generates the non-polyhedral surfaces shown in Fig. 4.23(c) even though there exist many polyhedral solutions ((e) is one example). The arrow in Fig. 4.23(c) shows the self intersecting portions of the surface. Even when the tiling result is a polyhedron, it might be physically unlikely. For example, Fig. 4.9(a) shows two cross section contours. Fig. 4.9(c) shows a vertical cross section of a reconstruction that tiles the interior of the top contour to the bottom contour (see Fig. 4.9(b)). The scalar data along the vertical line L of (c) flips its sign twice between two adjacent slices. This is an unlikely topology, especially when the distance between the two slices is small. Fig. 4.9(d) shows another tiling in which the vertices of the dissimilar portion tile to the medial axis of the dissimilar portion. The medial axis is placed at between two slices. Fig. 4.9(e) shows a cross section of (d). It shows that Fig. 4.9(d) is a highly likely topology. All solid surfaces should be single sheeted; that is, the scalar data along a vertical line changes its sign at most once, between two adjacent slices if the distance between slices is fine enough. The tiling method shown in Fig. 4.9(b) violates this claim.

Algorithms [66, 73, 88, 99, 101, 129, 130, 217, 235] which attempt to tile all contour vertices to the adjacent slice produce an unlikely topology, as shown in Fig. 4.9(b). Boissonnat [45] and Barequet et al. [38] produce horizontal triangles which lie on the slice, thus avoiding tilings like those in Fig. 4.9(b). This generates a result better than Fig. 4.9(b) without adding any intermediate vertices.

The branching problem A branching problem occurs when a contour in one slice can correspond to more than one contour in an adjacent slice. Fig. 4.10(a) shows that contour $C3$ of slice $S2$ branches into $C1$ and $C2$ of slice $S1$. A contour in one slice having no corresponding contour in an adjacent slice forms either a hole or the beginning/end of a vertical feature. The possibility of branching significantly complicates the task of tiling. It creates the problem of branching surface reconstruction.

Lin et al. [145] model branching regions by interpolating many intermediate contours. This method generates a smooth surface at the cost of a large number of triangles. Other branch processing approaches can be classified into the four methods shown in Fig. 4.10(b)-(e). Fig. 4.10(b) shows that a curve L or a point is added between two slices to model the valley or saddle point formed by the branching. The added curve L is placed at slice $S2$ in (c). One or more line segments are added to form a composite contour as in (d); thereafter the tiling between the composite contour and $C3$ becomes one-to-one. Fig. 4.10(e) also forms a composite contour, which is the convex hull of the branching contours, in order to have one-to-one tiling. The branching region between $C1$ and $C2$ is filled up by horizontal triangles. In terms of topological correctness, the best branching handling is (b) because it corresponds to the expected physical object better than the others do.

Christiansen et al. [66], Shantz [205] and Shinagawa et al. [212] use the method in Fig. 4.10(d). They dip down the middle of the bridge to model the saddle point of the branching region. This approach works well only in simple branching cases. Ekoule et al. [88] form an intermediate contour, similar to a composite contour, between two slices for the case of one-to-many branching. The intermediate contour is tiled to the merging contour as well as to the branching contours. This produces less distortion than the method of Fig. 4.10(d). Meyers et al. [162] use the scheme in Fig. 4.10(e). They improve the horizontal triangle problem associated with this approach by feeding the triangulation mesh into a surface fitting program to regenerate the surface. Barequet et al. [38] first matches and tiles similar portions between corresponding contours. Then, the clefts, which are the polygons formed by the untiled portions, are triangulated. If the XY projections of clefts are nested, bridges are added to break the nesting. Their bridge adding scheme solves the problem of the possible conflict between a bridge in one slice and the geometry of the other slice. In the case of Fig. 4.10(a), the contour portions along the branching area are unmatched, and the triangulation result is similar to Fig. 4.10(d) or (e) depending on whether bridges are added or not.

Boissonnat [45] invents a different approach than tiling. He applies 3D Delaunay triangulation to contour vertices of two adjacent slices. The surface of the polyhedron formed by the union of tetrahedra is the desired surface. He reduces the problem of tetrahedralization of the object delimited by the two slices into building tetrahedra from two 2D Delaunay triangulations of two adjacent slices. Geiger [104] improves Boissonnat's approach so it can handle complicated branching and dissimilar contours. He projects the external Voronoi skeleton (EVS) from one slice to the adjacent slice and adds the projection in the 2D Delaunay triangulation. Thus, the triangles of a merging contour are split into several regions corresponding to each of the branching contours. Tetrahedra can be constructed between these corresponding regions. His branching handling is as in Fig. 4.10(c). In the case of dissimilar contours such as in Fig. 4.9(a), his algorithm tiles the dissimilar area (i.e. the interior) of the top contour to its EVS projection on the adjacent slice. This results in a topology similar to Fig. 4.9(d).

An analysis of the algorithms summarized above shows that they each violate at least one of the following guidelines:

1. As explained in Fig. 4.9, one should not enforce connecting *every* vertex of one contour to another contour if these two contours are very dissimilar.

2. The re-sampling of the reconstructed surface should yield the original contours. The branching methods shown in Fig. 4.10 (c), (d), and (e) violate this guideline.

3. Composite contours should not be formed (e.g. as shown on the top slice of Fig. 4.10(d) and (e)) because they do not correspond well to the actual physical object.

From the above discussion, it is clear that the problem of shape reconstruction is underconstrained, which implies that there are many feasible solutions. Our goal is to define surface criteria which, by imposing reasonable constraints on the problem, lead to an algorithm that generates the most likely object. These criteria must therefore lead through derivation to explicit correspondence and tiling rules. Holes, branching regions and dissimilar portions of contours will be detected because they can not tile to other vertices based on the derived rules. Unlike many other algorithms which treat holes, one-to-many branching, many-to-many branching and dissimilar contours as different special cases, our postprocessing algorithm treats them all in the same manner.

The algorithm we derive from a set of surface criteria does not violate any of the three guidelines listed above. The overall procedure followed by our algorithm is similar to that of Barequet et al [38] − we will discuss the similarities and differences in the implementation section. Our post-processing of the untiled region is similar to adding the EVS from one slice to the adjacent slice, as in Geiger's approach [104].

We present the theory behind our algorithm in Section 4.1, the implementation in Section 4.1, and the results in Section 4.1. We discuss our contributions and their limitations in Section 4.1, and conclude in Section 4.1. Lastly, the proofs of Section 4.1 are listed in the Appendix.

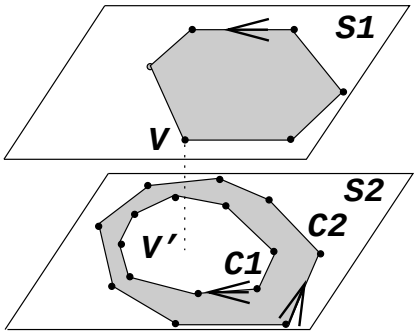


Figure 4.11: Examples of oriented contours on parallel slices. The shaded regions are the solid regions.

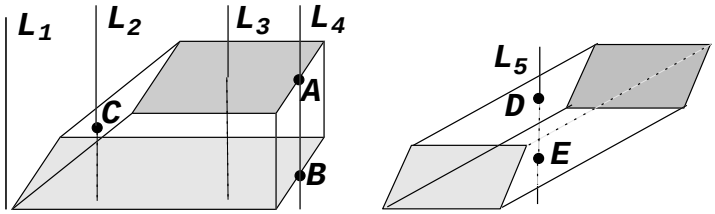


Figure 4.12: Criterion 2: The shaded regions are the solid regions of two slices, and the reconstructed surface is the surface covering the space between two solid regions. (a) A vertical line between two slices intersects the reconstructed surface at zero points (e.g. L_1 and L_3), one point (e.g. L_2), or one line segment (e.g. L_4). (b) The reconstructed surface shown is not allowed because the vertical line L_5 intersects the reconstructed surface at two points, D and E.

Correspondence and Tiling Rules

As discussed in Section 4.1, any surface-based approach must address the correspondence problem, the tiling problem and the branching problem. The tiling problem itself has two aspects: 1) obtaining an optimal tiling in terms of certain metrics, and 2) the detection and tiling of dissimilar portions of contours.

In this section, we address all of these problems simultaneously by first defining a set of criteria for the desired reconstructed surface. The criteria chosen can constrain the shape reconstruction problem so that the surfaces produced correspond well with expected physical models.

These criteria are then used to derive correspondence and tiling rules. The correspondence rules that are derived are local; they rely only on data in adjacent slices to determine the correspondence between contours. The tiling rules prohibit those tilings which result in undesired or nonsensical surfaces, and allow detection of branching regions and dissimilar portions of contours.

Section 4.1 will discuss how to use these rules to obtain a near-optimal tiling and to detect and process branching and dissimilar regions.

We define the input before presenting the surface reconstruction criteria. The input consists of two sets of contours, one set on each of two adjacent slices (Fig. 4.11). Each set contains zero or more contours which are simple polygons. A single contour divides the slicing plane into a *solid region* (shown as the shaded region of Fig. 4.11) and a *void region*. Like isocontours, contours are simple polygons and they can be inside other contours. They cannot, however, intersect each other on the same slice. A contour does not contain its interior. It is oriented so the solid region is on its left side. The solid region is thus inside a *CCW* (counter-clockwise) contour and is outside a *CW* (clockwise) contour. For the example in Fig. 4.11, contour $C1$ is *CW* and contour $C2$ is *CCW*. In the presence of looped contours, the solid region is is inside a *CCW* contour and is outside zero or more *CW* contours. for simplicity, we assume that a solid region does not intersect the image boundary. A vertex is one endpoint of a linear contour segment. We define it to have the same *CCW* or *CW* direction as the contour.

We now define the surface reconstruction criteria:

Criterion 1 *The reconstructed surface and solid regions form piecewise closed surfaces of polyhedra.*

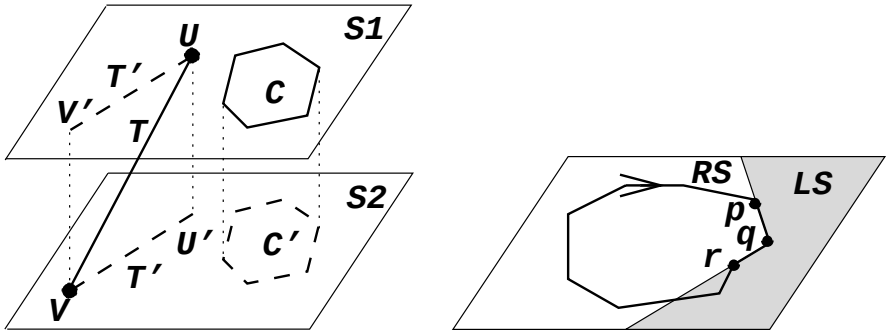


Figure 4.13: (a) Examples of projection. (b) A shadow region $\mathscr{LS}(q)$ of a vertex q.

Criterion 2 *Any vertical line (a line perpendicular to the slice) between two slices intersects the reconstructed surface at zero points, one point, or along line segment (Fig. 4.12(a)).*

Criterion 3 *Re-sampling of the reconstructed surface on the slice should produce the original contours.*

Criterion 1, which requires that surfaces be composed of polyhedra, prohibits such incorrect structures as self-intersecting surfaces. Criterion 2 is used to avoid the generation of unlikely topologies. This criterion is inspired from the unlikely topology of Fig. 4.9(c), in which a vertical line L intersects the reconstructed surface twice between two slices. This criterion may not be enforceable if the distance between two adjacent slices is large, or if the sampling plane is nearly tangent to the surface of a long, thin object. For example, the desired surface in Fig. 4.12(b) isn't allowed because there exists a vertical line L_5 intersecting the reconstructed surface at two points. Suggestions to handle these cases are presented in Section 4.1. The motivation behind Criterion 3 is obvious.

>From these three criteria, we derive explicit tiling and correspondence rules. The theorems stated in this section are directly used in our reconstruction algorithm; the lemmas are only used to prove these theorems. Theorems 1-5 are related to the tiling rules, while Theorems 6 - 7 describe the correspondence rules. Based on Theorems 6 and 7, the correspondence relationship is unique if the reconstructed surface satisfies Criteria 1-3.

The following lemmas and theorems hold for any pair of adjacent slices. Their proofs are given in the Appendix.

Lemma 1 *If a vertical line L not intersecting any contour does intersect the reconstructed surface, then L intersects exactly one solid region.*

Please recall that a contour does not contain its interior. For the example of Fig. 4.12(a), all vertical lines except L_4 don't intersect any contour.

Lemma 2 *Let L be a vertical line not intersecting any contour. Suppose it has M ($M = 0, 1,$ or 2) intersections with the solid regions on the two adjacent slices, and it has N ($N = 0$ or 1) intersections with the reconstructed surface. Then, M and N have the same parity, that is,*

[A] $M = 1 \Longleftrightarrow N = 1$

[B] $M = 0$ *or* $M = 2 \Longleftrightarrow N = 0$

Definition 1 *Cross: Two line segments that intersect, but not at their endpoints.*

Definition 2 *Projection: The projection of an object is denoted by appending a prime sign ($'$). For the example shown in Fig. 4.13(a), C', U', V' and T's are the projections of contour C, vertex U, vertex V and line segment T (which is $\overline{UV}$), respectively. Because T lies between two slices, it has projections, the two T's, on two slices. Unless stated otherwise, any projection is onto an adjacent slice.*

Lemma 3 *If the projection of one contour segment crosses any other contour segment, then these two contour segments cannot be tiled.*

Definition 3 *Augmented contours: New vertices are embedded in contours at those points where the projection of that contour would cross another contour. This breaking of contour segments ensures that any intersection between a contour and a contour projection is either a contour vertex or a contour segment.*

Because of Lemma 3, *augmented contours* are formed to allow the tiling of contour segments whose projections *cross* each other. They are achieved by adding the intersection to these two contour segments as a new vertex. From this point on, all contours are assumed to be *augmented contours*.

Definition 4 $\mathscr{NEC}$ *is the nearest enclosing contour of a point or a contour. Note that* $\mathscr{NEC}(C)$*, where C denotes a point or a contour, cannot intersect C. In Fig. 4.11,* $C1 = \mathscr{NEC}(V')$.

Definition 5 *Positive/negative vertex and overlapping vertex: Let V be a vertex (see Fig. 4.11). If its projection* V' *lies on a contour, then V is an overlapping vertex. An overlapping vertex pair represents the vertex pair* (V_1, V_2) *where* $V_1 = V_2'$*. If V is not an overlapping vertex, then V is said to be a positive or negative vertex if V and* $\mathscr{NEC}(V')$ *have opposite or identical orientations, respectively. For example, the vertex V of Fig. 4.11 is CCW and* $\mathscr{NEC}(V')$ *(contour C1) is CW, so V is a positive vertex.*

Theorem 1 *Any overlapping vertex must tile to its projection.*

(Note: This theorem does not mean an overlapping vertex cannot tile to any vertex other than its projection. A vertex can have more than one slice chord.)

Definition 6 $\mathscr{LS}(q)$ *and* $\mathscr{RS}(q)$ *are the left side and the right side of a vertex q, respectively.*
Suppose $\overline{pq}$ *and* $\overline{qr}$ *of Fig. 4.13(b) are two ordered contour segments. The two half lines* $\vec{qp}$ *and* $\vec{qr}$ *divide the slice into two regions. Then* $\mathscr{LS}(q)$ *is the shadow region which contains the left side of* $\widehat{pqr}$*. Neither* $\mathscr{LS}(q)$ *nor* $\mathscr{RS}(q)$ *contains* $\vec{qp}$ *or* $\vec{qr}$.

Theorem 2 *Let T be a slice chord that is incident with a contour vertex V, and let* $\mathscr{S}$ *be the slice.*

[A] Suppose V is not an overlapping vertex.
if $\mathscr{NEC}(V')$ *is CW* $\Longrightarrow T' \subset (\mathscr{S} - \mathscr{RS}(V))$*, or*
if $\mathscr{NEC}(V')$ *is CCW* $\Longrightarrow T' \subset (\mathscr{S} - \mathscr{LS}(V))$.

[B] If V is an overlapping vertex, its projection V' *is also a vertex because all contours are augmented contours. Then* $T' \subset (\mathscr{S} - ((\mathscr{LS}(V) \cap \mathscr{LS}(V')) \cup (\mathscr{RS}(V) \cap \mathscr{RS}(V'))))$.

Theorem 2 defines the region in which the projection of a slice chord must be located.

Theorem 3 *Suppose* $\triangle V_1V_2V_3$ *is a tiling triangle defined by vertices* V_1*,* V_2*, and* V_3*. Let* V_1 *and* V_2 *be two ordered adjacent contour vertices. Let* $\mathscr{LS}(\overline{V_1V_2})$ *and* $\mathscr{RS}(\overline{V_1V_2})$*, respectively, denote the left side and right side of the line passing through* V_1 *and* V_2.

[A] If either V_1 *or* V_2 *is not an overlapping vertex and its projection has a CW* $\mathscr{NEC}$*, then* $V_3' \notin \mathscr{RS}(\overline{V_1V_2})$.

[B] If either V_1 *or* V_2 *is not an overlapping vertex and its projection has a CCW* $\mathscr{NEC}$*, then* $V_3' \notin \mathscr{LS}(\overline{V_1V_2})$.

(Because all contours are *augmented contours*, it is impossible for one of V_1 or V_2 to satisfy Case 1 while the other satisfies Case 2.)

Theorem 3 is required in addition to Theorem 2 because some invalid slice chords can satisfy Theorem 2.

Definition 7 $\mathscr{I}(C)$ *and* $\mathscr{O}(C)$ *denote the inside and outside regions of a simple polygon C, respectively. Neither* $\mathscr{I}(C)$ *nor* $\mathscr{O}(C)$ *contains C.*

Theorem 4 *Let T be a slice chord, and C be any contour. Then* T' *cannot have intersections with both* $\mathscr{I}(C)$ *and* $\mathscr{O}(C)$.

Theorem 4 doesn't imply that T' cannot have any intersection with contours. Fig. 4.14 shows some valid and invalid slice chord projections.

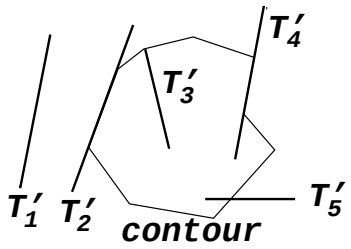


Figure 4.14: Theorem 4: The thick line segments denoted by T's are slice chord projections. T_1', T_2' and T_3' satisfy Theorem 4 while T_4' and T_5' violate Theorem 4.

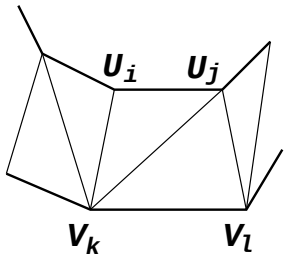


Figure 4.15: If $\overline{U_iV_k}$ and $\overline{U_jV_l}$ are two slice chords, then $i-j=k-l$. Here i, j, k and l are the sequence indices of vertices.

Theorem 5 *Let T_2 be any existing slice chord, and T_1 be the newly proposed slice chord.*

[A] If T_1' crosses T_2', then T_1 cannot be a slice chord.

[B] If T_1' intersects, but does not cross, T_2', and T_1 crosses T_2 (in 3D), then T_1 cannot be a slice chord.

Lemma 4 *Suppose $\overline{U_iU_j}$ and $\overline{V_kV_l}$ are two contour segments of two corresponding contours, and there exist two slice chords $\overline{U_iV_k}$ and $\overline{U_jV_l}$ as shown in Fig. 4.15, then $i-j=k-l=1$ or -1. Here i, j, k and l are the sequence indices of vertices.*

This lemma implies that the tiling sequence cannot be increasing for one contour and be decreasing for its corresponding contour.

Lemma 5 *Let T be a slice chord incident with a vertex V on a contour C. If V is a positive vertex, then $T' \subset (\mathscr{I}(C) \cup C)$. If V is a negative vertex, then $T' \subset (\mathscr{O}(C) \cup C)$.*

Lemma 5 simply restates Theorem 2 in a different way.

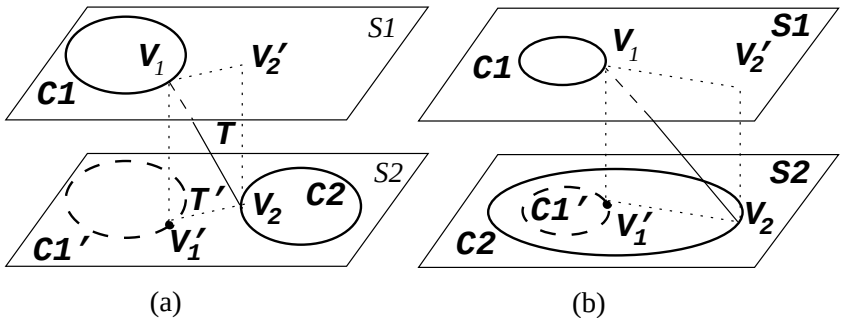


Figure 4.16: (a) $C1'$ and $C2$ are disjoint, (b) $C1'$ is inside $C2$.

Theorem 6 *If a tiling triangle can be placed between two contours $C1$ and $C2$, then one of the following is true (see Fig. 4.16).*

[A] $C1'$ *intersects* $C2$.

[B] If $C1'$ *and* $C2$ *are disjoint, then they have different orientations, each has at least one negative vertex, and their* $\mathcal{NEC}$*s do not* insulate $C1'$ *and* $C2$. *In other words, there exist non-overlapping vertices* $V_1 \in C1$ *and* $V_2 \in C2$ *such that* $\mathcal{NEC}(V_1') = \mathcal{NEC}(C2)$, *and* $\mathcal{NEC}(V_2') = \mathcal{NEC}(C1)$.

[C] If one contour's ($C1$*) projection is inside the other (*$C2$*), then they have the same orientation,* $C1$ *has at least one negative vertex,* $C2$ *has at least one positive vertex, and there is no contour* insulating $C1'$ *and* $C2$. *In other words, there exists a non-overlapping vertex* $V_2 \in C2$ *such that* $\mathcal{NEC}(V_2') = \mathcal{NEC}(C1)$, *and* $C2 = \mathcal{NEC}(C1')$.

Theorem 7 *If any of the three conditions of Theorem 6 holds, then there exists a path on the reconstructed surface linking these two contours. This does not imply that a tiling triangle always exists in between.*

Theorems 6 and 7 state the necessary and sufficient conditions for the correspondence of two contours in adjacent slices.

Implementation

Implementation of the correspondence and tiling rules of the preceding section are discussed in Section 4.1. This implementation guarantees that the reconstructed surfaces satisfies the three criteria stated in Section 4.1. It does not, however, guarantee numerical stability. Two minor modifications, which are discussed in Section 4.1, do make the implementation stable.

General implementation Section 4.1 provides precise rules to determine correspondences between contours on adjacent slices. The tiling rules on the other hand only prohibit bad tilings − they do not suggest good tilings. It was thus necessary to develop a multipass tiling algorithm to achieve reasonably good tiling. It first constructs tilings for any regions not violating any of the tiling rules. Regions that violate these rules correspond to holes, branching regions and dissimilar portions of contours. They are processed by tiling to their medial axes.

In addition to the notation in Section 4.1, we define OTV to be an optimal tiling vertex. V_1=$OTV(V_2)$ if $\overline{V_1V_2}$ is the shortest among all slice chords incident with V_2 that satisfy Theorems 2 and 4. An OTV pair (U,V) implies $OTV(V) = U$ and $OTV(U) = V$.

Our implementation has the following major steps:

Step 1: form closed contours from image slices.

Step 2: create any required *augmented contours* (Definition 3).

Step 3: find correspondences between contours.

Step 4: form the tiling region (Theorem 2) of each vertex.

Step 5: form the OTV table.

Step 6: construct the tiling.

Step 7: collect the boundaries of untiled regions.

Step 8: form triangles to cover untiled regions based on their edge Voronoi diagram (EVD).

As mentioned in Section 4.1, the overall procedure of our algorithm is very similar to the approach of Barequet et al [38]. The effect of our Step 2 is equivalent to enforcing short matches at the intersection of the contour projections in their algorithm. Our Step 5 is similar to the norm distance calculation between the consistently oriented contour portions of their approach. We employ a multi-pass tiling algorithm to collect pieces of tiling triangles while their algorithm uses voting to find a long match between contours. Our tiling algorithm gives much smaller untiled regions than their approach because of the dependence of their approach on the tolerance used in the matching. The use of a small tolerance in their approach results in large unmatched regions, while a large tolerance could result in multiply matched regions. We use Theorem 2 to avoid incorrect tiling while they rely on a tolerance selection. The procedure we use in Step 7, which collects the boundaries of untiled regions, is similar

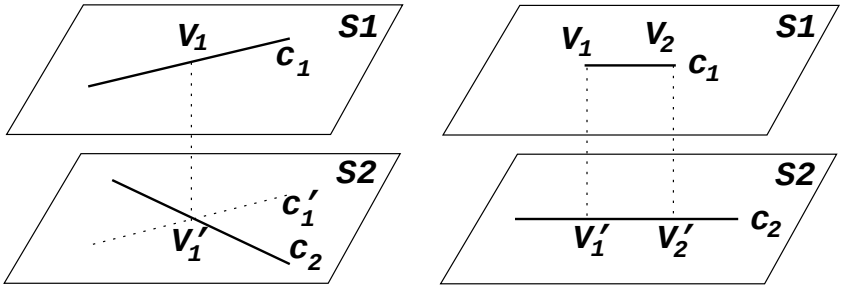


Figure 4.17: Projection of contour segment c_1 onto slice $S2$ containing contour segment c_2: (a) intersection, (b) overlap.

to Barequet's algorithm. The major difference between our approach and theirs is that we have theoretical foundation to justify our approach. Our postprocessing of untiled regions is similar to Geiger's approach [104]. The primary difference is that the EVD added in our postprocessing is a subset of the added external Voronoi skeleton (EVS) in the preprocessing of Geiger's approach.

Our program can have as input either image slices or contour data. In the case of contour data, Step 1 is skipped. Step 2 is required because our theorems are based on *augmented contours*. Step 3 finds the correspondences required by the calculation of the OTV table and tiling. Step 4 produces the data structure for the verification of Theorem 2. Step 5 precalculates the OTVs of every vertex. The OTV table is required for determining the optimality of a proposed slice chord during Step 6, which generates tiling triangles. Holes, branching regions and dissimilar portions of contours cannot be tiled, so they are detected in Step 7 and postprocessed in Step 8. The detailed implementation of each step is described in the following paragraphs.

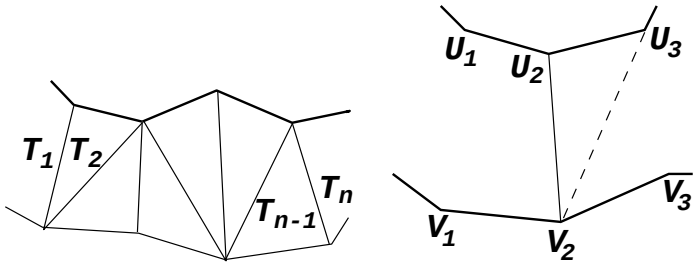
The 2D marching cubes algorithm [157] is used to generate contour segments from an image slice. We assume that the image objects do not intersect the slice boundary. Therefore, all generated contour segments can be linked to form simple polygons. Each contour segment from the 2D marching cubes algorithm is miniscule, so the contours are approximated by fewer contour segments under an error tolerance [92]. Choosing a half-pixel as the approximation tolerance effectively eliminates 75%-90% of contour segments. If the approximation causes intersections between contours of the same slice, the approximation process for the intersecting contours is repeated with a smaller tolerance.

Based on Lemma 3, *augmented contours* are formed. For the example in Fig. 4.17(a), $c_1' \cap c_2 = V_1'$, so V_1' is added to c_2 as an overlapping vertex. Similarly, V_1 is added to c_1. Although Lemma 3 does not require the insertion of any new vertices if the intersection of one contour segment projection with another contour segment is one line segment (e.g. Fig. 4.17(b)), we still insert new vertices (V_1' and V_2') so that the overlapping part is a contour segment in both slices. This allows the tiling algorithm to guarantee that the overlapping part in both slices will be used. An untiled region will thus not *cross* itself.

Theorem 6 judges the correspondence between contours on different slices. One table for each slice stores the contour relationships (disjointedness or enclosure) of that slice. These tables are used to derive the $\mathcal{NEC}$ (Definition 4) of a vertex V or a contour C. Another table stores the contour relationships (intersection, disjointedness, or enclosure) between adjacent slices. The $\mathcal{NEC}$ of every vertex projection is searched on the projection slice. Once the $\mathcal{NEC}$ of a vertex projection is available, whether the vertex is *positive* or *negative* can be determined by Definition 5. Finally, Theorem 6 determines the correspondence between contours of adjacent slices. This takes $O(n^2)$ time because checking a point inside a contour is an $O(n)$ time algorithm and there are $O(n)$ vertices.

The tiling region of each vertex is defined in Theorem 2. Two line equations and an $AND-OR$ flag define the tiling region described by Theorem 2.1. In the example of Fig. 4.13(b), the tiling region of the vertex q is on the left side of line pq or line qr. If V is an overlapping vertex, then (V,V') is an OTV pair. So V' can be indexed from the OTV table formed in Step 5, and no extra data structure is needed to verify Theorem 2.2.

The OTV table stores the OTVs of all vertices. The optimality is in terms of the shortest slice chord, as used by Christiansen et al. [66]. The OTV on the adjacent slice is searched for each vertex V. The current implementation sorts the candidate vertices based on their distances to V. Then the candidate vertices, starting from the closest vertex, are checked until one satisfies Theorems 2 and 4. Usually, the closest candidate is qualified. So the average time complexity is $O(n^2 \log n)$. But in the worst case of no qualified vertex, all vertices on the corresponding contour are tested. This implementation therefore has an $O(n^3)$ worst case time complexity. (There are $O(n)$ vertices. Checking Theorems 2 and 4 requires $O(1)$ and $O(n)$ time, respectively, and there are $O(n)$ checks for each V in the worst case.) This step is usually the most time consuming part of the whole process.



(a) (b)

Figure 4.18: (a) Only the boundary slice chords T_1 and T_n are needed to verify Theorem 5, (b) The optimality of forming $\triangle U_2V_2U_3$ can be classified into six cases.

Our method can be sped up by presorting all vertices into an RPO tree [69] so the closest-point query can be done in $O(\log n)$ time. Furthermore, the $O(\log n)$ algorithm described by Preparata [186] can be used to check Theorem 4. The time complexity of this approach is $O(n \log n)$ average case and $O(n^2 \log n)$ worst case.

During the tiling, the *legality* of each slice chord is checked frequently for compliance with Theorems 2 - 5). Verifying Theorems 2 and 3 for a proposed slice chord takes $O(1)$ time. Theorem 4 takes $O(n)$ or $O(\log n)$ time depending on the implementation. Verifying Theorem 5 takes $O(n)$ time because the number of existing slice chords is always increasing and they cannot be preprocessed. Fortunately, it is not necessary to check for intersections of proposed slice chords with all existing slice chords. Only the boundary slice chords of tiled regions need to be verified. For the example in Fig. 4.18(a), only the two boundary slice chords T_1 and T_n need to be checked. The existing slice chords T_2 to $T_(n-1)$ are bounded by T_1, T_n and contours, so it is impossible for the projection of the proposed slice chord to cross any of T_2 to T_{n-1} without crossing T_1, T_n, or any contour.

There are four passes in tiling. The tiling sequence is based on the optimality of the tiling pair, with optimality defined in terms of the shortest slice chord. Suppose $\overline{U_2U_3}$ and $\overline{V_2V_3}$ are two contour segments of two corresponding contours, as shown in Fig. 4.18(b). We are going to form a tiling triangle $\triangle U_2V_2V_3$ or $\triangle U_2V_2U_3$. Suppose slice chords $\overline{U_2V_2}$ and $\overline{U_3V_2}$ are *legal*, and $\overline{U_2U_3}$ has not been used. Considering only one triangle $\triangle U_2V_2U_3$, we can classify it into six cases in decreasing order of degree of optimality.

Case 1: (U_2, V_2) is an OTV pair, and so is (U_3, V_3).

Case 2: (U_2, V_2) is an overlapping vertex pair.

Case 3: (U_2, V_2) is an OTV pair, and $V_2 = OTV(U_3)$.

Case 4: (U_2, V_2) is an OTV pair, $V_2 \neq OTV(U_3)$, and V_2 and $OTV(U_3)$ are on the same contour.

Case 5: (U_2, V_2) isn't an OTV pair, and V_2, $OTV(U_2)$ and $OTV(U_3)$ are on the same contour.

Case 6: All other cases.

Cases 1-3 are all considered to be optimal. Cases 4, 5 and 6 are not optimal and their degree of optimality decreases with increasing case index.

There are four passes in tiling: The first pass handles cases 1-3, the second pass handles case 4, and so on.

In the first pass, the OTV table is scanned to look up an optimal tiling pair. Lemma 4 states that there are only two spanning directions for tiling. If case 1 is encountered, then the quadrilateral $U_2V_2V_3U_3$ is divided into two triangles using the shorter slice chord. In cases 2 or 3, only one triangle ($\triangle U_2V_2U_3$ or $\triangle U_2V_2V_3$) is chosen in one span direction. The selection is based on the shorter of $\overline{U_2V_3}$ and $\overline{V_2U_3}$ if both triangles satisfy cases 2 or 3. Tiling triangles are tested and may be formed on both sides of $\overline{U_2V_2}$. The boundary slice chords and their directions (spanning right or left) are put into a boundary slice chord array. If the to-be-stored boundary slice chord is already in the array, it is already shared by two tiling triangles, and thus it is deleted from the boundary slice chord array.

Cases 4-6 are handled in the three subsequent passes. One starting tiling pair with its direction is popped out from the boundary slice chord array, and the tiling spans one direction until no satisfying case is available. The starting tiling pairs of the

second pass also come from the OTV table. As in the first pass, the boundary slice chords are stored in an alternative boundary slice chord array. The tiling takes $O(n^2)$ time because checking Theorem 5 takes $O(n)$ time, and there are $O(n)$ proposed slice chords. Compared to making only one pass to do all tiling, our multipass approach does not significantly increase the number of proposed tiling pairs, which is proportional to the processing time.

Fig. 4.19 shows a rather complicated many-to-many branching tiling. Fig. 4.19 (a) shows the top view of contours of two slices. Two bottom contours (C_2^1 and C_2^2) branch into five top contours (C_1^1 - C_1^5). In addition to this complicated branching, a very dissimilar contour C_1^4 and a hole C_1^2 are involved. The tiling result after the first pass is shown in (b); only good tiling triangles are formed. As each pass proceeds, the algorithm loosens the optimality requirements in order to form more tiling triangles. The last pass uses any *legal* triangle. The result after the last pass is shown in (c).

After all four passes, there may be untiled regions in dissimilar contours, holes or branching regions. The processing of untiled regions is illustrated in Fig. 4.19 (c)-(f). The untiled regions can be traced from the unused contour segments (the dotted lines in (c)) and the boundary slice chord array. The boundary slice chord is assigned a direction from the bottom slice to the top slice if it spans left, or from top to bottom if it spans right. The directions of the top unused contour segments are reversed. These directions are required to trace a closed untiled polygon when more than two untiled contour segments or slice chords share the same vertex. Fig. 4.19 (d) shows the top view of the traced untiled polygons.

If the projection of an untiled region is convex, it is triangulated with its center of gravity. Otherwise, we triangulate using its medial axis. We use Lee's algorithm [142] to find the EVD (edge Voronoi diagram or medial axis). The EVD is approximated by a smaller number of line segments so fewer triangles are required to cover the untiled region. The Z values of the EVD are set to the middle of two slices so the reconstructed model corresponds to our expectations of the physical object. The result is shown in Fig. 4.19(e). The perspective view of the triangulated untiled regions is shown in Fig. 4.19(f) in which the contours are plotted as dotted line segments. Fig. 4.19(g) shows all triangles from tiling and EVD. Two different perspective views of the shaded result are presented in (h) and (i).

In the case of one untiled region's projection enclosing the projection of another untiled region, one can use the EVD algorithm of Srinivasan et. al. [220] to calculate the medial axis of nested polygons. Because our algorithm tries to do as much tiling as possible during the last tiling pass, the number of untiled regions left is near minimum. The case of nested untiled regions rarely happens; in fact, we have never encountered it in real image data.

Numerically stable implementation Two minor modifications of the algorithm sketched in Section 4.1 help avoid numerical instability. The first modification abolishes some augmented contours, and the second develops an algorithm to find the rough medial axis.

The *augmented contours* are required because of Lemma 3. Augmenting a contour may add vertices which are very close to existing vertices. This results in numerical instability in programming because it is difficult to consistently treat two very close vertices as one vertex or two separated vertices. Besides, this results in very sharp triangles. These problems can be solved by not requiring Criterion 2 at *crossed* contour segments. This minor modification affects the reconstructed surface only at *crossed* contour segments; the overall shape isn't changed.

Lemma 3 shows that two *crossed* contour segments cannot be tiled because of Criterion 2, so *augmented contours* are formed so there are no *crossed* contour segments. If we waive the Criterion 2 requirement only at crossed contour segments and directly tile *crossed* contour segments, there is no need to form *augmented contours*. The non-planar quadrilateral formed by two crossed contour segments c_1 and c_2 in Fig. 4.17(a) is divided into two triangles. In the case of overlapping projection contour segments (e.g. Fig. 4.17(b)), the quadrilateral is planar and is triangulated. This is done before the first tiling pass, and then the same four-pass tiling algorithm applies. If a contour segment has n intersections (counting multiplicities) with the projections of other contours of the adjacent slice, then $n-1$ new vertices are inserted to break this contour segment into n contour segments. A new vertex is placed at the middle of every two adjacent intersections. This step is necessary to ensure that one contour segment can *cross* at most one contour segment projection.

Because the added vertices of this implementation are at the middle of two intersections of one contour segment, they are rarely close to any existing vertex. This also adds many fewer new vertices compared to the general implementation because most intersected contour segments have only one intersection and no vertex is added. The lack of Criterion 2 in the numerically stable implementation is limited to crossed segments, and it does not affect the formation of other regions. It has few effects on the overall shape, and the problems of very close vertices are solved. Fig. 4.20 shows the results of the general implementation and the numerically stable implementation.

It is difficult to implement a numerically stable EVD algorithm for different situations. So we developed a program to find the rough medial axis. A polygon is decomposed into two polygons by adding a cutting edge (the dashed line in Fig. 4.21). The

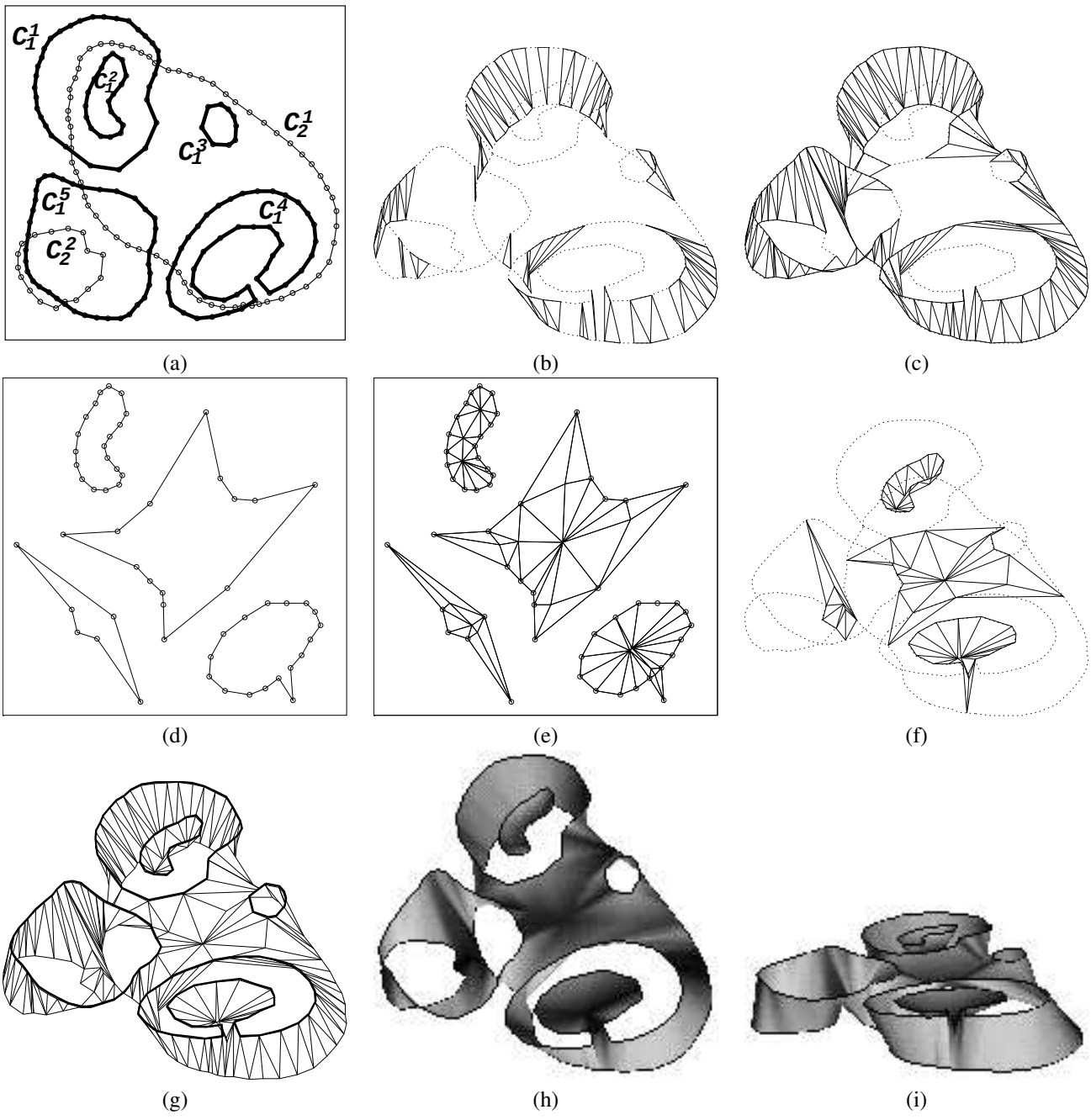
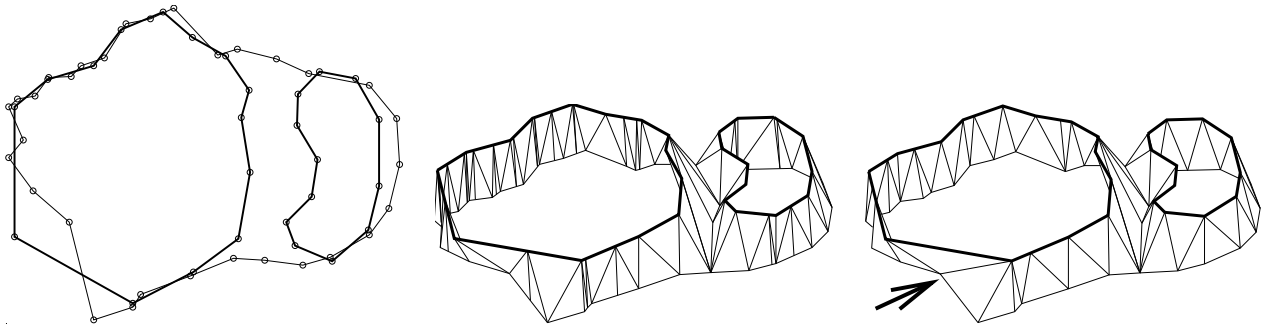


Figure 4.19: (a) Two slices of contours. Thicker contours are from the top slice. The small circles denote vertices. (b) result of the first tiling pass. Only good tiling triangles are formed. (c) the result of all tiling passes. (d) top view of untiled regions. (e) top view of untiled region triangulation by edge Voronoi diagram. (f) the perspective view of (e) with hidden lines removed. (g) the final result. (h) & (i) two different shaded views of the tiled surface shown in (g).



(a) (b) (c)

Figure 4.20: Results of the general implementation and the numerically stable implementation: (a) Top view of contours. Top contours are drawn in thick line segments. (b) The general implementation: it results in a very sharp triangle if the added vertex is very close to an existing vertex. (c) The numerically stable implementation: it does not generate very sharp triangles, but surfaces may be distorted (pointed by the arrow) because of not satisfying Criterion 2.

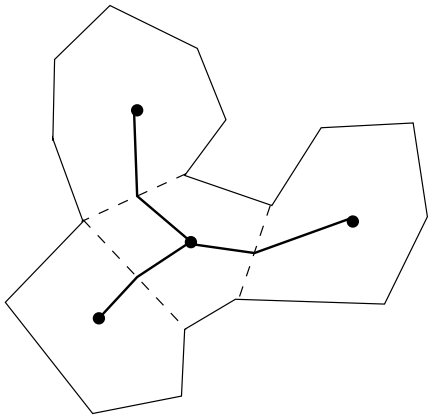


Figure 4.21: Finding the rough medial axis by repeatedly decomposing a polygon into two polygons.

decomposition is repeated until all generated polygons are convex. The rough medial axis (shown as the darker line segments in Fig. 4.21) is formed by linking the centers of the convex polygons and the middle points of the cutting edges.

Results

The algorithm has been implemented in C, and runs on Sun Sparc and Silicon Graphics Indigo2 workstations. It has also been ported to an Intel Paragon. Both the synthetic and real cases used to test these implementations are presented in this section.

Fig. 4.19 illustrates the capabilities of our algorithm when many-to-many branching, dissimilar contours and holes are present. As can be seen from Fig. 4.19, C_1^2 is reconstructed as a shallow hole since it has no corresponding contour on the bottom slice. The dissimilar portion of C_1^4 does not tile to the bottom contour C_2^1. It tiles to its medial axis and forms a shallow hole with a link to C_2^1. This is a highly likely topology. The handling of branching is shown in Fig. 4.19 (f) and in Fig. 4.22. The added curve could be a point to model the saddle point, as in Fig. 4.22 (b), if the branching region is not complicated, or it could be line segments to model canyons between contours, as shown in Fig. 4.22 (d).

Fig. 4.23 shows a dissimilar contour case which causes many tiling algorithms to fail. Fig. 4.23(a) illustrates the top view. The wire frame result of the minimum surface area optimizing algorithm is shown in (b). The result is redisplayed in (c) with hidden lines removed. The arrow points to the abnormality. Fig. 4.23 (d) is the result of the shortest slice chord heuristic algorithm. It is much worse than (c). Fig. 4.23 (e) shows at least one polyhedron solution. The result of our algorithm is shown in (f). Heuristic methods fail badly because 1) they do not check the limited region associated with a slice chord (Theorem 2), and 2) they cannot change any previous improper slice chord selection. The first drawback is also associated with the optimal

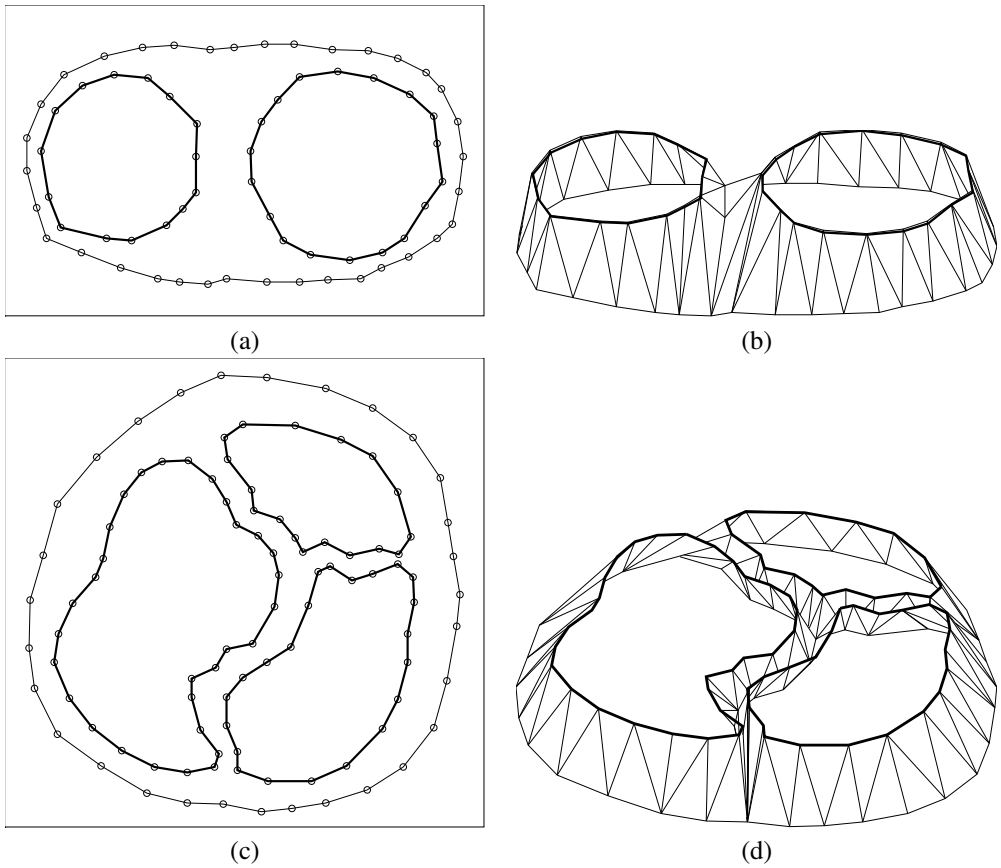


Figure 4.22: Tiling in the presence of branching: (a) & (c) top view of two different cases of branching contours. The thicker line segments are top contours and small circles represent vertices. (b) & (d) our results with hidden lines removed.

algorithm, but the optimal algorithm can recover from this drawback because it selects the optimal one from a large set of possible solutions. Our algorithm guarantees that the results are polyhedra because of Criterion 1. It forms a natural shape in the dissimilar region because of Criterion 2.

For a test with real data, we used the tiling algorithm to reconstruct different parts of the human body in order to build a database for a human body walkthrough. The algorithm worked very well in dense data, as will be seen with results from three sets of medical data.

The rendering tool (see Acknowledgment) is based on the hardware platform independent graphic library *XS* developed at the *Shastra* laboratory at the Computer Sciences Department, Purdue University. Fig. 4.24 shows the Gouraud shading and the wire frame of the reconstructed surface of a brain hemisphere. It was generated from a set of contour data that was manually traced from 52 MRI image slices. Fig. 4.25(a) shows a reconstructed skull. The noise around the teeth was present in the original image slices. The surface was automatically generated from 112 256*256 CT slices. The image slices were first enhanced by processing with a 3*3 median filter before automatic contour segmentation.

Figs. 4.25(b)-(d) show two adjacent slices around the nasal area and the tiling between them. As can be seen from (b) and (c), there are numerous holes, dissimilar areas, and branching between two adjacent slices. Fig. 4.26 shows three different views of the skeleton reconstruction of the human cadaver Freddy. The tiling of some cross sections is shown in Fig. 4.27. The image volume consisted of 920 256*256 slices. The resolution was 2.16mm in all dimensions. Each image slice was first enhanced by processing with a Gaussian weighted low pass filter before automatic contour segmentation. The global thresholding scheme employed in the 2D marching cubes approach did not work well in regions which were barely visible, such as at the shoulder bones. However, our tiling algorithm takes whatever contours are generated and produces surfaces based on those contours. Therefore, these three examples show the capability of our algorithm to handle complicated topologies.

The error tolerance during the approximation of the skull and Freddy contours was 0.5 pixel. The marching cubes approach

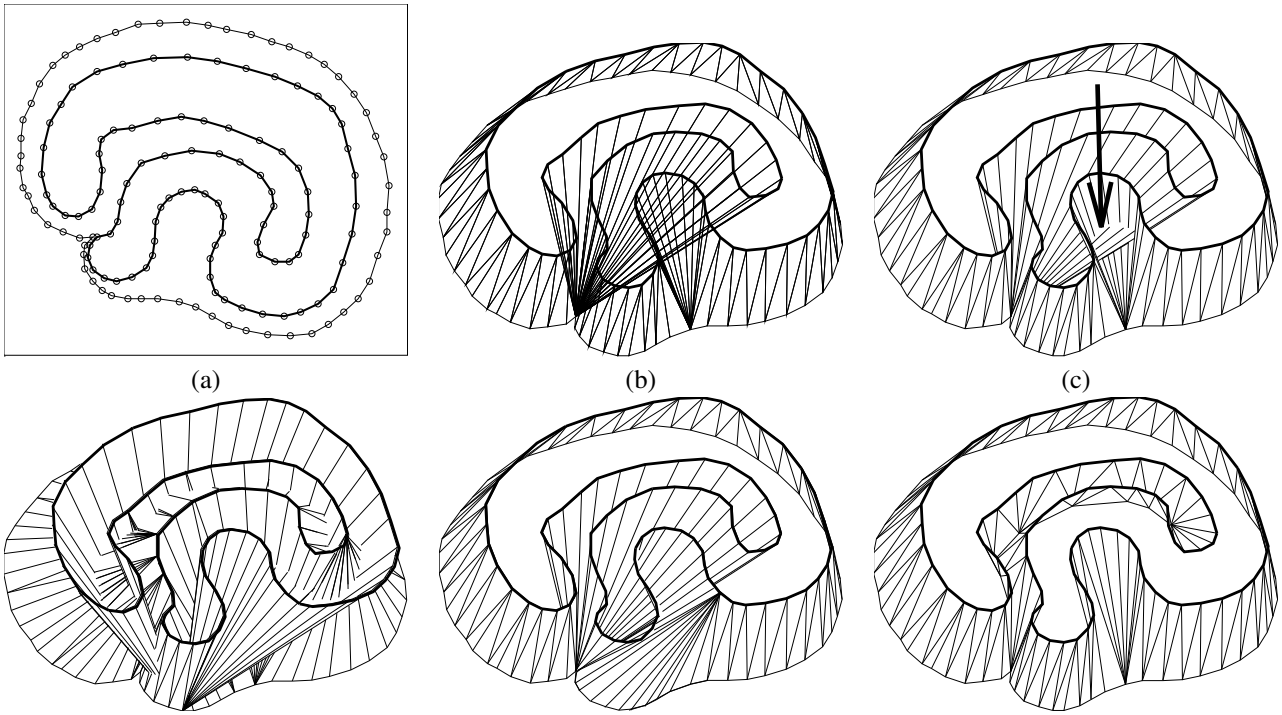


Figure 4.23: Tiling of two dissimilar contours: (a) top view of two contours. The thicker contour is the top contour and small circles represent vertices. (b) result of minimum surface area tiling, (c) same as (b) with hidden lines removed. The arrow points to the abnormality where triangles intersect non-adjacent triangles. (d) result of shortest slice chord heuristic algorithm. (e) existence of a non-selfintersecting surface. (f) our result.

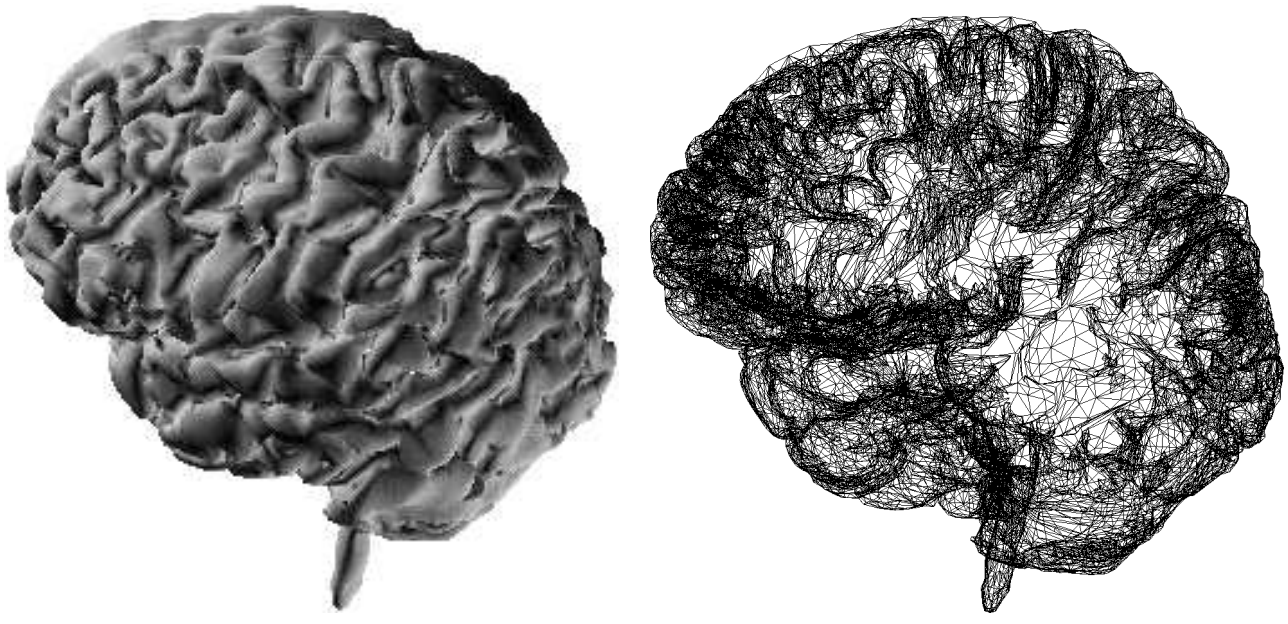
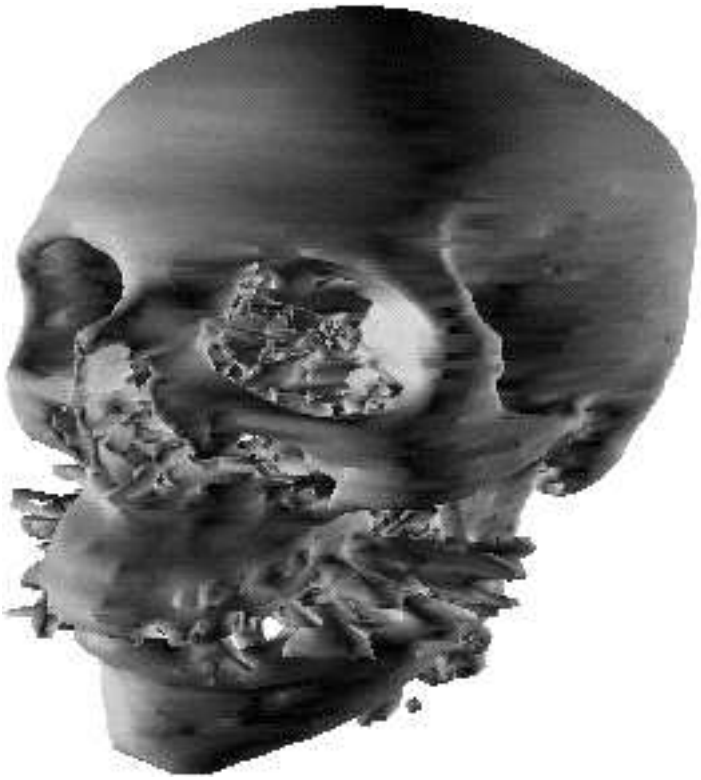
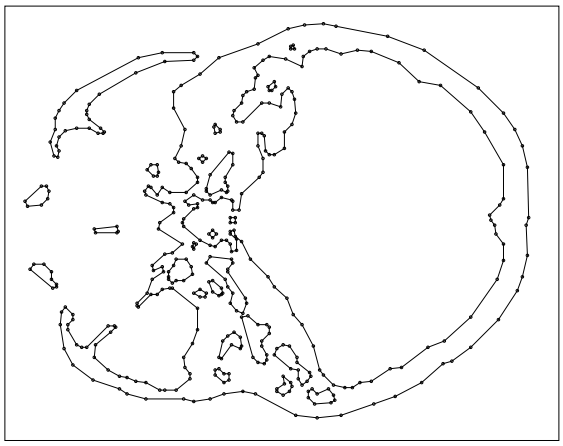


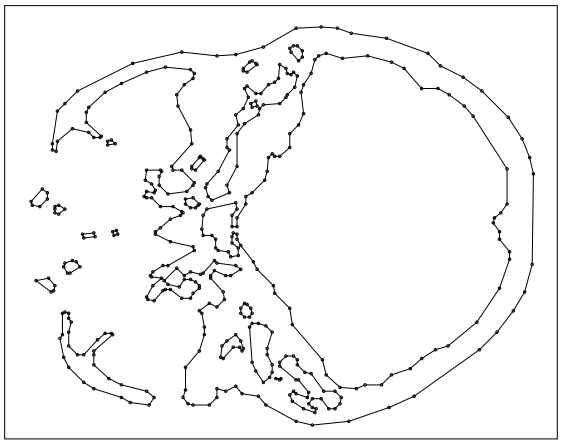
Figure 4.24: Visualization of a reconstructed brain hemisphere (a) Gouraud shading, (b) wire frame



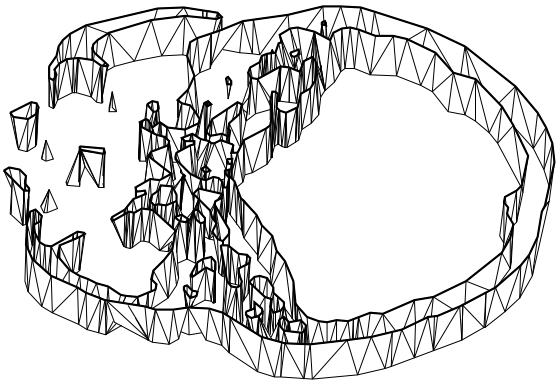
(a)



(b)



(c)



(d)

Figure 4.25: Visualization of a reconstructed skull: (a) Gouraud shading, (b) & (c) two sample slices, (d) the tiling of (b) & (c).

	segment. method	# of slices	# of △'s	CPU time (sec)	Step 3 time	Step 5 time	Step 6 time	marching cubes △'s
brain	manually	52	37,992	90	27%	60%	10%	N/A
skull	auto.	112	54,071	104	31%	51%	15%	554,500
Freddy	auto.	920	285,349	248	43%	32%	20%	1,416,584

Table 4.1: Results

generated 554,500 triangles from the skull data and 1.4 million triangles from the Freddy data, respectively. Our approach generated significantly fewer (54,071 and 285,349, respectively) triangles than the marching cubes approach. Table 1 summarizes our results. The CPU time in the table is based on SGI Indigo2 IMPACT workstations and does not include the image segmentation time.

Contributions and limitations

The theoretical approach and the algorithms developed in this paper differ significantly from those currently available. It is thus important to clearly describe the benefits they provide to both the theory and practice of surface reconstruction. It is equally important to describe the limitations of these results, both to ensure that the algorithms are used appropriately, and to clarify issues that should motivate future work.

Contributions The primary contribution of this paper is the theoretical foundation supporting our surface reconstruction algorithm. It consists of the three constraints imposed on the reconstructed surface and the derivation of precise correspondence and tiling rules that satisfy them. This constraint-based approach produces a reconstruction algorithm which can generate an expected physical surface from any topology that is encountered. This is not the case, for example, with algorithms based on heuristic methods for tiling and reconstruction. Such algorithms often fail on topologies that were not directly anticipated (see Section 4.1). The importance of these constraints is also made clear by the abnormal surface, shown in Fig. 4.23 (c), generated by a minimum-surface-area algorithm.

The second major contribution is the development of a robust algorithm whose reconstructed surfaces correspond well with the surfaces of the actual object. The aspects of our approach which make this possible are: (a) the data of our reconstructed model cannot flip its sign more than once along a perpendicular line between adjacent slices; (b) The re-sampling of our reconstructed surfaces is guaranteed to produce the original contours; (c) our method of handling branching method works well on very complex branchings; and (d) a region which cannot be tiled using our explicit rules is tiled with its medial axis.

The third major contribution of our research is our multipass tiling algorithm. Traditional tiling algorithms have problems in branching regions because it is difficult to know exactly when to switch from tiling rules based on the assumption of a one-to-one correspondence of contours to rules based on the assumption that a region is branched. Many papers try to avoid this problem by identifying branching regions and then inserting composite contours or intermediate contours in order to reduce the tiling of the branched topology to that of tiling uniquely paired contours. As discussed in Section 4.1, such approaches have drawbacks. We have avoided them by designing a tiling algorithm that makes several passes. Successive passes tile according to progressively weaker optimality rules − but in all cases satisfying the surface reconstruction constraints. At the end of these passes, branching regions, holes and dissimilar areas of contours are left untiled. We then tile these well-defined remaining regions. As discussed in Section 4.1 this new tiling algorithm produces very reasonable tilings for very complicated branching structures. The most probable reason for the success of this new approach is that it tiles "difficult" regions only after it has tiled as much of the rest of the surface as possible. It thus has as much surface information as possible when it has finally reached the point when it must tile the "difficult" regions.

Limitations As discussed above, one strength of our algorithm is that we know exactly what type of surfaces it will construct because of the constraints imposed on them.

One of these constraints, though, is actually a requirement on the sampling rate used when imaging an object. Specifically, Criterion 2 − which states that the reconstructed surface between adjacent slices can have at most one intersection with any line perpendicular to the slices − requires that the inter-slice spacing of the data be fine enough that certain topologies are unlikely. Examples of topologies which violate Criterion 2 because of undersampling in a given direction are provided in Fig. 4.28. In

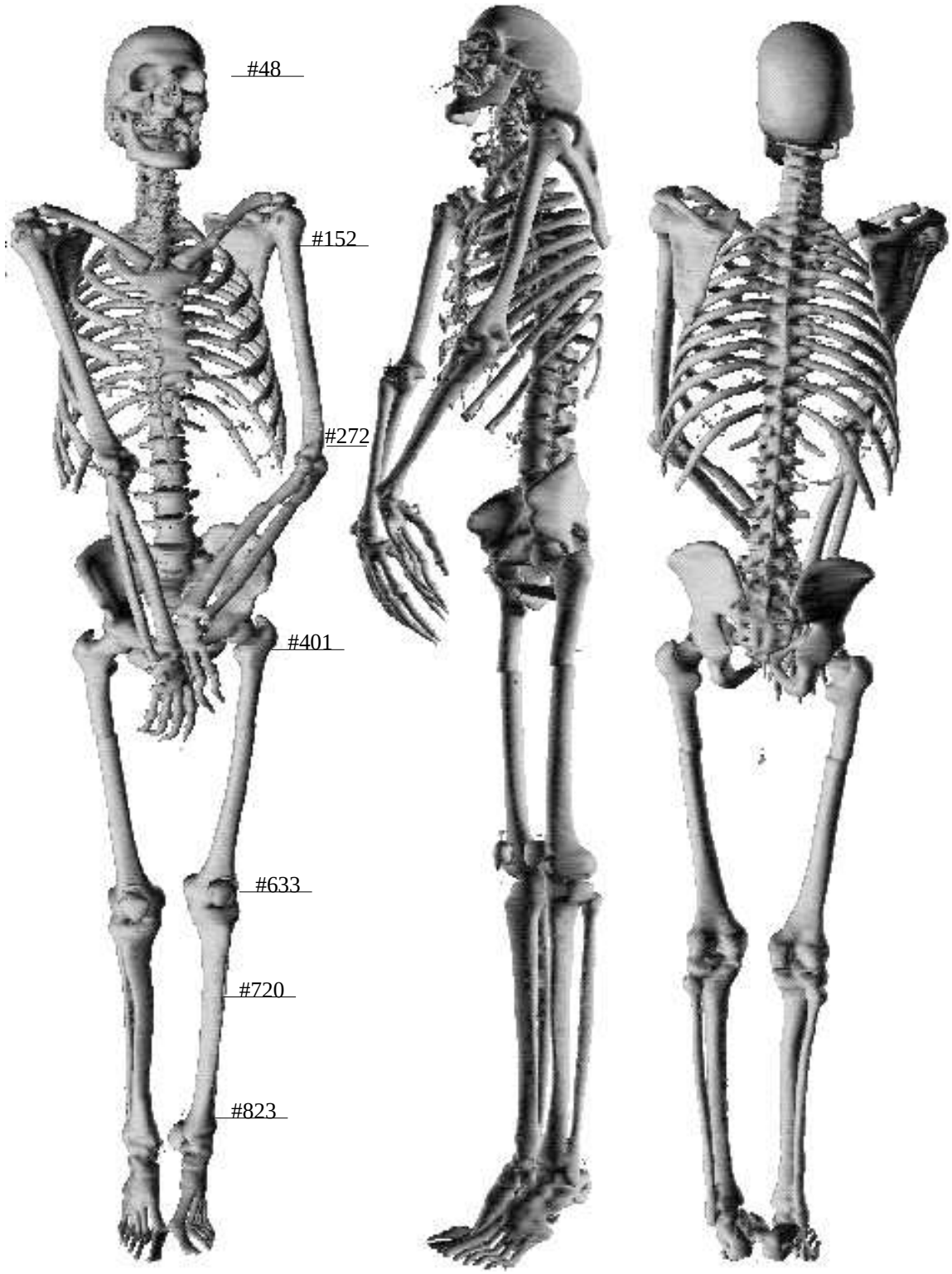


Figure 4.26: Tiling based visualization of an entire human skeleton (Freddy). The tilings of the numbered portions are shown in the next figure.

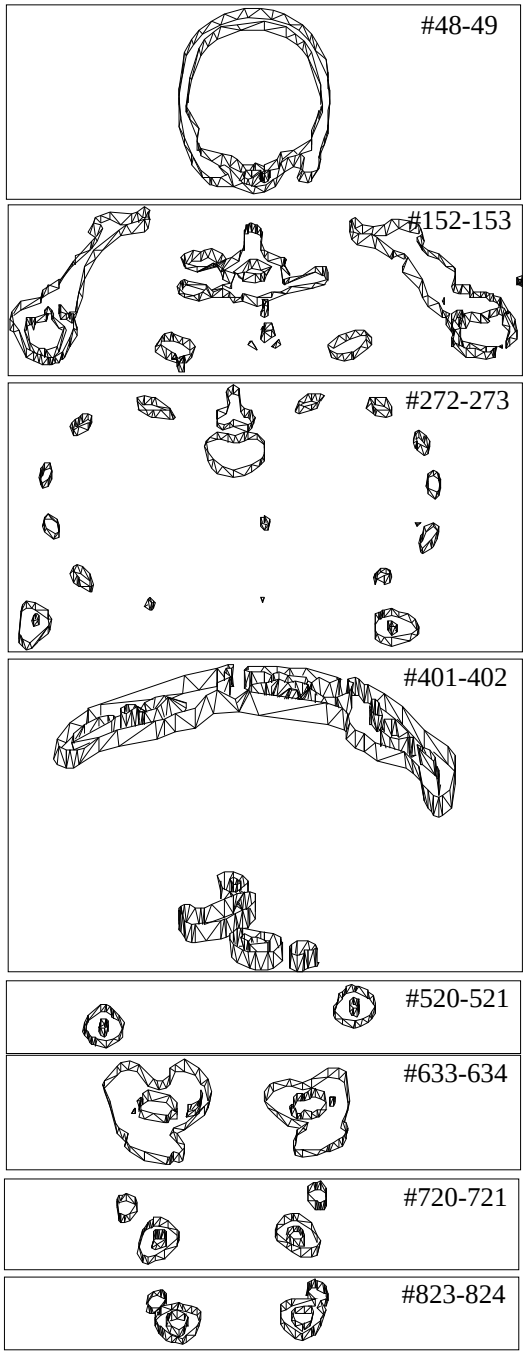


Figure 4.27: Multiple contours with tiling for different horizontal cross-sections through the human skeleton model.

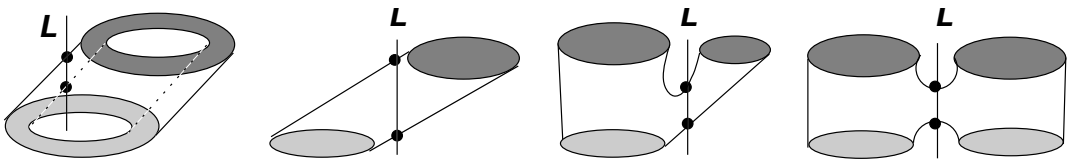


Figure 4.28: Some examples of topologies that can not be processed because they violate Criterion 2 by having two points of intersection with a line perpendicular to the slices.

each case, the true surface of the object can not be reconstructed by our algorithm because the surface it produces must satisfy Criterion 2. Our algorithm will still finish, but the surfaces produced in these cases will be incorrect.

In cases (a) through (c) in 4.28, a preprocessing stage would enable our algorithm to reconstruct the actual surface. For example, the surface in Fig. 4.28(a) can be generated if we apply our algorithm only to those contours with the same number of enclosing contours (see Definition 4). The surfaces in Figs. 4.28(b) and (c) could be correctly generated by translating and scaling the contours until Theorem 6 generates the actual correspondence, and then translating and scaling back the tiling results. O'Rourke [173] points out a potential flaw in scaling and suggests using a uniform scaling (with the same amount in both x and y). In this latter case, though, a significant amount of side information must be present in order to determine that the actual object must be as shown in the figure.

No amount of preprocessing would enable our algorithm to correctly reconstruct the surface of Fig. 4.28(d). In such badly undersampled data, the aliasing is severe enough that it is completely unclear what to do. The *best* way to handle this case would be to acquire additional data in order to decrease the inter-slice distance.

Conclusion

This paper presented a robust algorithm for reconstructing surfaces from a set of planar contours or image slices. The theoretical derivation of the correspondence and tiling rules allowed our algorithm, given any input data, to generate a unique topology satisfying the desired surface criteria. This new, unified approach led to reconstructed surfaces which correspond well with the surface of the physical objects that were imaged.

Our algorithm is not guaranteed to produce an optimal solution, but it does achieve the following improvements over previous surface-based techniques:

- It avoids such major drawbacks of general heuristic tiling procedures as the generation of twisted surfaces because of lack of global information;
- Unlike other tiling approaches which may generate self-intersecting surfaces, it guarantees that the tiling result is a polyhedron surface;
- It produces appropriate tilings in branching regions;
- It generates significantly fewer triangles than the marching cubes approach.

These improvements were all demonstrated with both real and synthetic medical data.

The strength of our approach is its ability to handle dissimilar contours and complicated branching. Its primary drawback is one faced by any surface-based algorithm – when the inter-slice distance is too large, it becomes very difficult to solve the correspondence problem without additional algorithms or human intervention. In our case, this problem manifests itself as a violation of Criterion 2. The probability that these violations occur increases as the inter-slice distance increases.

linear and non-linear (PDE) approach

4.2 Filtering

The following algorithm was taken from [25] (Bajaj2001).

Introduction

The solution for triangular surface mesh denoising (fairing) is achieved by solving a partial differential equation (PDE), which is a generalization of the heat equation customized to surfaces. The heat equation has been successfully used in the image processing for about two decades. The literature on this PDE based approach to image processing is large (see [192, 183, 187, 237]). It is well known that the solution of heat equation $\partial_t \rho - \Delta\rho = 0$, based on the Laplacian Δ, at time T for a given initial image ρ_0 is the same as taking a convolution of the Gauss filter $G_\sigma(x) = \frac{1}{2\pi\sigma}\exp\left(-\frac{|x|^2}{2\sigma^2}\right)$ with standard deviation $\sigma = \sqrt{2T}$ and image ρ_0. Taking the convolution of $G_\sigma(x)$ and image ρ_0 is performing a weighted averaging process to ρ_0. When the standard deviation σ become larger, the averaging is taken over a larger area. This explains the filtering effect of the heat equation to noisy images. The generalization of the heat equation for a surface formulation has recently been proposed in [67, 76] and shown to be very effective even for higher-order methods [26]. The counterpart of the Laplacian Δ is the Laplace-Beltrami operator Δ_M(see [83]) for a surface M. However, unlike the 2D images, where the grids are often structured, the discretized triangular surfaces are often un-structured. Certain regions of the surface meshes are often very dense, with a wide spectrum of noise distribution. Applying a single Gauss-like filter to such surface meshes would have the following side-effects:

(1). The lower frequency noise is not filtered (under-fairing) if the evolution period of time is suitable for removing high frequency noise,

(2). Detailed features are removed unfortunately, as higher frequency noise (over-fairing) if the evolution period of time is suitable for removing low frequency noisy components.

The bottom row of Fig 1 illustrates this under-fairing and over-fairing effects. For the input mesh on the top-left, three fairing results are presented at three time scales. The first figure exhibits the under-fairing for the head. The last figure exhibits the over-fairing for the ears, eyes, lips and nose. Hence, a phenomena that often appears for the triangular surface mesh denoising is that whenever the desirable smoothing results are achieved for larger features, the smaller features are lost. Prior work has attempted to solve the over-fairing problem by using an anisotropic diffusion tensor in the diffusion equation [26, 67]. However, this is far from satisfactory. The aim of this chapter is to overcome the under-fairing and over-fairing dilemma in solving the diffusion equation.

There are several situations where the produced triangular surface meshes have varying triangle density. One typical case is geometric modeling, where the detailed structures are captured by several small triangles while the simpler shapes are represented by fewer large ones. We may call such triangular meshes as feature-adaptive. Another case is the results of physical simulation, in which the researcher is interested in certain regions of the mesh. In these regions, accurate solutions are desired, and quite often, finer meshes are used. For example, in the acoustic pressure simulation [231], the interesting region is the ear canal for human hearing. Hence, more accurate and finer meshes are used there. We call such meshes as error-adaptive. One additional case arises from the multiresolution representation of surfaces, for example using wavelet transforms or direct mesh simplifications. Each resolution of the representation is a surface mesh that approximates the highest resolution surface. The approximation error is usually adaptive and can vary over the entire surface. For instance, the mesh simplification scheme in [24], which is driven by the surface normal variation, results in meshes that are both feature-adaptive and error-adaptive.

Previous Work For PDE based surface fairing or smoothing, there are several methods that have been proposed (see [26, 67, 76, 75]) recently. Desbrun et al in [76, 75] also use Laplacian, which is discretized as the umbrella operator in the spatial direction. In the time direction discretization, they propose to use the semi-implicit Euler method to obtain a stable numerical scheme. Clarenz et al in [67] generalize the Laplacian to the Laplace-Beltrami operator Δ_M, and use linear finite elements to discretize the equation. In [26], the problem is reformulated for 2-dimensional Riemannian manifold embedded in R^k aiming at smooth geometric surfaces and functions on surfaces simultaneously. The C^1 higher-order finite element space used is defined by the Loop's subdivision (box spline). One of the shortcomings of all these proposed methods that we address here is their non-adaptivity. All of them use uniform timesteps. Hence they quite often suffer from under-fairing or over-fairing problems.

Our Approach For a feature-adaptive or error-adaptive mesh, the ideal evolution strategy would be to correlate the evolution speed relative to the mesh density. In short, we desire the lower frequency errors use a faster evolution rate and the higher frequency errors succumbs to a slower evolution rate. To achieve this goal, we present a discretization in the time direction, which is mesh adaptive. We use a timestep $T(x)$, which depends on the position x of the surface. The part of the surface, that is coarse, uses larger $T(x)$. The idea is simple and it is easy to incorporate it into any uniform timestep diffusion implementation. The improvements achieved over the evolution results with uniform timestep are significant. The top row of Fig 1 shows this adaptive time evolution improvement over the uniform timestep evolution results, shown in the bottom row. The middle and

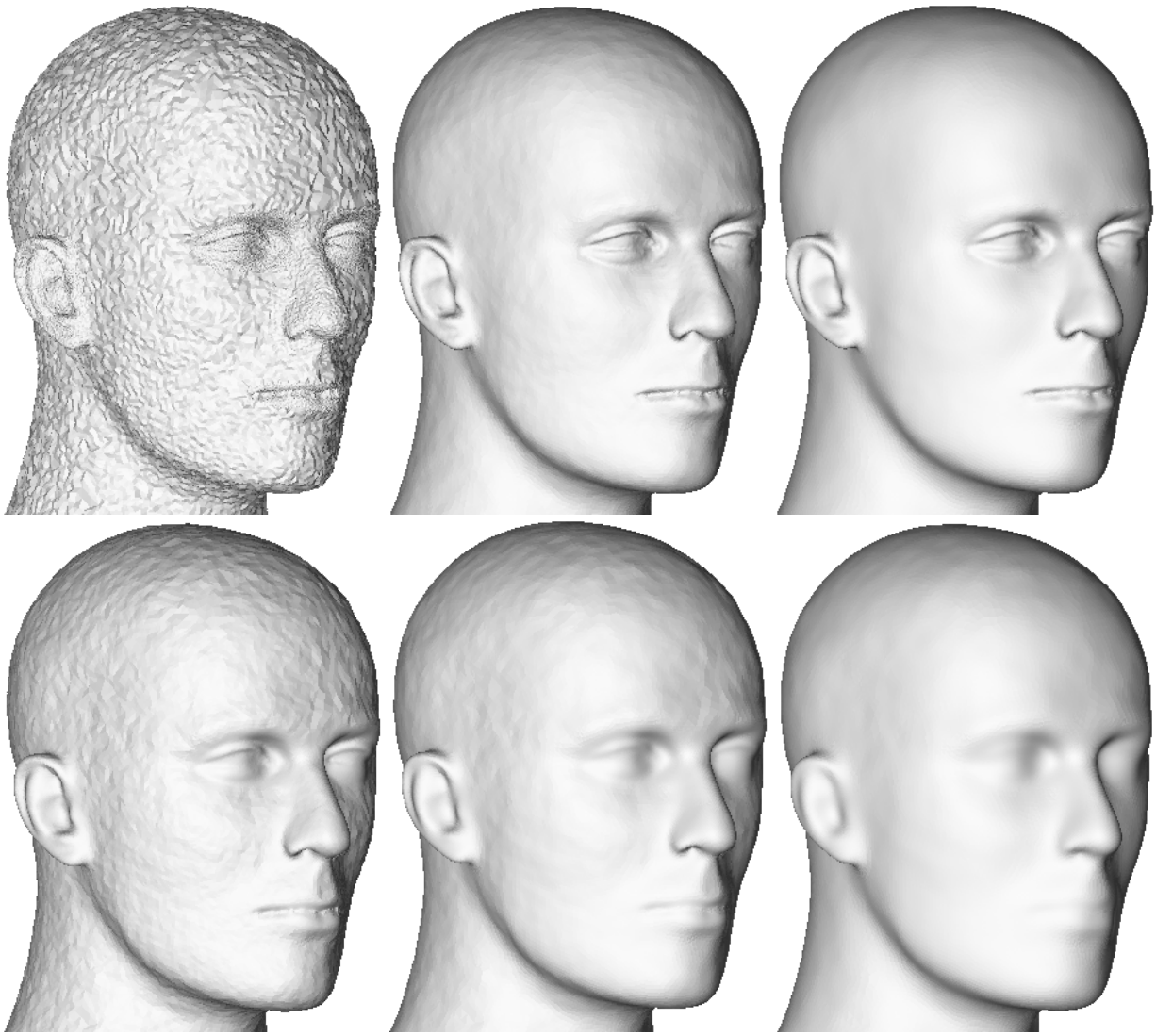


Figure 4.29: The left figure in the top row shows the initial geometry mesh. The top-middle and top-right figures illustrate the results of the adaptive timestep smoothing after 2 and 4 fairing steps with $\tau = 0.016$. The three figures in the bottom row are the results of the timestep t = 0.001 smoothing after 1, 2 and 4 fairing steps, respectively.

right figures in the top row are the smoothing results of the mesh on the left top after 2 and 4 fairing steps, respectively. As an alternative to the adaptive discretization in time direction, we also provide an approach for the adaptive choice of the diffusion tensor in the diffusion PDE equation.

The remaining of the chapter is organized as follows: Section 2 summarizes the diffusion PDE model used, followed by the discretization section 3. In the spatial direction, the discretization is realized using the C^1 smooth finite element space defined by the limit function of Loop's subdivision (box spline), while the discretization in the time direction is adaptive. The conclusion section 4 provides examples showing the superiority of the adaptive scheme.

Geometric Diffusion Equation

We shall solve the following nonlinear system of parabolic differential equations (see [26, 67]):

$$\partial_t x(t) - \Delta_{M(t)} x(t) = 0, \tag{2.20}$$

where $\Delta_{M(t)} = \mathrm{div}_{M(t)} \circ \nabla_{M(t)}$ is the Laplace-Beltrami operator on $M(t)$, $M(t)$ is the solution surface at time t and $x(t)$ is a point on the surface. $\nabla_{M(t)}$ is the gradient operator on the surface. This equation is a generalization of heat equation $\partial_t \rho - \Delta \rho = 0$ to surfaces, where Δ is the Laplacian. To enhance sharp features, a *diffusion tensor D*, acting on the gradient, has been introduced (see [26, 67]). Then (2.20) becomes

$$\partial_t x(t) - \mathrm{div}_{M(t)}(D \nabla_{M(t)} x(t)) = 0. \tag{2.21}$$

The diffusion tensor $D := D(x)$ is a symmetric and positive definite operator from $T_x M$ to $T_x M$. Here $T_x M$ is the tangent space of M at x. The detailed discussion for choosing the diffusion tensor can be found in [26, 67] for enhancing sharp features. In this chapter, we do not address the problem of enhancing sharp features. However, we shall use a scalar diffusion tensor for achieving an adaptive diffusion effect. The divergence $\mathrm{div}_M \psi$ for a vector field $\psi \in \mathrm{TM}$ is defined as the dual operator of the gradient (see [193]):

$$(\mathrm{div}_M v, \varphi)_M = -(v, \nabla_M \varphi)_{\mathrm{TM}}, \forall \varphi \in C_0^\infty(M), \tag{2.22}$$

where $C_0^\infty(M)$ is a subspace of $C^\infty(M)$, whose elements have compact support. TM is the tangent bundle, which is a collection of all the tangent spaces. The inner product $(\varphi, \psi)_M$ and $(u, v)_{\mathrm{TM}}$ are defined by the integration of $\varphi\psi$ and $u^T v$ over M, respectively. The gradient of a smooth function f on M is given by

$$\nabla_M f = [t_1, t_2] G^{-1} \left[\frac{\partial (f \circ x)}{\partial \xi_1}, \frac{\partial (f \circ x)}{\partial \xi_2} \right]^T, \tag{2.23}$$

where
$G^{-1} = \frac{1}{\det G} \begin{bmatrix} g_{22} & -g_{12} \\ -g_{21} & g_{22} \end{bmatrix}, G = \begin{bmatrix} g_{22} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}, g_{\mathrm{ij}} = \left(\frac{\partial}{\partial \xi_1} \right)^T \frac{\partial}{\partial \xi_2},$
$x(\xi_1, \xi_2)$ is a local parameterization of the surface. G is known as the first fundamental form. For a vector field $X = \sum_{i=1}^{2} X^i \frac{\partial}{\partial \xi_i} \in \mathrm{TM}$, an explicit expression for the divergence is given by (see [125], page 84)

$$\mathrm{div}_M X = \frac{1}{\sqrt{\det G}} \sum_{i=1}^{2} \frac{\partial}{\partial \xi_i} (\sqrt{\det G} X^i).$$

Then it is easy to derive that

$$\mathrm{div}_M (h \nabla_M f) = (\nabla_M f)^T \nabla_M h + h \blacksquare_M f, \tag{2.24}$$

where f, h are smooth functions on M. From (2.23), (2.24) and the fact that $\Delta_M x = 2H(x)n(x)$, we could rewrite (2.21) as

$$\partial_t x(x) = \nabla_M D(x) + 2D(x)H(x)n(x), \tag{2.25}$$

where $H(x)$ and $n(x)$ are the mean curvature and the unit normal of M, respectively. Equation (2.25) implies that the motion of the surface $M(t)$ can be decomposed into two parts: One is the tangential displacement caused by $\nabla_M D(x)$, and the other is the normal displacement (mean curvature motion) caused by $2D(x)H(x)n(x)$.

Using (2.22), the diffusion problem (2.21) could be reformulated into the following variational form

$$\begin{cases} \text{Find a smooth } x(t) \text{ such that} \\ (\partial_t x(t), \theta)_{M(t)} + (D\nabla_{M(t)} x(t), \nabla_{M(t)}\theta)_{\mathrm{TM}(t)} = 0 \ , \\ M(0) = M, \end{cases} \tag{2.26}$$

for any $\theta \in C_0^\infty(M(t))$. This variational form is the starting point for the discretization.

We already know that equation (2.20) describes the mean curvature motion. Its regularization effect could be seen from the following equation (see [67, 195])

$$\frac{d}{\mathrm{dt}}\mathrm{Area}(M(t)) = -\int_{M(t)} H^2 \mathrm{dx}, \frac{d}{\mathrm{dt}}\mathrm{Volume}(M(t)) = -\int_{M(t)} H\mathrm{dx} \tag{2.27}$$

,

where $\mathrm{Area}(M(t))$ and $\mathrm{Volume}(M(t))$ are the area of $M(t)$ and volume enclosed by $M(t)$, respectively, H is the mean curvature. From these equations, we see that the evolution speed depends on the mean curvature of the surface but not on the density of the mesh. Hence if the mesh is spatially adaptive, the dense parts that have detailed structures, have larger curvatures, which very possibly be over-faired.

Discretization We discretize equation (2.26) in the time direction first and then in the spatial direction. Given an initial value $x(0)$, we wish to have a solution $x(t)$ of (2.26) at $t = T(x(0))$. Using a semi-implicit Euler scheme, we have the following time direction discretization:

$$\begin{cases} \text{Find a smooth } x(t) \text{ such that} \\ \left(\frac{x(T)-x(0)}{T}, \theta\right)_{M(0)} + (D\nabla_{M(0)} x(T), \nabla_{M(0)}\theta)_{\mathrm{TM}(0)} = 0, \end{cases} \tag{2.28}$$

for any $\theta \in C_0^\infty(M(t))$. If we want to go further along the time direction, we could treat the solution at $t = T(x)$ as the initial value and repeat the same process. Hence, we consider only one time step in our analysis.

3.1 Spatial Discretization

The function in our finite element space is locally parameterized as the image of the unit triangle

$T = \left\{(\xi_1, \xi_2) \in R^2 : \xi_1 \geq 0, \xi_2 \geq 0, \xi_1 + \xi_2 \leq 1\right\}$.

That is, $(1 - \xi_1 - \xi_2, \xi_1, \xi_2)$ are the barycentric coordinate of the triangle. Using this parameterization, our discretized representation of M is $M = ?_{\alpha=1}^{k} T_\alpha$, $\circ T_\alpha \cap \circ T_\beta = \varphi$ for $\alpha \neq \beta$, where $\circ T_\alpha$ is the interior of T_α. Each triangular patch is assumed to be parameterized locally as $x^\alpha : T \to T_\alpha$; $(\xi_1, \xi_2) \mapsto x^\alpha(\xi_1, \xi_2)$. Under this parameterization, tangents and gradients can be computed directly. The integration on surface M is given by

$$\int_M \mathrm{fdx} := \sum_\alpha \int_T f(x^\alpha(\xi_1, \xi_2)) \sqrt{\det(g_{\mathrm{ij}})} d\xi_1 d\xi_2.$$

The integration on triangle T is computed adaptively by numerical methods.

Let M_d be the given initial triangular mesh, $x_i, i = 1, \cdots, m$, be its vertices. We shall use C^1 smooth quartic Box spline basis functions to span our finite element space. The piecewise quartic basis function at vertex x_i, denoted by φ_i, is defined by the limit of Loop's subdivision for the zero control values everywhere except at x_i where it is one (see [26] for detailed description of this). For simplicity, we call it the *Loop's basis*.

In Loop's subdivision scheme, the initial control mesh and the subsequent refined meshes consist of triangles only. In the refinement, each triangle is subdivided linearly into 4 sub-triangles. Then the vertex position of the refined mesh is computed as the weighted average of the vertex position of the unrefined mesh. Consider a vertex x_0^k at level k with neighbor vertices x_i^k for $i = 1, \cdots, n$ (see Fig 2), where n is the valence of vertex x_0^k. The coordinates of the newly generated vertices x_i^{k+1} on the edges of the previous mesh are computed as

$x_i^{k+1} = \frac{3x_\cdot^k + 3x_i^k + x_{i-1}^k + x_{i+1}^k}{8}, i = 1, \cdots, n,$

where index i is to be understood modulo n. The old vertices get new positions according to

$x_0^{k+1} = (1 - \mathrm{na})x_0^k + a(x_1^k + x_2^k + \cdots + x_n^k),$

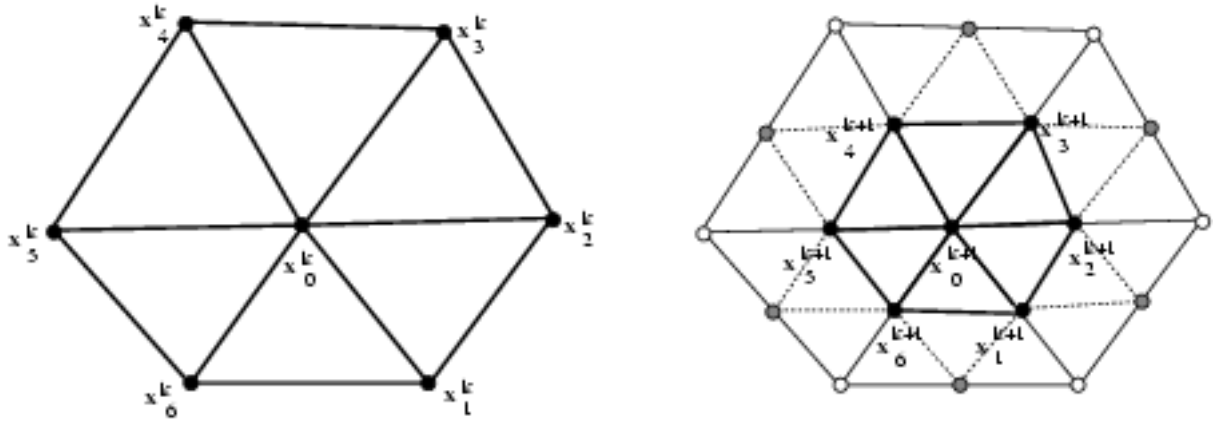


Figure 4.30: Loop's Subdivision and Finite Element Function Space. Refinement of triangular mesh around a vertex.

where
$a = \frac{1}{n}\left[\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n}\right)^2\right]$.

Note that all newly generated vertices have a valence of 6, while the vertices inherited from the original mesh at level zero may have a valence other than 6. We will refer to the former case as *ordinary* and to the later case as *extraordinary*.

Let e_j, $j = 1, \cdots, m_i$ be the 2-ring neighborhood elements. Then if e_j is regular (meaning its three vertices have valence 6), explicit Box-spline expressions exist (see [221, 236]) for φ_i on e_j. Using these explicit Box-spline expressions, we derive the BB-form expression for the basis functions φ_i (see Fig 3). These expressions could be used to evaluate φ_i in forming the linear system (2.30). If e_i is irregular, local subdivision is needed around e_i until the parameter values of interest are interior to a regular patch. An efficient evaluation method, that we have implemented, is the one proposed by Stam [221].

Compared with the linear finite element space, using the higher-order C^1 smooth finite element space spanned by Loop's basis does have advantages. The basis functions of this space have compact support (within 2-rings of the vertices). This support is bigger than the support (within 1-ring of the vertices) of hat basis functions that are used for the linear discrete surface model. Such a difference in the size of support of basis functions makes our evolution more efficient than those previously reported, due to the increased bandwidth of the affected frequencies. The reduction speed of high frequency noise in our approach is not that drastic, but still fast, while the reduction speed of lower frequency noise is not slow. Hence, the bandwidth of affected frequencies is wider. A comparative result showing the superiority of the Loop's basis function is given in [26].

Let $V_{M(0)}$ be the finite dimensional space spanned by the Loop's basis functions $\{\varphi_i\}_{i=1}^m$. Then $V_{M(0)} \subset C^1(M(0))$. Let
$x(0) = \sum_{i=1}^{m} x_i(0)\varphi_i \in M(0)$, $x(T) = \sum_{i=1}^{m} x_i(T_i)\varphi_i$, $\theta = \varphi_j$.

Then equation (2.28) is discretized in $V^3_{M(0)}$ as

$$\begin{aligned} &\sum_{i=1}^{m} (x_i(T_i) - x_i(0))(T_i^{-1}\varphi_i, \varphi_j)_{M(0)} + \\ &\sum_{i=1}^{m} x_i(T_i)(D\nabla_{M(0)}\varphi_i, \nabla_{M(0)}\varphi_j)_{\mathrm{TM}(0)} = 0 \end{aligned} \tag{2.29}$$

for $j = 1, \cdots, m$, where $x_i(0) := x_i$ is the i-th vertex of the input mesh M_d, $T_i = T(y_i(0))$ and $y_i(0)$ is a surface point corresponding to vertex $x_i(0)$. Equation (2.29) is a linear system for unknowns $x_i(T_i)$.

Adaptive Timestep We first use adaptive timesteps to achieve the adaptive evolution effect. In this case the diffusion tensor D is chosen to be identity, but $T(x)$ is not a constant function. Now (2.29) can be written in the following matrix form:

$$(S+L)X(T) = \mathrm{M}X(0), \tag{2.30}$$

where $X(T) = [x_1(T_1), \cdots, x_m(T_m)], X(0) = [x_1(0), \cdots, x_m(0)]$ and

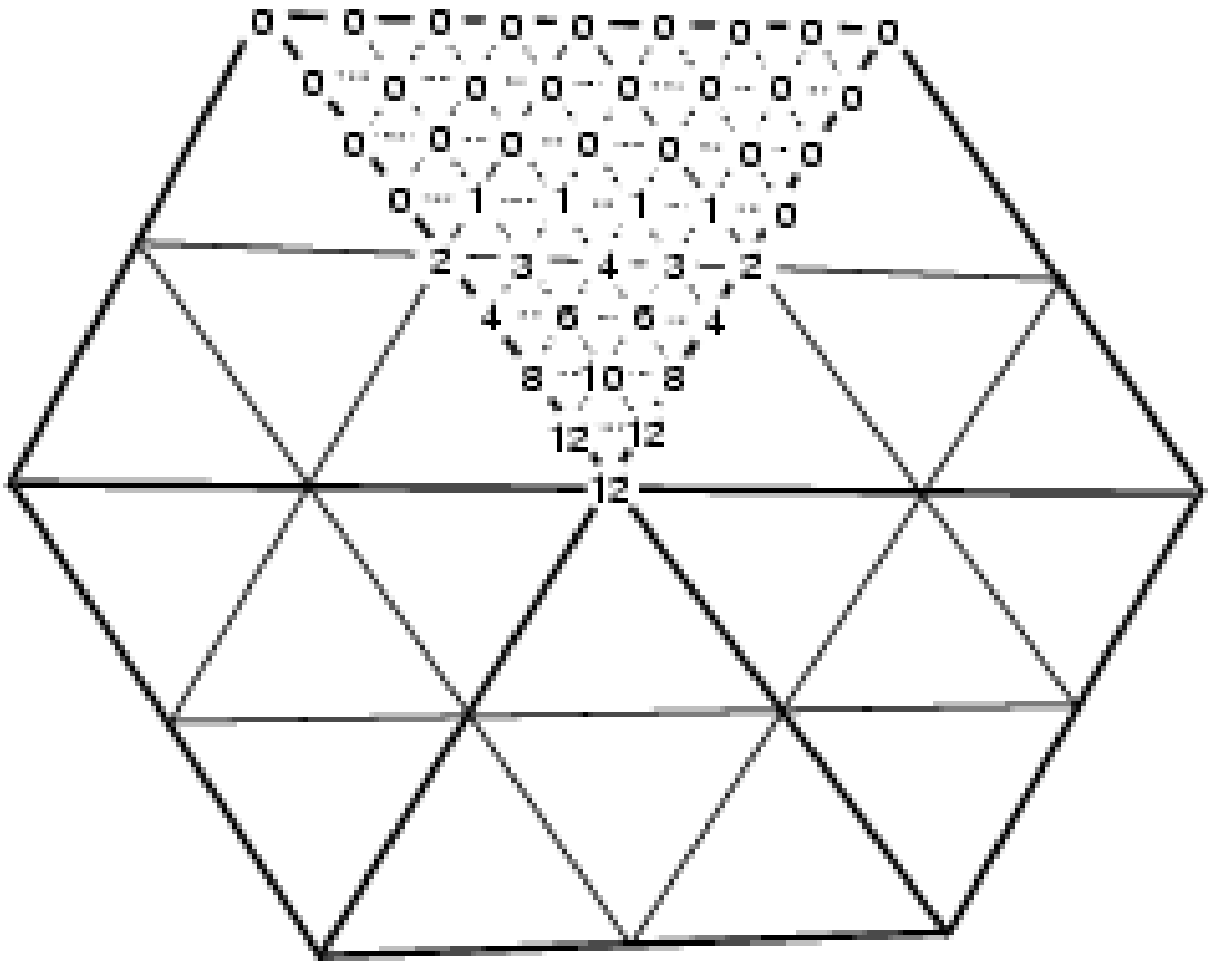


Figure 4.31: The definition of base: The quartic Bezier coefficients (each has a factor 1/24). The Bezier coefficients on the five macro-triangles are obtained by rotating the top macro-triangle around the center to the other five positions.

$$\begin{aligned} S &= \left((T_i^{-1}\varphi_i, \varphi_j)_{M(0)}\right)_{i,j=1}^m, \\ L &= \left((D\nabla_{M(0)}\varphi_i, \nabla_{M(0)}\varphi_j)_{\mathrm{TM}(0)}\right)_{i,j=1}^m. \end{aligned} \tag{2.31}$$

Note that both S and L are symmetric. Since $\varphi_1, \varphi_2, \ldots, \varphi_m$ are linearly independent and have compact support, S is sparse and positive definite. Similarly, L is symmetric and nonnegative definite. Hence, $S+L$ is symmetric and positive definite.

The coefficient matrix of system (2.30) is highly sparse. An iterative method for solving such a system is desirable. We solve it by the conjugate gradient method with a diagonal preconditioning.

Defining adaptive timesteps. Now we illustrate how $T(x)$ is defined. At each vertex x_i of the mesh M_d, we first compute a value $d_i > 0$, which measures the density of the mesh around x_i. We propose two approaches for computing it:

1. d_i is defined as the average of the distance from x_i to its neighbor vertices.

2. d_i is defined as the sum of the areas of the triangles surrounding x_i.

To make the $d_i's$ relative to the density of the mesh but not the geometric size, we always resize the mesh into the box $[-3,3]^3$. The experiments show that both approaches work well, and the evolution results have no significant difference. This value d_i is used as control value for defining timestep that is the same as defining the surface point:

$$T(x) = \tau D(x), D(x) = \sum_{i=1}^m d_i \varphi_i, \tag{2.32}$$

where $\tau > 0$ is a user specified constant. Hence, T is a function in the finite element space $V_{M(0)}$. Note that since T is not a constant any more, it is involved in the integration in computing the stiffness matrix S. Since $T(x) \in V_{M(0)}$, it is C^2, except at the extraordinary vertices, where it is C^1. However, $T(x)$ may also be noisy, since it is computed from the noisy data. To obtain a smoother $T(x)$, we smooth repeatedly the control value d_i at the vertex x_i by the following rule:

$$d_i^{(k+1)} = (1 - n_i l_i) d_i^{(k)} + l_i \sum_{j=1}^{n_i} d_j^{(k)}, \tag{2.33}$$

where $d_i^{(0)} = d_i$ for $i = 1, \cdots, m$, $d_j^{(k)}$ in the sum are the control values at the one-ring neighbor vertices of x_i, n_i is the valence of x_i, l_i and $a(n_i)$ are given as follows:

$l_i = \frac{1}{n_i + 3/8a(n_i)}, a(n_i) = \frac{1}{n_i}\left[\frac{5}{8} - \left(\frac{3}{8} + \frac{1}{4}\cos\frac{2\pi}{n_i}\right)^2\right]$.

The smoothing rule (2.33) is in fact for computing the limit value of Loop's subdivision (see [150], pp 41-42) applying to the control values $d_i^{(k)}$ at the vertices. In our examples, we apply this rule three times. Experiments show that even more times of smoothing of d_i are not harmful, but the influence to the evolution results are minor. The smoothing effect of (2.33) could be seen by rewriting it in the following form

$\frac{d_i^{(k+1)} - d_i^{(k)}}{n_i l_i} = \frac{1}{n_i}\sum_{j=1}^{n_i}\left(d_j^{(k)} - d_i^{(k)}\right)$.

The left-handed side could be regarded as the result of applying the *forward Euler method* to the function $d_i(t)$, the right-handed side is the umbrella operator (see [76]). Hence, (2.33) is a discretization of the equation $\frac{\partial D}{\partial t} = \Delta D$. Since $n_i l_i < 1$, the stability criterion for (2.33) is satisfied.

A different view of adaptive timestep approach

Consider the following diffusion PDE

$$\partial_t x(t) - D(x)\Delta_{M(t)} x(t) = 0, \tag{2.34}$$

where $D(x)$ is a function defined by (2.32). Again, equation (2.34) describes the mean curvature motion with a compression factor $D(x)$. If we use a semi-implicit Euler scheme to discretize the equation with constant timestep τ, we could arrive at the same linear system as (2.30). Hence, solving the equation (2.20) with an adaptive timestep $\tau D(x)$ is equivalent to solving equation (2.34) with a uniform timestep τ. But (2.34) may be easier to handle in the theoretical analysis.

Uniform Timestep and Adaptive Diffusion Tensor Now we use uniform timestep τ but a non-identity diffusion tensor $D(x)$. This $D(x)$ is the same as the one defined in (2.32), but we should regard it as $D(x)I$, where I is the identity diffusion tensor. The discretized equation (2.29) then becomes

$$\begin{aligned}&\sum_{i=1}^{m}(x_i(\tau)-x_i(0))(\varphi_i,\varphi_j)_{M(0)}+\\&\sum_{i=1}^{m}x_i(\tau)(\tau D\nabla_{M(0)}\varphi_i,\nabla_{M(0)}\varphi_j)_{\mathrm{TM}(0)}=0.\end{aligned}\tag{2.35}$$

From this, a similar linear system as (2.30) is obtained with

$$\begin{aligned}S&=\left((\varphi_i,\varphi_j)_{M(0)}\right)_{i,j=1}^{m},\\L&=\left((\tau D\nabla_{M(0)}\varphi_i,\nabla_{M(0)}\varphi_j)_{\mathrm{TM}(0)}\right)_{i,j=1}^{m}.\end{aligned}\tag{2.36}$$

We know that $D(x)$ is a smooth positive function that characterizes the density of the surface mesh. The effect of this diffusion tensor is suppressing the gradient where the mesh is dense, and hence slows down the evolution speed. Comparing equation (2.30) with equation (2.35), we find that they are similar (since $\tau D = T$), though not equivalent. Indeed, if $D(x)$ is a constant on each triangle of M, then they are equivalent. In general, $D(x)$ is not a constant, but approximately a constant on each triangle, hence the observed behavior of (2.30) and (2.35) are often similar. The last two figures in Fig 4 exhibit this similarity, where the third and fourth figures are the evolution results using an adaptive timestep and an adaptive diffusion tensor, respectively. Since the results of the two adaptive approaches are very close, in the other examples provided in this chapter, we use only the adaptive timestep approach.

Homogenization Effect of D

It follows from (2.25) that the non-constant diffusion tensor $D(x)$ causes tangential displacement of the vertices. For the diffusion tensor $D(x)$ defined in the last sub-section, we know that it is adaptive to the density of the mesh in the sense that it takes smaller values at denser regions of the mesh. Consider a case where a small triangle is surrounded by large triangles. In such a case, function $D(x)$ is small on the triangle and larger elsewhere. This implies that the gradient of $D(x)$ on the small triangle points to the outside direction, and the tangential displacement makes the small triangle become enlarged. If the density of the mesh is even, then $D(x)$ is near a constant. Then the tangential displacement is minor. Hence, the adaptive diffusion tensor we use has homogenizing effect. Such an effect is nice and important, as it avoids producing collapsed or tiny triangles in the faired meshes.

Algorithm Summary For a given initial mesh, stopping control threshold values $\varepsilon_i > 0$, $i = 1,2$ and $\tau > 0$, the adaptive timestep evolution algorithm could be summarized via the following pseudo-code:

```
Compute function and derivative values of φ_i on the integration points;
do {
  Compute d_i;
  Smooth d_i by (2.33);
  Compute matrices S and L by (2.31);
  Solve linear system (2.30);
  Compute H(t);
}  while (none of (2.37)-(2.38) is satisfied);
```

Note that the evolution process does not change the topology of the mesh. Hence the basis functions could be computed before the multiple iterations.

We use two of the three stopping criteria proposed in [26] for terminating the evolution process: Let

$$H(t)=\int_{M(t)}||H(t,x)||^2\mathrm{dx}/\int_{M(0)}||H(0,x)||^2\mathrm{dx},$$

where $H(t,x)$ is the mean curvature vector at the point x and time

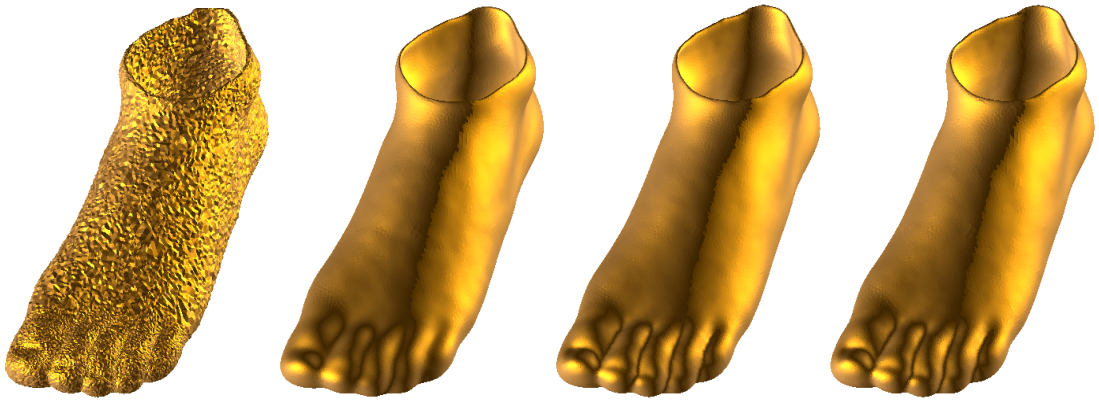


Figure 4.32: The left figure is the initial geometry mesh. The second figure is the faired mesh after 3 fairing iterations with uniform timestep t = 0.0011. The third and fourth figures are the faired meshes after 3 fairing iterations with adaptive timestep and alternatively adaptive diffusion tensor with uniform timestep $\tau = 0.025$, respectively.

tD(x). The stopping criteria are

$$|H'(t)| \leq \varepsilon_{1,} \tag{2.37}$$

or

$$H(t) \leq \varepsilon_2. \tag{2.38}$$

where ε_i are user specified control constants, $H'(t)$ is computed by divided differences.

Summary

We have proposed two simple adaptive approaches in solving the diffusion PDE by the finite element discretization in the spatial direction and the semi-implicit discretization in the time direction, aiming to solving the under-fairing/over-smooth problems that beset the uniform diffusion schemes. The implementation shows that the proposed adaptive schemes work very well.

Figure 4 and Fig 5 are used to illustrate the difference between the uniform timestep evolution and the adaptive timestep evolution. Since the adaptive timestep is not uniform, we cannot compare the evolution results for the same time. The comparing criterion we adopted here is we evolve the surface, starting from the same input, to arrive at similar smoothness for the rough/detailed features and compare the detailed/rough features. In Fig 4, the left figure is the input mesh, the second figure uses uniform timestep, the third and fourth figures use a adaptive timestep and a adaptive diffusion tensor with a uniform timestep, respectively. Comparing the three smoothing results, we can see that the large features look similar, but the toes of the foot are very different. The evolution results of the adaptive timestep and the adaptive diffusion tensor are much more desirable. Figure 5 exhibits the same effect. The top figure shows the input mesh, the next two are the results of the uniform timestep evolution. Comparing these to the bottom two figures, which are the results of the adaptive timestep evolution, many detailed features on the back and the snout of the crocodile are preserved by the adaptive approach. Furthermore, the large features of the uniform timestep evolution (compare the tails of the crocodiles) are less fairer than that of the adaptive timestep evolution, even though the detailed features are already over-faired.

4.3 Segmentation

linear and non-linear

4.4 Skeletonization

linear and non-linear

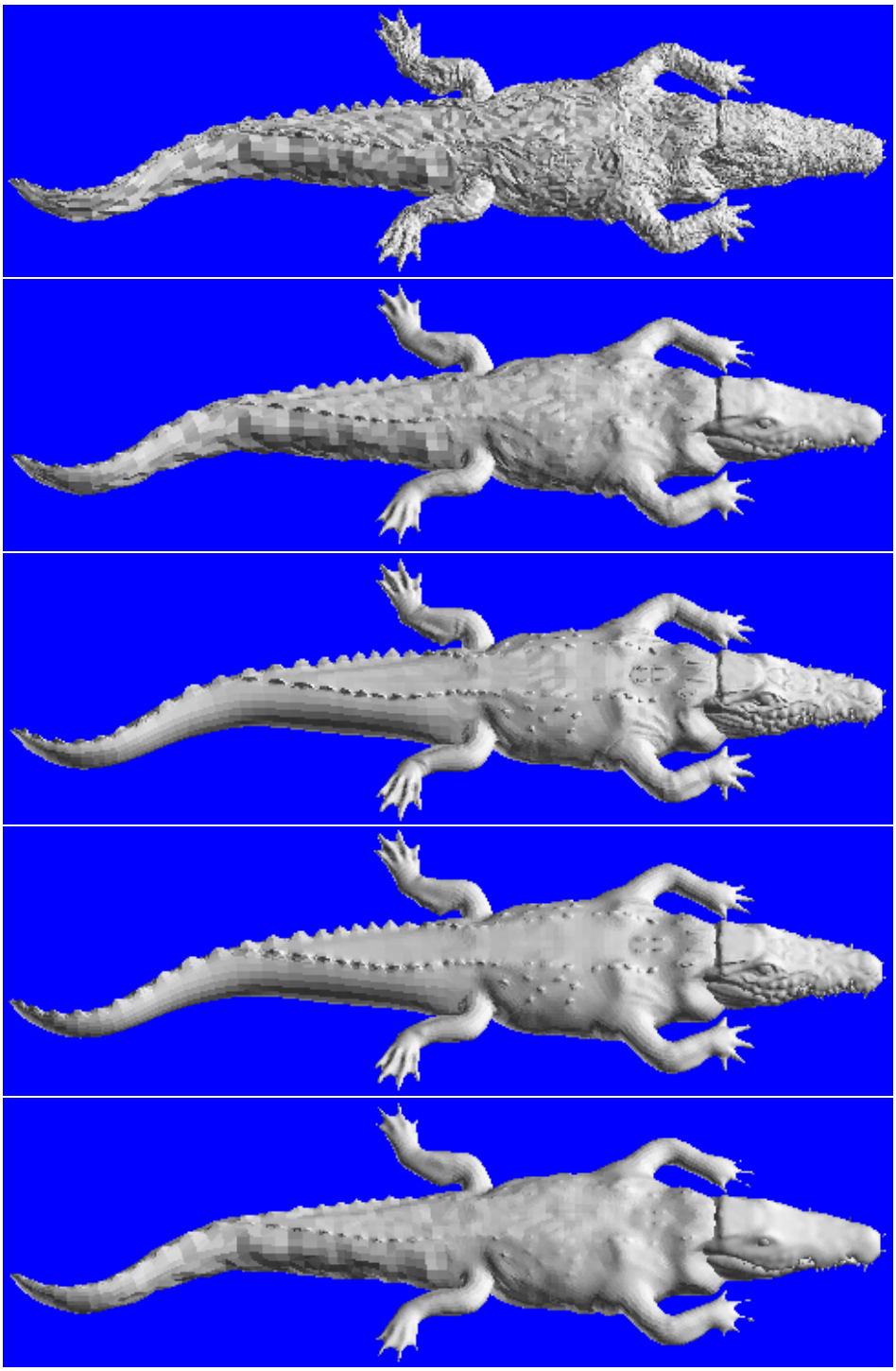


Figure 4.33: The top figure is the initial geometry mesh. The second and the third figures are the faired meshes after 2 and 4 fairing iterations with uniform timestep t = 0.0001. The last two are the faired meshes after 2 and 4 fairing iterations with adaptive timestep and τ = 0.0016.

The following algorithm is taken from [113] (Goswami2006Regions).

Problem and motivation. Many applications in shape modeling require to identify the salient features of a given shape. Some of them such as assembly planning, feature tracking, animations, structure elucidation of bio-molecules, human-body modeling benefit from a semantic annotation of the features. One such natural annotation is achieved by classifying the features as 'tubular' and 'flat'. Obviously, this annotation is ambiguous since the feature-space is a continuum resulting into features that cannot be simply classified as tubular or flat. Nevertheless, many designed and organic shapes have pronounced features that are perceived to be tubular and flat. We seek to identify these features using a topological method. The unstable manifolds induced by a shape distance function identify some one- and two-dimensional subsets of the medial axis. The preimage of a function that maps the points on the surface to the medial axis provides an association of the shape to these one- and two-dimensional subsets. The preimage of the one-dimensional subset is called tubular whereas that of the two-dimensional subset is called flat. Our experimental result shows that this classification can be effectively approximated for many datasets in practice.

Results. Given a compact surface Σ smoothly embedded in $\mathbb{R}^3$, a distance function h_Σ can be assigned over $\mathbb{R}^3$ that assigns to each point its distance to Σ.

$$h_\Sigma \ : \ \mathbb{R}^3 \to \mathbb{R}, \ \ x \mapsto \inf_{p \in \Sigma} \|x - p\|$$

In applications, Σ is often known via a finite set of sample points P of Σ. Therefore it is quite natural to approximate the function h_Σ by the function

$$h_P \ : \ \mathbb{R}^3 \to \mathbb{R}, \ \ x \mapsto \min_{p \in P} \|x - p\|$$

which assigns to each point in $\mathbb{R}^3$ the distance to the nearest sample point in P.

In this paper, we start with a finite sample P of Σ and identify the index 1 and index 2 saddle points of h_P from the Voronoi diagram $\mathrm{Vor}\,P$ and its dual Delaunay triangulation $\mathrm{Del}\,P$ of P. We then select only the saddle points of both indices which lie on the interior medial axis of Σ and compute their unstable manifolds. The unstable manifold of index 1 saddle points (U_1) are two dimensional whereas those of index 2 (U_2) are one dimensional. Exact computations of U_1 is prone to numerical error. So, we present an algorithm to compute them approximately. We then map the points belonging to U_1 and U_2 back to Σ. The image of U_1 under the mapping gives the flat regions of Σ and that of U_2 gives its tubular regions.

Thus, the main contributions of this paper are:

- Algorithms to compute the unstable manifolds of the index 2 saddles points of h_P exactly and those of the index 1 saddle points approximately,
- Identification of the tubular and flat features of Σ from its point sample P via the unstable manifolds of the saddle points,
- Experimental results exhibiting the performance of our algorithm on several point sampled objects.

Voronoi-Delaunay Diagram of P

In this paper we always assume the distance metric to be Euclidean unless otherwise stated. For a finite set of points P in $\mathbb{R}^3$, the Voronoi cell of $p \in P$ is

$$V_p = \{x \in \mathbb{R}^3 \ : \ \forall q \in P - \{p\}, \|x - p\| \leq \|x - q\|)\}.$$

If the points are in general position, two Voronoi cells with non-empty intersection meet along a planar, convex Voronoi facet, three Voronoi cells with non-empty intersection meet along a common Voronoi edge and four Voronoi cells with non-empty intersection meet at a Voronoi vertex. A cell decomposition consisting of the *Voronoi objects*, that is, Voronoi cells, facets, edges and vertices is the Voronoi diagram $\mathrm{Vor}\,P$ of the point set P.

The dual of $\mathrm{Vor}\,P$ is the Delaunay diagram $\mathrm{Del}\,P$ of P which is a simplicial complex when the points are in general position. The tetrahedra are dual to the Voronoi vertices, the triangles are dual to the Voronoi edges, the edges are dual to the Voronoi facets and the vertices (sample points from P) are dual to the Voronoi cells. We also refer to the Delaunay simplices as *Delaunay objects*.

Induced Flow

The distance function h_P induces a flow at every point $x \in \mathbb{R}^3$. This flow has been characterized earlier [111]. See also [86]. For completeness we briefly mention it here.

Critical Points. The critical points of h_P are those points where h_P has no non-zero gradient along any direction. These are the points in $\mathbb{R}^3$ which lie within the convex hull of its closest points from P. It turns out that the critical points of h_P are the intersection points of the Voronoi objects with their dual Delaunay objects.

- *Maxima* are the Voronoi vertices contained in their dual tetrahedra,
- *Index 2 saddles* lie at the intersection of Voronoi edges with their dual Delaunay triangles,
- *Index 1 saddles* lie at the intersection of Voronoi facets with their dual Delaunay edges, and
- *Minima* are the sample points themselves as they are always contained in their Voronoi cells.

In this discrete setting, the index of a critical point is the dimension of the lowest dimensional Delaunay simplex that contains the critical point.

Flow. For every point $x \in \mathbb{R}^3$, let $V(x)$ be the lowest dimensional Voronoi object that contains x and $D(x)$ be its dual. Now *driver* of x, denoted as $d(x)$, is defined as

$$d(x) = \text{argmin}_{y \in D(x)} \|x - y\|$$

The direction of steepest ascent can be uniquely determined by a unit vector in the direction of $x - d(x)$. The critical points coincide with their drivers. Now one can assign a vector v at every x with a zero vector assigned at the critical points. The resulting vector field is not necessarily continuous. Nevertheless, it induces a *flow* in $\mathbb{R}^3$. This flow tells how a point x moves in $\mathbb{R}^3$ along the steepest ascent of h_P and the corresponding path is known as the *orbit* of x. We can also define an *inverted orbit* of x where x moves in the direction of steepest descent.

Stable and Unstable Manifolds. For a critical point c its stable manifold is the set of points whose orbits end at c. The unstable manifold of a critical point c is the set of points whose inverted orbits end at c. The structure and computation of stable manifolds of the critical points of h_P were described in [111]. They can be computed from the Delaunay triangulations of the given point sets though they may not be subcomplexes of the Delaunay triangulations. For computational advantages they are also approximated by Delaunay subcomplexes as in [77].

We are interested in computing unstable manifolds and their approximations. As the Delaunay and Voronoi diagrams, the structures of stable and unstable manifolds have a duality. Interestingly, one can compute the unstable manifolds and their approximations from the Voronoi diagrams. Here we state some of the facts about the unstable manifolds of the critical points.

[A] MAXIMA. The unstable manifold is the local maximum itself.

[B] INDEX 2 SADDLES. The unstable manifold of an index 2 saddle point is a polyline starting at the saddle point and ending at a maximum.

[C] INDEX 1 SADDLES. The unstable manifold of an index 1 saddle point is a two dimensional surface patch which is bounded by the unstable manifold of index 2 saddle points.

[D] MINIMA. The unstable manifold of a local minimum is a three dimensional polytope bounded by the unstable manifold of critical points with higher indices.

In Sections 4.4 and 4.4 the computation of the unstable manifold of index 1 and index 2 saddle points is described.

Before we state the connection between the flow induced by h_P and the Vor-Del diagram of P, we would like to state some facts about the relative position of Voronoi and Delaunay objects. These relative positions can describe the nature of flows in the Voronoi objects. These facts were clearly explained in [86] for a more general setting of power distance.

Fact 1 *The unoriented normal to the supporting plane of a Voronoi facet is along its dual Delaunay edge and the plane passes through the midpoint of the edge. The Delaunay edge, though, may or may not intersect the dual Voronoi face.*

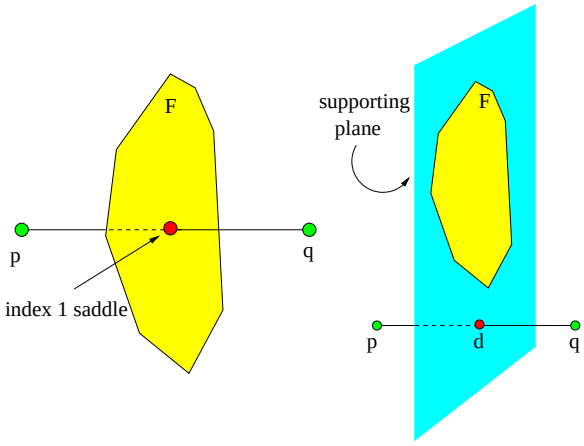


Figure 4.34: Relative position of a Voronoi facet F with respect to its dual Delaunay edge pq. The left picture shows the creation of an index 1 saddle point. The right picture shows the position of the driver d of F.

Figure 4.34 illustrates the two possibilities that may arise. The left figure corresponds to the situation that results in an index 1 saddle point.

Fact 2 *The supporting line of a Voronoi edge always intersects the plane of the dual Delaunay triangle at its circumcenter and is along its unoriented normal. The Voronoi edge may or may not intersect the interior of the Delaunay triangle.*

Figure 4.35 lists the four possible scenarios. The bottom right corresponds to the generation of an index 2 saddle point.

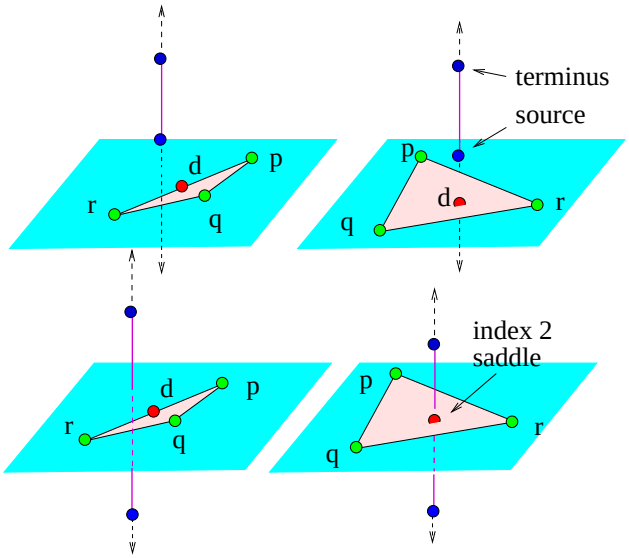


Figure 4.35: Relative position of a Voronoi edge e with respect to its dual Delaunay triangle pqr. The blue circles denote the two Voronoi vertices defining e. The driver of e is marked d and the supporting plane of triangle pqr is drawn in cyan.

We have already seen that the critical points of h_P can be computed from $\mathrm{Vor}\,P$ and $\mathrm{Del}\,P$. Also, the driver of a point x comes from the Delaunay object dual to the Voronoi object x lies in. In this context we would like to state the following lemma which is key to the further computations.

Lemma 1 *All interior points of a Voronoi object have the same driver.*

This result can be easily proved by considering all the different cases regarding the dimension of the Voronoi object and its position with respect to its dual Delaunay object.

By Lemma 1 and Facts 1 and 2 we can list the possible position of the drivers of the points lying in the interior of a certain dimensional Voronoi object.

Position of Drivers

Voronoi Cell

For a Voronoi cell V_p, the dual Delaunay object is a singleton set containing the sample point p and therefore all points x in the interior of V_p has p as their driver.

Voronoi Facet

Consider a Voronoi facet in the intersection of V_p and V_q. The dual Delaunay edge is pq and the midpoint of pq is the driver of all x lying in the interior of the Voronoi facet (Figure 4.34(right)).

Voronoi Edge

Next, consider a Voronoi edge in the intersection of V_p, V_q, V_r. As Fact 2 and Figure 4.35 indicate, the infinite line segment containing the Voronoi edge may or may not intersect the convex hull of p, q, r leading to two different possibilities

Case 1.1 In case of intersection, the circumcenter of pqr is the driver. Such Voronoi edges will be termed *non-transversal* edges as the flow is along the edge itself. The Voronoi edge has two Voronoi vertices as its endpoints. If both of them are in the same half-space defined by pqr, the closer Voronoi vertex is called *source* and the further one is called *terminus* of the Voronoi edge because the flow is directed from the closer to the further vertex. Figure 4.35 (top right) illustrates this case.

Case 1.2 If the Voronoi edge does not intersect the affine hull of $p, q,$ and r, the midpoint of the edge opposite to the largest angle of pqr is the driver. These Voronoi edges will be termed as *transversal*. If any point x moving along its orbit hits one such edge, the position of the driver implies that it will enter the Voronoi facet dual to the Delaunay edge opposite to the largest angle in pqr. Such Voronoi facet will be termed *acceptor* facets of that *transversal* Voronoi edge. Figure 4.36 illustrates the situation.

Voronoi Vertex

The case of Voronoi vertex again requires the analysis of two different cases. We assume, that it is outside its dual tetrahedron because otherwise it is a local maximum and hence is its own driver. Let v be a Voronoi vertex with the dual tetrahedron σ whose four neighbors are $\sigma_i, i = 1 \ldots 4$. Further, let the corresponding shared triangles between σ and σ_i be $t_i, i = 1 \ldots 4$ where $w_i, i = 1, \ldots 4$ is its opposite vertex in σ.

Case 2.1 There is only one triangle t_i of σ for which the Voronoi vertex v and the opposite vertex w_i lie in two different half-spaces defined by t_i. Let e_i be the Voronoi edge between the duals of σ and σ_i. Then, the driver for v (dual to σ) is same as the driver of e_i. In such cases, e_i is termed as the *outgoing* Voronoi edge of v. See top row of Figure 4.37 for an illustration.

Case 2.2 There are two triangles t_i, t_j of σ for which the Voronoi vertex v and the opposite vertex (w_i and w_j) lie in two different half-spaces defined by the corresponding triangles. Let e_i, e_j be the Voronoi edges defined as in Case 2.1. Note, in this case, both e_i, e_j are the *outgoing* Voronoi edges of v. There are two possibilities that we need to consider further.

Case 2.2.1 Both e_i, e_j are *transversal*: In this case the *acceptor*s of both of them is dual to the Delaunay edge $t_i \cap t_j$ and the corresponding driver is the midpoint of $t_i \cap t_j$. See bottom-left subfigure of Figure 4.37.

Case 2.2.2 One of e_i, e_j is *transversal*: The driver is same as that of the non-transversal Voronoi edge. See bottom-right subfigure of Figure 4.37.

In this context we state another lemma that is important for subsequent developments.

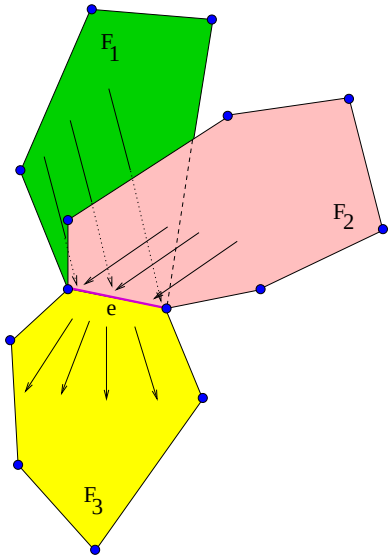


Figure 4.36: Transversal Voronoi edge e is shown in red with three incident Voronoi facets. Flow direction is shown with arrows. Flow from either of F_1 or F_2 hits e and enters F_3, the acceptor of e.

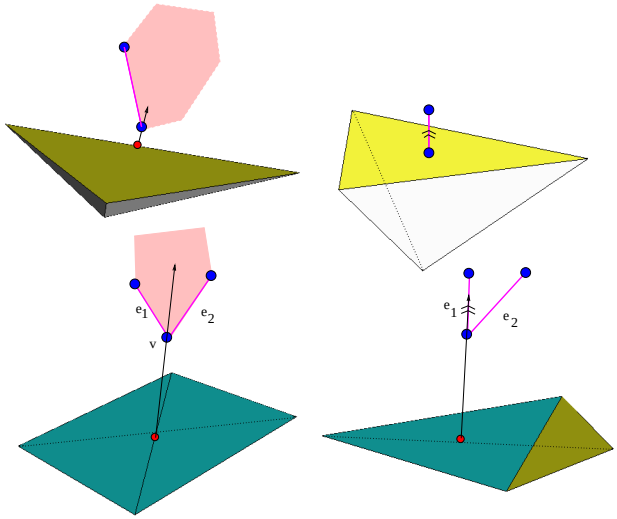


Figure 4.37: Possible driver positions of a Voronoi vertex v according to the cases 2.1 and 2.2.$(1-2)$. The acceptor Voronoi facet is shown in pink. The flow along a non-transversal Voronoi edge is shown with a double arrow. The driver is shown in red circle.

Lemma 2 *Let F be an acceptor Voronoi Facet for the transversal Voronoi edges $e_1 = (v_1, v_2) \dots e_k = (v_k, v_{k+1})$ around it.*

[A] The Voronoi edges $e_1 \dots e_k$ form a continuous chain around F.

[B] The Voronoi vertices $v_2 \dots v_k$ fall in the category 2.2.1. The Voronoi vertices v_1 and v_{k+1} fall in the category 2.2.2.

[C] F, $e_1 \dots e_k$, $v_2 \dots v_k$ have same driver which is the midpoint of the Delaunay edge dual to F.

We omit the proofs of all of the above claims.

Unstable Manifold of Index-2 Saddle Points

In this section we describe the structure and computation of the unstable manifolds of index 2 saddle points.

The unstable manifold of an index 2 saddle point is one dimensional. In our discrete setting it is a polyline with one endpoint at the saddle point and the other endpoint at a local maximum. The polyline consists of segments that are either subsets of non-transversal Voronoi edges or lie in the Voronoi facets. Due to the later case, the polyline may not be a subcomplex of Vor P.

Let us consider an index 2 saddle point, c, at the intersection of a Delaunay triangle t with a Voronoi edge e. Let the two tetrahedra sharing f be σ_1, σ_2. The edge e has the endpoints at the dual Voronoi vertices of σ_1 and σ_2, denoted as v_1, v_2 respectively. The unstable manifold $U(c)$ of c, has two intervals - one from c to v_1 and the other from c to v_2. We look at the structure of one of them, say the one from c to v_1, and the other one is similar.

At any point on the subsegment cv_1, the flow is toward v_1 from c. Once the flow reaches v_1, the subsequent flow depends on the driver of v_1. Instead of just looking at v_1, we consider a generic step, where the flow reaches at a Voronoi vertex v and we enumerate the possible situations that might occur depending on the position of driver of v. If v is a local maximum, the flow stops there, as the driver of v is v itself. Otherwise there are two cases to consider.

- v **falls into Case 2.1:** Let the dual tetrahedron be σ and the driver of v is same as that of the Voronoi edge e which is between the dual of σ and one of its neighbors, say σ'. If e is non-transversal, the flow will be along the Voronoi edge e till it hits the Voronoi vertex at the other endpoint (dual to σ'). Otherwise, the flow enters the acceptor Voronoi facet F of e. Due to Lemma 2, the driver of F is same as the driver of e. Therefore the next piece of the unstable manifold can be uniquely determined by the driver of e, say d and the Voronoi vertex v. It is the segment between v and the point where the ray $\overrightarrow{dv}$ intersects a Voronoi edge of F.

- v **falls under Case 2.2.x:** This situation is similar to the one described above. In case of both of the Voronoi edges being *transversal* (Case 2.2.1), the flow enters the acceptor Voronoi facet. In the other case (Case 2.2.2), the flow follows the non-transversal Voronoi edge.

Some segments of $U(c)$ are not along the Voronoi edges. Wherever the flow encounters a transversal Voronoi edge, it seizes to follow the Voronoi edge and enters a Voronoi facet which is acceptor for that Voronoi edge. This calls for the analysis of the flow when it crosses an acceptor Voronoi facet and hits a Voronoi edge. We have already characterized the position of the driver for a Voronoi edge and thereby classified those edges as either transversal or non-transversal. If the current edge intersected by the ray from the driver to v is a non-transversal edge, the flow will follow that Voronoi edge and hit a Voronoi vertex. Otherwise, it will enter the acceptor Voronoi facet of the Voronoi edge again. There is a technical difficulty we need to point out. Unless the acceptor for this Voronoi edge is different from the Voronoi facet the flow came from, we may encounter a cycle. The following lemma saves us from this awkward situation.

Lemma 3 *Let F be a Voronoi facet and let d be its driver. Let e be a Voronoi edge for which F is acceptor and x be any point on e. Also assume the ray from d to x intersects a Voronoi edge e'. If e' is transversal, the acceptor of e' is different from F.*

Figure 4.38 shows an example of the unstable manifold of an index 2 saddle point.

Following the above discussion on the structure of $U(c)$ we devise the algorithm to compute the unstable manifold of an index 2 saddle point c. We assume, the saddle point c carries the information about the two neighboring tetrahedra σ_1, σ_2 and additionally we have access to Del P which is used to evaluate the utility routines like acceptor() , terminus() etc. The pseudo-code of the algorithm is given in Figure 4.39.

Unstable Manifold of Index-1 Saddle Points

Unstable Manifold of index 1 saddle points are two dimensional. Due to hierarchical structure, they are bounded by the unstable manifold of index 2 saddle points. In this section we first describe the structure of the unstable manifolds and then describe an algorithm that computes an approximation of the unstable manifold of an index 1 saddle point.

Let us consider an index 1 saddle point, c. This point lies at the intersection of a Voronoi facet F and a Delaunay edge. For any point $x \in F \setminus c$, the driver is c. For all such x, if they are allowed to move in the direction of flow, they will move radially outward and hit the Voronoi edges bounding F. Thus F is in $U(c)$. Now we analyze the flow when a point hits a Voronoi edge.

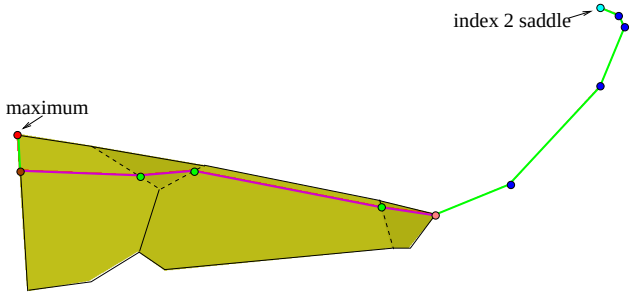


Figure 4.38: Unstable manifold $U(c)$ of an index 2 saddle point c. c is drawn with a cyan circle. The portion of $U(c)$ which is a collection of Voronoi edges is drawn in green with intermediate Voronoi vertices drawn in blue. The pink circle is a Voronoi vertex on $U(c)$ where the flow enters a Voronoi facet. The portion of $U(c)$ which lies inside the Voronoi facets is drawn in magenta. The transversal Voronoi edges intersected by this portion of $U(c)$ are dashed. $U(c)$ ends at a local maximum which is drawn in red.

We have characterized the position of the drivers for a Voronoi edge and we have also seen that depending on the driver, one can classify the Voronoi edges into two categories - transversal and non-transversal. For a non-transversal Voronoi edge, the flow is along the Voronoi edge. Such Voronoi edges lie on the boundary of $U(c)$. On the other hand, $U(c)$ grows via the acceptor facets of transversal Voronoi edges. Depending on the position of the driver, which by Lemma 2 is same for both the edge and the acceptor facet, a *truncated cone* defines the extension of $U(c)$ into the acceptor Voronoi facet. Consider the cone defined by the two rays emanating from the driver and passing through the endpoints of the transversal Voronoi edge. The intersection of the acceptor facet with the cone defines the truncated cone. The truncated cone hits a continuous chain of Voronoi edges in the acceptor facet. Some of them are completely contained in the truncated cone and some of them are intersected by the two rays and hence are partially contained in it. This chain of edges defines the new boundary of $U(c)$ through some of which $U(c)$ can be extended further recursively. Figure 4.40 shows an example truncated cone in a Voronoi facet F by the driver d and the end Voronoi vertices of the transversal Voronoi edge (green).

To compute $U(c)$ accurately, one therefore needs to compute the intersection of a ray and a line segment in three dimension. Such computations are prone to numerical errors. Therefore, we rely on an approximation algorithm that computes a superset of $U(c)$. The algorithm works as follows.

Starting from the Voronoi facet F containing c, we maintain a list of Voronoi facets which are already in $U(c)$ and a list of active Voronoi edges which are transversal edges and lie on the boundary of the current approximation of $U(c)$. Through these transversal edges we collect their acceptor facets and grow $U(c)$. Instead of computing the new set of active edges by an expensive numerical calculation of ray-segment intersection, we collect all the transversal edges of this new acceptor Voronoi facets. This way we grow $U(c)$ recursively till we have a set of Voronoi facets which are bounded by only a set of transversal Voronoi edges.

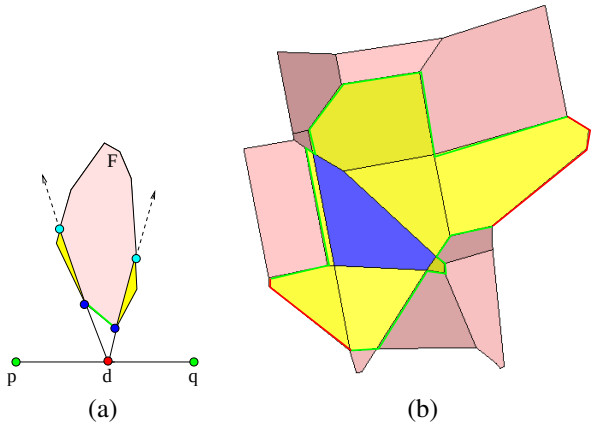
Figure 4.40(b) illustrates an intermediate stage of this computation. The index 1 saddle point c is contained in the blue Voronoi facet. The yellow Voronoi facets are already in $U(c)$. The red edges designate the static boundary as they are non-transversal and the green edges designate the active boundary through which the pink facets are included in $U(c)$ in the later stage of the algorithm. Following is the pseudo-code for this algorithm. Given an index 1 saddle point c it computes an approximation of $U(c)$. We assume c also has information about the Voronoi facet F it is contained in.

Classification of Medial Axis

In the previous two subsections we have described the structures of the unstable manifolds of an index 1 and index 2 saddle points. We have also given an accurate and an approximate algorithm to compute them. Our goal is to identify the unstable manifolds near the medial axis of Σ. Ultimately these manifolds are mapped back to Σ for the feature annotation. For this we first compute a Voronoi subcomplex that approximates the medial axis M_Σ and then identify different regions of this approximate medial axis as the unstable manifolds computed by the two subroutines UM_INDEX_2 and APPROX_UM_INDEX_1.

UM_INDEX_2(c)
1 $U_1 = cv_1$ and $U_2 = cv_2$
2 $v = v_1$
3 end$(U_1) = v_1$
4 while (v is not a maximum) do
5 if(v is not a Voronoi vertex)
6 $e =$ Voronoi edge containing v
7 if(e is non-transversal)
8 end$(U_1) =$ terminus(e)
9 $U_1 = U_1 \cup$ segment$(v,$ end(U_1)
10 $v =$ terminus(e)
11 else
12 $F =$ acceptor(e)
13 $d =$ driver$(F) =$ driver(e)
14 $x = \overrightarrow{dv} \cap e' \neq \emptyset$, e' is a Voronoi edge of F
15 end$(U_1) = x$
16 $U_1 = U_1 \cup$ segment$(v,$ end(U_1)
17 $v = x$
18 else
19 if(v falls under Case 2.1)
20 $e =$ *outgoing* Voredge (v)
21 repeat steps 7-17.
22 else if(v falls under Case 2.2)
23 $F =$ acceptor(v)
24 repeat steps 13-17.
25 endwhile
26 Similarly compute U_2.
27 return $U_1 \cup U_2$.

Figure 4.39: Pseudo-code for computation of unstable manifold of an index 2 saddle point.



(a) (b)

Figure 4.40: (a) Truncated Cone. Accurate computation selects only the pink region from the yellow Voronoi facet as part of unstable manifold of an index 1 saddle point c (not shown). (b) Snapshot of approximate computation of $U(c)$ at a generic stage.

APPROX_UM_INDEX_1(c)
1 $U = F$
2 $B =$ Voronoi edges of F
3 while ($B \neq \emptyset$) do
4 $e =$ pop(B)
5 if (e is transversal)
6 $U = U \cup$ acceptor(e)
7 $B = B \cup$ unvisited edges of acceptor(e)
8 endwhile
9 return U.

Figure 4.41: Pseudo-code for approximate computation of unstable manifold of an index 1 saddle point.

Before we describe our approach, we briefly mention a recent result by Dey, Giesen, Ramos and Sadri [78] where they proved that under sufficient sampling of Σ by P, the critical points of h_P lie either close to Σ or close to M_Σ. This motivates our approach. Applying the same result, we filter out only the index 1 and index 2 saddle points near M_Σ instead of Σ. Further, we consider only the components of M_Σ which lie in the interior of the solid bounded by Σ. For this purpose we use the TIGHTCOCONE algorithm by Dey and Goswami [79]. The implementation of this algorithm is freely available in the public domain [70] along with the software for medial axis approximations which is computed as a Voronoi subcomplex according to the algorithm by Dey and Zhao [82]. For the purpose of reconstruction, any other reconstruction algorithm also could be used [41, 7]. Applying TIGHTCOCONE followed by medial axis approximation we get the approximate interior medial axis of Σ. We perform the critical point detection only within the Voronoi subcomplex that approximates this medial axis. Let us call this set of index 1 saddle points C_1 and that of index 2 saddle points C_2. We then apply UM_INDEX_2(c) for all $c \in C_2$ and APPROX_UM_INDEX_1(c) for all $c \in C_1$. $U(c \in C_1)$ is two dimensional and $U(c \in C_2)$ is one dimensional. Therefore, by restricting the unstable manifold computation only within M_Σ we obtain two subsets of M_Σ. In the next section, we describe how this classification can be mapped back to Σ for automatic identification of its flat and tubular regions.

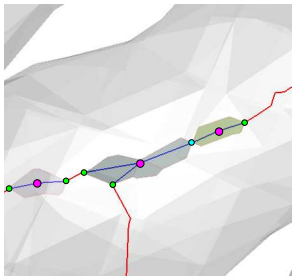


Figure 4.42: Removal of small patches in the tubular region via starring. Magenta circles indicate the centroids of these patches, green circles are the boundary vertices which connect a patch with a linear portion (red line) and cyan circle indicates where two different patches join at a common vertex. Blue lines are the replacements of these small patches obtained by the starring process.

Because of sampling artifacts, sometimes the interior medial axis in the tubular regions have a few index 1 saddle points. The unstable manifold of these saddle points need to be detected and approximated by lines. We partition the set C_1 based on the connectivity of their unstable manifolds via a common edge and every partition creates a patch which is the union of the unstable manifolds of all the index 1 saddle points falling into that partition. We further assign an *importance* value based on the area of the patch and sort the patches according to their *importance*. One could also employ other attributes like diameter, width etc. to evaluate the importance. The small clusters are then detected either by a user-specified threshold value or by simply selecting the k-smallest clusters where k is also a user-supplied parameter. These insignificant planar regions are then approximated by a set of straight lines emanating from the centroid of the patch to the boundary points which are connected to

either a polyline (green circles in Figure 4.42) or another patch (cyan circle in Figure 4.42). We call this process *starring*.

The resulting one dimensional and two dimensional subsets of the interior medial axis is shown in Figure 4.43. Left column shows the approximate medial axis computed by [82]. The right column shows the subset of medial axis captured by $U(C_2)$ and $U(C_1)$.

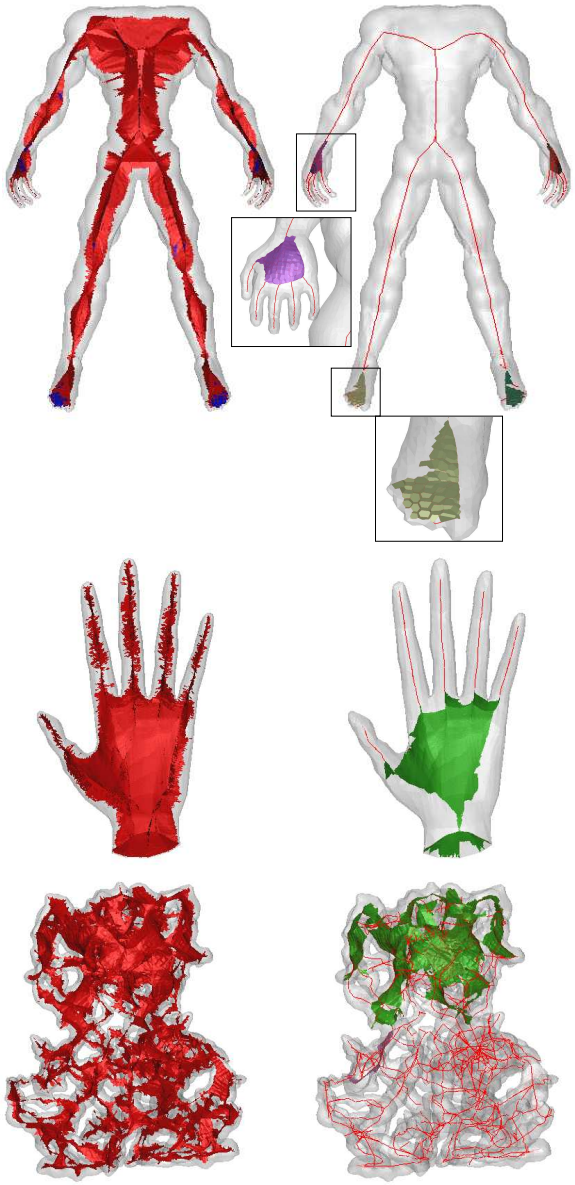


Figure 4.43: Results of Medial Axis classification. Top row shows the result for HEADLESS MAN. Two closeups have been shown to highlight the planar clusters in the palm of the hand and the feet. The closeup of hand has been rotated for visual clarity. The middle row shows the result on HAND dataset and the bottom row shows the result on a molecule data 1BVP.

Mapping of Unstable Manifolds to Σ

There is a natural association between the medial axis M_Σ and Σ via the map $\phi : \Sigma \to M_\Sigma$ where $\phi(x)$ is the center of the medial ball touching Σ at x. Following this map, any subset $A \subseteq M_\Sigma$ can be associated with $\phi^{-1}(A) \subseteq \Sigma$. Let A_1 and A_2 be the closure

of the unstable manifolds of index 2 and index 1 saddles in M_Σ defined by the distance function h_Σ. Recall that, generically, A_1 is one-dimensional and A_2 is two-dimensional. Ideally, we would like to identify $\phi^{-1}(A_1) \subseteq \Sigma$ as tubular and $\phi^{-1}(A_2) \subseteq \Sigma$ as flat. As we have an approximation of h_Σ by h_P, we compute these tubular and flat regions for the unstable manifolds in the approximate medial axis which we denote also as M_Σ for convenience.

We face a difficulty to compute an approximation of the preimage of ϕ from the approximate medial axis M_Σ. We are interested in computing an approximation of the preimage of $M'_\Sigma = A_1 \cup A_2 \subseteq M_\Sigma$ under the map ϕ.

Unfortunately, this requires an expensive computation to cover the entire M'_Σ which often spans a substantial portion of M_Σ. A naive approach is to take only a sample of M'_Σ, namely the Voronoi vertices, and then associate them to P, a sample of Σ, via the Voronoi-Delaunay duality. This also proves useless because M'_Σ does not contain all the Voronoi vertices and therefore many points in P cannot be covered by this Voronoi-Delaunay duality.

It turns out that the distance function h_P again proves to be useful to establish a correspondence between Σ and M'_Σ. Recall that, the stable manifold of a critical point is a collection of points whose orbits terminate at that critical point. Let X and Y be the set of maxima in $A_1 \subseteq M'_\Sigma$ and $A_2 \subseteq M'_\Sigma$ respectively. Consider the stable manifolds of the maxima in X and Y. The points in P that are in the stable manifolds of X are associated with the tubular regions and those in the stable manifolds of Y are associated with the flat regions. If a point belongs to the stable manifolds of maxima in X as well as in Y, we tag it arbitrarily. These points belong to the regions where a tubular part meets a flat part. Subsequently, every triangle of the surface reconstructed by TIGHT COCONE is tagged as flat or tubular if at least two of its vertices are already marked as flat or tubular respectively.

Computation of stable manifold of maxima has been described in [111] and its approximation was given in [77]. We follow the approximate algorithm to compute the stable manifolds of the local maxima lying on M'_Σ.

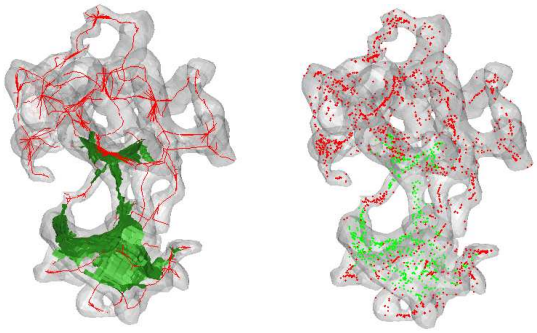


Figure 4.44: One dimensional subset of the interior medial axis is drawn in red and the two dimensional subset of the medial axis is drawn in green for the molecule data 1IRK. The right subfigure shows the selection of local maxima of the distance function in those two parts, colored accordingly.

Figure 4.44 shows the set M'_Σ of the molecule data 1IRK, and the set of maxima belonging to that set and identified as linear or planar. The corresponding flat and tubular portions of the surface captured by the mapping via stable manifold of these maxima - colored golden and cyan respectively - are shown in Figure 4.46. We collected the protein from Protein Data Bank [39] and blurred the molecule at a resolution 8 angstrom. Further we took the vertex set of a suitable level set as the input to our program. We verified the result with the existing literature in structural biology and we have seen that the flat regions identified by our algorithm correspond to the β-sheets of the protein molecule.

Annotation Algorithm

The modules described in the previous sections and subsections can thus be combined to devise an algorithm for automatic feature annotation of Σ. We give the pseudo-code of this annotation algorithm here.

Implementation Issues

The algorithm works on the Voronoi-Delaunay diagram of the set of sample points lying on the surface. To robustly compute the Delaunay triangulation and its dual Voronoi diagram for the input set of points we use the library CGAL [72] which is

```
IDENTIFY_FLAT_AND_TUBULAR_REGIONS(P)
 1  Compute Vor P and Del P.
 2  Compute the interior Medial Axis M_Σ
      by TIGHTCOCONE_AND_MA (P)
 3  C_1 = set of index 1 saddle points lying on M_Σ and
    C_2 = set of index 2 saddle points lying on M_Σ,
 4  A_1 = A_2 = ∅
 5  for all c ∈ C_2
 6     A_1 = A_1 ∪ UM_INDEX_2(c)
 7  for all c ∈ C_1
 8     A_2 = A_2 ∪ APPROX_UM_INDEX_1(c)
 9  X = maxima in A_1
10  Y = maxima in A_2
11  Σ_Tubular = MAPPING_VIA_STABLE_MANIFOLD(A_1)
12  Σ_Flat = MAPPING_VIA_STABLE_MANIFOLD(A_2)
13  return Σ_Tubular and Σ_Flat.
```

Figure 4.45: Pseudo-code of the feature annotation algorithm.

freely available.

Even in CGAL-framework, we sometimes face the degenerate case of five or more points being cospherical. This case has to be handled with special care because only one Voronoi vertex is repeated and therefore the flow along the Voronoi edges is not well-defined anymore. To deal with such situations, we modify the algorithm slightly. At the start of the algorithm we collect the sets of tetrahedra which are cospherical. While computing the unstable manifold of index 2 saddle points, if the polyline hits a Voronoi vertex whose dual is a member of one such cospherical cluster, the algorithm automatically advances through the non-degenerate Voronoi edges which are dual to the triangles bounding the cospherical lump. This degeneracy poses a more serious threat to the computation of unstable manifold of index 1 saddle points and at this stage, we do not extend the manifold through any Voronoi edge whose dual Delaunay triangle is shared by two cospherical tetrahedra.

There are some parameters involved in the full feature annotation process. For surface reconstruction and medial axis approximation we used the software [70]. The parameters for these routines are described in [79], [82]. For noisy inputs we replace TIGHT COCONE by ROBUST COCOCNE and the parameters for that step are again described in [80]. The rest of the algorithm requires only one parameter k which is the number of flat regions to be output.

Performance

Figure 4.46 shows the performance of the annotation algorithm on six datasets. The datasets have been chosen to represent different domains this algorithm can possibly be applied in. PIN is a CAD dataset which has two tubular parts joined in the middle through a flat portion. The algorithm can identify them correctly. Similarly the method can correctly identify the handle as the tubular and the body as the flat region for the MUG dataset. In the second row we show the performance of our method on two protein molecules obtained from Protein Data Bank [39]. We took the crystal structure of these two molecules (PDB ID 1CID and 1IRK) and blurred them with Gaussian kernel. We further took a level set which represents a molecular surface and used the vertex set of that isosurface as the input to our algorithm. The flat features identified by our method correspond to the β-sheets of the secondary structure of those two proteins. In the last row we show the result on two free form objects containing both flat and tubular features. As we can see, the palm of the HAND has been detected as flat whereas the fingers have been detected as tubular. Our method can also capture the major flat and tubular features of ALIEN.

We purposefully show the performance of the algorithm on ALIEN as it brings forth the limitations of our algorithm. We see that a portion of the arm has been identified as flat. This is because the initial reconstruction phase could not separate the beginning of the arm from the torso due to lack of sampling. Secondly, one of the feet could not be fully identified as flat by our algorithm. This is because the approximate medial axis, that we started with, is not a close approximation of the true medial axis in that region, again due to lack of sampling. Because of that, our method fails to collect sufficiently many index 1 saddle points leading to incomplete identification of flat features in that region.

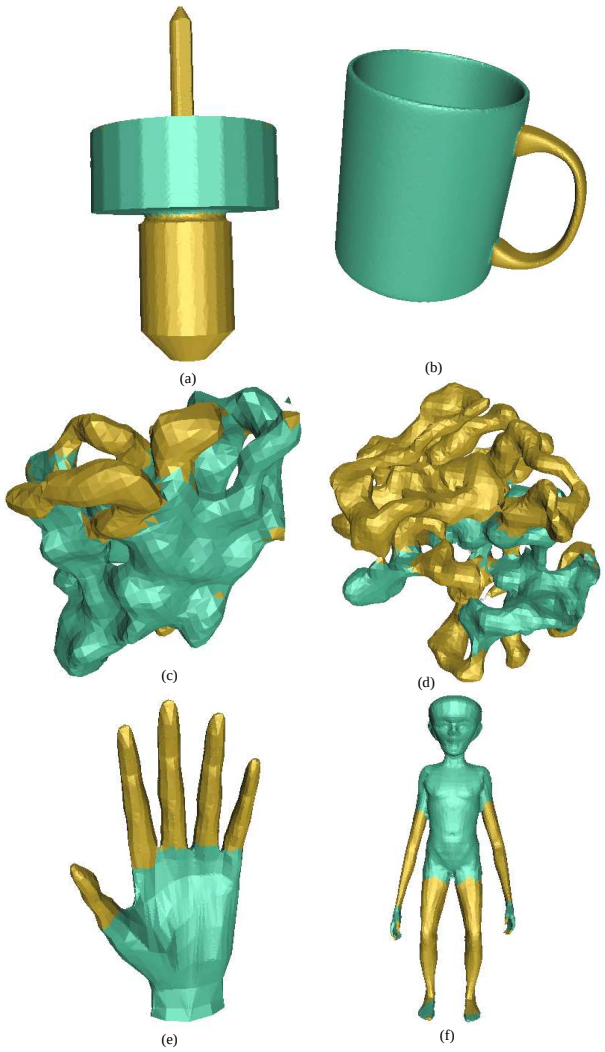


Figure 4.46: Performance of the feature annotation algorithm. The models are (a) PIN, (b) MUG, (c) molecule 1CID (d) molecule 1IRK, (e) HAND, (f) ALIEN

Figure 4.47 shows the performance of our method on noisy dataset HORSE. Instead of applying TIGHT COCONE, we first mark the interior and exterior of the closed surface from its noisy point sample by ROBUST COCONE ([80]) and then obtain the interior medial axis and proceed further with the unstable manifold computation and feature identification. Originally there were some thin flat regions due to the unstable manifold of some index 1 saddle points near the hind legs which we filtered out by thresholding in order to get a clean skeleton of the HORSE. In the rightmost picture we see some white triangles near the ears. These triangles appear as the mapping via stable manifold misses some points on the surface in that portion.

Timings

The time and space complexity of the algorithm is dominated by the complexity of Delaunay triangulation. We report the timings of the entire execution into four major steps

[A] Step 1: Building the Voronoi-Delaunay diagram of the point set (Line 1 of Figure 4.45).

[B] Step 2: Computation of interior medial axis. (Line 2 of Figure 4.45).

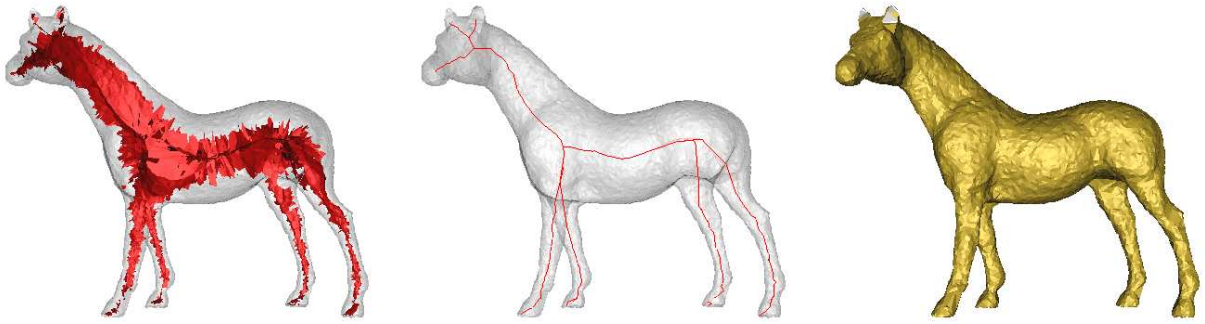


Figure 4.47: Results on Noisy Data.

[C] Step 3: Computation of unstable manifold of index 1 and index 2 saddle points lying on the interior medial axis. (Line 3-8 of Figure 4.45).

[D] Step 4: Mapping the maxima in the planar and linear portion of the medial axis back to the surface. (Line 9-13 of Figure 4.45).

We built the code using CGAL [72] and gnu C++ libraries. The code is compiled at an optimization level $-O2$. We run the experiments in a machine with INTEL XEON processor with 1GB RAM running at 1GHz cpuspeed. Table 4.2 reports the time taken in the four steps of the algorithm for a number of datasets. It is clear from the breakup of timing that the first two steps of building the Delaunay triangulation and then computing the interior medial axis are the two most expensive steps. For noisy datasets, additionally ROBUST COCONE is used to obtain an initial in-out marking. This step is comparatively inexpensive. For example, for NOISY HORSE (48,000 points) this step only adds 10 sec to the whole computation time which is approximately 100 sec.

object	# points	Step 1 (sec.)	Step 2 (sec.)	Step 3 (sec.)	Step 4 (sec.)
1CID	5170	7.59	15.63	6.69	0.39
1IRK	13940	29.88	43.93	15.63	1
HEADLESS MAN	16287	18.63	51.30	16.01	1.26
PIN	15530	15.73	41.4	21.53	0.92
CLUB	16864	20.54	47.3	19.83	1.24
MUG	27109	37.68	83.28	47.14	2.19
HAND	40573	53.48	120.16	40.67	2.69
P8	48046	33.46	136.59	39.97	3.22
1BVP	53392	148.18	159.52	62.19	3.53
ALIEN	78053	102.62	242.33	64.11	5.4

Table 4.2: Timings

In this work, we first described the structure of the unstable manifold of index 1 and index 2 saddle points of the distance function induced by a set of points sampled from a surface. We further used this analysis to compute the unstable manifold of an index 2 saddle point exactly and the unstable manifold of an index 1 saddle point approximately. We then used the unstable manifold of index 1 and index 2 saddle points near the medial axis of the surface to automatically detect the flat and tubular features of the shape.

We believe that this work will be useful in many areas of science and engineering. One natural connection to structural biology is the elucidation of secondary structural properties of protein molecules. Secondary structure of a protein molecule is made up of a collection of α-helices and β-sheets. α-helices are tubular and β-sheets are flat. The results of our algorithm on protein molecules have been verified against the true structural information obtained from Protein Data Bank [39] and the existing literatures in structural biology. We have seen that our method can identify the secondary structural motifs correctly. Often the applications in structural biology require to elucidate the secondary structural information in the absence of atomic level representation of a protein molecule. This is particularly the case when a protein molecule is present in a larger assembly such as in virus capsid and the input is obtained only at a resolution coarser than 4 angstrom via electron microscopy. This method will prove fruitful in analyzing the secondary structural properties in such situations.

This work has triggered several questions. We have collected the initial set of index 1 and index 2 saddle points only from the interior medial axis computed by [82]. This is only an approximation of the true medial axis. As a result, the collection thus obtained often misses some critical points which are close to the true medial axis but do not lie on the approximation. One needs to devise an algorithm to collect all the critical points near the true medial axis efficiently, say by the critical point separation algorithm of Dey et al. [78]. Most likely this will improve the performance of our algorithm.

The algorithm sometimes fails to collect a flat region completely as can be seen from the feet of ALIEN in Figure 4.46. This is partly due to the fact that we map the linear and planar regions of the medial axis via the stable manifold of the maxima lying in those regions. We apply the approximation algorithm of [77] to compute the stable manifolds of the maxima. This method computes these stable manifolds approximately as subcomplexes of Del$P$. To improve the performance of our algorithm, while still maintaining efficiency, we plan to investigate how we can use the exact computation presented in [111] only for those local maxima that border a one- and a two-dimensional region in the medial axis.

The following algorithm is taken from [16] (Bajaj2009Secondary).

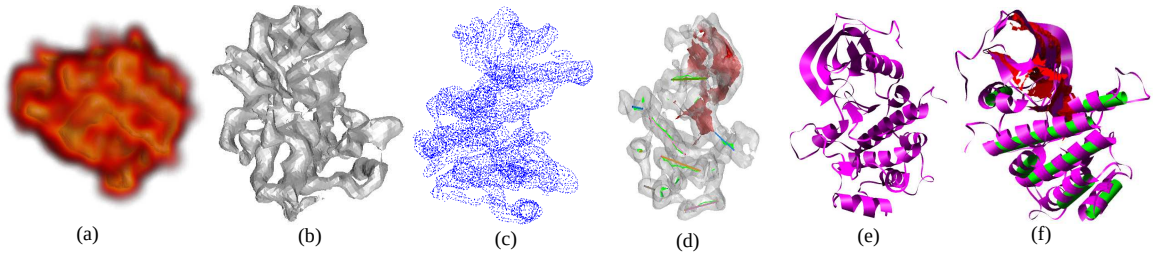


Figure 4.48: Secondary structure elucidation algorithm for Insulin Receptor Tyrosine Kinase Domain with pdbid: 1IRK. (a) volume rendering of its blurred 3D Map at 8 Å resolution. (b) surface rendering of the protein's molecular surface. (c) pointset sampling of the molecular surface. (d) The red patch inside the transparent surface depicts the β-sheet while the straight lines designate the axes of the cylinders which correspond to the α-helices. (e) The secondary structural motifs, documented in the Protein Data Bank, where the helices are shown as ribbon coils and the sheets are sets of ribbon strands. (f) combined display of (d) and (e).

Three dimensional Electron Microscopy reconstruction (3D EM) and in particular single particle cryo-EM reconstruction [94], has advanced rapidly over recent years, such that several macromolecules (complexes of proteins and ribo-nucleic acids or RNA) can be resolved routinely at low resolution (10-20 Å) and in some cases at sub-nanometer (intermediate) resolution (7-10 Å) [37]. The ultra-structure of these complexes once elucidated from the 3D EM (henceforth 3D Maps), provide not only insights into individual protein and RNA folds and structural motifs, but even more importantly provide information about how the various structural components interact. In addition, with the increasing capability of determining multiple secondary structural folds and conformers of a complex [278], there is the promise of studying the dynamics of such interacting systems.

Proteins are polypeptide chains of amino acids, (and nucleic acids are sugar-phosphate chains of acidic bases). The secondary structure of proteins are made up of a set of helical (or crudely, cylindrical) arrangement of sub-chains called α-helices and mostly planar arrangement of sub-chains called β-sheets. Relatively similar configuration of secondary structures also arise in RNA [197].

In this paper, we provide a solution to the problem of automatically elucidating the secondary structural folds of proteins (and nucleic acids) from 3D Maps of macromolecular complexes. A 3D Map is akin to a 3D spatial matrix of electron density values. With the improved reconstruction resolution of 3D Maps of macromolecules via 3D EM, the secondary structural elements of proteins and RNA can be fully elucidated. Often (as evidenced by structures in the PDB), the atomic resolution structures of individual proteins or RNA, that make up the macromolecule are also discerned via X-ray diffraction and/or Nuclear Magnetic Resonance techniques, allowing us to validate our 3D EM secondary/tertiary structure elucidation algorithms [31]. Depending on the resolution of the 3D Map (in Å), our goal here is to detect the secondary structural motifs using the information encoded in the distance function induced by a suitable molecular surface extracted from the density map.

Starting with an electron density map segmented from a macromolecular assembly, the processing pipeline consists of the following steps.

[A] Segmentation of the density map of a protein from the overall map of the macromolecular assembly. The description of this step is skipped in this paper as we use the implementation of the algorithm developed by Yu and Bajaj [260] in the publicly available VolRover tool [6].

[B] Extraction of a suitable molecular surface from the density map of the protein molecule under investigation and computation of the medial axis. (Section 4.4),

[C] Computation of *distance function* and its critical points.

[D] Computation of *unstable manifold* of the critical points.

[E] Selection of Secondary Structural Motifs (α-helices and β-sheets) from the unstable manifolds.

Before we go into the computational details, we survey the known techniques to detect secondary structural motifs from the density map.

Prior Approaches to Detect Secondary Structure from Density Map There are relatively few published algorithms that detect α-helices and β-sheets of proteins from 3D Maps at coarse resolution (> 6 Å). Wen *et al.*have devised an algorithm called *Helix Hunter* [239] for detecting α-helices in a low resolution map where they modeled the helices as cylinders and the density function of the cylinder is convolved with the original map to detect the peaks of the cross-correlation. The main disadvantage of this technique is that it searches exhaustively over the space of all rigid-body transformations and is therefore very slow.

A similar approach for β-sheet detection was adopted by Kong and Ma [137, 138] who modeled a β-sheet as a disk like primitive and searched through the input map to find the possible positions of the disk that matched best with the electron density. This work also, due to its exhaustive search paradigm, is extremely compute intensive.

Recently, Yu and Bajaj [261] have developed an algorithm for detection of both helices and sheets based on the features present in the electron density map. Their algorithm classifies a voxel into planar or cylindrical region based on the relative magnitudes of the eigenvalues of the local structure tensor. Further the cylindrical regions are connected to form the axes of the helices while the planar voxels are patched to form the sheets.

Our approach is distinct because it is the first boundary-based approach toward the elucidation of the secondary structure of proteins from EM maps. This also exploits the natural connection between the topological structures induced by the molecular surface with its bio-chemical features. The key ingredient of the algorithm presented in this paper is the concept of "distance" function. Variants of the topological structures of this function have been used earlier for modeling three dimensional objects. In computational geometry, distance function has been used to reconstruct piecewise triangulated surface from discrete point sample [56, 86, 111] and to compute the stable medial axis [58], [215]. While a surface approximation and its medial axis serve as crude shape descriptors, deeper insights about the feature space of the shape can also be obtained using the critical points of the function as were shown in the context of shape segmentation and annotation of point sampled three dimensional geometry in [77, 114].

In this work, we exploit the critical point structure of the distance function and show that the unstable manifolds of a subset of them have a strong connection with the structural folds of the corresponding protein molecule.

In this section, we review some concepts related to the distance function which is the key computational tool used in this paper. We assume familiarity with the Voronoi and Delaunay diagrams of a set of points in $\mathbb{R}^3$ which can otherwise be found in [74].

Critical Points and their Stable/Unstable Manifolds Given a compact surface Σ smoothly embedded in $\mathbb{R}^3$, a distance function h_Σ can be constructed that assigns to each point its distance to Σ. In applications, Σ is often known via a finite set of points P sampling Σ and in such cases h_Σ can be approximated with h_P that assigns the distance to the closest sample point in P.

The critical points of h_P are the points which lie inside the convex hull of their closest points from P. This also means that the critical points can be detected by the intersection of the Voronoi objects with their dual Delaunay objects [216]. In particular, the *maxima* are the Voronoi vertices which lie inside their dual Delaunay tetrahedra; the *minima* are the sample points themselves as they lie inside their Voronoi cells, the *index 2* saddle points are the intersections of the Delaunay triangles with dual Voronoi edges, and *index 1* saddle points are the intersections of the Voronoi faces with their dual Delaunay edges. Note, index of a critical point matches the dimension of the Delaunay simplex that contains the critical point. Figure 4.49 (a) shows the four possible configurations of Voronoi and dual Delaunay elements that result four types of critical points.

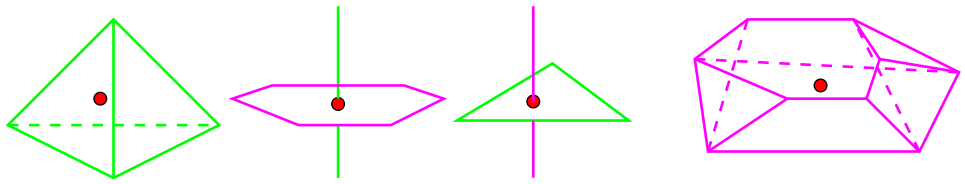


Figure 4.49: The possible cases of Voronoi-Delaunay duality that results the generation of four types of critical points of h_P - (from left to right) maxima, index-2 saddles, index-1 saddles and minima. The (non-zero dimensional) Delaunay objects are drawn in green and the Voronoi objects are drawn in magenta.

At every $x \in \mathbb{R}^3$, a unit vector can be assigned that is oriented in the direction of the steepest ascent of h_P. The critical points are assigned zero vectors. This vector field, which may not be continuous, nevertheless induces a *flow* in $\mathbb{R}^3$. This flow tells how a point x moves in $\mathbb{R}^3$ along the steepest ascent of h_P and the corresponding path is called the *orbit* of x. Analogously, one can also think of *inverted orbit* as the path a point traces when it follows the negative gradient. For a critical point c its *stable manifold* is the set of points whose orbits end at c and *unstable manifold* is the set of points whose inverted orbits end at c. The unstable manifold of a minimum is a three dimensional polytope whose boundary is composed of the unstable manifolds of the index 1 saddle points which in turn are bounded by the unstable manifolds of index 2 saddle points and maxima. The induced *flow* of the function h_P is the basis that helps compute the unstable manifolds of its critical points. Crudely the idea is as follows. To compute the unstable manifold of any particular critical point c, we investigate its local neighborhood and see how a *free* (or regular) point x will move so that the function h_P increases most along its trajectory. Note, the critical points themselves do not move because they are assigned zero flow vectors. Clearly, such a method will not lead to an algorithm as the domain is continuous and therefore this approach is not computationally feasible. However as we show in this subsection, a great deal of information about the behavior of a *free* point is encoded in the Voronoi-Delaunay diagram of the set of points P which originally induced the distance function h_P. In fact, as we shall see in the next subsection, simple combinatorial tests on the Voronoi-Delaunay duality helps compute the flow of any arbitrary regular point and additionally gives a handle to compute the unstable manifolds in a discrete setting.

Induced Flow via Voronoi-Delaunay Duality For the rest of this subsection, we assume that x is a regular (free) point which is allowed to move only in the direction of the gradient of the distance function. Depending on which Voronoi element x is in, we give a canonical description of its movement, restricted to that Voronoi element.

Voronoi Cell

For any point x in the Voronoi cell V_p, the function h_P increases most radially along the vector $\vec{px}$. Figure 4.50 (a) shows an illustration.

Voronoi Facet

Let F be a Voronoi facet at the intersection of V_p and V_q. The dual Delaunay edge is pq. For a point $x \in F$, the flow is along the vector $\vec{dx}$ where d is the midpoint of pq. Note, by the property of Voronoi-Delaunay duality relations, the point d is the closest point to F. Figure 4.50 (b) shows an illustration. When pq intersects F, d is actually an index-1 saddle point.

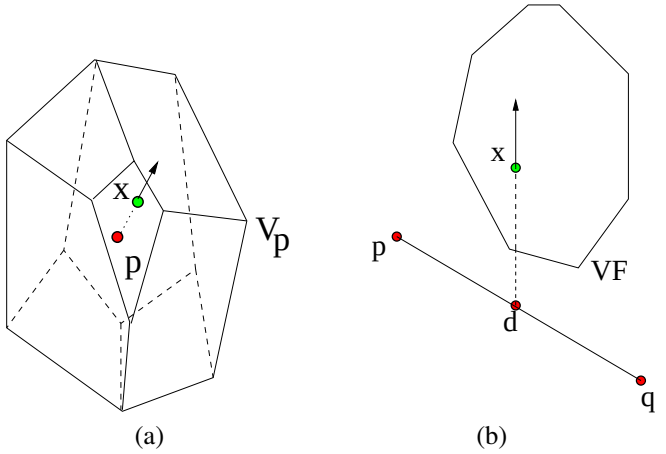


Figure 4.50: Flow path of a point x. (a) When x is inside V_p. (b) When x is inside a Voronoi face at the intersection of V_p and V_q.

Voronoi Edge

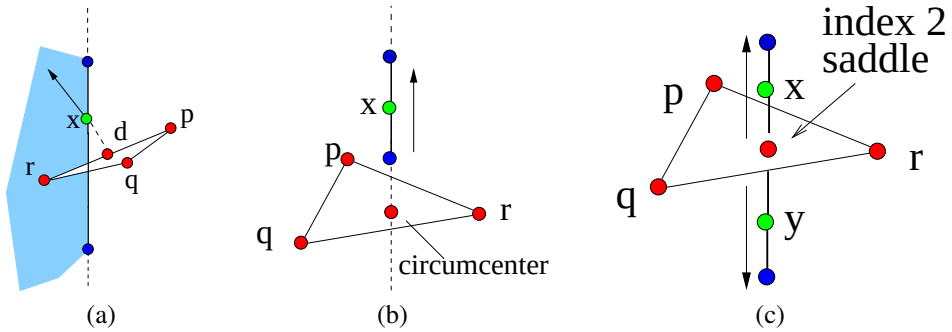


Figure 4.51: Flow path of a point x on a Voronoi edge. (a) The circumcenter of the dual triangle falls outside the triangle. (b) The circumcenter falls inside but the Voronoi edge segment is above the plane of the triangle. (c) The Voronoi edge segment pierces the plane of the triangle through the circumcenter. This results an index-2 saddle point.

Let E be a Voronoi edge at the intersection of V_p, V_q, V_r. The dual Delaunay triangle is denoted as $\triangle_{p,q,r}$. Note by the property of Voronoi-Delaunay duality, the supporting line of E intersects the plane containing p, q, r at the circumcenter of $\triangle_{p,q,r}$. Now consider two cases.

- $\triangle_{p,q,r}$ is obtuse-angled Let the angle at q be obtuse as seen in Figure 4.51 (a). In that case, a point on E shall enter the Voronoi facet dual to the longest side of $\triangle_{p,q,r}$ which, in this case, is pr. The course of the path from then on is guided by the rule for Voronoi facets as we described earlier. We give a special name to such Voronoi facets and call them *acceptor*s.
- $\triangle_{p,q,r}$ is acute-angled This case is straight-forward as any point x on E flows along the Voronoi edge itself.

Therefore for any point $x \in E$, one first decides if the dual $\triangle_{p,q,r}$ is acute or obtuse angled and decide if x should move in the direction of the Voronoi edge or if it should enter a Voronoi facet. If it does enter a Voronoi facet (named *acceptor*), the above rule for Voronoi facet shall dictate its course henceforth.

Voronoi Vertex

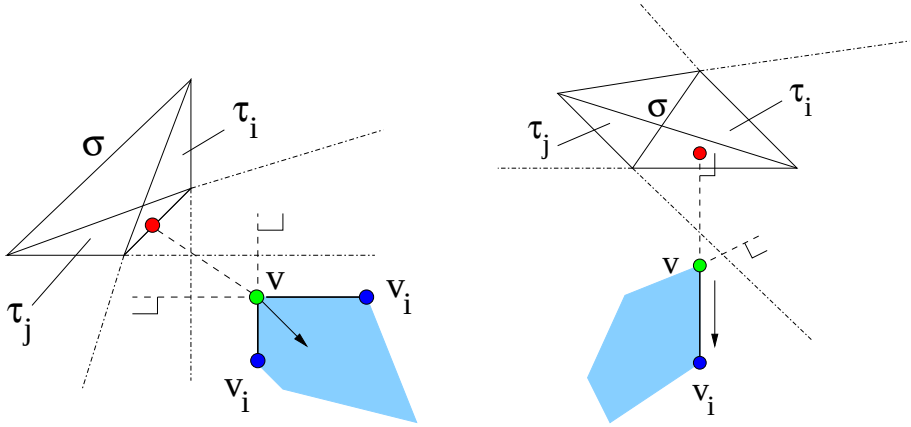


Figure 4.52: Flow path of a point x on a Voronoi edge. (a) The circumcenter of the dual triangle falls outside. (b) The circumcenter falls inside but the Voronoi edge segment is above the plane of the triangle. (c) The Voronoi edge segment pierces the plane of the triangle through the circumcenter. This results an index-2 saddle point.

The case of Voronoi vertex is the most intriguing. Let the point x be sitting on a Voronoi vertex v which is at the intersection of four Voronoi cells V_{p_1,p_2,p_3,p_4}. The dual Delaunay simplex is the tetrahedron σ spanning the four points. Now suppose, v is outside σ. Note, otherwise v is a maximum and has no *flow*.

Let τ_i be the triangle in σ opposite to p_i. If x and p_i are not in the same half-space of the supporting plane τ_i, we say τ_i *blocks* x (from seeing p_i). By the property of Delaunay triangulation, the number of *blockers* can be at most 2. So let us consider the following two cases.

- τ_i *blocks* x: This case can be derived from previous cases. Let E be the Voronoi edge dual to τ_i. The flow at x (coincident with v) is guided by the flow conditions of E (described earlier).
- τ_i and τ_j *block* x: There are two cases to consider. Both are illustrated in Figure 4.52.

  In case (a), both the perpendiculars from v to the supporting planes of $\tau_{i,j}$ fall outside the triangles. In this situation, the flow at v enters the Voronoi facet dual to the common Delaunay edge shared by τ_i and τ_j.

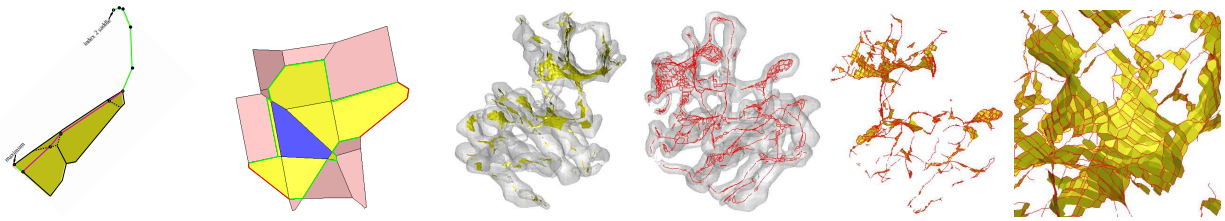
  In case (b), one of the perpendiculars fall inside. Without loss of generality, we take that facet to be τ_i and in that case, the flow is along the Voronoi edge dual to τ_i.

By the observations above, we know if a regular point x hits a Voronoi vertex v along its flow-path, it either stops there when v is a maximum, or there are simple combinatorial tests that decide the next course of its path.

Unstable Manifolds of Index-2 Saddle Points Once the flow-path of a regular point is fully characterized, we are in a position to build the unstable manifold of an index-2 saddle point. Note, an index-2 saddle point, say c, is at the intersection of a Voronoi edge E and its dual Delaunay triangle τ. Let the two tetrahedra sharing τ be σ_1, σ_2. The edge E has the endpoints at the dual Voronoi vertices of σ_1 and σ_2, denoted as v_1, v_2 respectively (two blue circles in Figure 4.51 (c)). The unstable manifold $U(c)$ of c, has two intervals - one from c to v_1 and the other from c to v_2. We look at the structure of one of them, say the one from c to v_1, and the other one is exactly similar.

At any point x on the subsegment cv_1, the flow is toward v_1 from c. Once the flow reaches v_1, the subsequent flow depends on the flow structure of v_1 as described in the previous subsection. Instead of just looking at v_1, we consider a generic step, where the flow reaches at a Voronoi vertex v and we list the possible situations. If v is a local maximum, the flow stops there and marks the endpoint of U_c. Otherwise depending on the flow structure of v, one can combinatorially decide if the flow line follows along a particular Voronoi edge, incident on v, or if it enters a Voronoi facet. If the flow is along a Voronoi edge, it eventually hits another Voronoi vertex and we apply the same tests. If it enters a Voronoi facet, again by the flow structure, we know the direction of the flow path and it is easy to identify the exit point where it hits the boundary Voronoi edge of the

Voronoi facet. The flow structure of the boundary Voronoi edge then decides the next course of the path. Figure 4.53 (a) shows an example U_c for an index 2 saddle point c.



(a) (b) (c)

Figure 4.53: (a) Unstable manifold $U(c)$ of an index 2 saddle point c. c is drawn with a cyan circle. The portion of $U(c)$ which is a collection of Voronoi edges is drawn in green with intermediate Voronoi vertices drawn in blue. The pink circle is a Voronoi vertex on $U(c)$ where the flow enters a Voronoi facet. The portion of $U(c)$ which lies inside the Voronoi facets is drawn in magenta. The transversal Voronoi edges intersected by this portion of $U(c)$ are dashed. $U(c)$ ends at a local maximum which is drawn in red. (b) Snapshot of computation of unstable manifold of an index 1 saddle c. The text describes the algorithm. (c) A collage of U_1 (yellow) and U_2 (red) of the molecule 1IRK. Note U_2 bounds U_1.

Unstable Manifolds of Index-1 Saddle Points The computation of the unstable manifold of an index-1 saddle point is done in a similar fashion as index-2 saddle point. Unstable Manifold of index 1 saddle points are two dimensional. Due to hierarchical structure, they are bounded by the unstable manifold of index 2 saddle points.

Let us consider an index 1 saddle point, c. This point lies at the intersection of a Voronoi facet F and a Delaunay edge. For any point $x \in F \setminus c$, the flow is radially outward in the direction of $\vec{cx}$. Clearly, F is in $U(c)$. As the flow hits the Voronoi edges bounding F, we examine the flow-structure of those edges. Again, by the description of the flow structure given in Section 4.4, we know there may be two cases. In the first case, when the flow is along the Voronoi edge, it must be in the boundary of U_c. In the second case, the flow enters one of the neighboring Voronoi facets and we recursively process that face in a similar fashion.

Figure 4.53(b) illustrates an intermediate stage of this computation. The index 1 saddle point c is contained in the blue Voronoi facet. The yellow Voronoi facets are already in $U(c)$. The red edges designate the static boundary as they are non-transversal and the green edges designate the active boundary through which the pink facets are included in $U(c)$ in the later stage of the algorithm.

Now we have all the computational gadgets in hand to proceed to the secondary structure elucidation algorithm. Once the density map of the protein molecule is segmented out from that of the macromolecular assembly, the process is as follows -

[A] Extraction of Molecular Surface and Medial Axis (Section 4.4),

[B] Computation of Unstable Manifolds

[C] Detection of α-helices and β-sheets (Section 4.4),

Molecular Surface Extraction The input to our algorithm is an electron density map, possibly segmented out from a larger macro-molecular assembly, like viruses using any standard algorithm, for example [261]. Starting with the density map, we first need to extract a molecular surface suitable for subsequent operations. Commonly, such operation is known as "contouring" or "isosurface" extraction.

Given a volumetric data, an isocontour is a two-dimensional surface on which all points have same density value. Marching Cubes algorithm is the most popular way to extract isosurfaces from scalar volume [154]. But the size of the triangles in marching cube is fairly uniform all over the extracted surface which poses a problem for the Voronoi-Delaunay based algorithm for unstable manifold computation that we sketched earlier.

One way to circumvent the problem is to make the triangulation of any extracted molecular surface Delaunay-conforming, as was done in [46], [60], [115]. However these algorithms are conservative and typically add many extra vertices. In this work, we therefore used Marching Tetrahedra - a version of Marching Cubes - algorithm which produces reasonably dense, uniform triangulation of the molecular surface for us to carry out the subsequent Voronoi based computations. In this method, we first

decompose every voxel in the volume data into a set of tetrahedra and compute the triangulation of the surface intersected by each tetrahedron. Once a suitable molecular surface is extracted, we compute its medial axis [82]. The medial axis is the starting point on which we compute the unstable manifolds of the index 1 and 2 saddle points of the distance function induced by the extracted molecular surface.

Detection of α-helices and β-sheets The unstable manifold of index 1 and index 2 saddle points restricted to the medial axis of the molecular surface decomposes the medial axis into linear (U_2) and planar (U_1) portions. The advantage of such an annotation is that the linear sub-part (U_2) gives the axial description of a tubular part while the planar sub-part (U_1) provides the planar axis of the relatively flat portion of the molecular surface. The next task is to select a subset of U_2 that gives the α-helices of the protein. Typically, an α-helix is a cylindrical subvolume of the molecule which is of width approximately 2.5 Å. Also the subvolume does not deviate much from a straight cylinder for proteins [48, 239]. These two conditions, dictate the following computational steps.

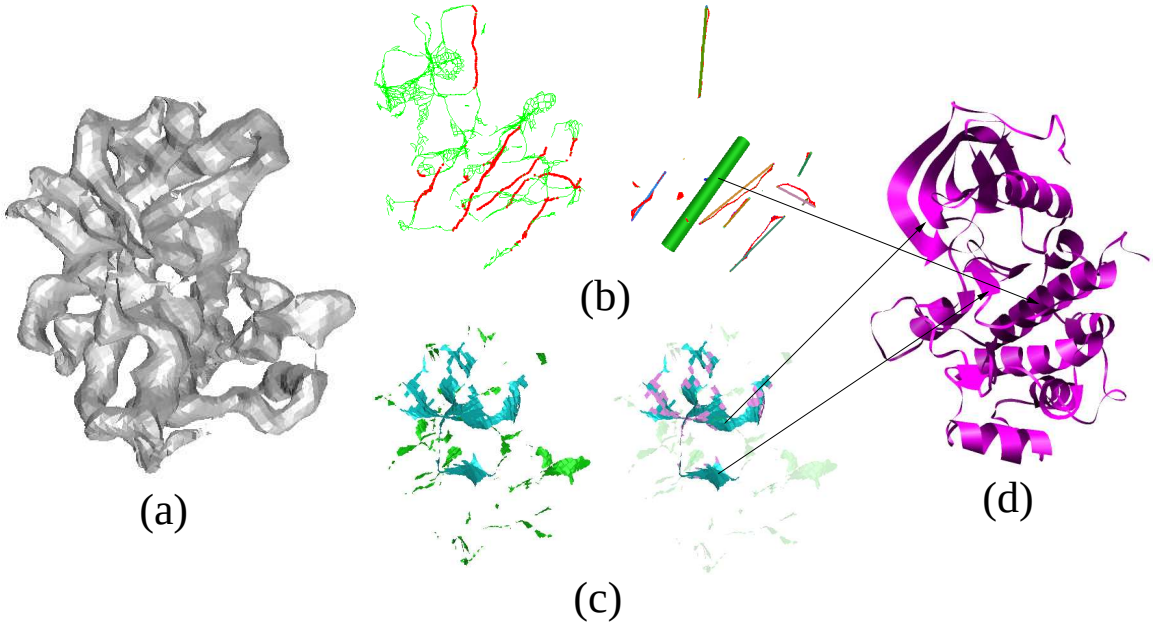


Figure 4.54: (a) The molecular surface of 1IRK. (b) The selected Voronoi vertices on U_2 and the fitted cylinder. (c) Filtering out subsets of U_1 which are small (green) or do not satisfy the width test (magenta). (d) shows the secondary structures obtained from the PDB and its correspondence with the computed structure (b,c).

The unstable manifold of every index 2 saddle on the medial axis is a polyline with Voronoi vertices at the joints. Every Voronoi vertex has h_P value which can be computed by the circumradius of the dual Delaunay tetrahedron. Locally this gives the radius of the cylinder that best fits the molecular surface in the cylindrical regions. We first populate the set H with the Voronoi vertices whose h_P values fall within 2 Å and 3 Å. A 3 Å neighborhood graph is then computed over H that clusters the points. The choice of 3 Å is dictated by the fact that pitch of the helices is 1.5 Å and usually there is more than 2 turns in every helix. The diametrical point pair in every cluster is then computed. The maximum deviation of any intermediate point from the straight line joining the point pair decided how straight the fitted cylinder to the cluster is. This way we select those clusters from H and the cylinders fitted to these clusters produce the detected α helices. The process is shown in 4.54(b).

The selection of β-sheets is similar. U_1 gives the possible candidates for β-sheets. First we notice that there are some tiny components that are created due to sampling artifacts, and they do not correspond to real planar substructures of the molecule. We first filter these small clusters out (green patches in Figure 4.54(c)). After this first stage of filtering, we are left with the planar subsets of the medial axis (cyan in Figure 4.54(c)). At this point we apply the knowledge, that β-sheets are of width roughly 1.5 Å [48, 137] and we filter out the planar patches which do not satisfy this "thickness" criterion. The width of a facet in U_1 is easy to check as they are the Voronoi facets and therefore have an 1-1 correspondence with their dual Delaunay

PDBID	Protein	PDBID	Protein
1IRK	Insulin Receptor Tyrosin Kinase Domain	1TIM	Triose Phosphate Isomerase
1PLQ	Proliferating Cell Nuclear Antigen	1RIE	Rieske Iron-sulphur Protein
1CID	T Cell Surface Glycoprotein CD4	1MBN	Myoglobin
1VDF	Cartilage Oligomeric Matrix Protein	1JPC	Agglutinin
1AOR	Aldehyde Ferrodoxin Oxydoreductase	1BBH	Cytochrome C'

Table 4.3: Name and PDBID of the proteins used in the calibration process.

edges which cross the medial axis. Therefore, we select only those Voronoi facets from U_1 whose dual Delaunay edges are of length between 1 Å and 2 Å. The portions of U_1 which are filtered out by this test are shown in magenta in Figure 4.54(c). The rest of U_1, which qualify for β-sheets, are shown in cyan. The sheets detected are shown to correspond well with the β-sheets recorded in Protein Data Bank.

For calibrating our structure elucidation algorithms, we downloaded atomic level descriptions of proteins from the Protein Data Bank [39]. For each protein, we first "blurred" them into a 3D map (to correspond to a reconstructed 3D EM map) at varying resolutions (5 to 15 Å), using publicly available software EMAN [158]. From these volumetric maps, we extracted the proteins molecular surface using TEXMOL [12]. Next we collected a pointset sampling of the molecular surface, and used the COCONE software for surface reconstruction and medial axis computation [70]. The next step was to detect the critical points and compute the stable/unstable manifolds of a subset of the critical points. These computations rely on the Voronoi Diagram - Delaunay triangulation of the pointset, and were done using the CGAL library [72].

The calibration process is essential before we apply them to 3D EM maps of unknown atomic descriptions. The datasets used in the calibration process are summarized in Table 4.3. Details of the molecules are available from the Protein Data Bank (PDB) via the pdbid. Figure 4.55 shows the snapshots of the key steps of the algorithm. Figure 4.55(a) shows the U_1 (green) and U_2 (red) of 1BBH. The clusters of points lying on U_2, selected by the width criterion of α-helix, and the axis of the cylinders to be fitted to the clusters are shown in Figure 4.55(b). Figure 4.55(c) shows the secondary structure of 1BBH documented in the PDB. It has total 8 helices and all of them are detected correctly (green cylinders) by our algorithm (Figure 4.55(d)). The second row in Figure 4.55 shows similar set of pictures for 1RIE. It has three β-sheets which are identified correctly (red, yellow and blue patches). Third and fourth row show the performance of the algorithm on six more molecules at varying resolutions.

Next we show the results on the cryo-EM density maps provided to us by our collaborators. We have experimented with the segmented capsid proteins of Rice Dwarf Virus. The architecture of the virus ([278]) suggests that it has a double-layer capsid. The outer capsid layer is composed of multiple copies of protein P8 and the inner layer is composed of protein P3. Figure 4.56 shows molecular surface extracted from segmented cryo-EM map (at 6.8 Å) passing through the data processing pipeline. The unstable manifolds of the index 1 and 2 saddle points are computed and subsequently the secondary structural units (helices and sheets) are detected. Figure 4.57 shows the performance of the algorithm on the P3 protein and the structural unit of chaperon GroEL.

4.5 Meshed Models

linear and non-linear

The following algorithm is taken from [116] (Goswami2008Efficient).

A wide variety of science and engineering applications rely on accurate level set triangulation. This is especially true for multiscale models in biology, such as macromolecular structures extracted from reconstructed single particle cryo-EM (Electron Microscopy), cell-processes and cell-organelles extracted from TEM (Tomographic Electron Microscopy), and even trabecular bone models extracted from SR-CT (Synchrotron Radiation Micro-Computed Tomography) imaging. Computational analysis of these models for estimation of nano, micro, or mesoscopic structural properties depends on the mesh representation of the contour components respecting their topological features.

Our goal is to find an algorithm to solve the following problem. The input to the algorithm is a rectilinear sampling of a bounded domain of an unknown scalar function F. The rectilinear grid need not be uniform; it may be adaptive as in the case of an octtree. The user then specifies a local interpolant to generate a level set approximation for any isovalue v; we use Σ to

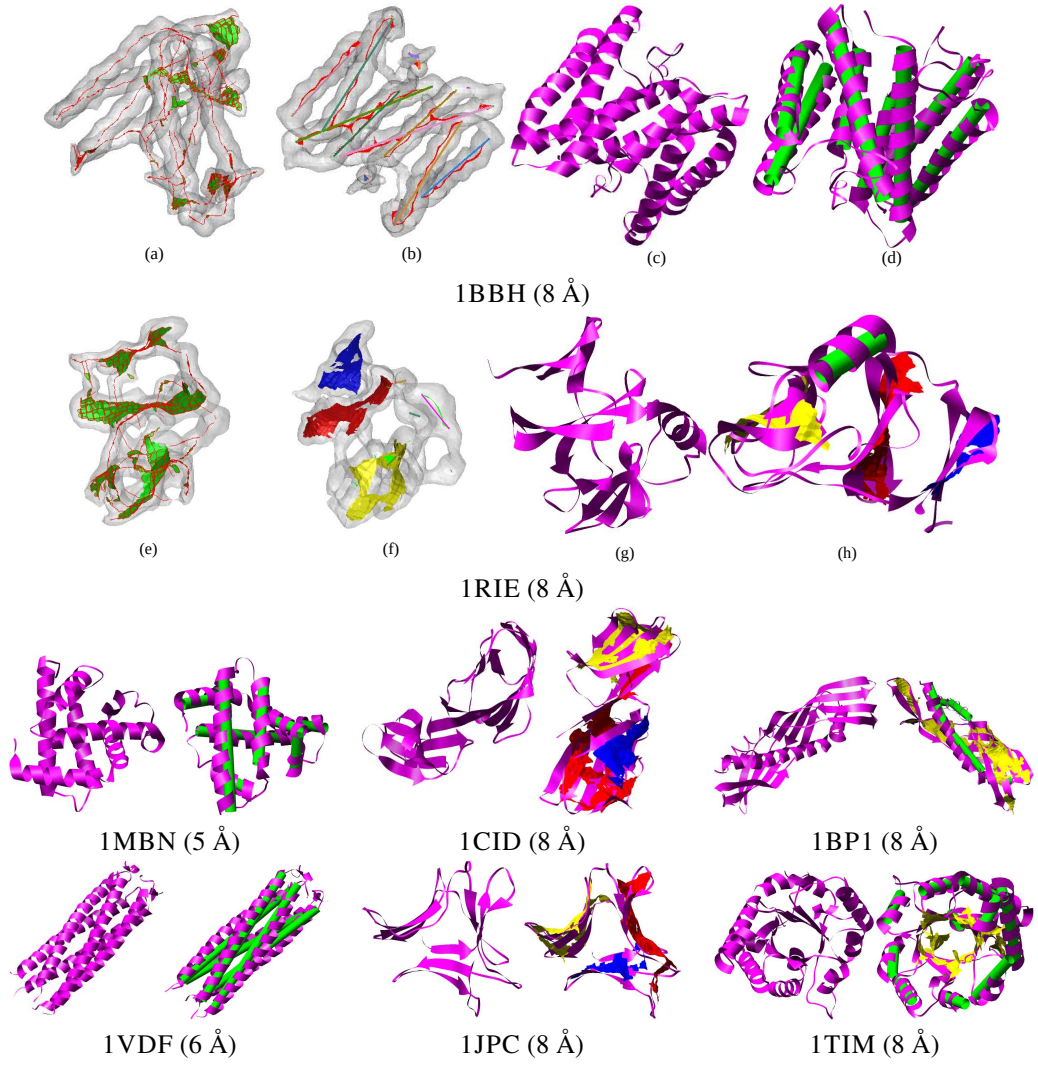


Figure 4.55: Performance of our α-helix and β-sheet detection algorithm.

denote this level set of the interpolating function. Since the function F is unknown, we must assume that the local interpolant produces a good approximation of the function F within each cell of the grid. Our algorithm is general enough to use any local interpolant, however, in our experience, a trilinear interpolant is the most natural choice.

Our goal is construct a mesh in an efficient manner such that the following properties hold:

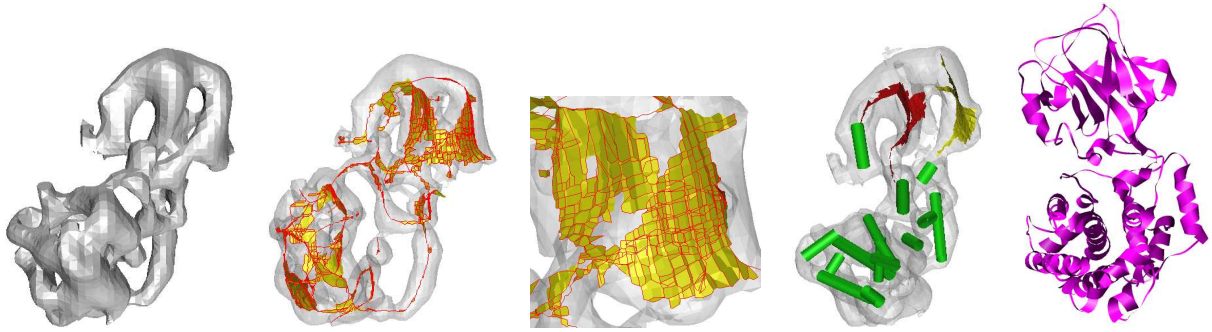
I **Topological Guarantee:** M is homeomorphic to Σ.

II **Geometrical Guarantee:** The Hausdorff distance from M to Σ is within a user-specified bound ε.

III **Delaunay Conformity:** M is a subcomplex of the Delaunay triangulation of the vertex set of M.

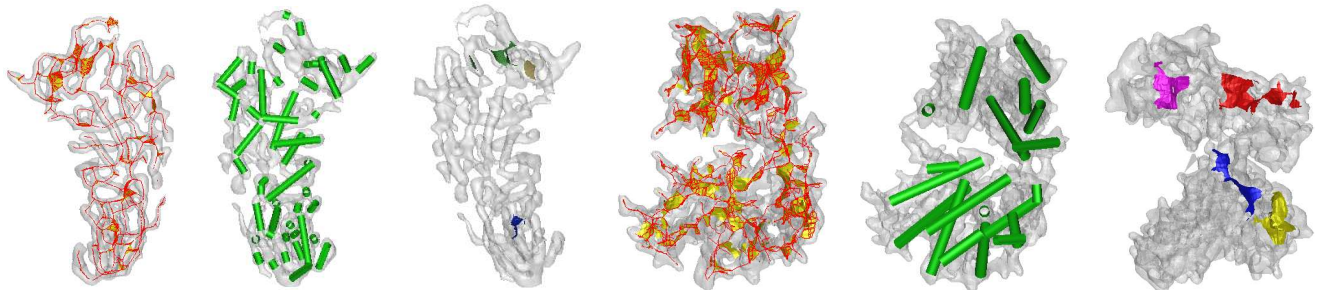
IV **Adaptivity:** The user can decimate part of the volumetric data and still preserve properties I, II, and III.

Once a surface M is generated that is Delaunay conforming, it is possible to improve the mesh quality by applying any Delaunay refinement algorithm. We detail such an algorithm in Section 4.5. Figure 4.58 visually illustrates a toy data set (a),



(a) (b) (c) (d) (e)

Figure 4.56: Different stages of the secondary structure elucidation process on the outer-capsid layer protein P8 of Rice Dwarf Virus. From left to right: the molecular surface, the unstable manifolds, detected helices and sheets, and the ribbon diagram of the secondary structure recorded in Protein Data Bank.



(a) (b) (c) (d) (e) (f)

Figure 4.57: (a) Unstable manifolds inside transparent molecular surface. (b) Detected helices and (c) sheets of inner capsid protein P3 of Rice Dwarf Virus. (d) Unstable manifolds inside transparent molecular surface. (b) Detected helices and (c) sheets of structural unit of GroEL.

the failure of a typical isocontouring method, in this case Marching Cubes, in reconstructing the level set (b), generation of the correct topology by our algorithm (c-d), geometric refinement (e-g), and the final Delaunay-conforming surface sample (h).

At this point, we emphasize the novelty of our approach. Although there exist algorithms which can be applied to solve the problem as stated, such algorithms are devised in a more general setting and thus do not exploit the natural structure of the volumetric data. Our approach has a number of unique advantages over its predecessors. As part of the algorithm, we collect some of the grid points around Σ and use them to build a Delaunay conforming mesh efficiently. Once the surface mesh is created and forced to conform to the Delaunay triangulation of the point set, these grid points can either be removed or used to construct an interior or exterior tetrahedral volume mesh.

Additionally, as a result of noise in the input data or a poor choice of the isovalue v, there may exist topological anomalies in the surface Σ. In mesh processing literature, such anomalies have been referred to as "topological noise" [244, 14]. The term "noise" indicates undesirable features of small geometric size that prevent the mesh from being used for further processing. Methods have been developed to remove such artifacts provided that the point sample of Σ is sufficiently dense near the anomalies; our method guarantees this density. Therefore, the output mesh M can be applied without any further refinement to any point processing algorithm that uses prior Delaunay-based reconstruction of geometry to detect and selectively remove these topological features.

Finally, algorithms involving volumetric data and meshes often become computationally demanding due to the large size of data sets. The adaptivity of the algorithm (property IV) provides one way to ease calculations by down-sampling less important regions of the volumetric data. The Voronoi and Delaunay diagrams of a point set P, denoted $\mathrm{Vor}\,P$ and $\mathrm{Del}\,P$ respectively, play an important role in the computations involved in this paper. Due to page limitations, we do not go into the detail of their

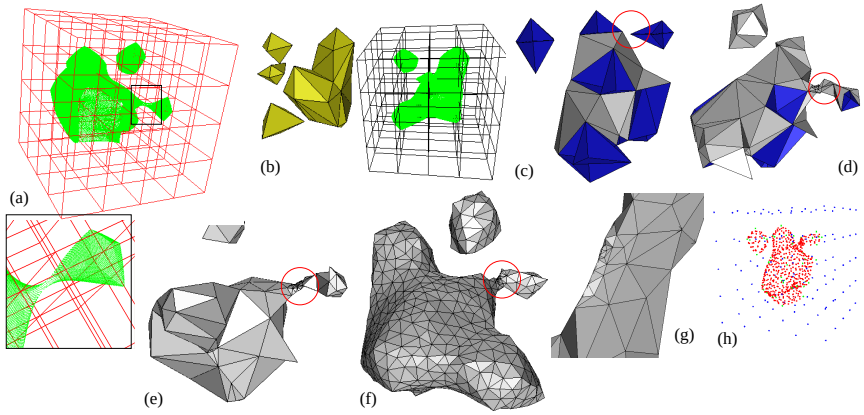


Figure 4.58: Various stages of our algorithm. (a) A rectilinear grid with sample values of an unknown function at the grid points. Within cells, the function is approximated with a trilinear interpolant. For the purpose of visualization only, we collect a set of points (green) on the surface and display them. A narrow region of the surface is magnified below. (b) Another view of the data (right) and the same view of the mesh generated by Marching Cubes [154]. Note that the mesh is disconnected in the thin region. (c) The mesh generated by the restricted Delaunay triangulation of only edge and grid points. Blue facets have a grid point as at least one of their vertices. This point set is still not sufficient to produce a Delaunay-conforming mesh. (d) The mesh generated by addition of sample points of Σ. The topology is now recovered (Property I). (e,f) Geometrical refinement for progressively smaller value of ε. (g) Even in the magnified portion of the thin region, the triangles approximate the geometry nicely (Property II). (h) All the points involved in construction of the mesh including grid points (blue), edge points (green), and new points added by the algorithm (red). Observe that in order to recover the topology and reduce the geometric error in the approximation, many surface sample points are added to the point set.

construction and refer the reader to any standard computational geometry textbook, e.g. [74].

Given a point set P chosen from the same space in which Σ is embedded, the *restricted Delaunay triangulation of* P, denoted $\mathrm{Del}\,P|_\Sigma$, is defined to be the set of Delaunay objects of $\mathrm{Del}\,P$ whose dual Voronoi objects have non-zero intersection with Σ. If P is chosen in a way that respects the structure and local feature size of Σ, $\mathrm{Del}\,P|_\Sigma$ will be a mesh with the desired properties.

Edelsbrunner and Shah gave a sufficient criterion called the *closed ball property* [87], sometimes referred to as the *topological ball property*, for $\mathrm{Del}\,P|_\Sigma$ to be homeomorphic to Σ. A Voronoi object V of dimension k satisfies the closed ball property if $V \cap \Sigma = \emptyset$ or $V \cap \Sigma$ is homeomorphic to a closed ball of dimension $k-1$. Accordingly, a point set P is said to satisfy the closed ball property if every Voronoi object of $\mathrm{Vor}\,P$ satisfies the closed ball property. Using the notion of *transversality* as defined in [118], their criterion can now be stated precisely.

Theorem 1 *[87] If* Σ *intersects each Voronoi object of* $\mathrm{Vor}\,P$ *transversally and* $\mathrm{Vor}\,P$ *satisfies the closed ball property, then* $\mathrm{Del}\,P|_\Sigma$ *is homeomorphic to* Σ.

We will show that our algorithm produces a mesh satisfying the closed ball property. This ensures that the mesh is Delaunay conforming (property III) and, by the theorem, that the mesh is homeomorphic to Σ (property I). In this section, we describe the algorithm, analyze its efficiency and present some simplifying results for a specific choice of local interpolant. Figure 4.59 shows an overview of the process in two dimensions.

Algorithm Description

Our algorithm is motivated primarily by the work of Cheng et al. in [60] who build a Delaunay conforming approximation of the level set of any general implicit function. Since our problem is focused on locally interpolated functions, we take advantage of the natural scaffolding of the input grid to substantially improve the computational efficiency of the algorithm. Moreover, we also show that once the Delaunay conforming surface mesh is extracted, it is quite straightforward to build a tetrahedral volumetric mesh of good quality. We call the algorithm for extracting a surface mesh DELSURFMESH and the extension to build a Delaunay tetrahedral interior or exterior mesh DELVOLMESH.

In [60], the authors start with a small sample of points lying on the surface to be meshed. They keep refining the mesh until its vertex set satisfies the closed ball property, thereby providing a Delaunay conforming mesh homeomorphic to Σ. To ensure

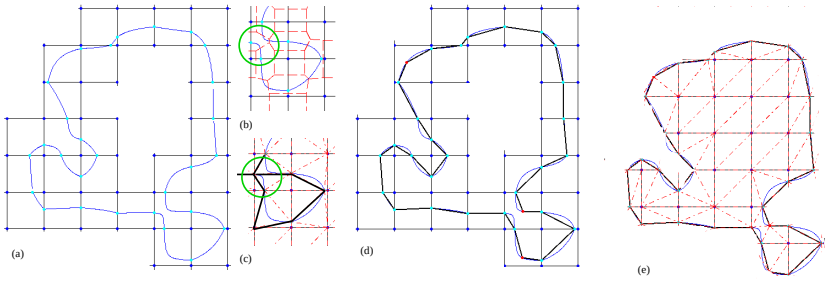


Figure 4.59: A 2D example using bilinear interpolation of Σ demonstrating the importance of the closed ball property. (a) Grid points (dark blue) and edge points (light blue) relevant to our algorithm are shown. Note that in the 3D case we form a layer of grid points twice as thick. (b) A portion of the Voronoi diagram is shown in red and a location where the closed ball property is violated is circled. (c) Since the closed ball property is violated, the restricted Delaunay diagram (black) has incorrect topology in the circled region. (d) Red points are inserted where the closed ball property is violated and the restricted Delaunay graph is formed (black). (e) By including those grid points interior to Σ, we efficiently produce an interior mesh that does not alter the Delaunay conforming surface mesh.

that a point set P satisfies the closed ball property, it is necessary to check the intersection of Σ with each Voronoi edge, facet, and cell of $\mathrm{Vor}\,P$. While checking intersections is a computationally expensive task for a general implicit function, it is even more burdensome for the case of a piecewise locally interpolated function as given in our problem. For example, to determine if a certain Voronoi edge intersects Σ more than once, it is necessary to search all voxels containing any subset of Σ which are stabbed by the Voronoi edge, as Voronoi edges may be incident upon many voxels. Matters become worse for Voronoi facets or cells which may touch large regions of the domain.

It is in regards to this difficulty that our approach becomes significant. We exploit the "gridded" nature of the input data set to put $O(1)$ bounds on our intersection calculations. To start, we construct an initial sampling by computing all the points where a grid edge intersects Σ. These points serve as the initial sampling of Σ. However, if we compute the Voronoi diagram of these E points alone, the Voronoi cells can intersect an arbitrary number of voxels, meaning we will still have trouble verifying and enforcing the closed ball property. To circumvent this problem, we compute a protective layer of grid points near Σ which we denote G. The selection of G traps Voronoi cells of E points into a few voxels. As the algorithm progresses, it adds more points to the existing samples and the nature of the insertion process ensures that the Voronoi cells of these new points are also trapped in a constant number of voxels. We derive bounds on the size of Voronoi cells of the initial samples and the new points for uniform and non-uniform rectilinear gridding (Octtree) in Section 4.5.

We now give the pseudocode of the algorithm DELSURFMESH and describe the specifics of the steps subsequently (Figure 4.60).

DELSURFMESH(Σ)
1 Compute the point set E sampling Σ.
2 Compute the protective layer G of grid points.
3 Compute the Voronoi and Delaunay diagrams of $E \cup G$.
4 Insert new sample points (N) repeatedly until $\mathrm{Vor}\,(E \cup G \cup N)$ satisfies
  closed ball property.
5 Output the Restricted Delaunay triangulation $\mathrm{Del}\,(E \cup G \cup N)|_{\Sigma}$.

Figure 4.60: Pseudo-code of the DELSURFMESH algorithm.

We have already described how we choose the initial set of points on (E) and near (G) the surface Σ. The next task is to ensure that the closed ball property holds for the set of points. For a general interpolant within every grid cell, we employ the method given in [60]. For completeness, we briefly describe how the closed ball property can be violated and what measures are to be taken. This process is thus divided into three sub-steps CBP_VE for Voronoi edges, CBP_VF for Voronoi facets, and CBP_VC for Voronoi cells. As we show in Section 4.5, one can simplify this process considerably further if the typical trilinear interpolant is used, thereby improving the robustness and efficiency of the algorithm.

- CBP_VE: A Voronoi edge VE violates the closed ball property if it intersects Σ in more than one point. If this occurs,

the intersection point which is farthest from the Delaunay triangle dual to VE is inserted into the triangulation.

- CBP_VF: A Voronoi facet VF violates the closed ball property if it intersects Σ in more than one component or if the intersection includes a closed loop in the interior of VF. In either case, the intersection point farthest from the Delaunay edge dual to VF is inserted into the triangulation.
- CBP_VC: A Voronoi cell VC violates the closed ball property if it intersects Σ in more than one component, if the intersection includes an isolated component of Σ inside VC, or if the intersection includes a surface of positive genus with one or more disks removed.

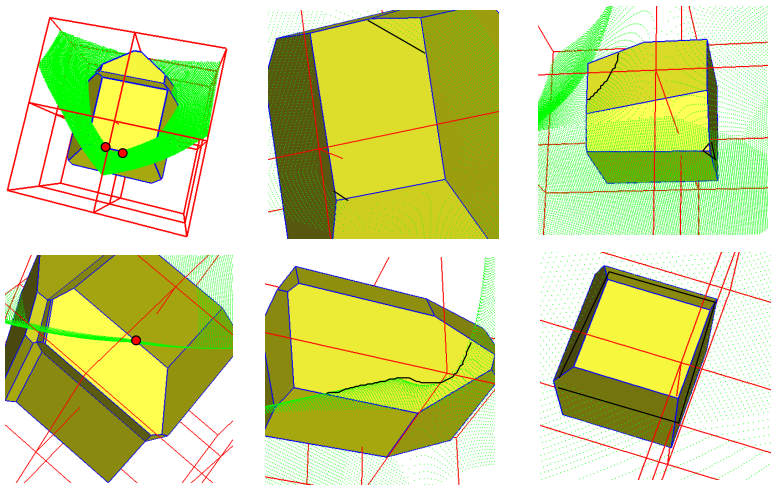


Figure 4.61: Top row shows three samples scenarios where the closed ball property is violated for a Voronoi Edge (left), Voronoi Facet (middle) and Voronoi Cell (right). The bottom row shows the cases where the closed ball property is satisfied for Voronoi objects of the corresponding dimension (in the top row).

Note, the above properties are to be checked in the order given. Once a violation is detected, a new point is inserted into the existing Delaunay triangulation, the triangulation is updated, and the processes must begin again. Figure 4.61 shows different situations that can arise in this context.

Mesh Refinement Although the topology of M is now correct, it may be possible that the geometry of M is not approximated sufficiently for an application purpose. Hence, we allow a user input ε and refine M as follows. For each Voronoi edge VE that intersects Σ, we compute the unique point $p \in VE \cap \Sigma$ and the circumcenter c of the dual Delaunay face to VE. If the distance between p and c is more than ε, we add p to the vertex set and regenerate the restricted Delaunay mesh. Every Voronoi edge dual to a restricted Delaunay facet is a normal approximation to Σ locally meaning the distance between p and c is an upper bound for the Hausdorff distance from Σ to the restricted Delaunay mesh. Hence, this process will yield a mesh satisfying property II.

In our current implementation, we maintain the 3D Vor/Del diagram of the point set throughout the process. Very recently, it was shown by Dey and Levine that this is not necessary [81]; one can recover the geometry while manipulating only the 2D mesh data structure of the surface, as long as there are enough points sampling a topologically correct approximation of Σ.

Tetrahedral Meshing of Interior/Exterior At this stage, we have an accurate mesh approximation of the level set both topologically and geometrically. Since the mesh is already embedded in a Delaunay mesh that includes some grid points, we already have a tetrahedral mesh of both the interior and exterior of Σ. In order to improve the quality of the mesh elements, we use the algorithm DELVOLMESH defined as follows.

Without loss of generality, we describe how DELVOLMESH is used to generate a tetrahedral mesh of the interior of Σ. The input to DELVOLMESH is $\mathrm{Del}(G \cup E \cup N)$, the volumetric mesh generated from DELSURFMESH. Note that $\mathrm{Del}(G \cup E \cup N)$ has the output of DELSURF MESH, $\mathrm{Del}(G \cup E \cup N)|_{\Sigma}$, as a subcomplex. We form a set G' of all grid vertices of the original rectilinear scaffolding which have function values less than the isovalue and do not belong to G. These points are distributed

through the interior of Σ evenly (or evenly relative to an adaptive gridding) and we add them to the Delaunay mesh of the volume.

Here, our protective layer of grid points is crucially important. As we add the points of G', some triangles of the Delaunay mesh will necessarily change but the Delaunay triangles among surface points (E and N vertices) will be unaffected. This is a direct consequence of the fact that a point of G' is, by construction of G, closer to points of G than to points of $E \cup N$. Therefore, $\text{Del}\,(E \cup N \cup G \cup G')$ will still have M as the restricted Delaunay diagram. By throwing out the points of G exterior to Σ, we are left with a tetrahedral mesh of the volume with good quality tetrahedra and a Delaunay conforming surface mesh. The 2D analogue of DELVOLMESH is shown in Figure 4.59 (d) and (e).

Efficiency

Since our algorithm uses the natural structure of the rectilinear input data to construct Voronoi and Delaunay diagrams, we are able to provide two important results that reduce the computational burden. We state and discuss the significance of each one.

Theorem 2 *There is an $O(1)$ bound on the number of voxels that a Voronoi cell of an E or N point may intersect. In the case of uniform rectilinear gridding with voxels that are cubes, this bound is four voxels.*

PROOF. We prove the following lemma for the simplest case in 2D which will be the crux of the argument for the proof of the more general cases.

Lemma 4 *For data given on an equally spaced 2D grid, the Voronoi cell of an E point is bounded within two pixels.*

PROOF. Let $e \in E$ lie on the edge ε be between grid points $g_1, g_2 \in G$ and let $VC(e)$ be the (2D) Voronoi cell defined by e. There exist two lines perpendicular to ε, one passing through the midpoint of g_1 and e and one passing through the midpoint of g_2 and e. By definition, $VC(e)$ is necessarily contained between these two lines which bounds $VC(e)$ to a single column of pixels.

Now consider one of of the two pixels containing both g_1 and g_2. Denote its other vertices as g_3 and g_4. Let s denote the side length of a pixel. Then both $||e - g_3|| > s/2$ and $||e - g_4|| < s/2$. For any point f on the edge between g_3 and g_4, observe that $||f - g_3|| \leq s/2$ or $||f - g_4|| \leq s/2$. Therefore, each point on the edge between g_3 and g_4 is closer to one of those grid points than it is to e, and hence cannot belong to $VC(e)$. This bounds $VC(e)$ to the two pixels containing ε, proving the lemma. A picture version of this proof appears in Figure 4.62a. □ □

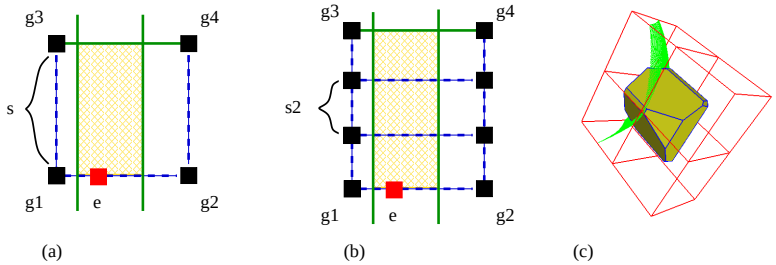


Figure 4.62: (a) A picture proof of Lemma 4. The Voronoi cell of the edge point e must lie in the shaded region bounded by the solid lines (green). By symmetry, this restricts the Voronoi cell to a two pixel range. (b) A picture proof of Theorem 2 in the general 2D case. Here, each pixel has dimensions s_1 by s_2 with $\lceil s_1/s_2 \rceil = 3$. By symmetry, the Voronoi cell of e is restricted to a six pixel range. (c) A Voronoi cell of an edge point in 3D, visibly contained within four voxels.

Continuing with the 2D case and the same notation, suppose that each pixel is a rectangle with side lengths s_1 and s_2. Suppose ε has length s_1. As in the proof of the lemma, we can immediately restrict $VC(e)$ to a column of pixels. Consider only the range of $2\lceil s_1/s_2 \rceil$ pixels closest to ε. This range contains the two squares of side length s_1 with ε as one edge. Therefore, we may repeat the analysis in the proof of the lemma using the corners of this range instead of the corners of the pixel. Hence, $VC(e)$ is bounded within $2\lceil s_1/s_2 \rceil$ pixels. Figure 4.62b shows a simple example.

Now consider the 3D case where each voxel has dimensions s_1, s_2, and s_3 and again assume $e \in E$ lies on the edge ε with side length s_1. There are two planes perpendicular to ε which bounds $VC(e)$ to a flat grid of voxels. In either of the two rectilinear dimensions of this grid, we apply the same analysis as above to conclude that $VC(e)$ intersects at most $2(\lceil s_1/s_2 \rceil + \lceil s_1/s_3 \rceil)$ voxels. Note that this constant reduces to 4 when $s_1 = s_2 = s_3$.

The proof for points of type N is similar. Finally, we note that as we add more E and N points to the diagram, the Voronoi cells can only get smaller. Therefore, the presence of other E or N points in the diagram is irrelevant to the bound. □ □

Theorem 2 has significant implications for solving the ray intersection problem. A priori, Voronoi cells may touch an arbitrary number of voxels which cause an explosion in computational time when checking if the closed ball property is satisfied. With Theorem 2, the computational time can be bounded in advance.

Notably, the actual bound depends on the gridding scheme of the input, not the choice of interpolant. Accordingly, the width of the protective layer of grid points collected during the algorithm depends on the gridding scheme as well. For the case of adaptive gridding in an octtree construction, the user must require the "level difference" between two adjacent cells to be no more than some fixed number k. Given k, a loose bound on the number of voxels a Voronoi cell of an E or N point may intersect is $4k^2$, as an edge is incident on at most k^2 cells in each of the other two orthogonal direction. This bound may be improved to 4 for E points and $3k+1$ for N points by considering limiting cases and using the convexity of Voronoi cells.

We have inserted the G points in order to decrease computations required to check the closed ball property, however we have increased the complexity of the Voronoi and Delaunay triangulations themselves. By Theorem 3 below, the new edges and facets formed by the addition of these G points do not add any significant burden to the closed ball property confirmation process. Further, these facets and edges are used in the algorithm DELVOLMESH described in Section 4.5.

Theorem 3 *Let VF be a Voronoi facet formed between two grid points and VE a Voronoi edge formed among three grid points at any point in the algorithm. Then $VF \cap \Sigma = \emptyset$ and $VE \cap \Sigma = \emptyset$.*

PROOF. It suffices to prove the theorem for E points, since the addition of N points only makes existing Voronoi elements smaller. First we treat the 2D case. Consider a pixel with grid points g_1, g_2, g_3, g_4 and two edge points p and q. (If the pixel has more (four) or fewer (zero) edge points, the theorem is vacuous.) The edge points may occur on adjacent or opposite edges of the pixel, as shown in Figures 4.63 a and b, respectively. We consider the adjacent case first and suppose that the edges on which p and q lie intersect at g_4. Let R denote the region bounded by the rectangle formed with p and q as opposite corners (the shaded yellow regions of Figure 4.63). Note that $\Sigma \subset R$ since we have chosen a bilinear interpolant. Hence, it suffices to show that the Voronoi edges formed between g_1, g_2 and between g_1, g_3 do not intersect R. If x is a point on the Voronoi edge between g_1, g_2 then by definition $||x - g_1|| = ||x - g_2||$ and $||x - z|| \geq ||x - g_2||$ for all $z \in G \cup E - \{g_1, g_2\}$. However, if $x \in R$ then $||x - q|| < ||x - g_2||$, based on the labelling of Figure 4.63a. Hence, the Voronoi edge between g_1, g_2 cannot intersect R. Similarly, the Voronoi edge between g_1, g_3 cannot intersect R, proving this case. The case where p and q lie on opposite edges is similar.

The proof for the 3D case is analagous to the 2D case. It suffices to prove the claim for a closed Voronoi facet between two grid points g_1, g_2 as this includes the relevant Voronoi edges. Consider just one of the voxels into which this facet extends. In each of the rectinear directions away from the g_1, g_2 edge, there exists a closest point of E. Taking these two E points and the g_1, g_2 edge, we uniquely define a box. Using the definitions of the Voronoi diagram construction, we can similarly show that Σ lies outside of the box and the Vornoi facet lies inside it, thereby proving the theorem. □ □

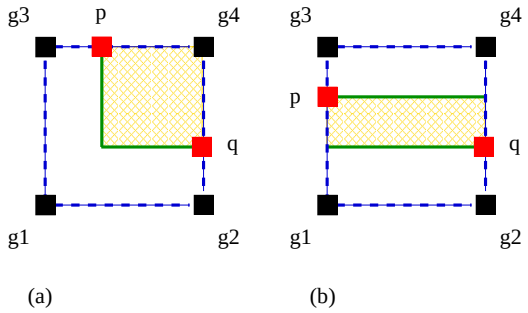


Figure 4.63: Proof of Theorem 3 in the 2D case. We show that Σ lies entirely inside the shaded yellow region, while the Voronoi edges formed between two grid points necessarily lie outside of it.

Simplifying Results

Thus far, we have addressed how the inclusion of the grid points G reduces computational overhead. We now examine how particular choices of the interpolant can further ease the calculations. First we consider the process CBP_VF. If $VF \cap \Sigma \neq \emptyset$, then the intersection is a compact 1-manifold by the transversality assumption. All compact 1-manifolds are homeomorphic to a finite collection of line segments and circles and we distinguish between the case of no circles and at least one circle. If no circles exist, the closed ball property is violated if and only if more than two edges of the facet intersect Σ which is easy to check.

If at least one circle occurs in the intersection of Σ and a Voronoi facet, calculations can become more subtle. In the case of trilinear interpolation, these types of intersections do arise in two fashions. First, there exist configurations of relative function values that produce a tunnel topology inside a single voxel, for example, Case 13.5.2 as defined in [153]. In some cases, a Voronoi facet will pass through the entire tunnel, causing an interior loop in its intersection with Σ. In this case, we can detect the topology of the cell by the function values and add the "shoulder points" as described in [153]. The shoulder points are positioned so that the Voronoi diagram breaks these interior loops on Voronoi facets, thereby easing calculations. Alternatively, it may occur that the interior loop of a Voronoi facet passes through multiple voxels, which still provides some difficulties. We conjecture that this phenomenon happens only if the interpolated function is non smooth at a voxel edge or vertex and are working to simplify this problem.

Now we turn to CBP_VC. For this process, use of the trilinear interpolant provides much stronger results to simplify calculations. First, if Σ intersects a Voronoi cell, the intersection manifold must have a boundary component by the following Theorem.

Theorem 4 *If each facet of a Voronoi cell VC has empty intersection with Σ and Σ is defined by a trilinear interpolant, then $VC \cap \Sigma = \emptyset$. That is, there cannot be a component of Σ entirely inside a single Voronoi cell.*

PROOF. The trilinear interpolation method precludes the existence of isolated surface components within a single voxel. Therefore, every component of Σ has a point of G in its interior and hence is sampled by at least six points of E, corresponding to the six rectilinear directions. Thus, each component of Σ intersects at least six Voronoi cells. A similar proof holds if the data is not given on a rectilinear grid. □ □

Therefore, the process of checking the closed ball property for a Voronoi cell is reduced to checking if the intersection along the facets of a cell is a single closed curve (or empty). A priori, the possibility remains that the intersection surface is a manifold of positive genus with a disc removed. Detection of such cases has been discussed in detail in [60]. However, if our algorithm is applied in the common case of trilinear interpolation with voxels that are cubes, these difficult cases can also be excluded, leading to a much simpler algorithm.

Theorem 5 *Let VC be a Voronoi cell whose facets and edges satisfy the closed ball property and let Σ be defined by a trilinear interpolant on a uniform grid with voxels that are cubes. Then the closed ball property is satisfied for VC if and only if $VC \cap \Sigma$ has a single boundary component.*

PROOF. It suffices to exclude the minimum case where $VC \cap \Sigma$ is homeomorphic to a torus with a disc removed. To generate such a surface by trilinear interpolation on a grid of cubes requires more than four voxels since the intersection of Σ with a voxel face cannot contain closed loops. By Theorem 2, the surface therefore intersects more than one Voronoi cell. □ □

We have used CGAL [72] to build and maintain the Voronoi and Delaunay diagram of the point set. We also needed to compute the intersection of a ray with the surface Σ inside certain voxels; for this task, we have used another publicly available library called SYNAPS [190].

Only the parameter ε is needed to run the algorithm on a given data set. This input dictates the desired upper bound on the Hausdorff distance between the surface approximation M and the interpolated surface Σ. Given this parameter, we reduce the amount of computation needed by snapping some of the points of E in the following manner. If there exist points $g \in G$ and $e_0, e_1, e_2 \in E$ such that the e_i are on grid edges incident to g and $d(g, e_i) < \varepsilon$ for $i = 0, 1, 2$, we snap the three e_i points onto the common point g. Since all e_i points are within ε distance of the grid point and Σ passes through each e_i, the closet point of Σ to g cannot be more than ε away, which is within the user specified limit of tolerance.

Although we have reduced the complexity of the ray-surface intersection problem, we still have to compute it for a constant number of voxels for every Voronoi edge. This was accomplished by parameterizing the Voronoi edge segment between two Voronoi vertices p_0 and p_1 with a real number t. Inside every candidate voxel, we then detect the intersection points of the segment with the interpolating polynomial function using the Algebraic solver available in the library SYNAPS. If an intersection point lies inside the corresponding voxel, we consider it as a valid intersection.

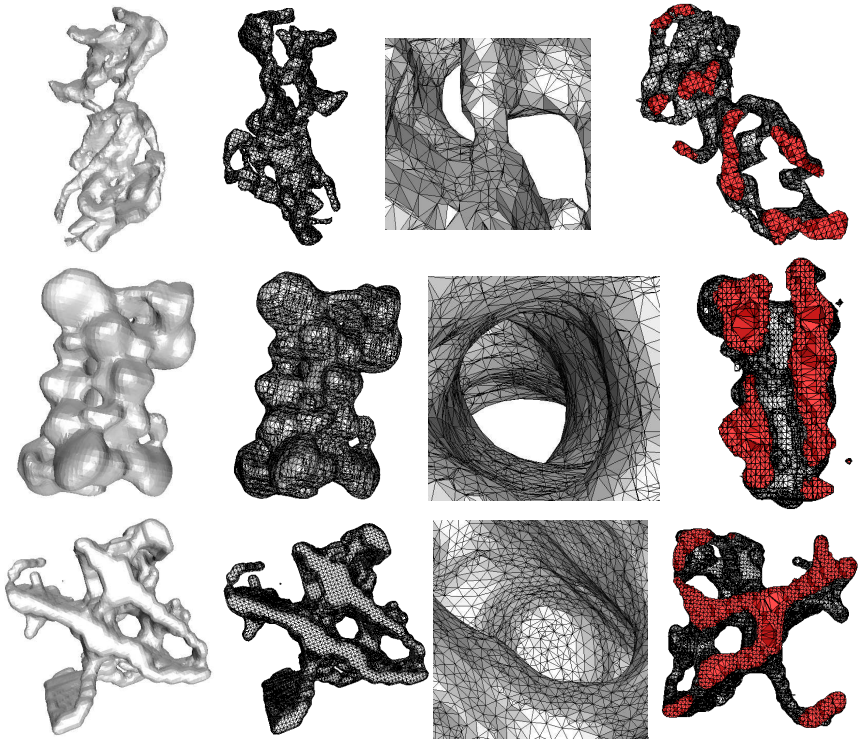


Figure 4.64: Performance of the meshing algorithm. The top row shows the geometry, surface mesh, a closeup on the surface mesh and a cut-away of the volumetric tetrahedral mesh for 1CID. The second and the third rows show the same for 1MAG and BONE, respectively.

The performance of the algorithm on biological entities at various scales is shown in Figure 4.64.

The 1CID data set was obtained via blurring the coordinates of the atoms available from the Protein Data Bank (PDB) [39]. The resulting 3D scalar volume represents the electron density of the protein molecule T Cell Surface Glycoprotein CD4 . The goal in this case is to extract a well-sampled Delaunay conforming surface mesh so that one can analyze the secondary structure using the unstable manifolds of the index 1 and index 2 saddle points of a suitable distance function [15].

1MAG, shown in the second row of Figure 4.64, is the PDB entry of the ion channel Gramicidin A, obtained from soil bacteria Bacillus brevis. The molecular surface has many tunnels as shown in the left most sub-figure in the second row. However, only the tunnel in the middle is topologically significant as it is important to preserve the property of the ion channel. Once the isosurface has been extracted, it is therefore necessary to remove all the other tunnels and preserve the main tunnel. The algorithm for such removal of topological features has been described in [14] which relies on a Delaunay conforming isosurface in order to detect and remove unwanted topological features.

Finally, BONE data is shown in the third row of Figure 4.64. This data set is provided by our collaborator from the University of Rome. The goal here is to mesh the internal bone structure for stress-strain analysis.

The size of the 1CID volume data set is 128^3. It took 1 second to build the initial triangulation, 45 seconds to enforce the closed ball property, and 35 seconds to recover the geometry for $\varepsilon = 0.00001$. The size of the 1MAG dataset is 128^3. It took 1.5 seconds to build the initial triangulation, 20 seconds to enforce the closed ball property, and 28 seconds to recover the geometry for $\varepsilon = 0.00001$. The size of the BONE dataset is 64^3. It took 5 seconds to build the initial triangulation, 65 seconds to enforce the closed ball property, and 89 seconds to recover the geometry for $\varepsilon = 0.0001$. Note, the time taken to mesh does not depend on the size of the volume data set since we were able to enforce that the ray-surface intersection calculation is of constant complexity. However, as the level set passes through more voxels, the number of points $(E \cup G)$ increases and that increases the time complexity of the algorithm. We have presented an improvement over traditional level set meshing approaches. In particular, we claimed that the isosurfaces extracted via well-known approaches such as marching cubes or dual contouring are not suitable as they not only fail to capture the topology in a provable manner but also do not produce sufficient samples for the extracted approximation to be embedded in the Delaunay triangulation of the sampled set of points. On the other hand,

there are elegant approaches for meshing general implicit surfaces, yet these algorithms suffer from the computational overhead of computing the intersection of a ray and the level set. Typically, the level set of a sampled scalar function is approximated via a number of piecewise interpolating functions and therefore finding the intersection of a ray with the level set requires a large number of checks. In light of this, we have presented an algorithm which not only overcomes the sampling issue of traditional isosurfacing techniques but also adopts a well-known provable algorithm [60] for generating a good sample of the isosurface efficiently. We have also shown that for the commonly used trilinear interpolant, many of the difficult cases that arise in implementation of the algorithm can be avoided, thereby improving numerical robustness.

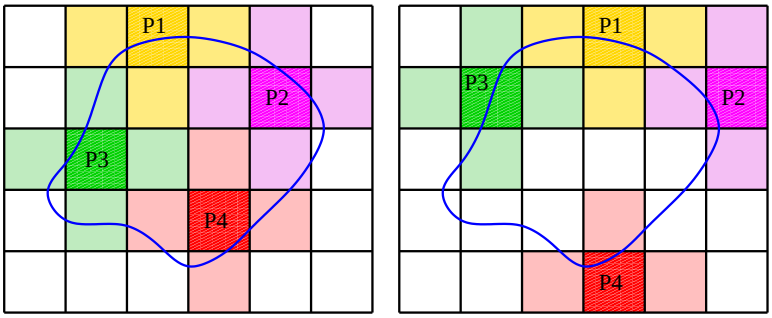


Figure 4.65: Data may be partitioned to run the algorithm in parallel. In this 2D example, the voxels indicated by the P_i are to be processed. One can send data from each P_i voxel to a separate processor, along with the collar of protective voxels around it indicated by the same color shading. Two iterations are shown.

Scope As noted earlier, the scope of this algorithm is immense. First, we have not assumed any particular interpolation scheme in order to prove that we need to check only a few voxels when detecting ray-surface intersection. Therefore, it is possible to extend the algorithm to any higher order interpolant, for example cubic A-patches [9]. Second, adaptive sampling of the scalar function poses no problem to this algorithm as explained previously. Finally, since we exploit the local nature of the interpolated surface Σ, the algorithm can easily be parallelized which is especially important for the large data sets often used in the multiscale biological models we have discussed. Figure 4.65 indicates how a data set might be partitioned to speed up computational time the analogous 2D case. The different colored pixels marked with P_i can be processed concurrently on separate processors as the Voronoi cells of the E and N points in those pixels are guaranteed to be inside the protective collar of surrounding pixels. We note that not all pixels can be processed concurrently as the protective collars must be non-overlapping. However, by iterating the process, the most difficult computations can be done in parallel.

Limitations One limitation of our algorithm is that it tends to generate more samples due to the layer of G points that we use to restrict the search space. Therefore it is necessary to apply adaptive sampling of the scalar function such that bigger voxels are automatically created in regions of less detail and smaller voxels in regions of high detail. By the results of Section 4.5, the efficiency of the algorithm remains the same.

Future Work The extracted mesh typically has a "gridding" artifact because of the influence of the grid. Further, sometimes the uniformity of the grid causes it to be over-sampled. Therefore, ideally, one would decimate the grid so that the G vertices lie close to the medial axis after the isovalue is known. We plan to develop a scheme for such an optimal placement of the grid vertices. Also, the efficiency of the core computation requires a scaffolding structure. Hence, once the Delaunay conforming surface mesh is extracted, it is not clear how the surface can be decimated (if needed) by throwing away some of the grid vertices, while still efficiently checking that the resulting (decimated) point sample satisfies the closed ball property.

The following algorithm is taken from [265] (zhang20053d).

Introduction

The development of finite element simulations in medicine, molecular biology and engineering has increased the need for high quality finite element meshes. Although there has been tremendous progresses in the area of surface reconstruction and

3D geometric modelling, it still remains a challenging process to generate 3D meshes directly from imaging data, such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and Signed Distance Function (SDF). The imaging data V is given in the form of sampled function values on rectilinear grids, $V = \{F(i,j,k) \mid i,j,k$ are indices of x,y,z coordinates in a rectilinear grid.$\}$. We assume a continuous function F is constructed through the trilinear interpolation of sampled values for each cubic cell in the volume.

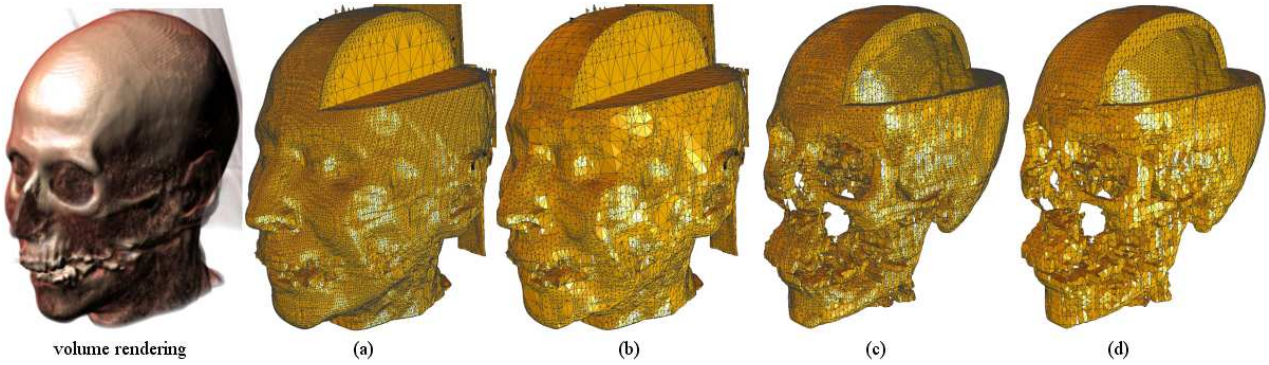


Figure 4.66: Adaptive tetrahedral meshes are extracted from a CT-scanned volumetric data(UNC Head). Depending on the selected isovalues, different meshes of a skin and a skull are constructed. The number of tetrahedra can be controlled by choosing a user specified error tolerance (tet# - (b) 935124, (c) 545269, (d) 579834, (e) 166271). Note that the extracted mesh has no crack and no hanging node.

For accurate and efficient finite element calculations, it is important to have adaptive and high quality geometric models with minimal number of elements. The studied object may have complicated topology. Figure 4.86 shows an interval volume between two isosurfaces from the SDF volumetric data of a knee. The two surfaces have the same topology in Figure 4.86(d), while the topology of the inner surface may be different from the topology of the outer one (Figure 4.86(b)). In this paper, we present a comprehensive approach to extract tetrahedral and hexahedral meshes directly from imaging data.

$$S_F(c) = \{(x,y,z) : F(x,y,z) = c\} \tag{5.39}$$

$$I_F(\alpha_1,\alpha_2) = \{(x,y,z) : \alpha_1 \leq F(x,y,z) \leq \alpha_2\} \tag{5.40}$$

Given volumetric imaging data, α_1 and α_2 ($\alpha_1 < \alpha_2$) represent two isovalues, each of which defines a boundary isosurface (Equation (1)) of an interval volume (Equation (2)). The main steps to extract tetrahedral/hexahedral meshes for the interval volume, I_F, between the two isosurfaces are as follows:

1 Preprocessing – volumetric denoising and isovalue selection.

2 Adaptive 3D meshing with correct topology and feature preservation.

3 Quality improvement

Noise often exists in imaging data, especially in CT and MRI data. The bilateral prefiltering coupled with anisotropic diffusion methods [23] is applied to smooth the volumetric data. Accurate gradient estimation can also be obtained. The Contour Spectrum [19] provides quantitative metrics of a volume to help us select two suitable isovalues for the interval volume. If the imaging data has no noise (for example, SDF data) and two isovalues are given to define the interval volume, the preprocessing step can be skipped.

We extend the idea of dual contouring to interval volume tetrahedralization and hexahedralization from volumetric Hermite data (position and normal information). Dual Contouring [126] analyzes those edges that have endpoints lying on different sides of the isosurface, called *sign change edge*. Each sign change edge is shared by four (uniform case) or three (adaptive case) cells, and one minimizer is calculated for each of them by minimizing a predefined Quadratic Error Function (QEF) [103]. The QEF is defined as follows:

$$QEF[x] = \sum_i (n_i \cdot (x - p_i))^2 \tag{5.41}$$

where p_i, n_i represent the position and unit normal vectors of the intersection point respectively. For each sign change edge, a quad or a triangle is constructed by connecting the minimizers. These quads and triangles provide an approximation of the isosurface.

Each sign change edge belongs to a boundary cell. We present a systematic way to tetrahedralize the volume in the boundary cell. For uniform grids, it is easy to deal with the interior cells. We only need to decompose each cell into five tetrahedra. The adaptive case is more complicated. In order to avoid introducing hanging nodes, which are strictly prohibited in finite element meshes, we design an algorithm to tetrahedralize the interior cell depending on the resolution levels of all its neighbors. Figure 4.66 shows an example of adaptive tetrahedral meshes extracted from a scanned CT data. As a byproduct, the uniform hexahedral mesh extraction algorithm is simpler. We analyze each interior vertex (a grid point inside the interval volume) which is shared by eight cells. One minimizer is calculated for each of them, and those eight minimizers construct a hexahedron.

Reconstructing a mesh with correct topology is important for accurate finite element calculations. We guarantee the resulting 3D mesh is topologically equivalent to the real interval volume $I_F(\alpha_1, \alpha_2)$. The topology of the 3D mesh is preserved during the simplification process. Unlike the dual contouring method [126], we use a different error function based on the function difference normalized by gradients. The function approximates the maximum difference between coarse and fine level isosurfaces to decide the level of adaptivity. Using this error measurement and a user specified error tolerance, we can identify octree cells of appropriate levels which satisfy the threshold criteria. The result shows the error function we use yields feature-sensitive adaptation as shown in Figure 4.78 (d). Since we still use QEF for computing minimizing vertices, we can also preserve sharp edges and corners.

The tetrahedral meshes extracted from volume data can not be used for finite element calculations directly, since some elements may have bad quality. The edge-ratio and Joe-Liu parameter are chosen to measure the mesh quality. The edge contraction method removes tetrahedra with bad edge-ratios, and the smoothing method improves the mesh quality measured by the Joe-Liu parameters. We applied our algorithm to extracting a tetrahedral mesh from the accessibility function of a mouse acetylcholinesterase (mAChE) biomolecule. The extracted meshes have been used for efficient and correct finite element calculation of a diffusion problem [218].

Previous Work

Isosurface Extraction In most cases, an isosurface is extracted in the form of piecewise linear approximation for the modeling and rendering purpose. The Marching Cubes algorithm (MC) [155] visits each cell in a volume and performs local triangulation based on the sign configuration of the eight vertices. To avoid visiting unnecessary cells, accelerated algorithms [241] [18] minimizing the

Preprocessing

Since noise in imaging data influences the accuracy of the extracted meshes, it is important to remove it before the mesh extraction process. The isotropic diffusion method can remove noise, but blurs features such as edges and corners. In order to preserve features during the process of noise smoothing, anisotropic diffusion [238] was proposed by introducing a diffusion tensor. Generally, a Gaussian filter is used to calculate the anisotropic diffusion tensor before smoothing, but it also blurs features. Bilateral filtering [227], which is a nonlinear filter combining domain and range filtering, was introduced to solve this problem. Anisotropic diffusion can be used for fairing out noise in both surface meshes and functions defined on the surface [27] [68]. Here we use the anisotropic diffusion method [23] to smooth noise. In order to obtain more accurate computation of curvature and gradient for anisotropic diffusion tensor, the bilateral prefiltering combining the domain and range filtering together is chosen instead of Gaussian filtering.

Mesh extraction from imaging data requires the selection of suitable boundary isosurfaces. We use a user interface called Contour Spectrum [19] to find significant isosurfaces. The Contour Spectrum computes quantitative properties such as surface area, volume, and gradient integral of contours in real time, and helps to choose suitable isosurfaces by showing the related spectrum in a 2D plane. For instance, we can obtain isosurfaces of a skin and a skull in CT scanned human head data (Figure 4.67) by taking isovalues with the local maximum of the gradient integral. In the case of the interval volume, the topology of inner and outer isosurfaces may need to be controlled depending on the application. A contour tree [52] can be used to capture the topological information on each isosurface and help choose isosurfaces with desirable topology. For example, we can choose the inner and outer isosurfaces with exactly the same topology by taking isovalues lying on the same edges in a contour tree.

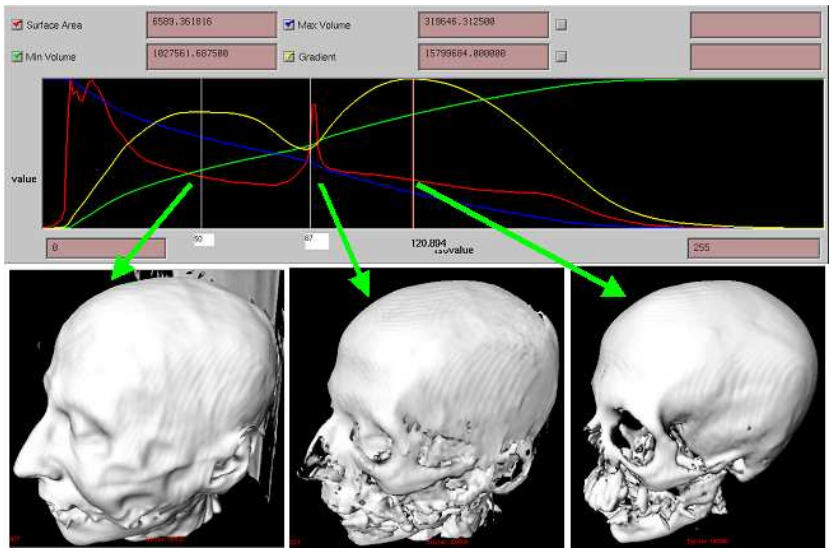


Figure 4.67: The contour spectrum of the UNC human head

3D Mesh Extraction

In this section, our goal is to tetrahedralize or hexahedralize the interval volume between two isosurfaces by using an octree-based data structure. We describe in detail how to extract adaptive tetrahedral meshes from volume data. First, we discuss the triangulation in 2D problems, then we extend it to 3D tetrahedralization. A hexahedral mesh generation algorithm is presented at the end of this section. Here are definitions used in the algorithm description:

Sign Change Edge: A sign change edge is an edge whose one vertex lies inside the interval volume (we call it the interior vertex of this sign change edge), while the other vertex lies outside.

Interior Edge in Boundary Cell: In a boundary cell, those edges with both vertices lying inside the interval volume are called interior edges.

Interior Face in Boundary Cell: In the boundary cell, those faces with all four vertices lying inside the interval volume are called interior faces.

Interior Cell: Different from the boundary cell, all the eight vertices of an interior cell lie interior to the interval volume.

Uniform Tetrahedral Mesh Extraction For isosurface extraction, we only need to analyze boundary cells – those cells that contain sign change edges, or those cells that contain the isosurface. There are four neighbor cubes which share the same sign change edge. Dual Contouring generates one minimal vertex for each neighbor cube by minimizing the QEF, then connects them to generate a quad. By marching all the sign change edges, the isosurface is obtained. For tetrahedral mesh extraction, cells inside the interval volume should also be analyzed besides the boundary cells.

Uniform 2D Triangulation Figure 4.68 is a uniform triangulation example of the area interior to the isocontour in two dimensions. There are three different cases which need to be dealt with separately.

1 Sign change edge – find the minimizers of the two cells which share the edge, then the two minimizers and the interior vertex of the edge construct a triangle (blue triangles).

2 Interior edge in boundary cell – find the QEF minimizer of the boundary cell, then the minimizer and this interior edge construct a triangle (yellow triangles).

3 Interior cell – decompose each interior cell into two triangles (pink triangles).

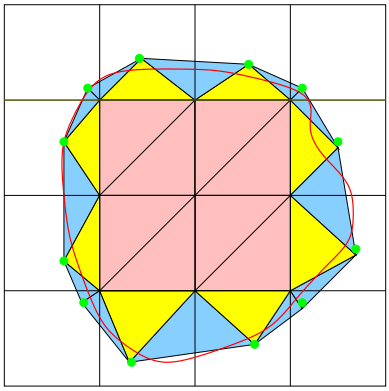


Figure 4.68: Uniform triangulation - the red curve represents the isocontour, green points represent minimizers.

Uniform 3D Tetrahedralization Compared to 2D triangulation, three dimensional tetrahedral meshing is more complicated. Figure 4.69 shows the case table of uniform tetrahedralization.

1 Sign change edge – decompose the quad into two triangles, then each triangle and the interior vertex of this edge construct a tetrahedron. In Figure 4.69(a), the red line represents the sign change edge, and two blue tetrahedra are constructed.

2 Interior edge in boundary cell – find the QEF minimizers of the boundary cell and its boundary neighbor cells, then two adjacent minimizers and the interior edge construct a tetrahedron. In Figure 4.69(b)(c), the red cube edge represents the interior edge. (b) shows four minimizers to construct four edges, each of which construct a tetrahedron with the interior edge, so totally four tetrahedra are constructed. While (c) assumes the cell below this boundary cell is interior to the interval volume, so there is no minimizer for it. Therefore we obtain three minimizers, and only two tetrahedra are constructed.

3 Interior face in boundary cell – find the QEF minimizer of the boundary cell, then the interior face and the minimizer construct a pyramid, which can be decomposed into two tetrahedra (Figure 4.69(f)). Figure 4.69(d)(e)(f) show a sequence of how to generate tetrahedra when there is only one interior face in the boundary cell. (d) analyzes four sign change edges, (e) deals with four interior edges and (f) fills the gap.

4 Interior cell – decompose the interior cube into five tetrahedra. There are two different decomposition ways (Figure 4.69(g)(h)). For two adjacent cells, we choose a different decomposition method to avoid the diagonal choosing conflict problem.

Our meshing algorithm assumes that there is only one minimizer point in a cell. This means two different boundary isosurfaces of an interval volume can not pass through the same cell. Therefore, we enforce every cell to have at most one boundary isosurface before actual meshing of cells. If a cell contains two boundary isosurfaces, we recursively subdivide the cell into eight identical sub-cells until each sub-cell contains at most one boundary isosurface as shown in Figure 4.70. The detailed subdivision algorithm is described in Section 4.5.

Adaptive Tetrahedral Mesh Extraction Uniform tetrahedralization usually generates an over-sampled mesh. Adaptive tetrahedral meshing is an effective way to minimize the number of elements while preserving the accuracy requirement. First, we split the volume data by using the octree data structure to obtain denser cells along the boundary, and coarser cells inside the interval volume (Figure 4.72). The QEF value is calculated for each octree cell. When the error tolerance ε is selected, the octree is traversed in a bottom-up manner to obtain the cells which satisfy the error tolerance requirement. We call this set of cells to be leaf cells of the octree assuming we pruned unnecessary nodes from the tree.

Each leaf cell may have neighbors at different levels. An edge in a leaf cell may be divided into several edges in its neighbor cells. Therefore it is important to decide which edge should be analyzed. The Dual Contouring method provides a good rule to follow – we always choose the minimal edge. Minimal edges are those edges of leaf cubes that do not properly contain an edge of a neighboring leaf.

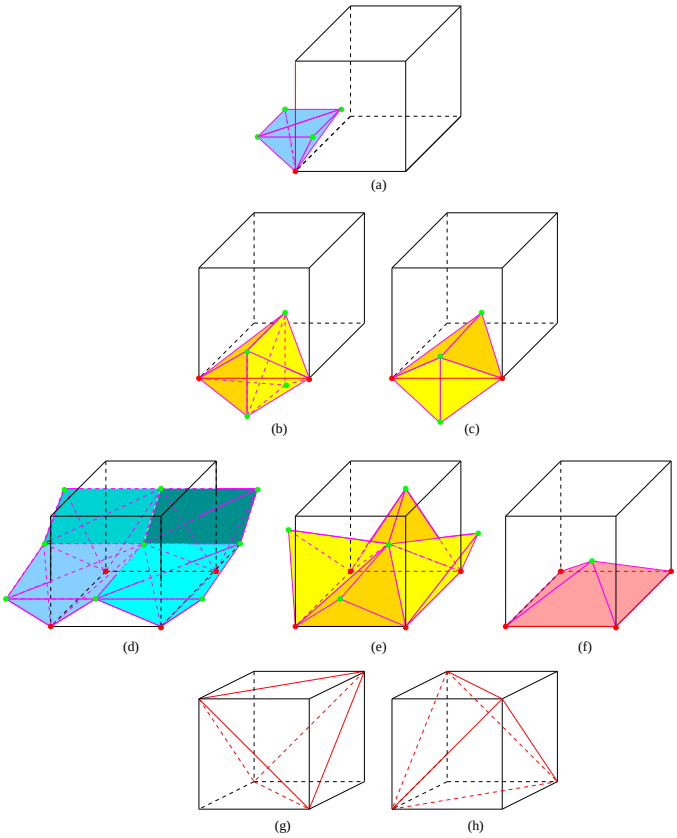


Figure 4.69: The case table of uniform tetrahedralization - the red vertex means it lies interior to the interval volume, otherwise, it is outside. Green points represent minimizers. (a) - a sign change edge; (b)(c) - an interior edge in a boundary cell; (d)(e)(f) - an interior face in a boundary cell; (g)(h) - an interior cell.

Similar to uniform tetrahedral mesh extraction, we need to analyze the sign change edge, the interior edge/face in the boundary cell, and the interior cell. When we analyze boundary cells, only minimal edges/faces are analyzed. Compared to the uniform case, the only difference is how to decompose interior cells into tetrahedra without hanging nodes.

Hanging Node: a hanging node is one that is attached to the corner of one triangle but does not attach to the corners of the adjacent triangles. Generally, a hanging node is a point that is a vertex for some elements (e.g., triangle, quad, tetra, hexa), but it is not for its other neighbor elements that share it. It lies on one edge or one face of its neighbors, for example, a T-Vertex.

Figure 4.71 shows two methods to remove hanging nodes - splitting and merging. In the T-Vertex example (Figure 4.71a), there is a hanging node (red point). Only the right two triangles (Number 2 and 3) need to be modified if we use the merging method (Figure 4.71b); while only the left one (Number 1) needs to be modified if we intend to split the mesh (Figure 4.71c). In order to maintain the accuracy, we adopt the splitting method in our algorithm.

Lemma: Only the interior cell needs to be modified if the splitting method is adopted.

Proof: All the leaf cells can be divided into two groups: the boundary cell and the interior cell.

1 Interior cell – Since its neighbor cells may have higher resolution levels, hanging nodes are unavoidable. In Figure 4.71d, there is a hanging node if we triangulate the interior cell as in the uniform case. Figure 4.71e shows a re-triangulating method to remove the hanging node.

2 Boundary cell – There are two rules for the sign change edge and the interior edge/face in boundary cells. The two rules guarantee that no hanging nodes need to be removed for the boundary cell if only the splitting method is chosen.

- Minimal Edge/Face Rule - only minimal edges/faces are chosen. This rule keeps the analyzed edges/faces owning the highest resolution level compared with their neighbors.

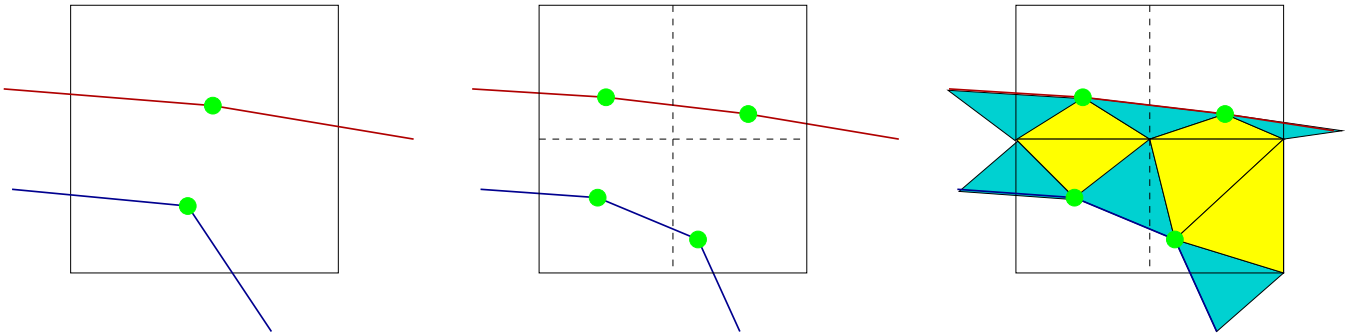


Figure 4.70: A two dimensional example of cell subdivision for enforcing each cell to have at most one boundary isocontour. When two boundary isocontours pass through the same cell, the cell is recursively subdivided until each sub-cell contains at most one minimizer.

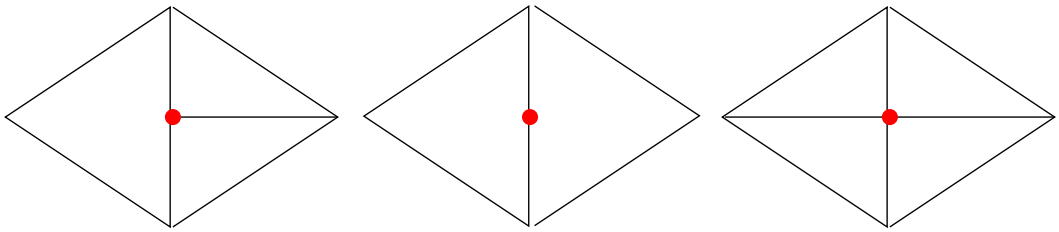


Figure 4.71: Hanging node removal - the red point is a hanging node. (a) - T-Vertex; (b) - merging two triangles; (c) - splitting method. (d) and (e) show an example of hanging node removal by re-triangulating the interior cell (pink). The red curve is the real isocontour, and green points are minimizer points for boundary cells. (d) - the triangulation of the interior cell without considering the hanging node; (e) - the re-triangulation of the interior cell to remove the hanging node.

- Only one minimizer is generated for each leaf cell.

Adaptive 2D Triangulation Figure 4.72(left) shows an example of how to triangulate the interior area of an isocontour. Similarly, we need to analyze the following three problems:

1 Sign change edge – if the edge is minimal, deal with it as in the uniform case (blue triangles).

2 Interior edge in the boundary cell – if the edge is minimal, deal with it as in the uniform case (yellow triangles).

3 Interior cell – Figure 4.72(Right) lists all the main cases of how to decompose an interior cell into triangles. The case table can be easily generalized to any other adaptive cases. In order to obtain triangles with good aspect ratio, we restrict the neighboring level difference to be ≤ 2.

Compared to the uniform case, the triangulation of interior cells is more complicated. All neighbors of an interior cell need to be checked because the neighbor cells are used to decide if there are any middle points on the shared edge. Suppose the resolution level of this cell is κ. We group into five cases according to the number of edges whose level is greater than κ. The i^{th} group means there are number i edges whose level is greater than κ, where $i = 0, \ldots, 4$. For each subdivided edge, it may be subdivided more than once, or the neighbor cell may have higher level than $(\kappa + 1)$. So we need to search all the middle points on this edge by looking at the resolution levels of the neighbor cells. Figure 4.72(right) lists all the main cases of how to decompose the interior cell into triangles according to its neighbors' resolution levels. If all the four edges have already been subdivided, then we can use the recursive method to march each of the four sub-cells with the same algorithm. In this way, hanging nodes are removed effectively.

Adaptive 3D Tetrahedralization For three dimensional adaptive tetrahedralization, we use the similar algorithm with the uniform case when we deal with the boundary cell.

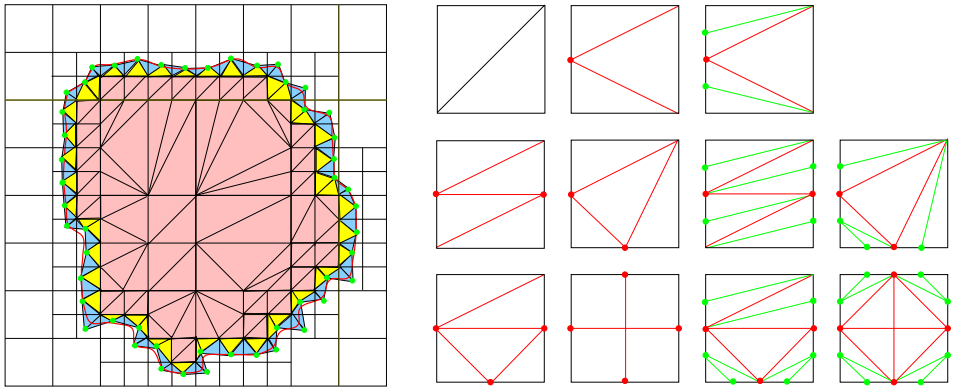


Figure 4.72: Left: adaptive triangulation. The red curve represents the isocontour, green points represent minimizers. Right: The case table for decomposing the interior cell into triangles in 2D. Suppose the resolution level of this cell is κ, and middle points appear on the shared edges if its neighbors have higher level than κ. Red points and red lines mean its neighbors have level $(\kappa+1)$; green points and green lines mean its neighbors have higher level than $(\kappa+1)$. The case table can be easily generalized to any other adaptive cases.

1 Sign change edge – if the edge is minimal, deal with it as in the uniform case.

2 Interior edge in the boundary cell – if the edge is minimal, deal with it as in the uniform case.

3 Interior face in the boundary cell – identify all the middle points on the four edges, and decompose the face into triangles by applying the same algorithm as in the adaptive 2D case, then calculate the minimizer of this cell, each triangle and this minimizer construct a tetrahedron.

4 Interior cell – decompose each face of the cube into triangles, just as how to deal with the interior cell for the adaptive 2D triangulation (Figure 4.72(right)), then insert a Steiner point at the cell center. Each triangle and the Steiner point construct a tetrahedron. Figure 4.73 shows how to construct tetrahedra when the neighboring level difference is one. Sometimes pyramids are constructed. In order to avoid the diagonal choosing conflict, we decide which diagonal is chosen to decompose one pyramid into two tetrahedra according to the odd-even property of the cell index.

By using the above algorithm, we extract tetrahedral meshes from volumetric imaging data successfully. Figure 4.74 (a) and (b) show the tetrahedral mesh of the human head model extracted from 65^3 volumetric data. The volume inside the skin isosurface is tetrahedralized.

Hexahedral Mesh Extraction Finite element calculations sometimes require hexahedral meshes instead of tetrahedral meshes. Each hexahedron has eight points. In the tetrahedralization process, we deal with edges shared by at most four cells. This means that we can not get eight minimizers for each edge. However, each vertex within the interval volume is shared by eight cells. We can calculate a minimizer for each of them. In the case of interior cells, we set the center point as the minimizer. These eight minimizers can then be used to construct a hexahedron. Figure 4.74 (c) and (d) show two hexahedral meshes for the head model, which are used to solve electromagnetic scattering simulations in finite element calculations.

Mesh Topology

Constructing an adaptive 3D mesh with correct topology plays an important role in accurate and efficient finite element calculations. Our goals in this section are (i) meshing with correct topology and (ii) topology preserving adaptive meshing. The topology of a mesh defined by an interval volume depends on the topology of two boundary isosurfaces enclosing the interval volume. Therefore we focus on the topology of an isosurface. We use the term 'dual contour' as a polygonized isosurface extracted using the dual contouring method [126]. 'Correct topology' means that an extracted dual contour is topologically equivalent to the real isosurface defined from an original input function. 'Topology preserving' means the topology of a dual contour is preserved during simplification.

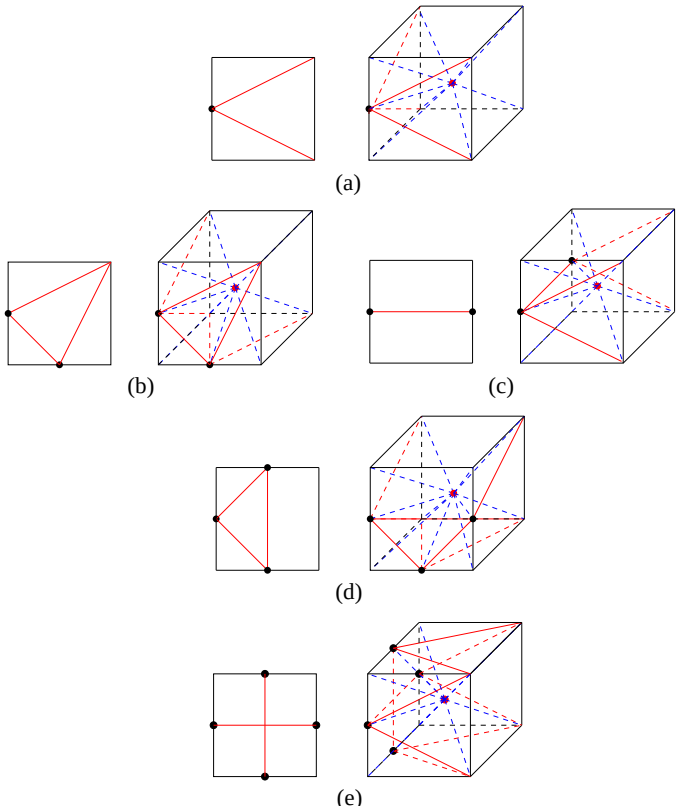


Figure 4.73: The case table for decomposing the interior cell into tetrahedra when the neighboring level difference is one – in (a $\sim$ e), the left picture shows the triangulation format of one face according to Figure 4.72; the right one shows how to decompose the cell into tetrahedra without hanging nodes. (a) - one subdivided edge; (b)(c) - two subdivided edges; (d) - three subdivided edges; (e) - four subdivided edges. Any other adaptive cases are easily generalized using the case table in Figure 4.72.

Assume the function F within a cubic cell is defined by the trilinear interpolation of the eight vertex values. The sign of a vertex is defined to be positive when its value is greater than or equal to an isovalue, and negative when its value is less than the isovalue. An isosurface within a cube, defined as $F(x,y,z) = \alpha$, may have different local topology depending on the configuration of signs at the eight vertices and the isovalue. There are 2^8 sign configurations which can be reduced into 14 cases using symmetry [155]. Several cases may have more than one local topology, and are termed ambiguous. We refer to [152] for all the cases of different local topology of an isosurface in a cube. When a case is ambiguous, the standard dual contouring method causes a non-manifold at the minimizing vertex in the cube. This makes the topology of the dual contour different from that of the real isosurface. On the other hand, if a cube has no ambiguity, then the real isosurface is topologically equivalent to the dual contour, which is always a simple manifold. Whether a case in a cube is ambiguous or not can be checked by collapsing each edge which has two vertices with the same sign into a vertex [126]. If the cube can be collapsed into an edge, then the cube has no ambiguity and the topologically correct dual contour is generated in the cube. An example of a cell with an ambiguous case is shown in Figure 4.76 (a) where the non-manifold dual contour is generated using a naive approach.

We use the recursive cell subdivision in the finest level to reconstruct a dual contour with correct topology when the cell contains a non-manifold dual contour. The subdivision algorithm is very similar with what we used for enforcing each cell to have at most one boundary isosurface. As a first step, all boundary cells in the finest level are identified. If a cell contains a non-manifold dual contour, we subdivide the cube into eight identical sub-cubes. The function values defined on newly generated vertices are calculated by the trilinear interpolation of the values at the eight cube vertices. We recursively repeat this process for the sub-cubes containing a non-manifold dual contour until each sub-cube contains a manifold dual contour. Figure 4.75 shows a 2D example.

We justify the correctness of the algorithm as follows. The function defined in a sub-cube is exactly same as the original

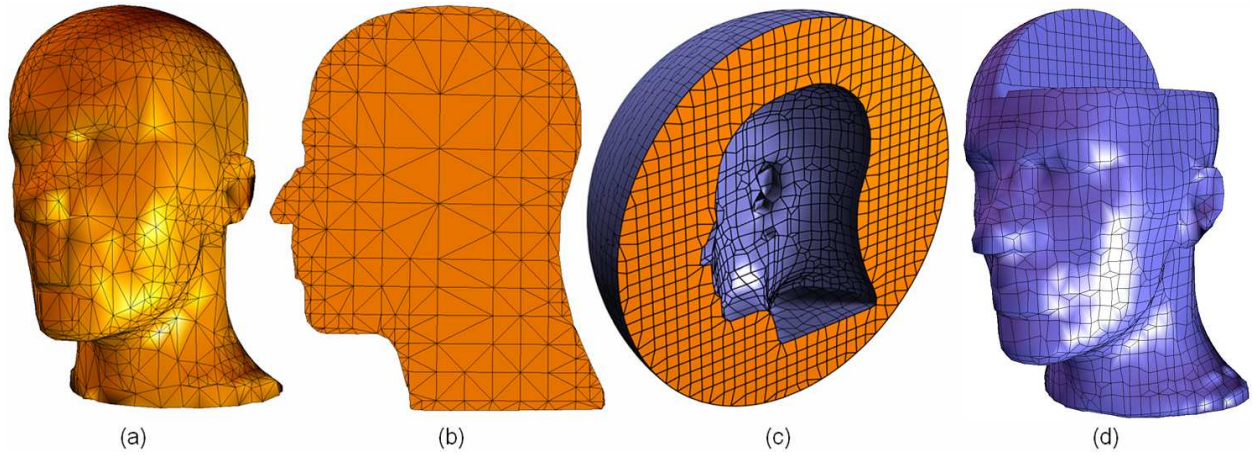


Figure 4.74: Tetrahedral/hexahedral meshes of the human head model - (a) and (b) show adaptive tetrahedral meshes, (c) and (d) show the hexahedral meshes. (a): the tetrahedralization of the volume inside the head; (b): a cross section of (a); (c): the hexahedralization of the volume between the human head and a sphere boundary; (d): the hexahedralization of the volume inside the head.

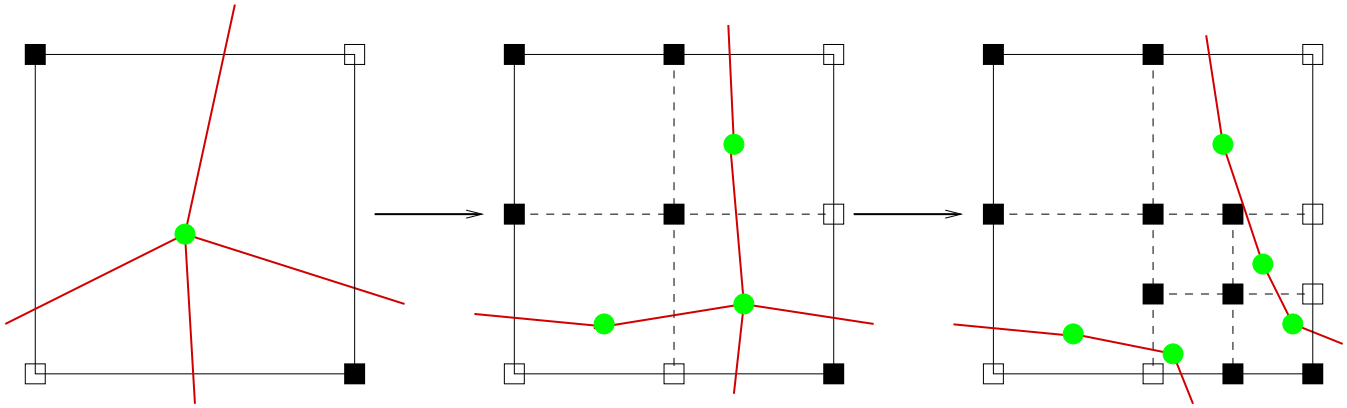


Figure 4.75: A two dimensional example on the recursive subdivision of a cubic cell in the finest level for reconstructing a dual contour with correct topology.

function because we use the trilinear interpolation. If a sub-cube has no ambiguity and hence contains a manifold dual contour, then the topology of the dual contour in the sub-cube is correct in the sense that the dual contour is topologically equivalent to the real isosurface. Therefore, if every sub-cube contains a manifold dual contour, then the dual contour in each sub-cube has correct topology. Since we recursively subdivide a cube until every sub-cube has a manifold dual contour, the resulting dual contour within the finest cubic cell has correct topology.

In this way, we can obtain a dual contour with correct topology from every finest cell in an octree structure. Then we traverse the octree in a bottom-up manner to get an adaptively simplified dual contour. During the traverse from children cells to a parent cell, the topology of a dual contour can change. This may not be desirable. The paper [126] described an algorithm to check whether the fine dual contour is topologically equivalent to the coarse one or not. The sign of the middle vertex of a coarse edge/face/cube must be the same as the sign of at least one vertex of the edge/face/cube which contains the middle vertex. We restrict the octree simplification process to preserve the topology by using their algorithm.

Figure 4.76 shows an example of a finest-level cell with an ambiguous case. A non-manifold dual contour is generated in the cell (a). However, the real isosurface in the cell can have either two disks (b) or a tunnel shape (c)(d) depending on an isovalue. (b) and (c) show two results of a dual contouring after the cell subdivision on (a). The tunnel shape is correctly reconstructed as shown in (d). (e) and (f) show a real example on topologically correct reconstruction of a knee surface.

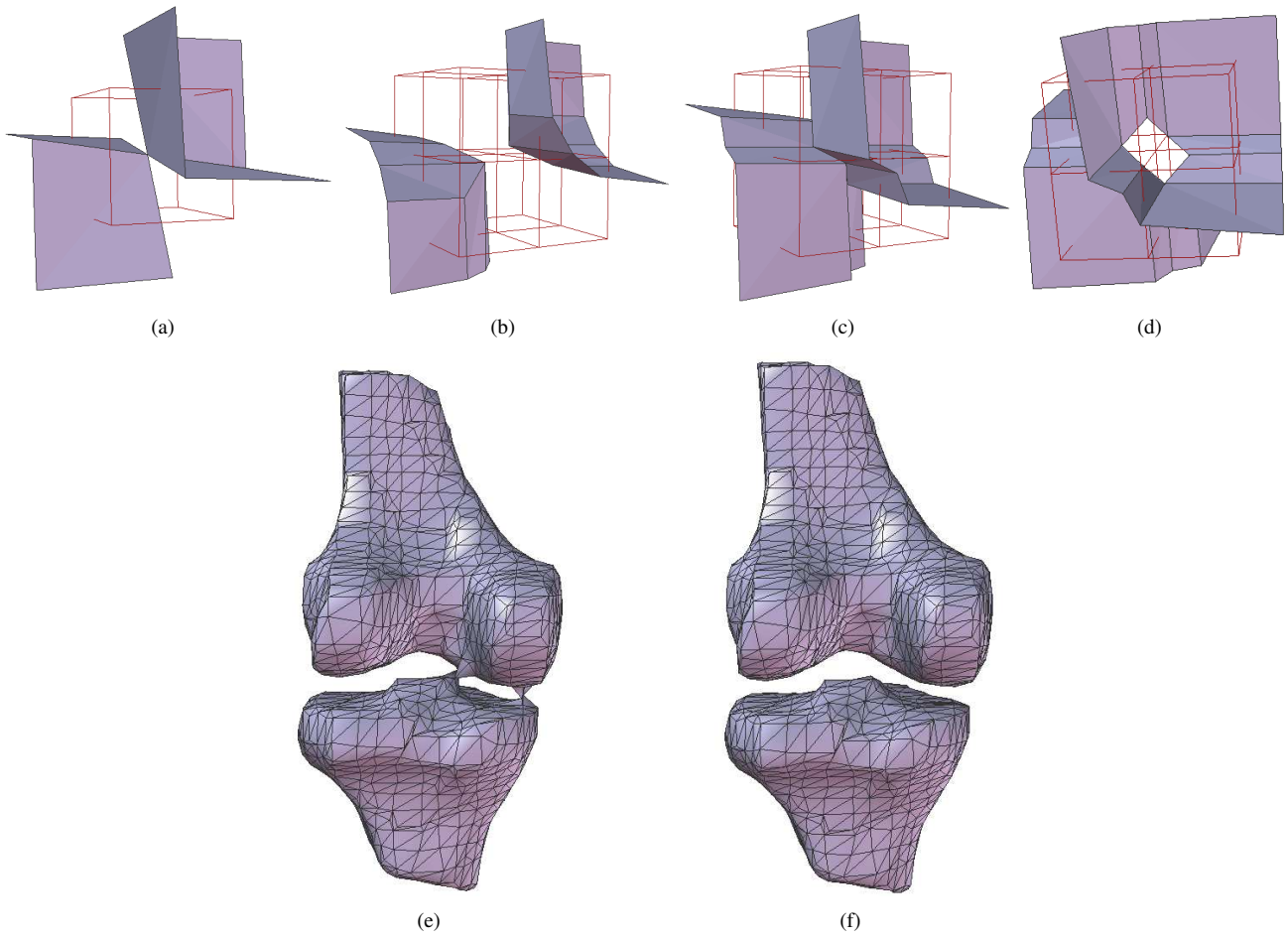


Figure 4.76: (a)-(d): An example of an ambiguous case for a finest-level cell. Both the front-right-up and back-left-down vertices have the positive sign, all the other vertices have the negative sign. (a): A non-manifold dual contour is generated in a cube with an ambiguous case. In this case, the real isosurface in the cube is topologically equivalent to either two simple disks (b) or a tunnel (c) [152]. The topologically correct dual contours, (b) and (c), can be constructed by the simple subdivision of a cell in (a). (d) is displayed in a different view angle from (c) to emphasize that the tunnel shape is correctly reconstructed. (e)-(f): A real example (the human knee) on topologically correct reconstruction of a dual contour. (e): before subdivision, (f): after subdivision. Note that non-manifolds in (e) are removed in (f).

Feature Sensitive Adaptation

Finite element applications require a minimal number of elements while preserving important features on boundary surfaces for efficient and accurate calculations. For a given precision requirement, uniform meshes are always over-sampled with unnecessary small elements. Adaptive meshes are therefore preferable.

The level adaptivity can be controlled manually by regions or automatically by using an error function and a tolerance. For example, in the calculation of ligand binding rate constants on mAChE data (Figure 4.83), the geometric accuracy on the cavity area mostly affects the accuracy of the calculation. Therefore, we refine the cavity area as much as possible, while keep coarse meshes in other regions. In most cases, we want to control the level adaptivity with an error tolerance ε. For this purpose, we use an error function (Equation 5) which approximates the difference between isosurfaces defined from two neighboring levels. Large geometric change of surfaces is considered as features. The error is measured as large in a region which contains important features, therefore the features are not easily lost during the process of adaptive simplification. Similar error metric is used in [181]. The details of the difference approximation can be found in [223].

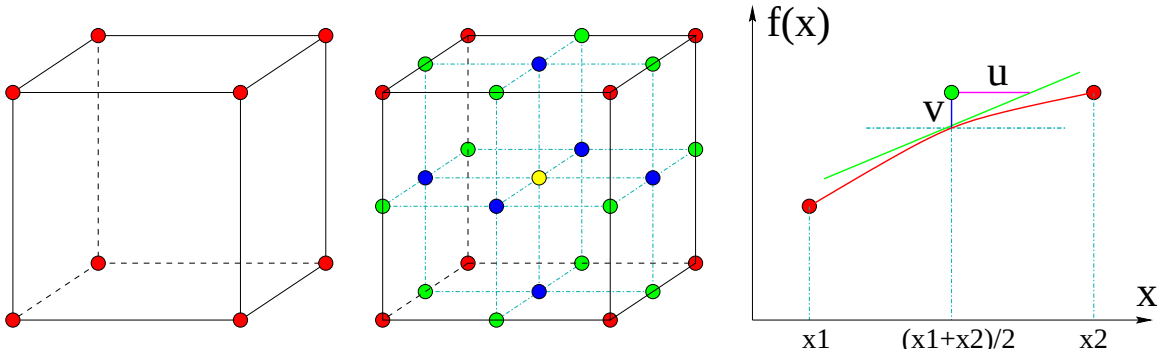


Figure 4.77: Sampling points for the feature sensitive error function (Left - Level (i); Middle - Level (i+1).) and the isosurface error calculation in 1D (Right). In the right picture, the red curve represents the trilinear function in Level (i) (it becomes to a straight line in 1D), and the green straight line represents the tangent line of the trilinear function at the middle point.

In Figure 4.77, the left picture shows eight red vertices for resolution level (i), and the right picture shows twelve green edge middle points, six blue face middle points and one yellow center point besides those eight red vertices for level (i+1). For level (i), the eight red vertices' function values are given, and a trilinear function is defined in Equation (4), from which the function values of 12 edge middle points (green), 6 face middle points (blue) and 1 center point (yellow) can be obtained. For level (i+1), the function values of all vertices and all middle points are given. We want to estimate the difference of the isosurface between the two neighboring levels.

$$\begin{aligned} f^i(x,y,z) &= f_{000}(1-x)(1-y)(1-z) + f_{001}(1-x)(1-y)z \\ &+ f_{010}(1-x)y(1-z) + f_{011}(1-x)yz \\ &+ f_{100}x(1-y)(1-z) + f_{101}x(1-y)z \\ &+ f_{110}xy(1-z) + f_{111}xyz \end{aligned} \tag{5.42}$$

$$Error = \sum \frac{|f^{i+1} - f^i|}{|\nabla f^i|} \tag{5.43}$$

The error function is defined in Equation (5). The right picture of Figure 4.77 shows the calculation of the isosurface error U at the green middle point in 1D. The blue line segment V represents the difference of the function values at the middle point, or $V = |f^{i+1} - f^i|$. The isosurface error U can be computed by $U = V/k$, where k is the slope of the green tangent line. In higher dimensions, the slope k becomes the magnitude of the gradient. In Equation (5), we only need to sum the isosurface error over all the middle points, including edge middle points, face middle points and the center point, since the function values at the eight vertices are the same for the two neighboring levels.

We still use QEF to calculate minimizer points, which are connected to approximate the isosurfaces, so we can also preserve sharp edges and corners as shown in Figure 4.78 (a) and (b). QEF and the error function in Equation (5) are compared in

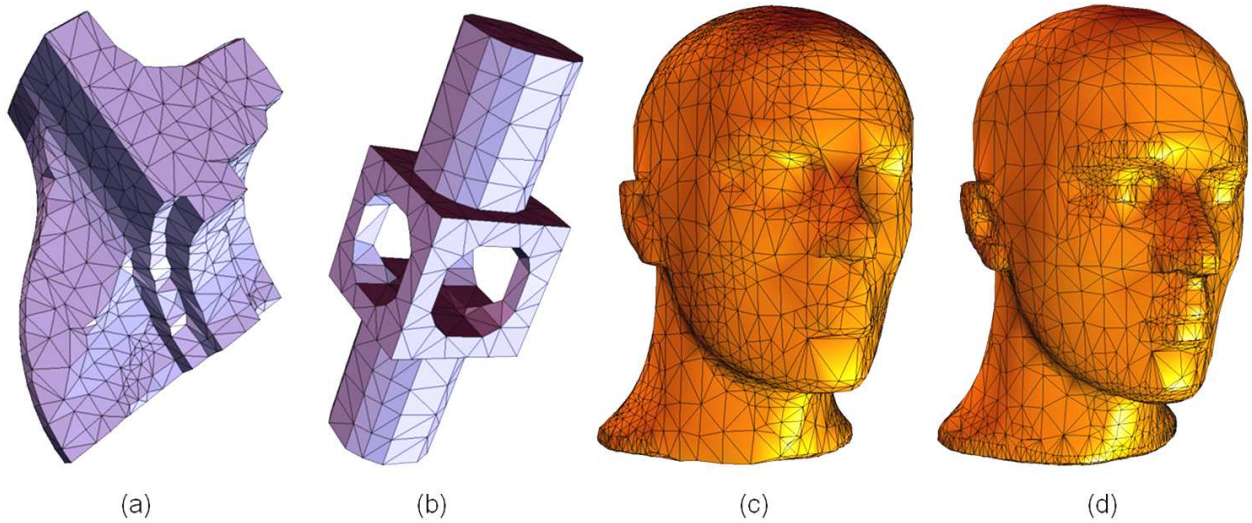


Figure 4.78: (a) , (b) : Tetrahedral meshes of (a) fandisk and (b) mechanical part. Note that sharp edges and corners are accurately reconstructed ; (c) , (d) : the comparison of QEF ((c), 2952 triangles) and the feature sensitive error function ((d), 2734 triangles). The facial features are better refined in (d).

controlling the level adaptivity for the human head model (Figure 4.78, (c) and (d)), and Equation (5) yields more sensitive adaptivity for facial features, like the areas of nose, eyes, mouth and ears. The feature sensitive adaptivity is important for finite element meshes to identify and preserve necessary geometric and topological properties of the object while minimizing the number of elements.

Quality Improvement

Poorly shaped elements influence the convergence and stability of the numerical solutions. Since the extracted meshes may have such undesirable elements, we need an additional step for mesh quality improvement. First we need to define criteria to measure the mesh quality. Various functions have been chosen to measure the quality of a mesh element. For example, Freitag [95] defined poor quality tetrahedra using dihedral angles. George [105] chose the ratio of the element diameter such as the longest edge over the in-radius. We use the edge-ratio, the Joe-Liu parameter [147], and a minimum volume bound. These quality metrics are used to detect slivers and sharp elements which need to be removed. With these measures, the mesh quality can be judged by observing the worst element quality, and the distribution of elements in terms of their quality values.

- **Edge-ratio:** The edge-ratio of a tetrahedron is the ratio of the longest edge length over the shortest edge length.
- **Joe-Liu parameter:** Let $|s|$ denote the volume (which may be negative) of a given d-simplex 's', $v_i (i = 0, ..., d)$ denote the vertices of s, and e_{ij} denote the edges of s that connect v_i to v_j. We compute the Joe-Liu parameter of a d-simplex s as:

$$F(s,d) = \frac{f(s,d)}{g(s,d)} = \frac{2^{2(1-\frac{1}{d})} \times 3^{\frac{d-1}{2}} \times |s|^{\frac{2}{d}}}{\sum_{0 \le i < j \le 3} |e_{ij}|^2} \tag{5.44}$$

  For tetrahedral meshes, d is set to be 3. $F(s,3)$ is normalized to yield maximal value of 1 for the unit tetrahedron (with volume 1/6).

- **Minimum volume bound:** Calculate the volume for each tetrahedron and find the minimum volume of the extracted mesh. This volume parameter needs to be improved if the minimum volume is less than a threshold (e.g. 1.0^{-4}).

We perform iterative edge contractions to improve the worst edge-ratio. For each iteration, we remove the tetrahedron with the maximum edge-ratio by contracting the shortest edge. We keep removing the tetrahedron with the maximum edge-ratio until

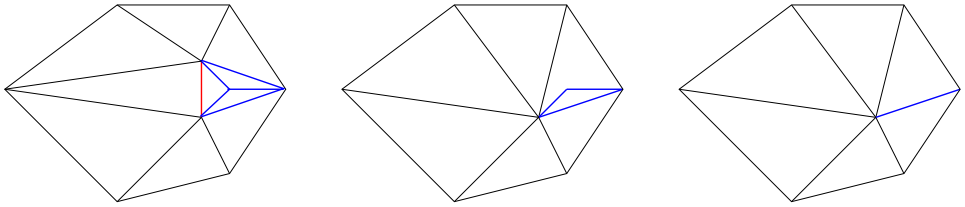


Figure 4.79: A special case for the edge-contraction method. Left - the original mesh, the red edge is to be contracted; Middle - the red edge is contracted; Right - the additional triangles are removed.

the maximum edge-ratio is below the given threshold. During the edge contraction, we merge an interior vertex to a boundary vertex, an interior vertex to another interior vertex, or a boundary vertex to another boundary vertex. A special case may occur when we contract edges as shown in Figure 4.79, where two blue triangles (left picture) degenerate into the same triangle, and which should also be removed from the mesh. The special case can be detected by checking the number of elements sharing the same edge (2D) or face (3D). If there are more than two elements sharing the same edge/face, then the special case occurs. This can also be removed by contracting one shared edge. If edge contraction is not enough to arrive the threshold, then the longest edge bisection method is used to continue reducing the largest edge-ratio. But the number of vertices and the number of elements will increase.

In the mesh extraction process, a pyramid/diamond may be generated when we analyze the sign change edge/the interior edge in boundary cells. There are two or three ways to decompose a pyramid/diamond into tetrahedra, we compare the worst Joe-Liu parameters for different splitting ways and choose the better one. This process is executed when we extract meshes, therefore our mesh generation method tends to produce meshes with good overall quality.

Finally smoothing techniques are used to improve the Joe-Liu parameter and the minimum volume. The simplest discretization of the Laplacian operator for a node is the average of all its neighbors. Laplacian smoothing is an efficient heuristic, but it is possible to produce an invalid mesh containing inverted elements or elements with negative volume. We choose a 'smart' Laplacian smoothing [95], which relocates the point only if the quality of the local mesh is improved.

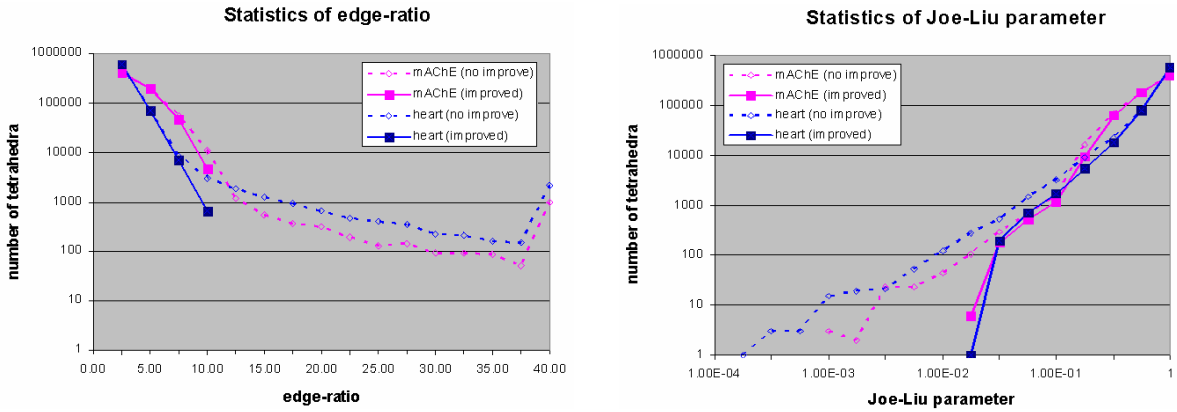


Figure 4.80: The histogram of edge-ratios (left) and Joe-Liu parameters (right) for mAChE and the human heart. The number of tetrahedra at edge-ratio 40.0 represents the number of all the elements, whose edge-ratios are greater than 40.0.

The histogram of edge-ratios and Joe-Liu paramters (Figure 4.80) shows the overall quality of extracted tetrahedral meshes for a biomolecule mAChE (Figure 4.83) and the heart model (Figure 4.84). By comparing the two quality metrics before and after applying techniques of quality improvement, we can see the worst parameters are improved significantly. Figure 4.81 shows the improvement of the worst values of the edge-ratio, the Joe-Liu parameter and the minimum volume.

Results and Applications

We have developed an interactive program for 3D mesh extraction and rendering from volume data. In the program, error tolerances and isovalues can be changed interactively. Our results were computed on a PC equipped with a Pentium III 800

	Vertex Number	Tetra Number	Edge-ratio (best, worst)	Joe-Liu (best, worst)	Volume (minimal, maximal)
mAChE^b	124180	670642	(1.03, 1.19×10^4)	(1.0, 6.24×10^{-4})	(2.43×10^{-7}, 9.40×10^6)
mAChE^a	121670	656823	(1.03, 8.5)	(1.0, 1.73×10^{-2})	(1.40×10^{-4}, 9.40×10^6)
Heart^b	140425	689020	(1.02, 1.20×10^5)	(1.0, 1.23×10^{-4})	(4.88×10^{-8}, 2.56×10^2)
Heart^a	138072	676494	(1.02, 8.5)	(1.0, 1.99×10^{-2})	(4.39×10^{-4}, 4.31×10^2)

Figure 4.81: The comparison of the three quality criteria (the edge-ratio, the Joe-Liu parameter and the minimal volume) before/after the quality improvement for mAChE and the human heart model. DATA^b – before quality improvement; DATA^a – after quality improvement.

Dataset	Type	Size	Number of Tetrahedra (Extraction Time (unit : ms))			
			(a)	(b)	(c)	(d)
mAChE	Given	257^3	670642 (10891)	–	–	–
Heart	SDF	257^3	689020 (11110)	–	–	–
Skin	CT	129^3	935124 (17406)	545269 (10468)	–	–
Skull	CT	129^3	–	–	579834 (10203)	166271 (3063)
Poly	CT	257^3	–	276388 (5640)	63325 (1672)	14204 (672)
Head	SDF	65^3	143912 (2547)	76218 (1391)	40913 (766)	10696 (203)
Knee	SDF	65^3	70768 (1360)	94586 (1782)	93330 (1750)	72366 (1406)

Figure 4.82: Datasets and Test Results. The CT data sets are re-sampled to fit into the octree representation. Rendering results for each case is shown in Figure 4.83, 4.84, 4.66, 4.85, 4.86 and 4.87. the Skull and Skin isosurfaces are extracted from the UNC Head model.

MHz processor and 1 GB main memory.

Our algorithm has been used to generate tetrahedral meshes for a molecular dataset (mAChE) and the human heart model in two projects. We also tested our algorithm on volumetric data from CT scans, the *UNC Human Head* and *Poly (heart valve)*, and signed distance volumes generated from the polygonal surfaces of a human head and a knee. Figure 4.82, 4.83, 4.84, 4.66, 4.85, 4.86 and 4.87 provide information about datasets and test results. The results consist of the number of tetrahedra, extraction time, and corresponding images with respect to different isovalues and error tolerances. As a preprocessing, we calculate min/max values for each octree cell to visit only cells contributing to mesh extraction and to compute QEF values only in those cells at run time. Extraction time in the table includes octree traversal, QEF computation and actual mesh extraction, given isovalues and error tolerance values for inner and outer surfaces as run time parameters. If we fix isovalues, and change error tolerance interactively, the computed QEF is reused and thus the whole extraction process is accelerated. The results show that the mesh extraction time scales linearly with the number of elements in the extracted mesh.

Figure 4.83 shows the extracted adaptive tetrahedral mesh of mAChE, which has been used as the geometric model in solving the steady-state Smoluchowski equation to calculate ligand binding rate constants using the finite element method (FEM) [218]. The most important part in the geometric structure of mAChE is the cavity, where fine meshes are required. We first find the position of the cavity, then control the adaptivity according to it. The area of the cavity is kept the finest level, while coarser meshes are obtained everywhere else. After improving the mesh quality, convergent results have been obtained in the finite element calculation, and they match with some experimental results well.

A good geometric model of the human heart is important for the simulation of the human cardiovascular system, which is very useful for the predictive medicine applications in cardiovascular surgery. To extract 3D meshes from the surface heart model provided by New York University, we computed the signed distance function from the surface data and performed the mesh extraction (Figure 4.84). An adaptive and quality tetrahedral mesh is extracted with correct topology and feature sensitive adaptation. Meshes are refined in the areas of heart valves, while coarse meshes are kept for other regions.

Figure 4.85 and 4.86 show the resulting tetrahedral meshes extracted from two signed distance function data with the size of 65^3. The results from CT data are shown in Figure 4.66 (skull, skin) and 4.87 (heart valve). The number of elements in the extracted mesh is controlled by changing error tolerance. It is clear that adaptive tetrahedral meshes are extracted from the interval volume, and facial features are identified sensitively and preserved (Figure 4.85). In Figure 4.86, the sequence of images are generated by changing the isovalue of the inner isosurface. The topology of the inner isosurface can change

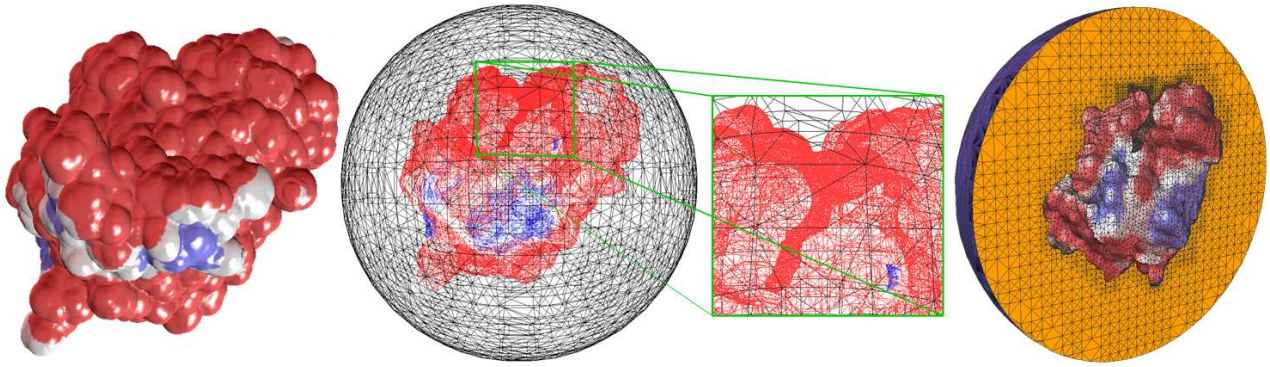


Figure 4.83: mAChE – (a): the isosurface at 0.5 from the accessibility function; (b): the wire frame of the isosurface and an outer sphere; (c): a zoomed-in picture to show the refined cavity; (d): the adaptive tetrahedral mesh of the volume between the isosurface and an outer sphere. The color on the isosurface represents the distribution of the potential function, the color map is: (-∞, -0.5) - red; [-0.5, 0.5] - white; (0.5, +∞) - blue. Note the region around the cavity has fine meshes, while other areas have relatively coarse meshes.

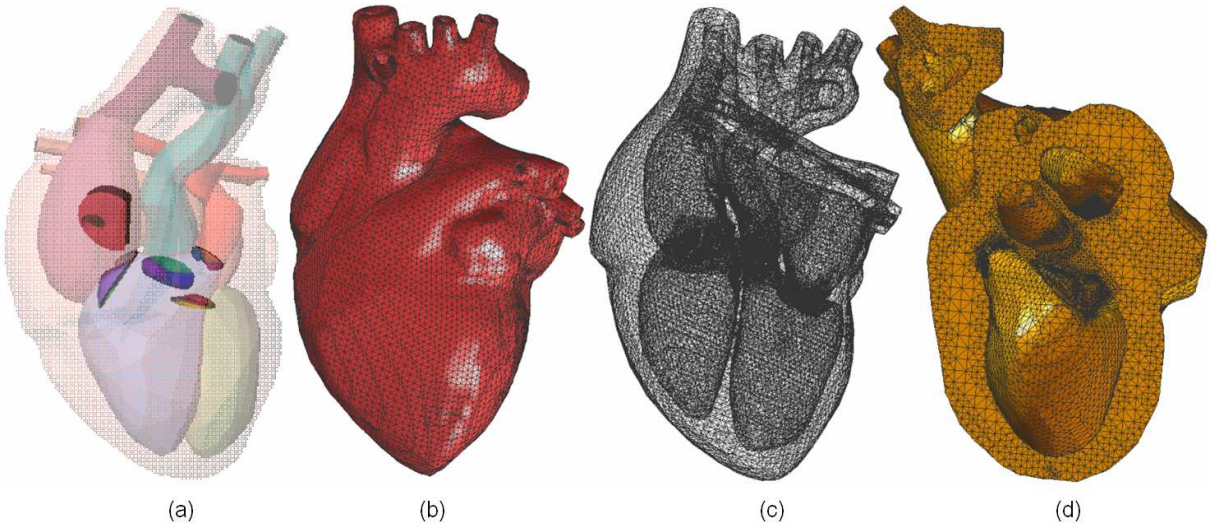
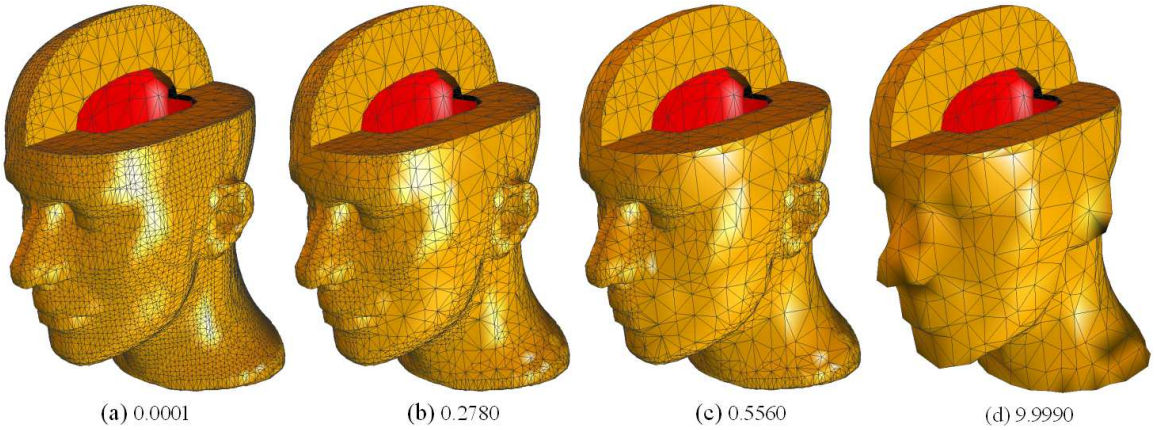


Figure 4.84: Heart (SDF) – (a): the triangular surface of a human heart with valves (data courtesy from New York University (NYU)); (b): The tetrahedral mesh extracted from SDF of the heart surface. The smooth surface and the wire frame on the mesh is rendered; (c): the wire frame of the extracted heart model. Note that the region of heart valves are refined; (d): Cross-section of the adaptive tetrahedral mesh.

arbitrarily.

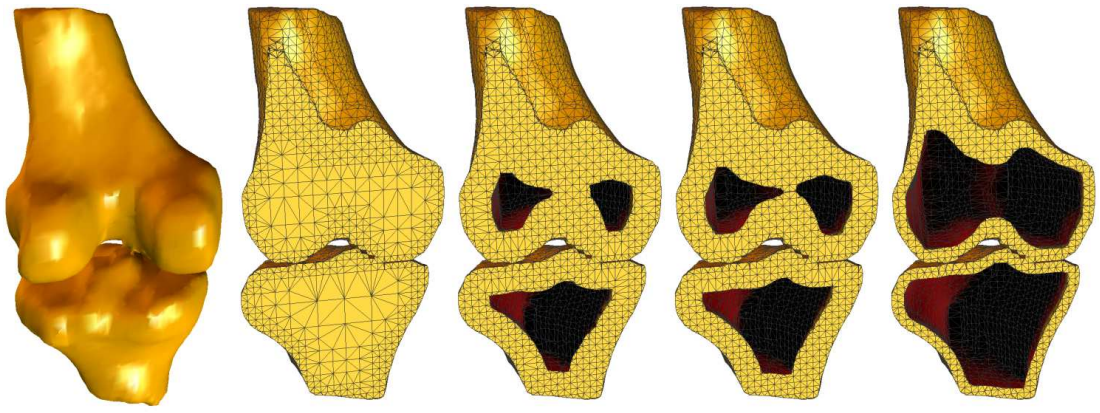
Conclusion

We have presented an algorithm to extract adaptive and high quality 3D meshes directly from volumetric imaging data. By extending the dual contouring method described in [126], our method can generate 3D meshes with good properties such as no hanging nodes, sharp feature preservation and good aspect ratio. Using an error metric which is normalized by the function gradient, the resolution of the extracted mesh is adapted to the features sensitively. The resulting meshes are useful for efficient and accurate finite element calculations.



(a) 0.0001 (b) 0.2780 (c) 0.5560 (d) 9.9990

Figure 4.85: Head (SDF) – Isovalues α_{in} = -9,17464, α_{out} = 0.0001; error tolerances ε_{in} = 1.7, ε_{out} are listed below each picture.



(a) –7.70715 (b) –4.17492 (c) –3.79098 (d) –1.71771

Figure 4.86: Knee (SDF) – Error tolerances $\varepsilon_{in} = \varepsilon_{out}$ = 0.0001; isovalues α_{out} = -0.02838, α_{in} are listed below each picture.

The following algorithm is taken from [269] (zhang2009surface).

Introduction

The quality of unstructured quadrilateral/hexahedral meshes plays an important role in finite element simulations. Although a lot of efforts have been made, it still remains a challenging problem to generate quality quad/hex meshes for complicated structures such as the biomolecule Ribosome 30S shown in Figure 4.88. We have described an isosurface extraction method to

(a) volume rendering (b) 1.852 (c) 3.549 (d) 9.999

Figure 4.87: Heart Valve (Poly) – Isovalues α_{in} = 1000.0, α_{out} = 75.0; error tolerances ε_{in} = 0.0001, ε_{out} are listed below each picture.

generate quad/hex meshes for arbitrary complicated structures from volumetric data and utilized an optimization-based method to improve the mesh quality [267] [264] [268], but the surface needs to be smoothed and the mesh quality needs to be further improved.

Geometric partial differential equations (GPDEs) such as Laplacian smoothing have been extensively used in surface smoothing and mesh quality improvement. There are two main methods in solving GPDEs, the finite element method (FEM) and the finite difference method (FDM). Although FDM is not robust sometimes, people still prefer to choosing FDM instead of FEM because FDM is simpler and easy to implement. Recently, a discretized format of the Laplacian-Beltrami (LB) operator over triangular meshes was derived and used in solving GPDEs [159] [252] [249]. In this paper, we will discretize the LB operator over quadrilateral meshes, and discuss an approach to apply the discretizated format on surface smoothing and quality improvement for quadrilateral or hexahedral meshes.

The main steps to smooth the surface and improve the quality of quadrilateral and hexahedral meshes are as follows:

1 Discretizing the LB operator and denoising the surface mesh - vertex adjustment in the normal direction with volume preservation.

2 Improving the aspect ratio of the surface mesh - vertex adjustment in the tangent direction with feature preservation.

3 Improving the aspect ratio of the volumetric mesh - vertex adjustment inside the volume.

For quadrilateral meshes, generally only Step 1 and Step 2 are required, but all the three steps are necessary for surface smoothing and quality improvement of hexahedral meshes.

Unavoidly the quadrilateral or hexahedral meshes may have some noise over the surface, therefore the surface mesh needs to be smoothed. In this paper, we derive a discretized format of the LB operator, and choose the surface diffusion flow (Equation (5.45)) to smooth the surface mesh by relocating vertices along their normal directions. The surface diffusion flow is volume preserving and also preserves a sphere accurately if the initial surface mesh is embedded and close to a sphere, therefore it is especially suitable for surface smoothing of biomolecular meshes since biomolecules are usually modelled as a union of hard spheres.

The aspect ratio of the surface mesh can be improved by adjusting vertices in the tangent plane, and surface features is preserved since the movement in the tangent plane doesn't change the surface shape ([196], page 72). For each vertex, the mass center is calculated to find its new position on the tangent plane. Since the vertex tangent movement is an area-weighted relaxation method, it is also suitable for adaptive quadrilateral meshes.

Besides the movement of surface vertices, interior vertices also need to be relocated in order to improve the aspect ratio of hexahedral meshes. The mass center is calculated as the new position for each interior vertex.

Although our relaxation-based method can not guarantee that no inverted element is introduced for arbitrary input meshes, it works well in most cases with the properties of noise removal, feature preservation, mesh quality improvement. Furthermore, it is especially suitable for surface smoothing and quality improvement of biomolecular meshes. As the 'smart' Laplacian smoothing [51] [95], this method is applied only when the mesh quality is improved in order to avoid inverted elements. This method can also be combined with the optimization-based method to obtain a high quality mesh with relatively less computational cost.

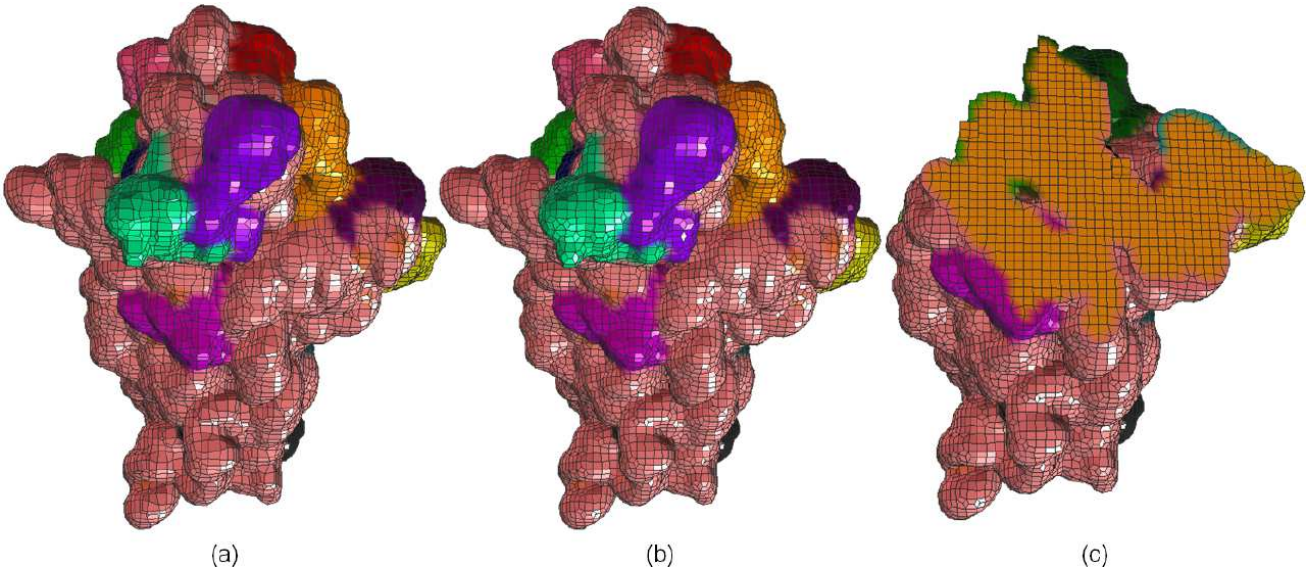


Figure 4.88: The comparison of mesh quality of Thermus Thermophilus small Ribosome 30S (1J5E) crystal subunit. The pink color shows 16S rRNA and the remaining colors are proteins. (a) the original quadrilateral mesh (13705 vertices, 13762 quads); (b) the improved quadrilateral mesh; (c) the improved hexahedral mesh (40294 vertices, 33313 hexes).

Previous Work

It is well-known that poor quality meshes result in poorly conditioned stiffness matrices in finite element analysis, and affect the stability, convergence, and accuracy of finite element solvers. Therefore, quality improvement is an important step in mesh generation.

Some quality improvement techniques of triangular and tetrahedral meshes, such as the edge-contraction method, can not be used for quadrilateral and hexahedral meshes because we do not want to introduce any degenerated elements. Therefore, the mesh smoothing methods are selected to improve the quality of quad/hex meshes by adjusting the vertex positions in the mesh while preserving its connectivity. As reviewed in [179] [226], Laplacian smoothing and optimization are the two main quality improvement techniques.

As the simplest and most straight forward method for node-based mesh smoothing, Laplacian smoothing relocates the vertex position at the average of the nodes connecting to it [93]. There are a variety of smoothing techniques based on a weighted average of the surrounding nodes and elements [107] [277] [214]. The averaging method may invert or degrade the local quality, but it is computationally inexpensive and very easy to implement, so it is in wide use. Winslow smoothing is more resistant to mesh folding because it requires the logical variables are harmonic functions [132].

Instead of relocating vertices based on a heuristic algorithm, people utilized an optimization technique to improve mesh quality. The optimization algorithm measures the quality of the surrounding elements to a node and attempts to optimize it [97]. The algorithm is similar to a minimax technique used to solve circuit design problems [57]. Optimization-based smoothing yields better results but it is more expensive than Laplacian smoothing, and it is difficult to decide the optimized iteration step length. Therefore, a combined Laplacian/optimization-based approach [51] [95] [96] was recommended. Physically-based simulations are used to reposition nodes [148]. Anisotropic meshes are obtained from bubble equilibrium [211] [47].

When we use the smoothing method to improve the mesh quality, it is also important to preserve surface features. Baker [36] presented a feature extraction scheme which is based on estimates of the local normals and principal curvatures at each mesh node. Local parametrization was utilized to improve the surface mesh quality while preserving surface characteristics [102], and two techniques called trapezium drawing and curvature-based mesh improvement were discussed in [202].

Staten et al. [222] [131] proposed algorithms to improve node valence for quadrilateral meshes. One special case of cleanup in hexahedral meshes for the whisker weaving algorithm is presented in [163]. Schneiders [200] proposed algorithms and a series of templates for quad/hex element decomposition. A recursive subdivision algorithm was proposed for the refinement of hex meshes [22].

Quadrilateral Mesh

Noise may exist in quadrilateral meshes, therefore we need to smooth the surface mesh. The quality of some quadrilateral meshes may not be good enough for finite element calculations, and the aspect ratio also needs to be improved.

There are two steps for the surface smoothing and the quality improvement of quadrilateral meshes: (1) the discretization of Laplace-Beltrami opertor and the vertex movement along its normal direction to remove noise, (2) the vertex movement on its tangent plane to improve the aspect ratio while preserving surface features.

Geometric Flow Various geometric partial differential equations (GPDEs), such as the mean curvature flow, the surface diffusion flow and Willmore flow, have been extensively used in surface and imaging processing [252]. Here we choose the surface diffusion flow to smooth the surface mesh,

$$\frac{\partial x}{\partial t} = \Delta H(x)\vec{n}(x). \tag{5.45}$$

where Δ is the Laplace-Beltrami (LB) operator, H is the mean curvature and $\vec{n}(x)$ is the normal vector at the node x. In [91], the existence and uniqueness of solutions for this flow was discussed, and the solution converges exponentially fast to a sphere if the initial surface is embedded and close to a sphere. It was also proved that this flow is area shrinking and volume preserving [252].

In applying geometric flows on surface smoothing and quality improvement over quadrilateral meshes, it is important to derive a discretized format of the LB operator. Discretized schemes of the LB operator over triangular meshes have been derived and utilized in solving GPDEs [159] [252] [249].

A quad can be subdivided into triangles, hence the discretization schemes of the LB operator over triangular meshes could be easily used for quadrilateral meshes. However, since the subdivision of each quad into triangles is not unique (there are two ways), the resulting discretization scheme is therefore not unique. Additionally in the discretization scheme, the element area needs to be calculated. If we choose to split each quad into two triangles and calculate the area of a quad as the summation of the area of two triangles, then the area calculated from the two different subdivisions could be very different because four vertices of a quad may not be coplanar. Therefore, a unique discretized format of the LB operator directly over quad meshes is required.

Discretized Laplace-Beltrami Operator Here we will derive a discretized format for the LB operator over quadrilateral meshes. The basic idea of our scheme is to use the bilinear interpolation to derive the discretized format and to calculate the area of a quad. The discretization scheme is thus uniquely defined.

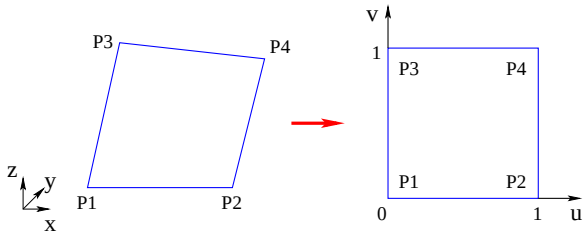


Figure 4.89: A quad $[p_1p_2p_4p_3]$ is mapped into a bilinear parametric surface.

Area Calculation: Let $[p_1p_2p_4p_3]$ be a quad in $\mathbb{R}^3$, then we can define a bilinear parametric surface S that interpolates four vertices of the quad as shown in Figure 4.89:

$$\begin{aligned} S(u,v) &= (1-u)(1-v)p_1 + u(1-v)p_2 \\ &+ (1-u)vp_3 + uvp_4. \end{aligned} \tag{5.46}$$

The tangents of the surface are

$$S_u(u,v) = (1-v)(p_2-p_1) + v(p_4-p_3), \tag{5.47}$$

$$S_v(u,v) = (1-u)(p_3-p_1) + u(p_4-p_2). \tag{5.48}$$

Let ∇ denote the gradient operator about the (x,y,z) coordinates of the vertex P_1, then we have

$$\begin{aligned} \nabla S_u(u,v) &= -(1-v), \quad (5.49)\\ \nabla S_v(u,v) &= -(1-u). \quad (5.50) \end{aligned}$$

Let A denote the area of the surface $S(u,v)$ for $(u,v) \in [0,1]^2$, then we have

$$\begin{aligned} A &= \int_0^1 \int_0^1 \sqrt{\| S_u \times S_v \|^2} du dv \\ &= \int_0^1 \int_0^1 \sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2} du dv. \end{aligned} \quad (5.51)$$

It may not be easy to obtain the explicit form for integrals in calculating the area, numerical integration quadrature could be used. Here we use the following four-point Gaussian quadrature rule to compute the integral

$$\int_0^1 \int_0^1 f(u,v) du dv \approx \frac{f(q_1)+f(q_2)+f(q_3)+f(q_4)}{4}, \quad (5.52)$$

where

$$\begin{aligned} q^- = \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad q^+ = \frac{1}{2} + \frac{\sqrt{3}}{6}, \\ q_1 = (q^-, q^-), \quad q_2 = (q^+, q^-), \\ q_3 = (q^-, q^+), \quad q_4 = (q^+, q^+). \end{aligned}$$

The integration rule in Equation (5.52) is of $O(h^4)$, where h is the radius of the circumscribing circle.

Discretized LB Operator: The derivation of the discretized format of the LB operator is based on a formula in differential geometry [159]:

$$\lim_{diam(R)\to 0} \frac{\nabla A}{A} = \vec{H}(p), \quad (5.53)$$

where A is the area of a region R over the surface around the surface point p, $diam(R)$ denotes the diameter of the region R, and $\vec{H}(p)$ is the mean curvature normal.

From Equation (5.51), we have

$$\begin{aligned} \nabla A &= \int_0^1 \int_0^1 \nabla \sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2} du dv \\ &= \int_0^1 \int_0^1 \frac{S_u (S_v, (v-1)S_v - (u-1)S_u))}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \\ &+ \int_0^1 \int_0^1 \frac{S_v (S_u, (u-1)S_u - (v-1)S_v)}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \\ &= \alpha_{21}(p_2 - p_1) + \alpha_{43}(p_4 - p_3) \\ &+ \alpha_{31}(p_3 - p_1) + \alpha_{42}(p_4 - p_2), \end{aligned} \quad (5.54)$$

where

$$\begin{aligned} \alpha_{21} &= \int_0^1 \int_0^1 \frac{(1-v)(S_v, (v-1)S_v - (u-1)S_u))}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \\ \alpha_{43} &= \int_0^1 \int_0^1 \frac{v(S_v, (v-1)S_v - (u-1)S_u))}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \\ \alpha_{31} &= \int_0^1 \int_0^1 \frac{(1-u)(S_u, (u-1)S_u - (v-1)S_v)}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \\ \alpha_{42} &= \int_0^1 \int_0^1 \frac{u(S_u, (u-1)S_u - (v-1)S_v)}{\sqrt{\| S_u \|^2 \| S_v \|^2 - (S_u, S_v)^2}} du dv \end{aligned}$$

∇A could be written as

$$\nabla A = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 + \alpha_4 p_4 \tag{5.55}$$

with

$$\begin{aligned} \alpha_1 &= -\alpha_{21} - \alpha_{31}, \quad & \alpha_2 &= -\alpha_{21} - \alpha_{42}, \\ \alpha_3 &= -\alpha_{31} - \alpha_{43}, \quad & \alpha_4 &= -\alpha_{43} - \alpha_{42}. \end{aligned} \tag{5.56}$$

Here we still use the four-point Gaussian quadrature rule in Equation (5.52) to compute the integrals in the α_{ij}. It follows from Equation (5.56) that $\sum_{i=1}^{4} \alpha_i = 0$, we have

$$\nabla A = \alpha_2(p_2 - p_1) + \alpha_3(p_3 - p_1) + \alpha_4(p_4 - p_1). \tag{5.57}$$

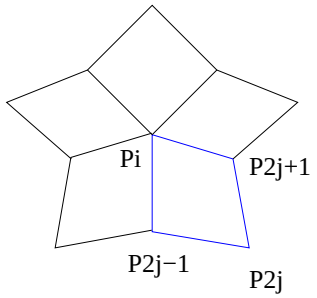


Figure 4.90: A neighboring quad $[p_i p_{2j-1} p_{2j} p_{2j+1}]$ around the vertex p_i.

Now let p_i be a vertex with valence n, and p_{2j} $(1 \le j \le n)$ be one of its neighbors on the quadrilateral mesh, then we can define three coefficients α_2, α_3, α_4 as in (5.57). Now we denote these coefficients as α_j^i, β_j^i and γ_j^i for the quad $[p_i p_{2j-1} p_{2j} p_{2j+1}]$ as shown in Figure 4.90. By using Equation (5.57), the discrete mean curvature normal can be defined as

$$\begin{aligned} \vec{H}(p_i) &\approx \frac{1}{A(p_i)} \sum_{j=1}^{n} [\alpha_j^i (p_{2j-1} - p_i) \\ &+ \beta_j^i (p_{2j+1} - p_i) + \gamma_{j+1}^i (p_{2j} - p_i)] \\ &= \sum_{k=1}^{2n} w_k^i (p_k - p_i) \end{aligned} \tag{5.58}$$

where $\vec{H}(p_i)$ denotes the mean curvature normal, $A(p_i)$ is the total area of the quads around p_i, and

$$w_{2j}^i = \frac{\gamma_j^i}{A(p_i)}, \quad w_{2j-1}^i = \frac{\alpha_j^i + \beta_{j-1}^i}{A(p_i)}, \quad w_{2j+1}^i = \frac{\alpha_{j+1}^i + \beta_j^i}{A(p_i)}.$$

Using the relation $\Delta x = 2H(p_i)$ ([242], page 151), we obtain

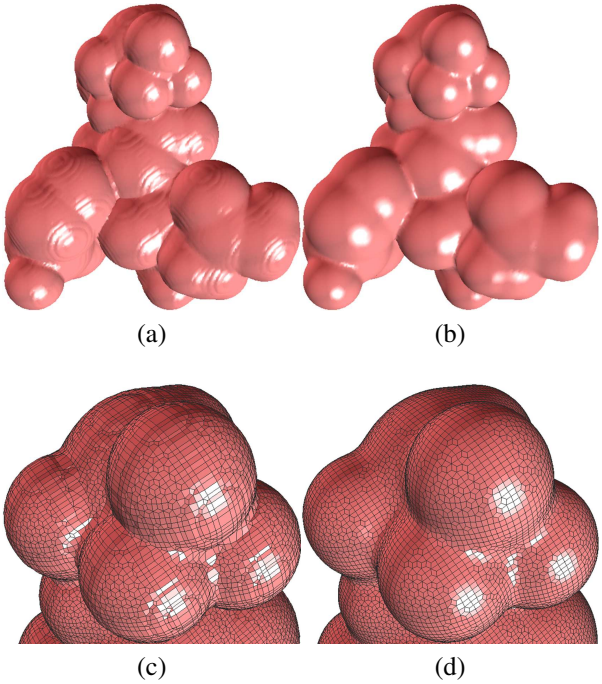
$$\Delta f(p_i) \approx 2 \sum_{k=1}^{2n} w_k^i (f(p_k) - f(p_i)). \tag{5.59}$$

Therefore,

$$\begin{aligned} \Delta H(p_i)\vec{n}(p_i) &\approx 2 \sum_{k=1}^{2n} w_k^i (H(p_k) - H(p_i))\vec{n}(p_i) \\ &= 2 \sum_{k=1}^{2n} w_k^i \left[\vec{n}(p_i)\vec{n}(p_k)^T \vec{H}(p_k) - \vec{H}(p_i) \right], \end{aligned} \tag{5.60}$$

where $\vec{H}(p_k)$ and $\vec{H}(p_i)$ are further discretized by (5.58). Note that $\vec{n}(p_i)\vec{n}(p_k)^T$ is a 3×3 matrix.

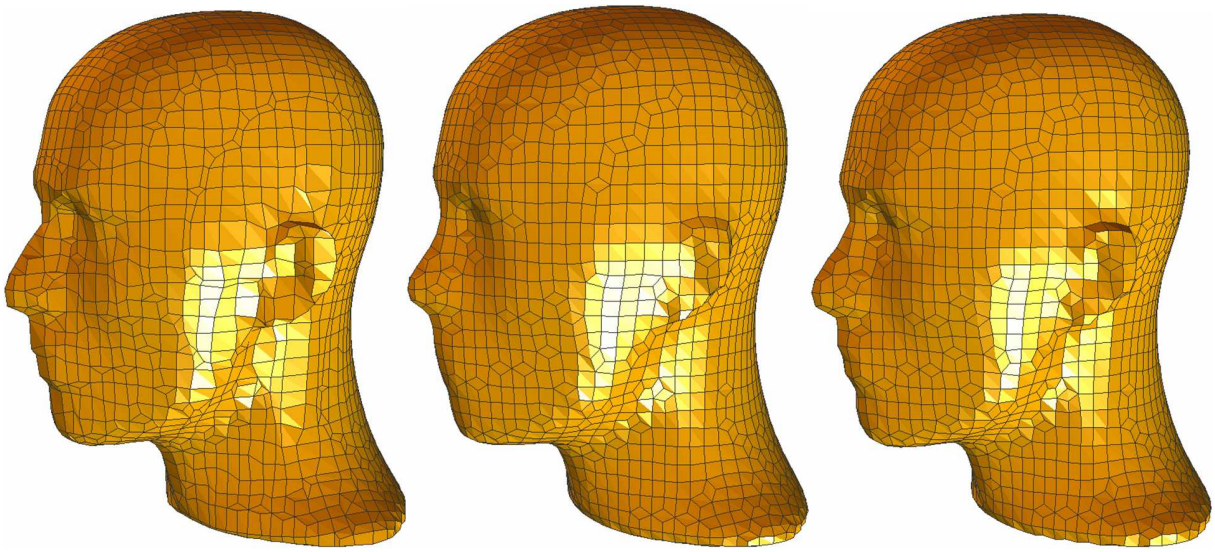
Figure 4.91 shows one example of the molecule consisting of three amino acids (ASN, THR and TYR) with 49 atoms. The molecular surface was bumpy as shown in Figure 4.91(a) since there are some noise existing in the input volumetric data, the surface becomes smooth after the vertex normal movement as shown in Figure 4.91(b).



(a) (b)

(c) (d)

Figure 4.91: Surface smoothing and quality improvement of the molecule consisting of three amino acids (ASN, THR and TYR) with 49 atoms (45534 vertices, 45538 quads). (a) and (c) - the original mesh; (b) and (d) - after surface smoothing and quality improvement.



(a) (b) (c)

Figure 4.92: The quality of a quadrilateral mesh of a human head model is improved (2912 vertices, 2912 quads). (a) The original mesh; (b) Each vertex is relocated to its mass center, some facial features are removed; (c) Only tangent movement is applied.

Tangent Movement In order to improve the aspect ratio of the surface mesh, we need to add a tangent movement in Equation (5.45), hence the flow becomes

$$\frac{\partial x}{\partial t} = \triangle H(x)\vec{n}(x) + v(x)\vec{T}(x), \tag{5.61}$$

where $v(x)$ is the velocity in the tangent direction $\vec{T}(x)$. First we calculate the mass center $m(x)$ for each vertex on the surface, then project the vector $m(x)-x$ onto the tangent plane. $v(x)\vec{T}(x)$ can be approximated by $[m(x)-x]-\vec{n}(x)^T[m(x)-x]\vec{n}(x)$ as shown in Figure 4.93.

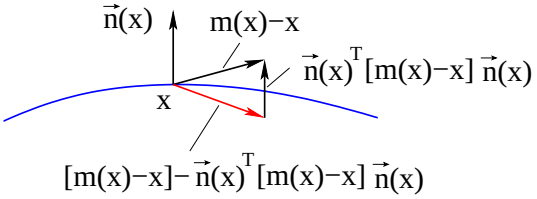


Figure 4.93: The tangent movement at the vertex x over a surface. The blue curve represents a surface, and the red arrow is the resulting tangent movement vector.

Mass Center: A mass center p of a region S is defined by finding $p \in S$, such that

$$\int_S \| y-p \|^2 \, d\sigma = min. \tag{5.62}$$

S is a piece of surface in $\mathbb{R}^3$, and S consists of quads around vertex x. Then we have

$$\sum(\frac{p_i+p_{2j-1}+p_{2j}+p_{2j+1}}{4}-p_i)A_j=0, \tag{5.63}$$

A_j is the area of the quad $[p_i p_{2j-1} p_{2j} p_{2j+1}]$ calculated from Equation (5.51) using the integration rule in Equation (5.52). Then we can obtain

$$m(p_i)=\sum_{j=1}^{n}(\frac{p_i+p_{2j-1}+p_{2j}+p_{2j+1}}{4}A_j)/A^i_{total}, \tag{5.64}$$

where A^i_{total} is the total of quad areas around p_i. The area of a quad can be calculated using Equation (5.51).

In Figure 4.91, the vertex tangent movement is used to improve the aspect ratio of the quadrilateral mesh of the molecule consisting of three amino acids. Compared with Figure 4.91(c), it is obvious that the quadrilateral mesh becomes more regular and the aspect ratio is better as shown in Figure 4.91(d).

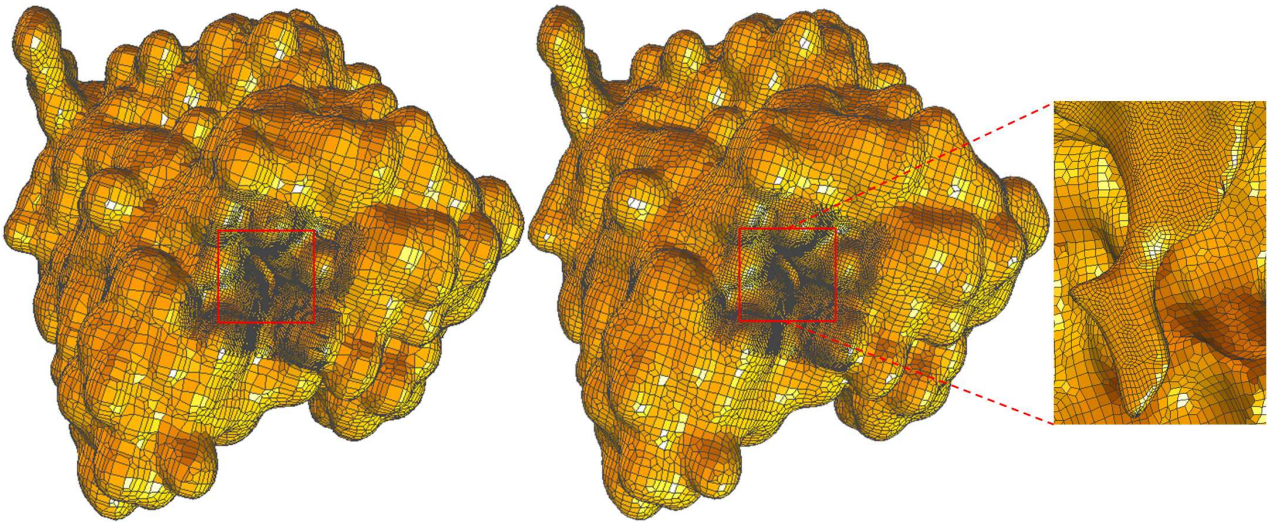
Temporal Discretization In the temporal space, $\frac{\partial x}{\partial t}$ is approximated by a semi-implicit Euler scheme $\frac{x_i^{n+1}-x_i^n}{\tau}$, where τ is the time step length. x_i^n is the approximating solution at $t=n\tau$, x_i^{n+1} is the approximating solution at $t=(n+1)\tau$, and x_i^0 serves as the initial value at x_i.

The spatial and temporal discretization leads to a linear system, and an approximating solution is obtained by solving it using a conjugate gradient iterative method with diagonal preconditioning.

Discussion **Vertex Normal Movement:** The surface diffusion flow can preserve volume. Furthermore, it also preserves a sphere accurately if the initial mesh in embedded and close to a sphere. Suppose a molecular surface could be modelled by a union of hard spheres, so it is desirable to use the surface diffusion flow to evolve the molecular surface. Figure 4.91 shows one example, the molecular surface becomes more smooth and features are preserved after surface denoising.

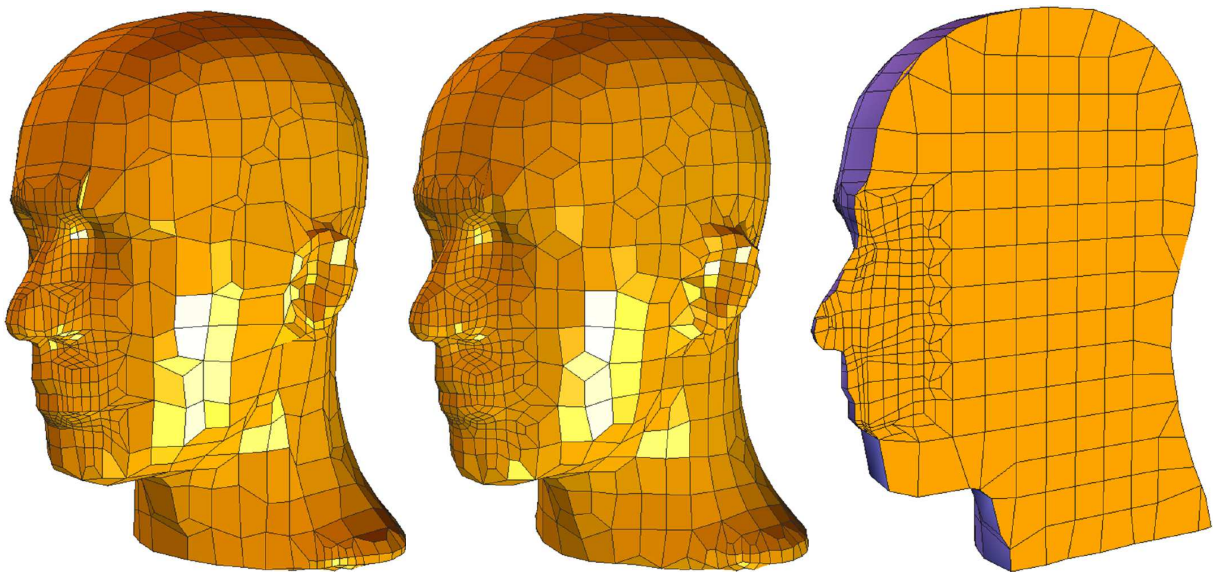
Vertex Tangent Movement: If the surface mesh has no noise, we can only apply the tangent movement $\frac{\partial x}{\partial t}=v(x)\vec{T}(x)$ to improve the aspect ratio of the mesh while ignoring the vertex normal movement. Our tangent movement has two properties:

- The tangent movement doesn't change the surface shape ([196], page 72). Figure 4.92 shows the comparison of the human head model before and after the quality improvement. In Figure 4.92(b), each vertex is relocated to its mass center, so both normal movement and tangent movement are applied. After some iterations, the facial features, such as the nose, eyes, mouth and ears, are removed. In Figure 4.92(c), the vertex movement is restricted on the tangent plane, therefore facial features are preserved.



(a) (b)

Figure 4.94: The quality of an adaptive quadrilateral mesh of a biomolecule mAChE is improved (26720 vertices, 26752 quads). (a) the original mesh; (b) after quality improvement.



(a) (b) (c)

Figure 4.95: Adaptive quadrilateral/hexadedral meshes of the human head. (a) the original quad mesh (1828 vertices, 1826 quads); (b) the improved quad mesh; (c) the improved hex mesh (4129 vertices, 3201 hexes), the right part of elements are removed to shown one cross section.

- The tangent movement is an area-weighted averaging method, which is also suitable for adaptive quad meshes as shown in Figure 4.94 and 4.95. In Figure 4.94, there is a cavity in the structure of biomolecule mouse acetylcholinesterase (mAChE), and denser meshes are generated around the cavity while coarser meshes are kept in all other regions. In Figure 4.95, finer meshes are generated in the region of facial features of the human head.

From Figure 4.92, 4.94 and 4.95, we can observe that after tangent movement, the quadrilateral meshes become more regular and the aspect ratio of the meshes is improved, as well as surface features are preserved.

Hexahedral Mesh

There are three steps for surface smoothing and quality improvement of hexahedral meshes, (1) surface vertex normal movement, (2) surface vertex tangent movement and (3) interior vertex relocation.

Boundary Vertex Movement The dual contouring hexahedral meshing method [267] [264] [268] provides a boundary sign for each vertex and each face of a hexahedron, indicating if it lies on the boundary surface or not. For example, a vertex or a face is on the surface if its boundary sign is 1, while lies inside the volume if its boundary sign is 0.

The boundary sign for each vertex/face can also be decided by checking the connectivity information of the input hexahedral mesh. If a face is shared by two elements, then this face is not on the boundary; if a face belongs to only one hex, then this face lies on the boundary surface, whose four vertices are also on the boundary surface.

We can use the boundary sign to find the neighboring vertices/faces for a given vertex. For each boundary vertex, we first find all its neighboring vertices and faces lying on the boundary surface by using the boundary sign, then relocate it to its new position calculated from Equation (5.61). There is a special situation that we need to be careful, a face/edge, whose four/two vertices are on the boundary, may not be a boundary face.

Interior Vertex Movement For each interior vertex, we intend to relocate it to the mass center of all its surrounding hexahedra. There are different methods to calculate the volume for a hexahedron. Some people divide a hex into five or six tetrahedra, then the volume of the hex is the summation of the volume of these five or six tetrahedra. This method is not unique since there are various dividing formats. Here we use an trilinear parametric function to calculate the volume of a hex.

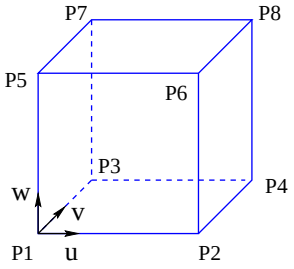


Figure 4.96: The trilinear parametric volume V of a hexahedron $[p_1 p_2 \dots p_8]$.

Volume Calculation: Let $[p_1 p_2 \dots p_8]$ be a hex in $\mathbb{R}^3$, then we define the trilinear parametric volume $V(u,v,w)$ that interpolates eight vertices of the hex as shown in Figure 4.96:

$$\begin{aligned} V(u,v,w) &= (1-u)(1-v)(1-w)p_1 \\ &+ u(1-v)(1-w)p_2 + (1-u)v(1-w)p_3 \\ &+ uv(1-w)p_4 + (1-u)(1-v)wp_5 \\ &+ u(1-v)wp_6 + (1-u)vwp_7 \\ &+ uvwp_8. \end{aligned} \tag{5.65}$$

The tangents of the volume are

$$\begin{aligned} V_u(u,v,w) &= (1-v)(1-w)(p_2-p_1) + v(1-w)(p_4-p_3) \\ &+ (1-v)w(p_6-p_5) + vw(p_8-p_7), \\ V_v(u,v,w) &= (1-u)(1-w)(p_3-p_1) + u(1-w)(p_4-p_2) \\ &+ (1-u)w(p_7-p_5) + uw(p_8-p_6), \\ V_w(u,v,w) &= (1-u)(1-v)(p_5-p_1) + u(1-v)(p_6-p_2) \\ &+ (1-u)v(p_7-p_3) + uv(p_8-p_4). \end{aligned}$$

Let V denote the volume of $V(u,v,w)$ for $(u,v,w) \in [0,1]^3$, then we have

$$V = \int_0^1 \int_0^1 \int_0^1 \sqrt{\bar{V}} du dv dw \tag{5.66}$$

Type	DataSet	MeshSize (Vertex♯, Elem♯)	Scaled Jacobian (best,aver.,worst)	Condition Number (best,aver.,worst)	Oddy Metric (best,aver.,worst)	Inverted Elem♯
quad	Head[1]	(2912, 2912)	(1.0, 0.92, 0.02)	(1.0, 1.13, 64.40)	(0.0, 1.74, 8345.37)	0
	Head[2]	-	(1.0, 0.93, 0.16)	(1.0, 1.11, 6.33)	(0.0, 0.63, 78.22)	0
	Head[3]	-	(1.0, 0.96, 0.47)	(1.0, 1.05, 2.12)	(0.0, 0.22, 6.96)	0
	Ribosome 30S[1]	(13705, 13762)	(1.0, 0.90, 0.03)	(1.0, 1.17, 36.90)	(0.0, 1.38, 2721.19)	0
	Ribosome 30S[2]	-	(1.0, 0.90, 0.03)	(1.0, 1.17, 34.60)	(0.0, 1.37, 2392.51)	0
	Ribosome 30S[3]	-	(1.0, 0.93, 0.06)	(1.0, 1.08, 16.14)	(0.0, 0.38, 519.22)	0
hex	Head[1]	(8128, 6587)	(1.0, 0.91, 1.7e-4)	(1.0, 2.99, 6077.33)	(0.0, 29.52, 1.80e5)	2
	Head[2]	-	(1.0, 0.91, 0.005)	(1.0, 1.96, 193.49)	(0.0, 6.34, 5852.23)	0
	Head[3]	-	(1.0, 0.92, 0.007)	(1.0, 1.80, 147.80)	(0.0, 4.50, 1481.69)	0
	Ribosome 30S[1]	(40292, 33313)	(1.0, 0.91, 2.4e-5)	(1.0, 2.63, 4.26e4)	(0.0, 34.15, 2.27e6)	5
	Ribosome 30S[2]	-	(1.0, 0.91, 0.004)	(1.0, 1.74, 263.91)	(0.0, 4.97, 8017.39)	0
	Ribosome 30S[3]	-	(1.0, 0.92, 0.004)	(1.0, 1.59, 237.36)	(0.0, 3.42, 5133.25)	0

Figure 4.97: The comparison of the three quality criteria (the scaled Jacobian, the condition number and Oddy metric) before/after the quality improvement for quad/hex meshes of the human head and Ribosome 30S. DATA[1] – before quality improvement; DATA[2] – after quality improvement using the optimization scheme in [264] [268]; DATA[3] – after quality improvement using the combined geometric flow/optimization-based approach.

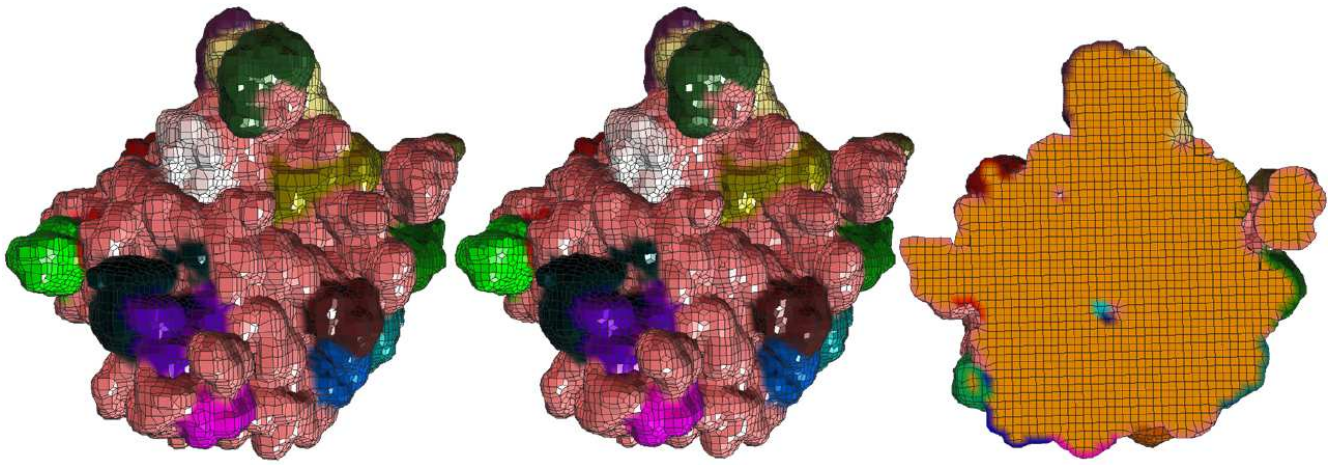


Figure 4.98: The comparison of mesh quality of Haloarcula Marismortui large Ribosome 50S (1JJ2) crystal subunit. The light yellow and the pink color show 5S and 23S rRNA respectively, the remaining colors are proteins. (a) the original quad mesh (17278 vertices, 17328 quads); (b) the improved quad mesh; (c) the improved hex mesh (57144 vertices, 48405 hexes).

where

$$\bar{V} \quad = \quad \| \, (V_u \times V_v) \cdot V_w \, \|^2 \tag{5.67}$$

Numerical integration quadrature could be used. Here we choose the following eight-point Gaussian quadrature rule to compute the integral

$$\int_0^1 \int_0^1 \int_0^1 f(u,v,w) du dv dw \approx \frac{\sum_{j=1}^{8} f(q_j)}{8}, \tag{5.68}$$

where

$$q^- = \frac{1}{2} - \frac{\sqrt{3}}{6}, \qquad q^+ = \frac{1}{2} + \frac{\sqrt{3}}{6},$$

$$q_1 = (q^-,\ q^-,\ q^-), \quad q_2 = (q^+,\ q^-,\ q^-),$$
$$q_3 = (q^-,\ q^+,\ q^-), \quad q_4 = (q^+,\ q^+,\ q^-),$$
$$q_5 = (q^-,\ q^-,\ q^+), \quad q_6 = (q^+,\ q^-,\ q^+),$$
$$q_7 = (q^-,\ q^+,\ q^+), \quad q_8 = (q^+,\ q^+,\ q^+).$$

The integration rule in Equation (5.68) is of $O(h^4)$, where h is the radius of the circumscribing sphere.

Mass Center: A mass center p of a region V is defined by finding $p \in V$, such that

$$\int_V \| y - p \|^2 \, d\sigma = min. \tag{5.69}$$

V is a piece of volume in $\mathbb{R}^3$, and V consists of hexahedra around vertex x. Then we have

$$\sum (\frac{1}{8} \sum_{j=1}^{8} p_j - p_i) V_j = 0, \tag{5.70}$$

V_j is the volume of the hex $[p_1 p_2 \dots p_8]$ calculated from the trilinear function, then we can obtain

$$m(p_i) = \sum_{j \in N(i)} (\frac{1}{8} \sum_{j=1}^{8} p_j V_j) / V^i_{total}, \tag{5.71}$$

where $N(i)$ is the index set of the one ring neighbors of p_i, and V^i_{total} is the total of hex volume around p_i.

The same Euler scheme is used here for temporal discretization, and the linear system is solved using the conjugate gradient iterative method.

Results and Applications

There are many different ways to define the aspect ratio for a quad or a hex to measure the mesh quality. Here we choose the scaled Jacobian, the condition number of the Jacobian matrix and Oddy metric [172] as our metrics [133][134][136].

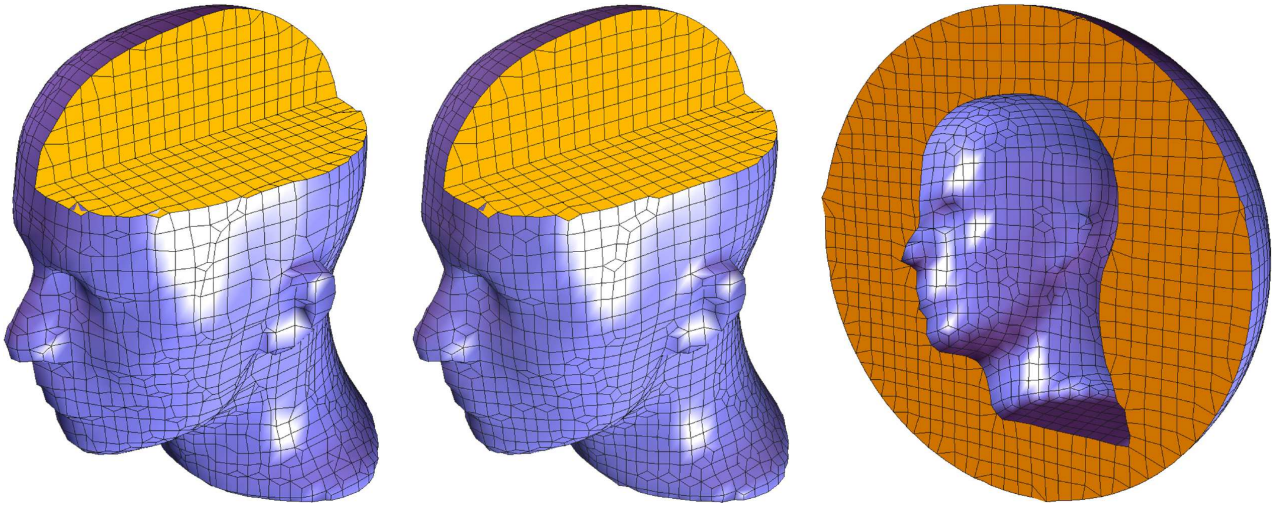
Assume $x \in \mathbb{R}^3$ is the position vector of a vertex in a quad or a hex, and $x_i \in \mathbb{R}^3$ for $i = 1, \dots, m$ are its neighboring vertices, where $m = 2$ for a quad and $m = 3$ for a hex. Edge vectors are defined as $e_i = x_i - x$ with $i = 1, \dots, m$, and the Jacobian matrix is $J = [e_1, ..., e_m]$. The determinant of the Jacobian matrix is called *Jacobian*, or *scaled Jacobian* if edge vectors are normalized. An element is said to be *inverted* if one of its Jacobians ≤ 0. We use the *Frobenius norm* as a matrix norm, $|J| = (tr(J^T J)^{1/2})$. The condition number of the Jacobian matrix is defined as $\kappa(J) = |J||J^{-1}|$, where $|J^{-1}| = \frac{|J|}{det(J)}$. Therefore, the three quality metrics for a vertex x in a quad or a hex are defined as follows:

$$Jacobian(x) = det(J) \tag{5.72}$$
$$\kappa(x) = \frac{1}{m} |J^{-1}||J| \tag{5.73}$$
$$Oddy(x) = \frac{(|J^T J|^2 - \frac{1}{m}|J|^4)}{det(J)^{\frac{4}{m}}} \tag{5.74}$$

where $m = 2$ for quadrilateral meshes and $m = 3$ for hexahedral meshes.

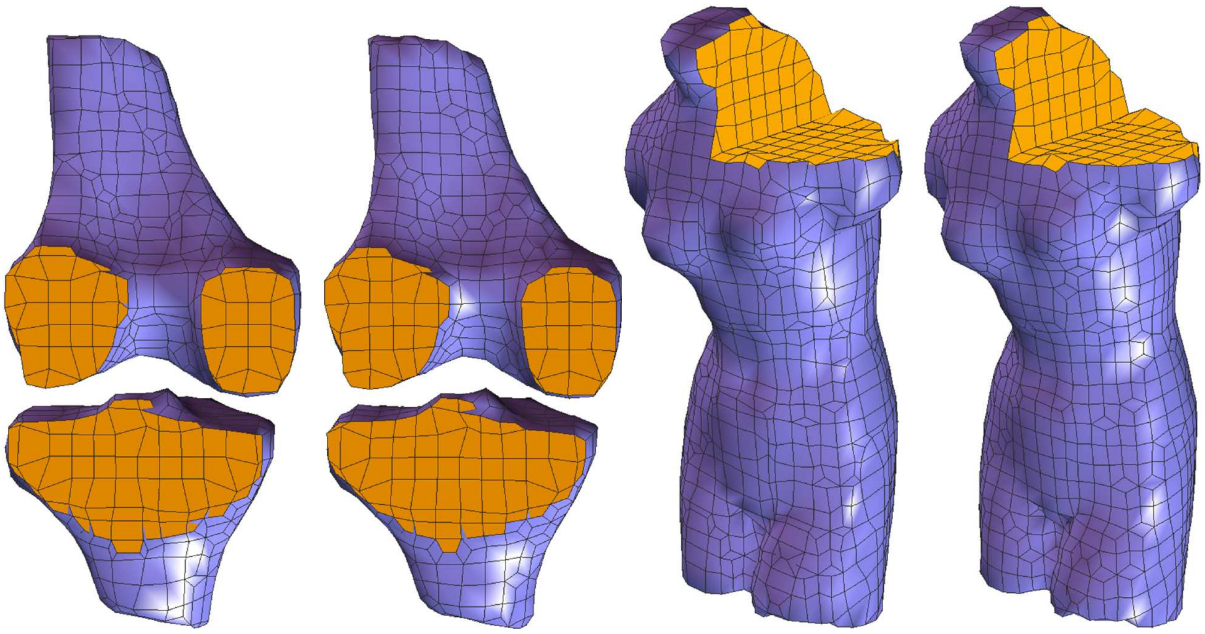
In [264] [268], an optimization approach was used to improve the quality of quad/hex meshes. The goal is to remove all the inverted elements and improve the worst condition number of the Jacobian matrix. Here we combine our surface smoothing and quality improvement schemes with the optimization-based approach. We use the geometric flow to improve the quality of quad/hex meshes overall and only use the optimization-based smoothing when necessary. Table 4.97 shows the comparison of the three quality criteria before and after quality improvement. We can observe that the aspect ratio is further improved by using the combined approach.

We have applied our surface smoothing and quality improvement technique on some biomolecular meshes. In Figure 4.91, the surface of a molecule consisting of three amino acids is denoised, the surface quadrilateral mesh becomes more regular and the aspect ratio is improved. The comparison of the quality of quad/hex meshes of Ribosome 30S/50S are shown in Figure 4.88,



(a) (b) (c)

Figure 4.99: The comparison of mesh quality of the interior and exterior hexahedral meshes. (a) the original interior hex mesh (8128 vertices, 6587 hexes); (b) the improved interior hex mesh; (c) the improved exterior hex mesh (16521 vertices, 13552 hexes).

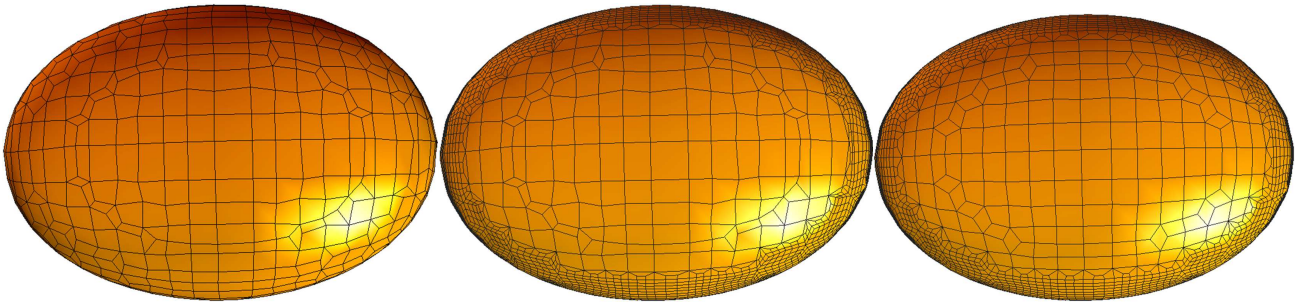


(a) (b) (c) (d)

Figure 4.100: The comparison of mesh quality of the human knee and the Venus model. (a) the original hex mesh of the knee (2103 vertices, 1341 hexes); (b) the improved hex mesh of the knee; (c) the original hex mesh of Venus (2983 vertices, 2135 hexes); (d) the improved hex mesh of Venus.

Figure 4.98 and Table 4.97. The surface diffusion flow preserves a sphere accurately when the initial mesh is embedded and close to a sphere and the tangent movement of boundary vertices doesn't change the shape, therefore features on the molecular surface are preserved. Our quality improvement scheme also works for adaptive meshes as shown in Figure 4.94.

From Figure 4.92 and 4.95, we can observe that the mesh, especially the surface mesh, becomes more regular and facial features of the human head are preserved as well as the aspect ratio is improved (Table 4.97). The interior and exterior



(a) (b) (c)

Figure 4.101: The comparison of mesh quality of the bubble. (a) a uniform quad mesh (828 vertices, 826 quads); (b) an adaptive quad mesh (5140 vertices, 5138 quads); (c) the improved adaptive quad mesh.

hexahedral meshes of the human head have been used in the electromagnetic scattering simulations. Figure 4.100 shows the quality improvement of hexahedral meshes, as well as the surface quadrilateral meshes, of the human knee and the Venus model.

Figure 4.101 shows the quadrilateral meshes for a bubble, which was used in the process of bubble elongation simulation using boundary element method. First a uniform quad mesh (Figure 4.101(a)) is extracted from volumetric data for the original state of the bubble, then we use the templates defined in [264] [268] to construct an adaptive mesh as shown in Figure 4.101(b), the boundary element solutions such as the deformation error are taken as the refinement criteria. Finally we apply our quality improvement techniques to improve the mesh quality. The improved quad mesh is shown in Figure 4.101(c).

Conclusions

We have presented an approach to smooth the surface and improve the quality of quadrilateral and hexahedral meshes. The surface diffusion flow is selected to denoise surface meshes by adjusting each boundary vertex along its normal direction. The surface diffusion flow is volume preserving, and also preserves a sphere accurately when the input mesh is embedded and close to a sphere, therefore it is especially suitable for surface smoothing of biomolecular meshes because biomolecules are usually modelled as a union of hard spheres. The vertex tangent movement doesn't change the surface shape, therefore surface features can be preserved. The interior vertices of hex meshes are relocated to their mass centers in order to improve the aspect ratio. In a summary, our approach has the properties of noise removal, feature preservation and mesh quality improvement. The resulting meshes are extensively used for efficient and accurate finite element calculations. Some of them have been used successfully.

The following algorithm is taken from [263] (Zhang2010MultiDomain).

Introduction

The problem of generating of multi-domain non-intersecting meshes from volumetric data is of great interest in the biological [270], chemical, medical [247, 232], material [42], geological [2] fields. Such multi-domain meshes are often of use in computer graphics and visualization and play a pivotal role in numerical techniques such as the finite element and the boundary element methods. For instance, in CT imaging or MRI of the human body, the domain of interest often consists of regions of different functionality. In such cases, it is desirable to segment and individually mesh these regions for the purpose of visualization and analysis. Figure 4.102 gives an example of such a case where an MRI of the brain has been segmented into 41 domains and meshed individually. Each color represents a single brain region and a small gap exists between different regions. Figure 4.102. A shows the original volumetric data, figure (B) shows the multi-domain surface and figure (C) shows a cut-view of figure B and zooming the part in the green box. We can clearly see small gaps between different sub-meshes.

Apart from the field of biological imaging, multi-domain meshing is also useful in studying composite materials, whose use has increased rapidly in the past decade due to the rise of emerging technologies [109]. Applications of these materials range from bio-medical products, aerospace structures, to meta-materials, with demonstrated performance improvements in structural, mechanical, electronic, and/or sensing and control properties[234]. Analysis of such domains with with varying

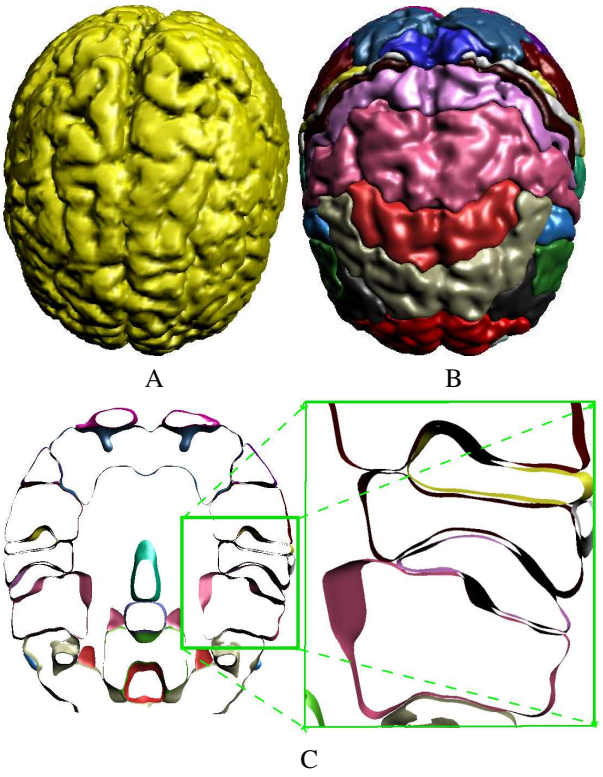


Figure 4.102: Multi-domain meshing of human brain. Figure (A) shows the original data. Figure (B) shows the multi-domain meshing result. Cut-view and zooming are shown in figure (C).

material properties across spatial dimensions are common in biomedical, geophysical and nanoscale modeling. In finite element problems in these areas, regions with identical physical/chemical attributes or material coefficients sometimes need to be meshed together such that these material regions do not intersect. Hence, it is essential that a reliable method of producing such meshes be developed for accurate analysis and simulation.

Multi-domain meshes are also useful in the field of structural biology. Macromolecular machines are composed of multiple components, either from a single molecule repeated several times (e.g. GroEL) or different molecules (e.g. ribosome). For instance, capsids of virii like the Bacteriophage P22 and the Rice Dwarf Virus (RDV) are a collection of proteins arranged spatially with icosahedral symmetry. The structural behavior of the capsid is believed to play a role in infectivity, and is also interesting from the point of view of a possible use of these capsids as nanocontainers. In [108], models of native and swollen copwea chlorotic mottle virus (CCMV) capsids from X-ray crystal structures were constructed and used in finite element simulations of atomic force microscopy (AFM) indentation along two-, three-, and fivefold icosahedral symmetry orientations. The force response of the swollen capsid model was observed to be softer by roughly a factor of two, significantly more nonlinear, and more orientation-dependent than that of a native capsid with equivalent elastic moduli, demonstrating that capsid geometric heterogeneity can have significant effects on global structural response. In order to account for the heterogeneity, it is necessary that the different capsids are separated out with independent interfaces between each capsomer in order to allow for inter-capsomer forces to be simulated. Such a study of the mechanical behavior of these viral capsids using non-linear elastic models necessitate a multi-domain mesh since it allows for different elasticity parameters to be assigned for different material regions/domains and allows one to account for the interactions between components, thereby providing a more accurate picture.

The problem of generating a manifold mesh from volumetric data is an isosurface discretization problem. Two widely used methods for producing a single domain mesh are the Marching Cubes (MC) and the Dual Contouring methods, due to Lorensen and Cline [156] and Ju et al. [127] respectively. Following the publication of these seminal methods, a lot of research has been done towards improving them[170]. These methods have also been extended for generating multi-domain meshes with varying degrees of success. The M3C algorithm [247] enhances the MC algorithm to extract the boundary surfaces between different materials to produce a multiple domain mesh from volumetric data. Wang [232] directly extracts the surface meshes

from heterogeneous volumes by extracting and segmenting coarse patches, constructing signed distance fields for different material domains and by adaptive re-meshing. The journal *Computer-Aided Design* delivered a special issue [1] in 2005 on treating heterogeneous object models and their applications with four papers on the topic. A common theme underlying all these meshing techniques is that different material domains share a common boundary. However, such an assumption may not be valid or useful for all possible applications.

In this paper, a new multi-domain meshing pipeline is proposed for producing a high quality mesh composed of an assembly of multiple non-intersecting 2-manifold sub-meshes. The sub-meshes do not share a common boundary, and the separation between them can be controlled by a parameter known as the gap threshold. A mesh separation algorithm guarantees that there are no intersections or osculations between any two sub-meshes. Hence, the produced mesh is of a high quality and may be directly used in scientific computation and in other application areas like surgical planning. The above pipeline has been implemented and sample meshes have been generated and illustrated in the following pages.

Sketch of Multi-domain Meshing Algorithm

The problem considered in this paper can be formulated as follows. Given a volumetric density map, expressed by a trivariate function as

$$\begin{aligned} f : \Omega \subset \mathbb{R}^3 &\rightarrow \mathbb{R} \\ (x_i, y_j, z_k) &\rightarrow f(x_i, y_j, z_k), i, j, k = 1, \ldots, n. \end{aligned}$$

The goal is to partition Ω into multiple domains Ω_i, such that $\Omega = \cup\Omega_i$ and for each domain of interest Ω_i, to produce a mesh $M_i (i = 1, \ldots, N)$ such that $M_i \cap_{i \neq j} M_j = \emptyset$.

The index $i (i = 1, \ldots, N)$ is referred to as the domain id of Ω_i. The volume corresponding to domain Ω_i is V_i and the isovalue for mesh M_i from volume V_i is c_i. Note that mesh M_i may be composed of multiple components. Since the mesh M_i is operated upon in several steps along the pipeline, the same symbol will be used through this paper to refer to the mesh; if an ambiguity arises with notation, new notation will be introduced as necessary.

The complete pipeline is outlined in Figure 4.103, where the gray boxes represents the data being operated upon or obtained from the previous step, and the yellow boxes summarize the approach or technique used. And the step in the yellow box with green boundary is the main technique which will be detailed in next section.

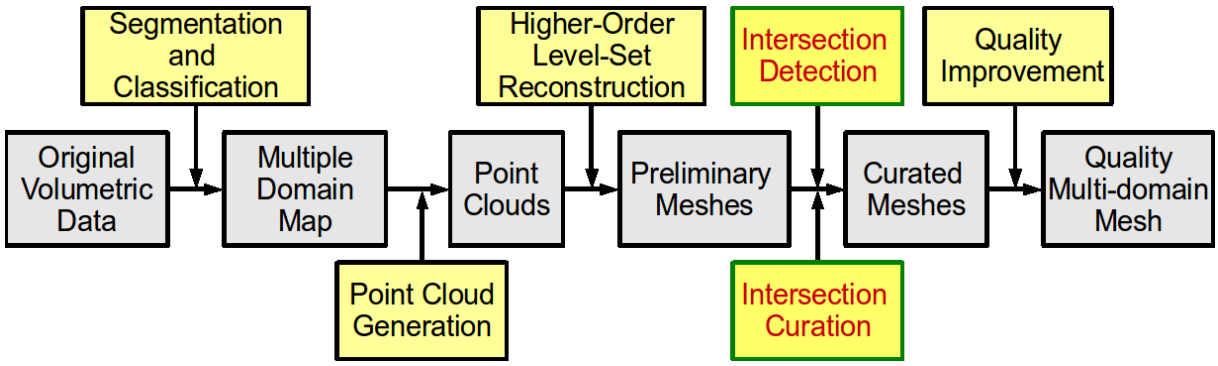


Figure 4.103: Multi-domain meshing pipeline

Algorithm Details

Segmentation and Classification Current multi-seeded fast marching methods such as [203] are used to segment 3D density maps into individual domains of interest [262, 34]. The basic idea of this method is that a contour, initialized from a seed point chosen a priori, is allowed to grow until a certain stopping condition is reached. Each voxel is assigned with a value called "time", which is initially set to zero for seed points and infinity for all other voxels. The voxel on the marching contour with minimal time value is removed from the contour iteratively and the time values of its neighbors are updated according to the equation

$$\|\nabla T(\mathbf{x})\| F(\mathbf{x}) = 1,$$

where $F(\mathbf{x})$ is called the speed function, and is usually determined by the gradients of the density map. The neighbors are then inserted into the contour if they are being updated for the first time. In order to achieve effective results, the three key problems in this approach which need to be solved are the formulation of the speed function, the initialization of the seed points and determination of the stopping criterion.

The density map is then classified into several different domains after segmenting it into multiple components of interest. Multiple components are classified as belonging to the same domain if their density maps are nearly identical after symmetry, rotation and reflection are taken into consideration. Prior knowledge from the field of study is also greatly useful for classification. Furthermore, an exact classification is difficult to obtain since the volumetric data from methods such as MRI or cryoEM always contains noise and other artifacts which can cause inaccuracies. Although preprocessing the initial volumetric data [33] can alleviate the impact of noise, exact segmentation and classification are still a challenging problem. After classification, a multi-domain volumetric map is obtained:

$$\begin{aligned} g : \mathbb{R}^3 &\rightarrow \mathbb{N} \\ (x_i, y_j, z_k) &\rightarrow g(x_i, y_j, z_k), \end{aligned}$$

here $\mathbb{N}$ is the set of natural numbers. As an example, for the original volume (see the top figure of Figure 4.104. A) of Penicillium Stoloniferum Virus F (PSVF) with a resolution of 4.5Å, the segmentation result is shown in the bottom figure of Figure 4.104. A.

Point Cloud Generation This step consists of partition surface extraction and preliminary smoothing and point cloud generation steps.

Partition surface extraction In this step, partition surfaces are produced from the multi-domain map produced from the previous segmentation and classification step. A partition surface is a surface whose interior includes voxels from a single domain only. In order to produce a partition surface for a domain with id i, a cube (or cell) with each vertex with domain id i is taken and a dual cube with six faces with face id i is produced. If two dual cubes have common faces with the same face id, say i, then two cubes are combined into one bigger cuboid, and so on. If two dual cubes have common faces with different face id, say i and j ($i \neq j$), two different surfaces are produced. Note that the produced surfaces may not be manifold since they may include non-manifold vertices and/or edges.

Preliminary smoothing The obtained partition surfaces may not be smooth in general, and a HLS reconstruction may result in a mesh with excessive, unnecessary features. Furthermore, it was found experimentally that a preliminary smoothing of the partition mesh results in a final multi-material mesh with much lesser inter-mesh intersections. Hence, geometric flow is used to perform a preliminary smoothing step. Other smoothing methods such as Laplacian smoothing [230] and similar approaches [224, 198] may also be utilized to carrying out this task. Geometric PDEs, such as the mean curvature flow, the surface flow and Willmore flow have been extensively used in surface and image processing [254]. In the pipeline described in this paper, surface diffusion flow with a tangential movement modification is chosen for smoothing:

$$\frac{\partial \mathbf{x}}{\partial t} = \Delta H \mathbf{n} + \nu \mathbf{t}, \tag{5.75}$$

where $\mathbf{n}, \mathbf{t}$ are the normal and tangent vectors of vertex $\mathbf{x}$, H is the mean curvature, and Δ is the Laplace-Beltrami operator, ν is the velocity in the tangential direction and t is a marching parameter. We use surface diffusion flow because it owns a volume preserving property.

Each term of the above expression (5.75) is discretized over the partition mesh, including the mean curvature H and the Laplace-Beltrami operator [250], and the equation is solved over the mesh using the forward Euler method [251].

We also use geometric flow smoothing scheme for quality improvement in the last step of the pipeline.

Point cloud generation Although the partition surfaces have been smoothed, it is very difficult to reconstruct a smooth surface since in general a partition surface is non-manifold. Hence, a different approach is used where a point cloud is produced to reconstruct a smooth manifold surface. In order to generate the point cloud, each triangle of the partition surface is subdivided into four triangles if the length of any edge of the triangle is greater than a threshold. Other subdivision methods, such as Catmull-Clark subdivision [53], Doo-Sabin subdivision [84], Loop subdivision [151] and others [184] may also be used;

however, the non-manifold property of the partition surface mesh must be taken into consideration. In Figure 4.104.B, we show the point clouds of three monomers of PSVF and zoom one intersection part. From this figure, we can see that some points in one monomer are inclosed in the space generated by the other points of another monomer.

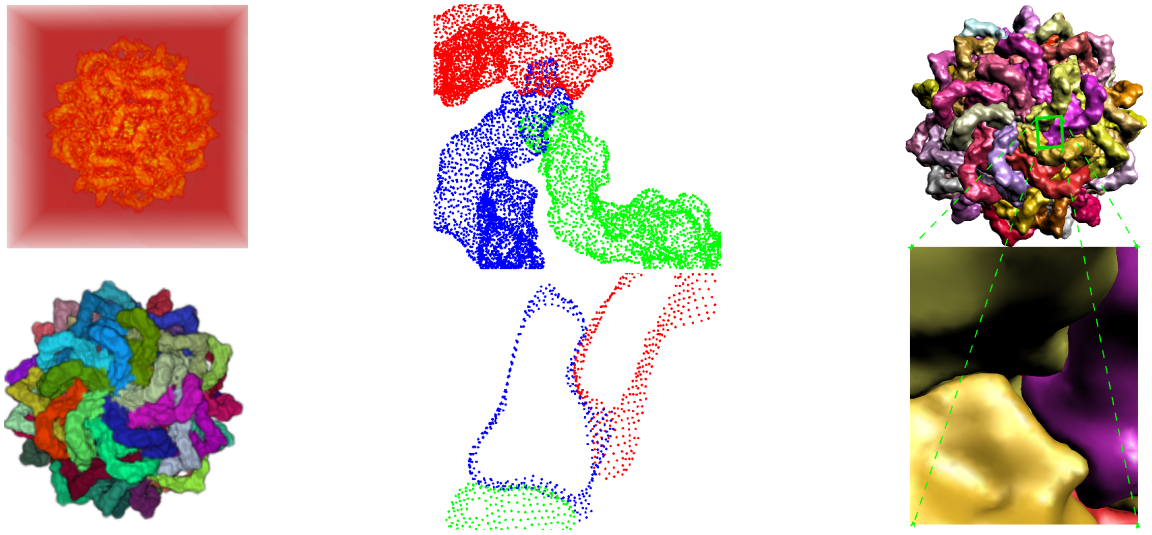
Higher-Order Level Set Method Based Surface Reconstruction The produce of smooth surfaces from a point cloud is a challenging task and large amount of research has been done in the past on this problem. Although multiple approaches exist, the implicit approach is described here. The signed distance function was used by Hoppe et al. [122]as an implicit function and the reconstructed surface was represented by the zero iso-surface. Zhao et al. [275, 276] proposed the weighted minimal surface model, using a fast sweeping method to compute distance function and computing the signed distance function by level set methods proposed by Osher et al. [176], to fit the point cloud data dynamically. In [256], an improved energy model is proposed to construct a smooth surface from noisy scattered data and a level-set method is utilized to solve the geometric flow equation from the improved energy model. In [28], a higher-order level-set method is proposed to construct smooth surfaces for molecules and scattered data sets. The reconstructed smooth surface is represented by the level set of a tri-cubic B-spline function. From a theoretical viewpoint, the surface is almost C^2 continuous. Several important issues related to this approach here are initial surface construction, thin shell technique, adaptive reinitialization of the signed distance function, conversion of piecewise tri-linear function to tri-cubic B-spline and iso-contouring methods. Two steps to construct the initial surface which is represented by the level set of a signed distance function include distance computation and sign prescription. Thin shell technique means the computation is only performed in a thin shell of the moving surface and is used to speed up the computation to overcome the shortcomings of level set method. Adaptive reinitialization of signed distance function can improve the precision of computation and decrease the development of flat and/or steep regions of moving surface. This task is performed by solving a Hamilton-Jacobi equation. The conversion of piecewise tri-linear function to tri-cubic B-spline can be carried out by fast iteration. Since the geometric flow equation involves the second order derivatives of the level set function of moving surface, a tri-cubic B-spline is used to represent the moving surface as well as signed distance function. Iso-contouring technique includes primal and dual contouring. To produce a topologically correct intersection-free, crack-free mesh, an improved primal contouring method is used, which handles the ambiguities in the original marching cubes method. In general, any meshing algorithm, adaptive or uniform, which can produce topologically correct, manifold surface meshes can be used with this pipeline.

Intersection Detection and Curation Following the surface reconstruction from a point cloud (subsection 4.5), a volume V_i and a mesh M_i with prescribed isovalue c_i for each domain Ω_i are obtained. Although a small isovalue may be selected to prevent intersections between different meshes, doing so may result in large separations between sub-meshes which is generally not acceptable. On the other hand, if surfaces with a larger isovalue are reconstructed from the volumes, intersections between different domains may appear. Hence, there is a tradeoff between intersections and the extent of separation between sub-meshes. A balanced way of selecting a suitable isovalue is by taking into account the initial mesh obtained after separation but keep no serious intersections between different meshes. It was found experimentally that this approach produces a quality mesh with correct topology and acceptable gaps between sub-meshes.

Intersection detection In order to prevent intersection between surfaces, the presence of an intersection between any two meshes must be checked. The proposed algorithm checks whether vertices belonging to one mesh exist within the interior of another mesh and vice versa. This intersection detection test is valid since it is known that a point in the volume exists in the interior of the mesh representing the isosurface if and only if the function value at that point is less than the isovalue of the mesh. The following example explains this more clearly.

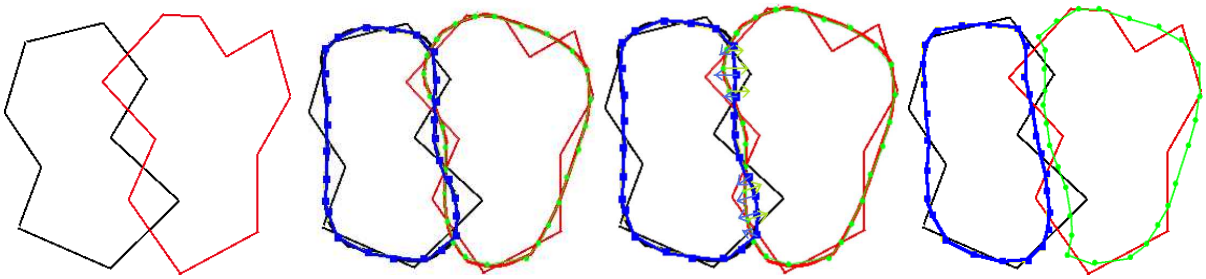
Suppose there are two meshes M_1 with P_1 vertices and M_2 with P_2 vertices from their corresponding volumes V_1 and V_2 with isovalue c_1 and c_2. For the vertex $\mathbf{v}_{1i}$ from mesh M_1, if $f(\mathbf{v}_{1i}) < c_2$, then it must be in the interior of mesh M_2. If $f(\mathbf{v}_{1i}) > c_2$ for $\mathbf{v}_{1i}, i = 1, \ldots, P_1$ in M_1, there is no intersection between M_1 and M_2. The same is for the vertices in M_2 with mesh M_1. The total number of vertex $\mathbf{v}_{1i}$ which exist in the interior of M_2 are denoted by p_{12} and the total number of vertex $\mathbf{v}_{2i}$ which exist in the interior of M_1 is denoted by p_{21}.

Intersection curation Suppose M_1 and M_2 have intersections with p_{12} vertices of M_1 in M_2 and p_{21} vertices of M_2 in M_1. In order to obtain meshes M_1' and M_2' without intersections, it is required that $p_{12}' = p_{21}' = 0$. Although the mesh M_1 is extracted from volume V_1 with isovalue c_1, the function value $f(\mathbf{v}_{1i})$ is not identical to c_1 and depends on the isosurface extraction method used. For instance, if the improved primal contouring is used the error between the function value and isovalue is small. On the



(a) (b) (c)

Figure 4.104: Multi-domain meshing of PSVF. The top figure of (A) shows the original volumetric data and the bottom shows the segmentation result. Figure (B) shows the points clouds of three domains and the intersection of point clouds is shown in the bottom figure. Figure (C) shows the multi-domain meshing result and zoomed part for intersection free viewing.



A B C D

Figure 4.105: Schematic illustration of detection and curation of intersections in 2D. Figure (A) shows the orginal meshes. Figure (B) shows the isocontours generated for each mesh and sampling points in the contours. (C) shows the evolution direction for the points in question. (D) shows the intersection and touch free meshes.

other hand, if a dual contouring method is used, the error is larger since the vertices of the mesh from dual contouring approach is just a minimizer which is not guaranteed to lie on the isosurface. Hence, a remedial step is carried out where the vertices of each mesh are projected back to their corresponding isosurface within an small error tolerance ε so that $f(\mathbf{v}_{ji}) = c_j \pm \varepsilon$.

For vertex $\mathbf{v}_{2i}(i = 1,\ldots,p_{21})$ of mesh M_2, the following partial differential equation is then used to move it outside of the mesh M_1,

$$\frac{\partial \mathbf{v}'_{2i}}{\partial t} = \tilde{d}_{21i} * \mathbf{n}_{21i}, \tag{5.76}$$

where $\mathbf{v}'_{2i}$ is the new position of vertex $\mathbf{v}_{2i}$, $\tilde{d}_{21i}$ is an improved version of the distance d_{21i} between vertex $\mathbf{v}_{2i}$ with the isosurface from volume V_1 with isovalue c_1. $\mathbf{n}_{21i}$ is the normal of vertex $\mathbf{v}_{2i}$ calculated from volume V_1. There are several approaches to compute normal from volume V_1 like a simple finite difference method, or an explicit form like the tricubic B-spline function. The explicit form for $\tilde{d}_{21i}$ is given by

$$\tilde{d}_{21i} = (1+\delta)d_{21i},$$

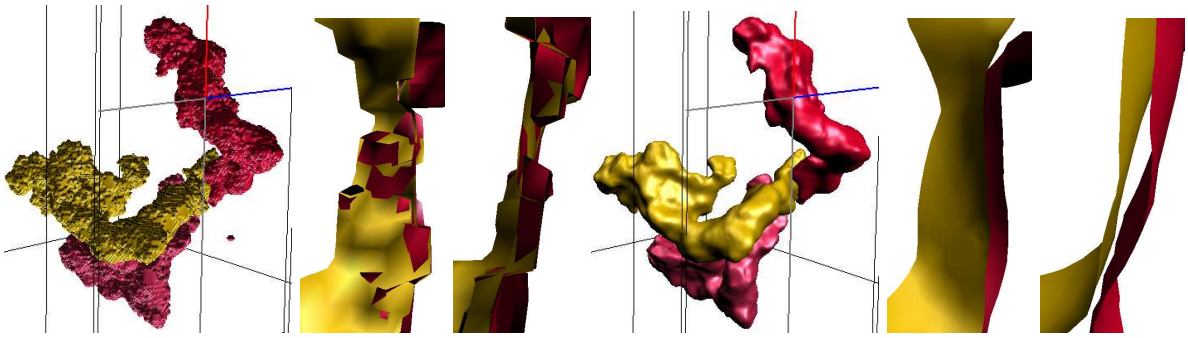


Figure 4.106: 3D intersection curation. Figure (A) shows the partition mesh after geometric flow smoothing. (B) and (C) show two different views for the thin shell in figure (A). (D) shows the result after point clouds reconstruction by higher-order level-set method and intersection detection and curation. (E) and (F) show two different views for the thin shell in figure (D).

where $\delta > 0$ is called gap measure which should be than ε. d_{21i} is the distance between vertex $\mathbf{v}_{2i}$ with the isosurface from volume V_1 with isovalue c_1 and it can be calculated as $d_{21i} = \frac{|f(\mathbf{v}_{2i})|}{\|\nabla f(\mathbf{v}_{2i})\|}$ [17]. Generally, the bigger δ is, the wider the gap. Taking the error of normal computation and speed into consideration , the Euler forward method is used to solve this equation. The process is iterated until $|f(\mathbf{v}'_{2i}) - (1+\delta)d_{12i}| < \varepsilon$, where $\varepsilon > 0$ is small.

A schematic figure in 2D is shown in Figure 4.105 as an illustration of the idea of intersection detection and curation. In this figure, A shows the original meshes, B shows the isocontours generated using improved Marching Cube methods. Figure C shows the evolution directions for the vertices which are in the interior of the other mesh. Figure D represents the evolution results and the intersection and touch free meshes. During the intersection detection and curation, the number of vertices does not change.

We also show a 3D example for PSVF in Figure 4.106, where figure (A) shows three meshes of PSVF monomers after domain partition and smoothing. The thin shell in figure (A) is shown in figure (B) and (C) as two different views, and we can see clearly the intersection of the red and yellow meshes. Figure 4.106. D shows the result after reconstruction from point clouds and intersection detection and curation steps. Figures (E) and (F) show two different views for the thin shell in figure (D). One can alos see that the small piece in figure (A) is eliminated.

After intersection detection and curation, the result is obtained. In Figure 4.104.C the multi-domain meshing result of PSVF and one zoom view are shown.

Results

Our pipeline has been implemented on a 8-core Sun Workstation running Ubuntu 9.04 using the C++ language. The algorithm was tested on the P22, PSV, Brain and Neuron datasets. The results are illustrated in various figures across this paper. The percentage of intersections between multiple domains was less than 0.02% consistently across all datasets, indicating the effectiveness of the algorithm.

Results for P22 As another example, the multi-domain meshing result for bacteriophage P22 is shown with resolution 9.5Åand EMDB id 1101 [124]. In Figure 4.107, the original volumetric data (A) and segmentation result (B), the mesh produced by the pipeline (C), which is intersection and touch free are shown. In Figure 4.107. D, the cut-view of (C) is illustrated. In Figure 4.108, one hexamer and pentamer are first partitioned into monomers. The meshing results and the shape of one monomer are also shown.

Result for neuron The Neuron dataset is difficult to deal with due to the large amount of noise. Usually, a mesh model of the neurons is reconstructed from pre-segmented neuron image slices using a contour tiling algorithm. However, the mesh obtained by this algorithm cannot be used for simulation since cellular elements such as dendrites, axons and glia intersect using this approach. Our approach on the other hand, can solve this problem directly. In Figure 4.109, one example is illustrated, where Figure (A) shows an image of one slice of the neurons; Figure (B) shows part of the result produced by the tiling scheme, where

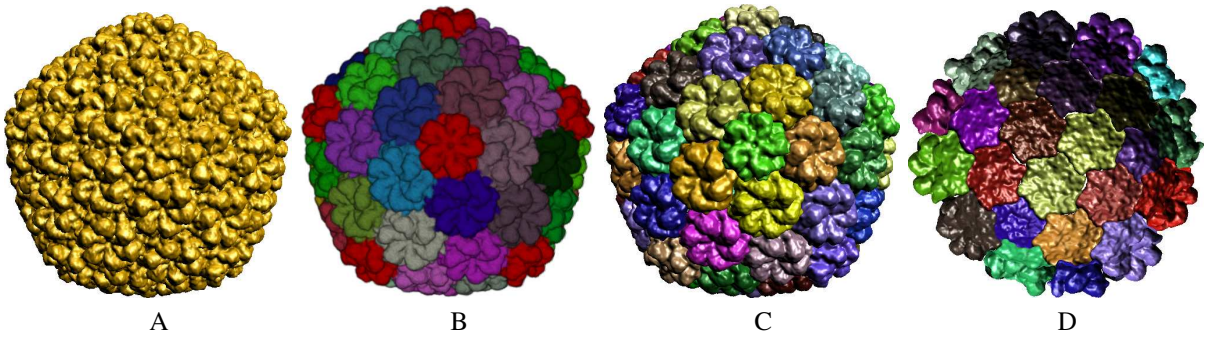


Figure 4.107: Multi-domain meshing of P22. (A) shows the original volumertric data. (B) shows the segmentation result. (C) shows the intersection and touch free mesh. (D) shows the cut-view of the mesh.

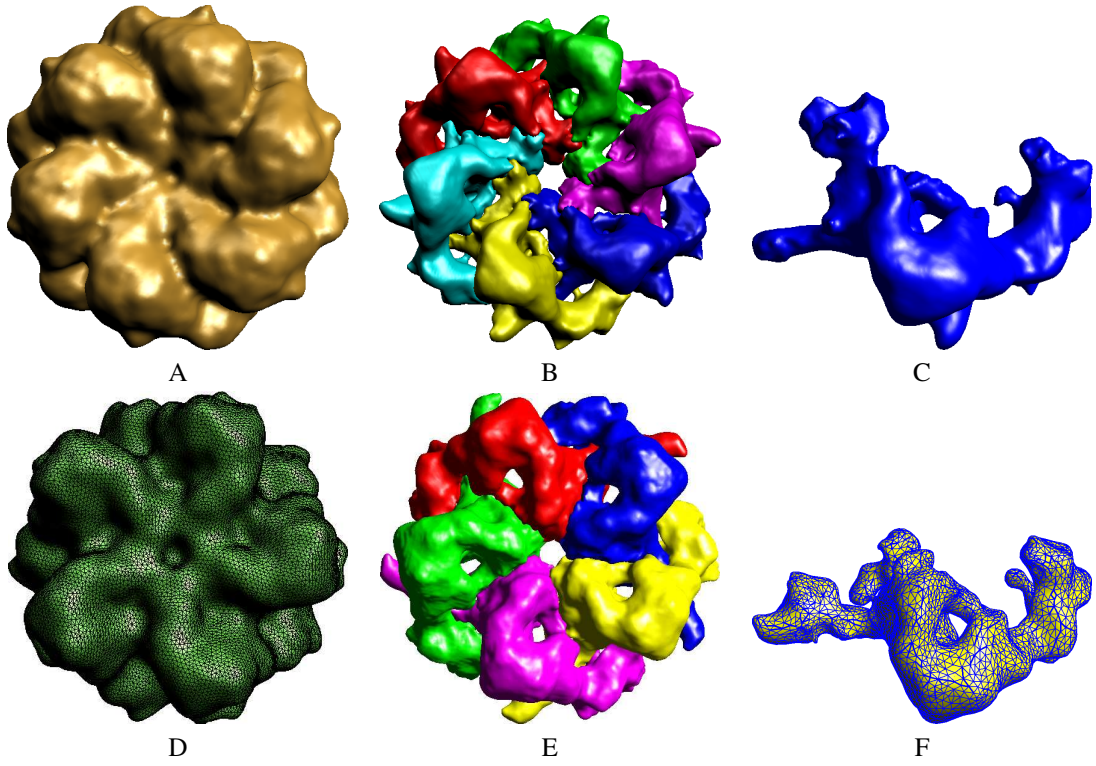


Figure 4.108: Further partition and meshing each hexamer and pentamerof P22. (A) and (D) show one hexamer and one pentamer separately. (B) and (E) show multi-domain meshing for (A) and (D). (C) shows one monomer of hexamer in (A). (F) shows one monomer of pentamer in (D) with quality mesh.

the dendrite and the axons intersect in some places. Figure (C) shows the curation of the intersection of some axons; Figure (D) shows a part of Figure (C) in greater detail. Figure (E) shows part result of axons and dendrites (where the large yellow region represents the mesh of one dendrite); Figure (F) shows a different view of Figure (E). One can see the these meshes are tightly packed but there is no intersection between them.

Conclusions and Future Work

In this paper, a new pipeline for generating quality mesh from volumetric data is proposed. The mesh is composed of many 2-manifold non-intersecting sub-meshes. There is a small gap between adjacent sub-meshes and which may be controlled by a prescribed threshold. An efficient intersection detection and curation technique has also been presented. Finally, the

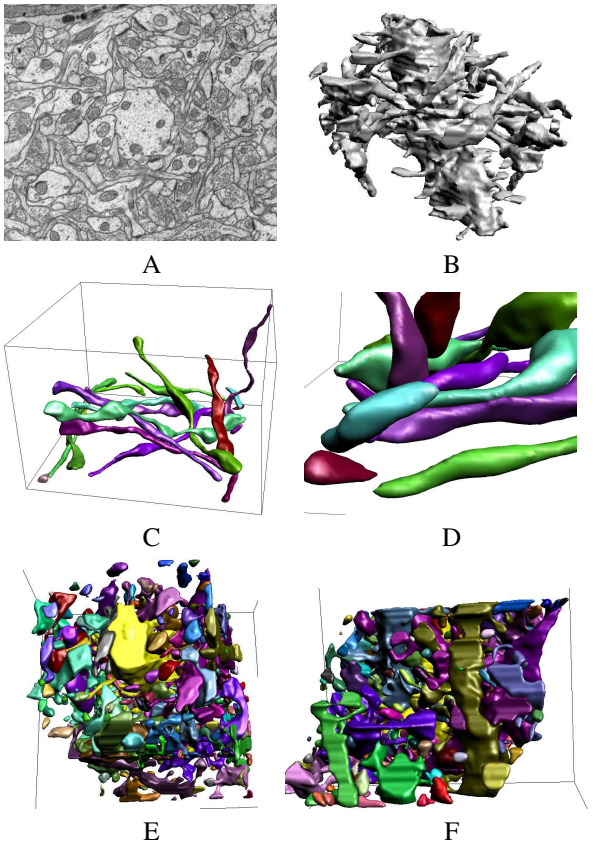


Figure 4.109: Multi-domain meshing of neuron. (A) one slice of image. (B) shows one dendrite and some axons by the contour tiling algorithm. (C) shows part result of the curation of the intersection of axons. (D) zooms in part of figure (C). (E) shows one intermediate result of neuron, which includes axons and dendrites. (F) shows another view of figure (E).

implemented pipeline has been described and numerical experiments performed using the pipeline have been illustrated. All the results provided have been tested using the open source software MeshLab (`http://meshlab.sourceforge.net/`).

4.5.1 Branching Structures

4.5.2 Free-form Surface Models

4.5.3 Volumetric Models

4.6 Biological Structure Elucidation

4.6.1 molecules — meshed models of molecular surfaces and volumes

4.6.2 cells with organelles —- meshed models of branching structures, forest, weight functions

4.6.3 tissues cells + extra-cellular space/filling —- volumetric meshes of T-tubules, bone tissue

4.6.4 organs — surface and volumetric meshed models of heart, brain

4.6.5 systems — meshed time dependent geometry

4.7 Bibliographic Notes

4.7.1 Reconstruction

4.7.2 Filtering

4.7.3 Segmentation

4.7.4 Skeletonization

The following is taken from [113] (Goswami2006Regions).

Previous results. Because of the significance of the problem, quite a few work spanning various approaches have been reported in the literature. To mention a few, we refer to the curvature based methods of [228] and [166, 167], the fuzzy clustering method of [128], the method based on PCA of surface normals by [185], the hybrid variational surface approximation by [246] and the Reeb graph approach of [213] and [229]. Remarkably the distance function over $\mathbb{R}^3$ which is defined by the distance to the boundary of the shape has not been fully used for feature annotation. In the context of surface reconstruction, topological structures induced by distance functions have been analyzed by Edelsbrunner [86], Chaine [56] and Giesen and John [111]. Chazal and Lieutier [58] and Siddiqi et al. [215] have used it for medial axis approximations. Dey, Giesen and Goswami used the topological structures induced by the distance function to segment a shape [77]. However, this work stops short of using the topological structures for feature annotations. In this paper we complete this step.

4.7.5 Meshed Models

The following is taken from [116] (Goswami2008Efficient).

Mesh generation techniques have received significant attention in the past two decades. Two works in particular, one by Chew [64] and one by Edelsbrunner and Shah [87], have spawned important and relevant results in the field. We will address the prominent successors of each of these works and compare the relative advantage of our approach.

Chew provided one of the first meshing algorithms for curved surfaces with provably good geometry in [64], although the algorithm as stated in the paper could not guarantee topological correctness. Boissonnat and Oudot showed how Chew's algorithm can be applied to produce a dense sample of Σ and subsequently mesh it. Oudot, Rineau and Yvinec recently improved that method by including sliver exudation, that is, the process of removing tetrahedra with very small dihedral angles from a mesh [178]. Alliez, Cohen-Steiner, Yvinec and Desbrun have also adapted the method to produce nicely graded meshes, i.e. meshes where the tetrahedra vary in size gradually based on their distance to the surface [4].

Separately, Edelsbrunner and Shah established a criterion called the *closed ball property* for ensuring that a mesh is Delaunay conforming [87]. Recent work by Cheng, Dey, Ramos, and Ray uses this property to provide a method for constructing Delaunay conforming meshes that avoids the need to estimate local feature size [60]. This is a significant development as approximating local feature size is computationally expensive and not always numerically robust. Cheng, Dey and Ramos have extended this strategy for piecewise smooth complexes [62]. Very recently, Dey and Levine [81] have given a two phase algorithm to mesh isosurfaces from imaging data.

All of these approaches to mesh generation are useful, however, they all rely on an oracle to know whether an arbitrary ray intersects the surface Σ. The implementation of such an oracle becomes computationally prohibitive when applied to piecewise interpolated surfaces. For example, a common data set size is 100^3 vertices, meaning there exist 100^3 functions in the piecewise decomposition. Thus, a single ray may pass through over a hundred separate function domains, making intersection calculations expensive. As we detail in Section 4.5, a major advantage of the algorithm presented in this paper is that we take advantage of the original rectilinear scaffolding from the data to substantially reduce the computational overhead required.

Our work also improves upon existing methods for isosurface construction. Many well-known techniques exist for isosurfacing including marching cubes [154], active snakes, dual contouring [266], and higher order interpolation [30]. A variety of approaches have also been developed to provide hierarchical isosurface extraction and interior meshing [245, 119, 123]. Shewchuk and Labelle recently provided a straightforward isosurfacing and stuffing algorithm with good quality tetrahedra [139]. Relatively few works, however, take into account the effect of a trilinear interpolant within each grid cell [168, 63, 153]. Attali and Lachaud gave an algorithm for construction of Delaunay conforming isosurfaces [5]. Their method, however, relies on a specific rule established by Lauchaud in [140] that does not accommodate interpolation within grid cells. Further, none of these techniques generalize easily to data that is sampled adaptively or to arbitrary interpolants.

The following is taken from [265] (zhang20053d).

time to search for contributing cells are developed. The isosurfaces of a function defined by the trilinear interpolation inside a cubic cell may have a complicated shape and topology which cannot be reconstructed correctly using MC. The function values of face and body saddles in the cell can be used to decide the correct topology and consistent triangulation of an isosurface in the cell [169]. Lopes and Brodlie [152] provided a more accurate triangulation.

Main drawbacks of MC and its variants are (i) the produced mesh is uniform, (ii) badly shaped triangles are generated. and (iii) sharp features in the data are not preserved. An adaptive isosurface can be generated by triangulating cells with different levels. When the adjacent cubes have different resolution levels, the cracking problem will happen. To keep the face compatibility, the gravity center of the coarser triangle is inserted, and a fan of triangles are used to approximate the isosurface [240]. The chain-gang algorithm [141] was presented for isosurface rendering of super adaptive resolution (SAR) and resolves discontinuities in SAR data sets. Progressive multiresolution representation and recursive subdivision are combined effectively, and isosurfaces are constructed and smoothed by applying the edge bisection method [180]. A surface wave-front propagation technique [243] is used to generate multiresolution meshes with good aspect ratio.

The enhanced distance field representation and the extended MC algorithm [135] can detect and reconstruct sharp features in the isosurface. By combining SurfaceNets [110] and the extended Marching Cubes algorithm [135], octree based Dual Contouring [126] can generate adaptive isosurfaces with good aspect ratio and preservation of sharp features. Elements in the extracted mesh often have bad aspect ratio. These elements can not be used for finite element calculations. The grid snapping method reduces the number of elements in an approximated isosurface and also improves the aspect ratio of the elements [165]. [40] studied how to generate triangular meshes with bounded aspect ratios from a planar point set. [164] proposed an algorithm, called QMG, to triangulate a d-dimensional region with a bounded aspect ratio.

Tetrahedral Mesh Generation Octree based, advancing front based and Delaunay like techniques were used for tetrahedral mesh generation. The octree technique recursively subdivides the cube containing the geometric model until the desired resolution is reached [206]. Advancing front methods start from a boundary and move a front from the boundary towards empty space within the domain [149] [98] [204]. The Delaunay criterion is called 'empty sphere', which means that no node is contained within the circumsphere of any tetrahedra of the mesh. Delaunay refinement is to refine the triangles or tetrahedra locally by inserting new nodes to maintain the Delaunay criterion. Different approaches to define new nodes were studied [65] [208] [61]. Sliver exudation [59] was used to eliminate those slivers. A deterministic algorithm [61] was presented for generating a weighted Delaunay mesh with no poor quality tetrahedra including slivers. Shewchuk [207] solved the problem of enforcing boundary conformity by constrained Delaunay triangulation (CDT). Delaunay refinement [208], edge removal and multi-face removal optimization algorithm [209] were used to improve the tetrahedral quality. Shewchuk [210] provided some valuable conclusions on quality measures for finite element method.

MC was extended to extract tetrahedral meshes between two isosurfaces directly from volume data [100]. A Branch-on-

Need octree was used as an auxiliary data structure to accelerate the extraction process . A different algorithm, Marching Tetrahedra (MT), was proposed for interval volume tetrahedralization [171]. A multiresolution framework [279] was generated by combining recursive subdivision and edge-bisection methods. Since many 3D objects are sampled in terms of slices, Bajaj et. al introduced an approach to construct triangular surface meshes from the slice data [11], and tetrahedralize the solid region bounded by planar contours and the surface mesh [32].

Hexahedral Mesh Generation Eppstein [90] started from a tetrahedral mesh to decompose each tetrahedron into four hexahedra. Although this method avoids many difficulties, it increases the number of elements. There are four distinct methods for unstructured all-hex mesh generation: grid-based, medial surface, plastering and whisker weaving. The grid-based approach generates a fitted 3D grid of hex elements on the interior of the volume [199] [201]. Medial surface methods involve an initial decomposition of the volume [189] [188]. Plastering places elements on boundaries first and advances towards the center of the volume [50] [43]. Whisker weaving first construct the spatial twist continuum (STC) or dual of the hex mesh, then the hex elements can be fitted into the volume using the STC as a guide [225].

Quality Improvement Algorithms for mesh improvement can be classified into three categories [226] [179]: local coarsening/refinement by inserting/deleting points, local remeshing by face/edge swapping and mesh smoothing by relocating vertices.

Laplacian smoothing, in its simplest form, relocates the vertex position at the average of the nodes connecting to it. This method generally works quite well for meshes in convex regions. However, it can result in distorted or even inverted elements near concavities in the mesh. [106] weighted the contribution of each neighboring node in the average function. [93] constrained the node movement in order to avoid the creation of invert elements. [160] discretized the Laplacian operator using Voronoi cells and the mixed Finite Element/Finite Volume method. The discretized format was used to solve surface modelling problems using the finite difference method [253]. [146] and [120] developed methods to extend to anisotropic meshes. Instead of relocating vertices based on a heuristic algorithm, people searched a optimization technique to improve the mesh quality. The optimization algorithm measures the quality of the surrounding elements to a node and attempts to optimize it. The algorithm is similar to a minimax technique used to solve circuit design problem [57]. The optimization-based smoothing yields better results while it is more expensive than Laplacian smoothing. Therefore, [51] [95] [96] recommended a combined Laplacian/optimization-based approach, which uses Laplacian smoothing when possible and only uses optimization-based smoothing when necessary. Physically-based simulations are used to reposition nodes [148]. Anisotropic meshes are obtained from bubble equilibrium [211] [47].

The following is taken from [269] (zhang2009surface).

Branching Structures

Free-form Surface Models

Volumetric Models

4.7.6 Time Dependent Geometry

Bibliography

[1] Heterogeneous object models and their applications. 37(3):285–353, 2005. Special issue edited by Alexander Pasko and Vadim Shapiro.

[2] V. Adzhiev, E. Kartasheva, T. Kunii, A. Pasko, and B. Schmitt. Cellular-functional modeling of heterogeneous objects. In *SMA '02: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, pages 192–203. ACM, 2002.

[3] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discr. Appl. Math.*, 71:5–22, 1996.

[4] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun. Variational tetrahedral meshing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 617–625. ACM Press, 2005.

[5] D. Attali and J.-O. Lachaud. Delaunay conforming iso-surface, skeleton extraction and noise removal. *Computational Geometry: Theory and Applications*, 19:175–189, 2001.

[6] C. Bajaj. Volrover: An interactive volume visualization tool.

[7] C. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *ACM SIGGRAPH*, pages 109–118, 1995.

[8] C. Bajaj, J. Chen, and G. Xu. Modeling with Cubic A-Patches. *ACM Transactions on Graphics*, 14(2):103–133, 1995.

[9] C. Bajaj, J. Chen, and G. Xu. Modeling with cubic A-patches. *ACM Transactions on Graphics*, 14(2):103–133, 1995.

[10] C. Bajaj, E. Coyle, and K. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graphical models and image processing*, 58(6):524–543, 1996.

[11] C. Bajaj, E. Coyle, and K.-N. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Modeling and Image Processing*, 58(6):524–543, 1996.

[12] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Interactive visual exploration of large flexible multi-component molecular complexes. In *Proc. of the Annual IEEE Visualization Conference*, pages 243–250, Oct. 2004.

[13] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Texmol: interactive visual exploration of large flexible multi-component molecular complexes. In VIS'04: Proceedings of the conference on Visualization '04, Washington, DC, USA, 2004.

[14] C. Bajaj, A. Gillette, and S. Goswami. Topology based selection and curation of level sets. In *TopoInVis 2007*, 2007.

[15] C. Bajaj and S. Goswami. Automatic fold and structural motif elucidation from 3d em maps of macromolecules. In *ICVGIP 2006*, pages 264–275, 2006.

[16] C. Bajaj and S. Goswami. Secondary structure elucidation from medium resolution cryo-em maps. Technical report, Institute of Computational Engineering and Sciences, University of Texas at Austin, 2009.

[17] C. Bajaj, I. Ihm, and J. Warren. Higher-order interpolation and least-square approximation using implicit algebraic surfaces. *ACM Transactions on Graphics*, 12(4):327–347, 1993.

[18] C. Bajaj, V. Pascucci, and D. Schikore. Fast isocontouring for improved interactivity. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 39–46, 1996.

[19] C. Bajaj, V. Pascucci, and D. Schikore. The contour spectrum. In *IEEE Visualization*, pages 167–174, 1997.

[20] C. Bajaj and V. Siddavanahalli. *Adaptive Grid Based Methods for Computing Molecular Surfaces and Properties*. Technical report, Institute for Computational and Engineering Sciences, The University of Texas at Austin, 2006.

[21] C. Bajaj and V. Siddavanahalli. *Fast Error-bounded Surfaces and Derivatives Computation for Volumetric Particle Data*. Technical report, Institute for Computational and Engineering Sciences, The University of Texas at Austin, 2006.

[22] C. Bajaj, J. Warren, and G. Xu. A subdivision scheme for hexahedral meshes. *The Visual Computer*, 18(5-6):343–356, 2002.

[23] C. Bajaj, Q. Wu, and G. Xu. Level set based volumetric anisotropic diffusion. In *ICES Technical Report 301, The Univ. of Texas at Austin*, 2003.

[24] C. Bajaj and G. Xu. Smooth adaptive reconstruction and deformation of free-form fat surfaces. Technical report, Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin, 1999. techreport.

[25] C. Bajaj and G. Xu. Adaptive surface fairing by geometric diffusion. In *Information Visualisation, International Symposium of Computer Aided Geometric Design, IV2001-CAGD, England*, pages 731–737. IEEE Computer Society, 2001. inproceedings.

[26] C. Bajaj and G. Xu. Anisotropic diffusion of noisy surfaces and noisy functions on surfaces. In *Proceedings of Information Visualization 2001 Cmputer Aided Geometric Design*, 2001.

[27] C. Bajaj and G. Xu. Anisotropic diffusion of subdivision surfaces and functions on surfaces. *ACM Transactions on Graphics*, 22(1):4–32, 2003.

[28] C. Bajaj, G. Xu, and Q. Zhang. Smooth surface construction via a higher-order level-set method. Technical Report 06-18, Institute of Computatioanl Engineering and Sciences, the University of Texas at Austin, 2006.

[29] C. Bajaj, G. Xu, and Q. Zhang. A higher-order level-set method with applications to smooth surface constructions. Technical report, 2010.

[30] C. Bajaj, G. Xu, and Q. Zhang. Smooth surface constructions via a higher-order level-set method. In *Proc. of CAD/Graphics 2007*, Accepted.

[31] C. Bajaj and Z. Yu. Geometric processing of reconstructed 3D maps of macromolecular complexes. In S. Aluru, editor, *Handbook of Computational Molecular Biology*, page Chap 45. Chapman and Hall, CRC Press, 2005.

[32] C. L. Bajaj, E. J. Coyle, and K.-N. Lin. Tetrahedral meshes from planar cross sections. In *Comput. Methods Appl. Mech. Engineering*, 1999.

[33] C. L. Bajaj and Z. Yu. Geometric processing of reconstructed 3D maps of molecular complexes. In S. Aluru, editor, *Handbook of Computational Molecular Biology*. Chapman & Hall, 2005.

[34] M. L. Baker, Z. Yun, W. Chiu, and C. L. Bajaj. Automated segmentation of molecular subunits in electron cryomicroscopy density maps. *J. Struct. Bio.*, 156:432–441, 2006.

[35] N. Baker, D. Sept, S. Joseph, M. Holst, and J. McCammon. Electrostatics of nanosystems: application to microtubules and the ribosome. In Proc. Natl. Acad. Sci. , USA, 2001.

[36] T. Baker. Identification and preservation of surface features. In *13th International Meshing Roundtable*, pages 299–310, 2004.

[37] T. S. Baker, N. H. Olson, and S. D. Fuller. Adding the third dimension to virus life cycles: three-dimensional reconstruction of icosahedral viruses from cryo-electron micrographs. *Microbio. and Mol. Bio. Reviews*, 63(4):862–922, 1999.

[38] G. Barequet and M. Shair. Piecewise-linear interpolation between polygonal slices. In *10th Computational Geometry*, pages 93–102, 1994.

[39] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, pages 235–242, 2000.

[40] M. W. Bern, D. Eppstein, and J. R. Gilbert. Provably Good Mesh Generation. In *IEEE Symposium on Foundations of Computer Science*, pages 231–241, 1990.

[41] F. Bernardini, C. Bajaj, J. Chen, and D. Schikore. Automatic reconstruction of 3D cad models from diital scans. *Int. Journal. on Comp. Geometry. and Applications*, 9(4-5):327, August & October 1999.

[42] A. Biswas, V. Shapiro, and I. Tsukanov. Heterogeneous material modeling with distance fields. 21:215–242, 2004.

[43] T. Blacker and R. Myers. Seams and Wedges in Plastering: a 3D Hexahedral Mesh Generation Algorithm. *Engineering With Computers*, 2:83–93, 1993.

[44] J. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[45] J. D. Boissonnat. Shape reconstruction from planar cross sections. *Comp. Vis. Grap. & Image Proc.*, (44):1–29, 1988.

[46] J.-D. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67:405–451, 2005.

[47] F. J. Bossen and P. S. Heckbert. A Pliant Method for Anisotropic Mesh Generation. In *5th International Meshing Roundtable*, pages 63–76, 1996.

[48] C. Branden and J. Tooze. *An Introduction to Protein Structure*. Garland Publishing Inc., New York, 1999.

[49] Y. Bresler, J. A. Fessler, and A. Macovski. A bayesian approach to reconstruction from incomplete projections of a multiple object 3d domain. *IEEE Trans. on Patt. Anal. Mach. Intell.*, 11(8):840–858, Aug. 1989.

[50] S. Canann. Plastering and Optismoothing: New Approaches to Automated, 3D Hexahedral Mesh Generation and Mesh Smoothing. *Ph.D. Dissertation, Brigham Young University, Provo, UT*, 1991.

[51] S. Canann, J. Tristano, and M. Staten. An Approach to Combined Laplacian and Optimization-Based Smoothing for Triangular, Quadrilateral and Quad-Dominant Meshes. In *7th International Meshing Roundtable*, pages 479–494, 1998.

[52] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Computational Geometry : Theory and Applications*, 24(2):75–94, 2003.

[53] E. Catmull and J. Clark. Recursively generated *b*-spline surfaces on arbitrary topological surfaces. 10(6):350–355, 1978.

[54] F. Cazals and F. Proust. On the topology and the geometry of (voronoi) molecular interfaces. part I: algorithms. Technical report, INRIA, Oct. 2004.

[55] F. Cazals and F. Proust. Revisting the description of protein-protein interface, Pert II: Experimental study. Technical report, Inria Technical Report, 2005.

[56] R. Chaine. A geometric convection approach of 3-d reconstruction. In *Proc. Eurographics Sympos. on Geometry Processing*, pages 218–229, 2003.

[57] C. Charalambous and A. Conn. An Efficient Method to Solve The Minimax Problem Directly. *SIAM Journal of Numerical Analysis*, 15(1):162–187, 1978.

[58] F. Chazal and A. Lieutier. Stability and homotopy of a subset of the medial axis. In *Proc. 9th ACM Sympos. Solid Modeling and Applications*, pages 243–248, 2004.

[59] S.-W. Cheng, T. Dey, H. Edelsbrunner, M. Facello, and S. Teng. Sliver Exudation. *Proceeding Journal of ACM*, 47:883–904, 2000.

[60] S.-W. Cheng, T. Dey, E. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. In *Proc. of the 20th Annual Symposium on Computational Geometry*, pages 280–289, 2004.

[61] S.-W. Cheng and T. K. Dey. Quality Meshing with Weighted Delaunay Refinement. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms (SODA 2002)*, pages 137–146, 2002.

[62] S.-W. Cheng, T. K. Dey, and E. A. Ramos. Delaunay refinement for piecewise smooth complexes. In *SODA*, pages 1096–1105, 2007.

[63] E. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. Technical report, 1995.

[64] L. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proc. SoCG '93*, pages 274–280, 1993.

[65] P. Chew. Guaranteed-quality delaunay meshing in 3D. In *ACM Symposium on Computational Geometry*, pages 391–393, 1997.

[66] H. N. Christiansen and T. W. Sederberg. Conversion of complex contour line definitions. *Computer Grapics*, pages

187–192, Aug. 1978.

[67] U. Clarenz, U. Diewald, and M. Rumpf. Anisotropic geometric diffusion in surface processing. In *VIS '00: Proceedings of the Conference on Visualization*, pages 397–405, Los Alamitos, CA, USA, 2000. inproceedings.

[68] U. Clarenz, U. Diewald, and M. Rumpf. Nonlinear Anisotropic Diffusion in Surface Processing. In *Proceedings of the 11th IEEE Visualization*, pages 397–405, 2000.

[69] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[70] Cocone. Tight Cocone Software for surface reconstruction and medial axis approximation. *http://www.cse.ohio-state.edu/∼tamaldey/cocone.html*.

[71] M. Connolly. Analytical molecular surface calculation. *J. Appl. Cryst.*, 16:548–558, 1983.

[72] C. Consortium. CGAL: Computational Geometry Algorithms Library. *http://www.cgal.org*.

[73] L. T. Cook, P. N. Cook, K. R. Lee, S. Batnitzky, B. Wong, S. L. Fritz, J. Ophir, S. J. D. III, L. R. Bigongiari, and A. W. Templeton. An algorithm for volume estimation based on polyhedral approximation. *IEEE Trans. Biomed. Eng.*, 10(6):949–955, Nov. 1988.

[74] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[75] M. Desbrun, M. Meyer, P. Schr
"oder, and A. Barr. Discrete differential-geometry operators in nD. *preprint, the Caltech Multi-Res Modeling Group*. http://www.multires.caltech.edu/pubs/.

[76] M. Desbrun, M. Meyer, P. Schr
"oder, and A. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, page 324. ACM Press/Addison-Wesley Publishing Co., 1999.

[77] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In F. Dehne, J.-R. Sack, and M. Smid, editors, *Proc. Workshop Algorithms Data Strucutres (WADS 03)*, LNCS 2748, pages 25–36, Berlin, Germany, 2003.

[78] T. K. Dey, J. Giesen, E. Ramos, and B. Sadri. Critical points of the distance to an epsilon-sampling of a surface and flow-complex-based surface reconstruction. In *Proc. 21st ACM-SIAM Sympos. Comput. Geom.*, pages 218–227, 2005.

[79] T. K. Dey and S. Goswami. Tight cocone: A water-tight surface reconstructor. In *Proc. 8th ACM Sympos. Solid Modeling and Applications*, pages 127–134, 2003.

[80] T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. In *Proc. 20th ACM-SIAM Sympos. Comput. Geom.*, pages 330–339, 2004.

[81] T. K. Dey and J. A. Levine. Delaunay meshing of isosurfaces. In *SMI '07: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007*, pages 241–250, Washington, DC, USA, 2007.

[82] T. K. Dey and W. Zhao. Approximating the Medial axis from the Voronoi diagram with convergence guarantee. *Algorithmica*, 38:179–200, 2004.

[83] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Boston, Basel, Berlin, 1992. book.

[84] D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. 10(6):356–360, 1978.

[85] B. S. Duncan and A. J. Olson. Shape analysis of molecular surface. *Biopolymers*, 33:231–238, 1993.

[86] H. Edelsbrunner. Surface reconstruction by wrapping finite point sets in space. In B. Aronov, S. Basu, J. Pach, and M. Sharir, editors, *Ricky Pollack and Eli Goodman Festschrift*, pages 379–404. Springer-Verlag, 2002.

[87] H. Edelsbrunner and N. Shah. Triangulating topological spaces. *Intl. Journal of Comput. Geom. and Appl.*, 7:365–378, 1997.

[88] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Trans. Graphics*, 10(2):182–199, Apr. 1991.

[89] T. T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, Aug. 1992.

[90] D. Eppstein. Linear Complexity Hexahedral Mesh Generation. In *Symposium on Computational Geometry*, pages 58–67, 1996.

[91] J. Escher, U. F. Mayer, and G. Simonett. The surface diffusion flow for immersed hypersurfaces. *SIAM Journal of Mathematical Analysis*, 29(6):1419–1433, 1998.

[92] D. Eu and G. T. Toussaint. On approximating polygonal curves in two and three dimensions. *Comp. Vis. Grap. & Image Proc.*, (3):231–246, 1994.

[93] D. Field. Laplacian Smoothing and Delaunay Triangulations. *Communications in Applied Numerical Methods*, 4:709–712, 1988.

[94] J. Frank. *Three-Dimensional Electron Microscope of Macromolecular Assemblies*. San Diego: Academic Press, 1996.
[95] L. Freitag. On Combining Laplacian and Optimization-Based Mesh Smoothing Techniqes. *AMD-Vol. 220 Trends in Unstructured Mesh Generation*, pages 37–43, 1997.
[96] L. Freitag and C. Ollivier-Gooch. Tetrahedral Mesh Improvement Using Swapping and Smoothing. *Int. J. Numer. Meth. Engng*, 40:3979–4002, 1997.
[97] L. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *International Journal for Numerical Method in Engineering*, 49:109–125, 2000.
[98] P. Frey, H. Borouchaki, and P.-L. George. Delaunay Tetrahedralization Using An Advancing-Front Approach. In *5th International Meshing Roundtable*, pages 31–48, 1996.
[99] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal surface reconstruction from plannar contours. *Com ACM*, 20:693–702, Oct. 1977.
[100] I. Fujishiro, Y. Maeda, H. Sato, and Y. Takeshima. Volumetric Data Exploration Using Interval Volume. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):144–155, 1996.
[101] S. Ganapathy and T. G. Dennehy. A new general triangulation method for planar contours. *Computer Grapics*, pages 69–75, 1982.
[102] R. V. Garimella, M. J. Shashkov, and P. M. Knupp. Optimization of surface mesh quality using local parametrization. In *11th International Meshing Roundtable*, pages 41–52, 2002.
[103] M. Garland and P. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization*, pages 263–270, 1998.
[104] B. Geiger. *Three-dimensional modeling of human organs and its application to diagnosis and surgical planning*. PhD thesis, INRIA, France, 1993.
[105] P. L. George. Tet Meshing: Construction, Optimization and Adaptation. In *8th International Meshing Roundtable*, pages 133–141, 1999.
[106] P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing, Application to Finite Elements*, pages 230–234, 1998.
[107] P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing, Application to Finite Elements*. 1998.
[108] M. M. Gibbons and W. S. Klug. Influence of nonuniform geometry on nanoindentation of viral capsids. *Biophys. J.*, 95:3640–3649, 2008.
[109] L. J. Gibson and M. F. Ashby. *Cellular Solids: Structure and Properties*. UK: Cambridge University Press, second edition, 1999.
[110] S. Gibson. Using Distance Maps for Accurate Surface Representation in Sampled Volumes. In *IEEE sympos. on Volume visualization*, pages 23–30, 1998.
[111] J. Giesen and M. John. The flow complex: a data structure for geometric modeling. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*, pages 285–294. ACM, 2003.
[112] C. Gitlin, J. O'Rourke, and V. Subramanian. On reconstructing polyhedra from parallel slices. Technical Report 25, Dept. Comput. Sci., Smith College, Northampton, MA, 1993.
[113] S. Goswami, T. Dey, and C. Bajaj. Identifying flat and tubular regions of a shape by unstable manifolds. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, page 37. ACM, 2006.
[114] S. Goswami, T. K. Dey, and C. L. Bajaj. Identifying flat and tubular regions of a shape by unstable manifolds. In *Proc. 11th Sympos. Solid and Physical Modeling*, pages 27–37, 2006.
[115] S. Goswami, A. Gillette, and C. Bajaj. Efficient Delaunay mesh generation from sampled scalar function. In *Proc. 16th Int. Meshing Roundtable*, page to appear, 2007.
[116] S. Goswami, A. Gillette, and C. Bajaj. Efficient Delaunay mesh generation from sampled scalar functions. In *Proceedings of the 16th International Meshing Roundtable*, pages 495–512. Springer, 2008.
[117] J. Grant and B. Pickup. A Gaussian description of molecular shape. *Journal of Phys. Chem.*, 99:3503–3510, 1995.
[118] V. Guillemin and A. Pollack. *Differential Topology*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.
[119] I. Guskov, A. Khodakovsky, P. Schroder, and W. Sweldens. Hybrid meshes: multiresolution using regular and irregular refinement. In *SCG '02: Proc. of the 18th Annual Symposium on Computational Geometry*, pages 264–272, 2002.
[120] P. Hansbo. Generalized Laplacian Smoothing of Unstructured Grids. *Communications in Numerical Methods in Engineering*, 11:455–464, 1995.
[121] M. Holst and F. Saied. Multigrid solution of the poisson-boltzmann equation. *J. Comput. Chem*, 14:105–113, 1993.
[122] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. pages 71–78, 1992.

[123] K. Hormann, U. Labsik, M. Meister, and G. Greiner. Hierarchical extraction of iso-surfaces with semi-regular meshes. In *SMA '02: Proc. of 7th ACM Symposium on Solid Modeling and Applications*, pages 53–58, 2002.

[124] W. Jiang, Z. Li, Z. Zhang, M. L. Baker, P. E. P. Jr., and W. Chiu. Coat protein fold anf maturation transition of bacteriophoage P22 seen at subnanometer resolutions. *Nature Structural Biology*, 10(2):131–135, 2003.

[125] J. Jost. *Riemannian geometry and geometric analysis*. Springer Verlag, second edition, 1998.

[126] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual Contouring of Hermite Data. In *SIGGRAPH*, pages 339–346, 2002.

[127] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of Hermite data. pages 339–346, 2002.

[128] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *Trans. on Graphics*, volume 3, pages 954–961, 2003.

[129] N. Kehtarnavaz, L. R. Simar, and R. D. Figueiredo. A syntactic/semantic technique for surface reconstruction from cross-section contours. *Comp. Vis. Grap. & Image Proc.*, (42):399–409, 1988.

[130] E. Keppel. Approximating complex surface by triangulation of contour lines. *IBM Journal of Research & Development*, (19):2–11, 1975.

[131] P. Kinney. CleanUp: Improving Quadrilateral Finite Element Meshes. In *Proceedings of the 6th International Meshing Roundtable*, pages 437–447, 1997.

[132] P. Knupp. Winslow smoothing on two dimensinoal unstructured meshes. *Engineering with Computers*, 15:263–268, 1999.

[133] P. Knupp. Achieving Finite Element Mesh Quality via Optimization of The Jacobian Matrix Norm and Associated Quantities. Part I - A Framework for Surface Mesh Optimization. *Int. J. Numer. Meth. Engng*, 48:401–420, 2000.

[134] P. Knupp. Achieving Finite Element Mesh Quality via Optimization of The Jacobian Matrix Norm and Associated Quantities. Part II - A Framework for Volume Mesh Optimization and The Condition Number of The Jacobian Matrix. *International Journal of Numerical Methods in Engineering*, 48:1165–1185, 2000.

[135] L. Kobbelt, M. Botsch, U. Schwanecke, and H. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM, 2001.

[136] C. Kober and M. Matthias. Hexahedral Mesh Generation for the Simulation of The Human Mandible. In *Proceedings of the 9th International Meshing Roundtable*, pages 423–434, 2000.

[137] Y. Kong and J. Ma. A structural-informatics approach for mining β- sheets: locating sheets in intermediate-resolution density maps. *J. of Mol. Bio*, 332:399–413, 2003.

[138] Y. Kong, X.Zhang, T.S.Baker, and J. Ma. A structural-informatics approach for tracing β-sheets: building pseudo-Ca traces for β-strands in intermediate-resolution density maps. *J. of Mol. Bio.*, 339:117–130, 2004.

[139] F. Labelle and J. Shewchuk. Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles. In *SIGGRAPH (to appear)*, 2007.

[140] J.-O. Lachaud. Topologically defined iso-surfaces. In *DCGA '96: Proc. of the 6th Intl. Workshop on Discrete Geometry for Computer Imagery*, pages 245–256, 1996.

[141] R. Laramee and R. Bergeron. An Isosurface Continuity Algorithm for Super Adaptive Resolution Data. *Advances in Modelling, Animation, and Rendering: Computer Graphics International (CGI)*, pages 215–237, 2002.

[142] D. T. Lee. Medial axis transformation of a planar shape. *IEEE Trans. on Patt. Anal. Mach. Intell.*, 4(4):363–369, July 1982.

[143] J. Liang, H. Edelsbrunner, P. Fu, and P. V. Sudhakar. Analytical shape computation of macromolecules: I. molecular area and volume through aplha shape. 33:1–17, 1998.

[144] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical Shape Computation of Macromolecules: II. Inaccessible Cavities in Protein. *Proteins: Structure, Function, and Genetics*, 33:18–29, 1998.

[145] W. C. Lin and S. Y. Chen. A new surface interpolation technique for reconstructing 3D objects from serial cross-sections. 48:124–143, 1989.

[146] E. B. D. l'Isle and P. L. George. Optimization of Tetrahedral Meshes. *IMA Volumes in Mathematics and its Applications*, 75:97–128, 1995.

[147] A. Liu and B. Joe. Relationship Between Tetrahedron Shape Measures. *BIT*, 34(2):268–287, 1994.

[148] R. Lohner, K. Morgan, and O. Zienkiewicz. Adaptive Grid Refinement for Compressible Euler Equations. *Accuracy Estimates and Adaptive Refinements in Finite Element Computations, I. Babuska et. al. eds. Wiley*, pages 281–297, 1986.

[149] R. Lohner and P. Parikh. Three Dimensional Grid Generation by The Advancing-Front Method. *International Journal for Numerical Methods in Fluids*, 8:1135–1149, 1988.

[150] C. Loop. Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics*, 1987.

[151] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, 1987.
[152] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. In *IEEE Transactions on Visualization and Computer Graphics*, pages 19–26, 2000.
[153] A. Lopes and K. Brodlie. Improving the robustness and accuracy of the marching cubes algorithm for isosurfacing. In *IEEE Transactions on Visualization and Computer Graphics*, volume 9, pages 16–29, 2003.
[154] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *SIGGRAPH*, pages 163–169, 1987.
[155] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of SIGGRAPH*, pages 163–169, 1987.
[156] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. pages 163–169, 1987.
[157] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface reconstruction algorithm. *Computer Grapics*, pages 163–169, 1987.
[158] S. Ludtke, P. Baldwin, and W. Chiu. EMAN: semiautomated software for high-resolution single-particle reconstructions. *J. of Struct. Bio.*, 128:82–97.
[159] M. Meyer, M. Desbrun, P. Schröder, and A. Burr. Discrete differential-geometry operators for triangulated 2-manifolds. *VisMath'02, Berlin*, 2002.
[160] M. Meyer, M. Desbrun, P. Schröder, and A. Burr. Discrete Differential-Geometry Operators for Triangulated 2-Manifolds. *VisMath'02, Berlin*, 2002.
[161] D. Meyers. *Reconstruction of Surfaces From Planar Contours*. PhD thesis, University of Washington, 1994.
[162] D. Meyers, S. Skinner, and K. Sloan. Surfaces from contours. *ACM Trans. Graphics*, 11(3):228–258, July 1992.
[163] S. Mitchell and T. Tautges. Pillowing Doublets: Refining a Mesh to Ensure That Faces Share at Most One Edge. In *Proceedings of the 4th International Meshing Roundtable*, pages 231–240, 1995.
[164] S. A. Mitchell and S. A. Vavasis. Quality Mesh Generation in Higher Dimensions. *SIAM Journal on Computing*, 29(4):1334–1370, 2000.
[165] D. Moore. Compact Isocontours from Sampled Data. *Graphics Gems III*, pages 23–28, 1992.
[166] M. Mortara, G. Patanè, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Blowing bubbles for the multi-scale analysis and decomposition of triangle meshes. *Algorithmica*, 38:227–248, 2004.
[167] M. Mortara, G. Patanè, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Plumber: a multi-scale decomposition of 3D shapes into tubular primitives and bodies. In *Proc. 9th ACM Sympos. Solid Modeling and Applications*, pages 139–158, 2004.
[168] B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11:52–62, 1994.
[169] B. K. Natarajan. On generating topologically consistent isosurfaces from uniform samples. *The Visual Computer*, 11(1):52–62, 1994.
[170] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30:854–879, 2006.
[171] G. Nielson and J. Sung. Interval Volume Tetrahedrization. In *IEEE Visualization*, pages 221–228, 1997.
[172] A. Oddy, J. Goldak, M. McDill, and M. Bibby. A Distortion Metric for Isoparametric Finite Elements. *Transactions of CSME, No. 38-CSME-32, Accession No. 2161*, 1988.
[173] J. O'Rourke. On the scaling heuristic for reconstruction from slices. *CVGIP GMIP*, 56(3):420–423, May 1994.
[174] S. Osher and R. Fedkiw. *Level Set Method and Dynamic Implicit Surfaces*. Springer, New York, 2003.
[175] S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.
[176] S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. 79(1):12–49, 1988.
[177] S. Osher and C. Shu. High-order essentially nonoscillatory schemes for hamilton-jacobi equations. *SIAM Journal of Numerical Analysis*, 28(4):907–922, 1991.
[178] S. Oudot, L. Rineau, and M. Yvinec. Meshing volumes bounded by smooth surfaces. In *Proc. 14th Intl. Meshing Roundtable*, pages 203–219, 2005.
[179] S. Owen. A Survey of Unstructured Mesh Generation Technology. In *7th Internat. Meshing Roundtable*, pages 26–28, 1998.
[180] V. Pascucci and C. Bajaj. Time critical adaptive refinement and smoothing. In *Proceedings of the ACM/IEEE Volume Visualization and Graphics Symposium*, pages 33–42, 2000.

[181] V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *IEEE Visualization Conference*, pages 187–194, 2002.

[182] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang. A PDE-Based Fast Local Level Set Method. *Journal of Computational Physics*, 155:410–438, 1999.

[183] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Machine Intell*, 12:629–639, 1990. TY - JOUR.

[184] J. Peters and U. Reif. *Subdivision Surfaces*. Springer Berlin Heidelberg, 2008.

[185] H. Pottmann, M. Hofer, B. Odehnal, and J. Wallner. Line geometry for 3D shape understanding and reconstruction. *Computer Vision - ECCV*, 3021(1):297–309, 2004.

[186] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[187] T. Preuber and M. Rumpf. An adaptive finite element method for large scale image processing. *Scale-Space Theories in Computer Vision*, pages 232–234, 1999.

[188] M. Price and C.Armstrong. Hexahedral Mesh Generation by Medial Surface Subdivision: Part II, Solids with Flat and Concave Edges. *Int. J. Numer. Meth. Engng*, 40:111–136, 1997.

[189] M. A. Price and C. G. Armstrong. Hexahedral Mesh Generation by Medial Surface Subdivision: Part I. *International Journal for Numerical Methods in Engineering*, 38(19):3335–3359, 1995.

[190] G. D. Reis, B. Mourrain, R. Rouillier, and P. Trébuchet. An environment for symbolic and numeric computation. In *Proc. Internat. Conf. on Mathematical Software*, pages 239–249, 2002.

[191] F. Richards. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioeng.*, 6:151–176, 1977.

[192] B. Romeny. *Geometry-driven diffusion in computer vision*. Kluwer Academic Publishers Norwell, MA, USA, 1994.

[193] S. Rosenberg. *The Laplacian on a Riemannian manifold: an introduction to analysis on manifolds*. Cambridge Univ Pr, 1997.

[194] M. Sanner, A. Olson, and J. Spehner. Reduced Surface: An Efficient Way to Compute Molecular Surfaces. *Biopolymers*, 38:305–320, 1996.

[195] G. Sapiro. *Geometric partial differential equations and image analysis*. Cambridge Univ Pr, 2001.

[196] G. Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge, University Press, 2001.

[197] B. Schneider, Z. Morveki, and H. Berman. RNA conformational classes. *Nucleic Acids Research*, 32, 5:1666–1677, 2004.

[198] R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free-form surface design. 18(4):359–379, 2001.

[199] R. Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering With Computers*, 12:168–177, 1996.

[200] R. Schneiders. Refining Quadrilateral and Hexahedral Element Meshes. In *5th International Conference on Grid Generation in Computational Field Simulations*, pages 679–688, 1996.

[201] R. Schneiders. An Algorithm for The Generation of Hexahedral Element Meshes Based on An Octree Technique. In *Proceedings of the 6th International Meshing Roundtable*, pages 195–196, 1997.

[202] I. B. Semenova, V. V. Savchenko, and I. Hagiwara. Two techniques to improve mesh quality and preserve surface characteristics. In *13th International Meshing Roundtable*, pages 277–288, 2004.

[203] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Visison, and Materials Science*. Cambridge Monographs on Applied and Computational Mathematical. Cambridge University Press, second edition, 1999.

[204] E. Seveno. Towards An Adaptive Advancing Front Method. In *6th International Meshing Roundtable*, pages 349–362, 1997.

[205] M. Shantz. Surface definition for branching contour defined objects. *Computer Grapics*, pages 242–270, July 1981.

[206] M. Shephard and M. Georges. Three-Dimensional Mesh Generation by Finite Octree Technique. *International Journal for Numerical Methods in Engineering*, 32:709–749, 1991.

[207] J. Shewchuk. Constrained Delaunay Tetrahedrizations and Provably Good Boundary Recovery. In *11th International Meshing Roundtable*, pages 193–204, 2002.

[208] J. R. Shewchuk. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the 4th Annual Symposium on Computational Geometry, Association for Computational Machinery*, pages 86–95, 1998.

[209] J. R. Shewchuk. Two Discrete Optimization Algorithms for The Topological Improvement of Tetrahedral Meshes. In *Unpublished manuscript*, 2002.

[210] J. R. Shewchuk. What is A Good Linear Element? Interpolation, Conditioning, and Quality Measures. In *Proceedings of the 11th International Meshing Roundtable, Sandia National Lab*, pages 115–126, 2002.

[211] K. Shimada, A. Yamada, and T. Itoh. Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles. In *6th International Meshing Roundtable*, pages 375–390, 1997.

[212] Y. Shinagawa and T. L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *The Visual Computer*, 7:72–86, 1991.

[213] Y. Shinagawa, T. Kunni, A. Belayev, and T. Tsukioka. Shape modeling and shape analysis based on singularities. *Internat. J. Shape Modeling*, 2:85–102, 1996.

[214] S. M. Shontz and S. A. Vavasis. A mesh warping algorithm based on weighted laplacian smoothing. In *12th International Meshing Roundtable*, pages 147–158, 2003.

[215] K. Siddiqi, A. Shokoufandeh, J. Dickinson, and S. Zucker. Shock graphs and shape matching. *Computer Vision*, pages 222–229, 1998.

[216] D. Siersma. Voronoi diagrams and morse theory of the distance function. In O. E. Barndorff and E. B. V. Jensen, editors, *Geometry in Present Day Science*, pages 187–208. 1999.

[217] K. R. Sloan and J. Painter. Pessimal guesses may be optimal: a counterintuitive search result. *IEEE Trans. on Patt. Anal. Mach. Intell.*, 10(6):949–955, Nov. 1988.

[218] Y. Song, Y. Zhang, T. Shen, C. L. Bajaj, J. A. McCammon, and N. A. Baker. Finite element solution of the steady-state smoluchowski equation for rate constant calculations. *Accepted in Biophysical Journal*, 2003.

[219] B. I. Soroka. Generalized cones from serial sections. (15):154–166, 1981.

[220] V. Srinivasan and L. R. Nackman. Voronoi diagram for multiply-connected polygonal domains I: Algorithm. *IBM jrd*, 31(3):361–372, May 1987.

[221] J. Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, page 404. ACM, 1998.

[222] M. Staten and S. Canann. Post Refinement Element Shape Improvement for Quadrilateral Meshes. *AMD-Trends in Unstructured Mesh Generation*, 220:9–16, 1997.

[223] G. Taubin. Distance Approximations for Rasterizing Implicit Curves. *ACM Transactions on Graphics*, 14(3–42), 1994.

[224] G. Taubin. A signal processing approach to fair surface design. In *SIGGRAPH '95 Proceedings*, pages 351–358, 1995.

[225] T. Tautges, T. Blacker, and S. Mitchell. The Whisker-Weaving Algorithm: A Connectivity Based Method for Constructing All-Hexahedral Finite Element Meshes. *Int. J. Numer. Meth. Engng*, 39:3327–3349, 1996.

[226] S.-H. Teng and C. Wong. Unstructured Mesh Generation: Theory, Practice and Perspectives. *International Journal of Computational Geometry and Applications*, 10(3):227–266, 2000.

[227] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of IEEE International Conference on Computer Vision*, pages 839–846, 1998.

[228] T. Várady, R. Martin, and J. Cox. Reverse engineering of geometric models - an introduction. *Computer Aided Design*, 29:255–268, 1997.

[229] A. Verroust and F. Lazarus. Extracting skeletal curves from 3d scattered data. *The Visual Computer*, 16:15–25, 2000.

[230] J. Vollmer, R. Mencl, and H. Müller. Improved Laplacian smoothing of noisy surface meshes. *Computer Graphics Forum*, 18:131–138, 1999.

[231] T. F. Walsh. *P Boundary Element Modeling of the Acoustical Transfer Properties of the Human Head/Ear*. PhD thesis, The University of Texas at Austin, Austin, TX, 2000.

[232] C. C. L. Wang. Direct extraction of surface meshes from implicitly represented heterogeneous volumes. 39:35–50, 2007.

[233] J. Wang, Z. Yang, and J. Deng. Blending surface with algebraic tensor-product b-spline surfaces. *Journal of University of Science and Technology of China*, 36(6):598–603, 2006.

[234] M. Y. Wang and X. Wang. A level-set based variational method for design and optimization of heterogeneous objects. 37:321–337, 2005.

[235] Y. F. Wang and J. K. Aggarawal. Surface reconstruction and representation of 3-d scenes. *Pattern Recognition*, 19(3):197–207, 1986.

[236] J. Warren. *Subdivision method for geometric design*. 1995.

[237] J. Weickert. *Anisotropic Diffusion In Image Processing*. ECMI Series, Teubner, Stuttgart, ISBN 3-519-02606-6, 1998. TY - BOOK.

[238] J. Weickert. Anisotropic diffusion in image processing. Teubner Verlag, Stuttagart, 1998.

[239] J. Wen, M. Baker, S. Ludtke, and W. Chiu. Bridging the information gap: computational tools for intermediate resolution structure interpretation. *J. of Mol. Bio.*, 308:1033–1044, 2001.

[240] R. Westermann, L. Kobbelt, and T. Ertl. Real-Time Exploration of Regular Volume Data by Adaptive Reconstruction of Isosurfaces. *The Visual Computer*, 15(2):100–111, 1999.

[241] J. Wilhelm and A. V. Gelder. Octrees for faster isosurface generation. In *ACM Transactions on Graphics*, pages 57–62, 1992.

[242] T. J. Willmore. Riemannian geometry. *Clareden Press, Oxford*, 1993.

[243] Z. Wood, M. Desbrun, P. Schroder, and D. Breen. Semi-regular mesh extraction from volumes. In *Visualization Conference Proceedings*, pages 275–282, 2000.

[244] Z. Wood, H. Hoppe, M. Desbrun, and P. Schroder. Removing excess topology from isosurfaces. *ACM Transactions on Graphics*, 23(2):190–208, Apr. 2004.

[245] Z. Wood, P. Schroder, D. Breen, and M. Desbrun. Semi-regular mesh extraction from volumes. In *VIS '00: Proc. of the Conference on Visualization 2000*, pages 275–282. IEEE Computer Society Press, 2000.

[246] J. Wu and L. Kobbelt. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum*, 24(3):277–284, 2005.

[247] Z. Wu and S. J. M. Jr. Multiple material marching cubes algorithm. *International Journal for Numerical Methods in Engineering*, 58:189–207, 2003.

[248] G. Xu. Adaptive and smooth surface construction by triangular a-patches. *Computational geometry: lectures at the Morningside Center of Mathematics*, page 213, 2003.

[249] G. Xu. Discrete Laplace-Beltrami Operators and Their Convergence. *Computer Aided Geometry Design*, 21:767–784, 2004.

[250] G. Xu. Discrete Laplace-Beltrami operators and their convergence. 21(8):767–784, 2004.

[251] G. Xu. *Geometric Partial Differential Equation Methods in Computational Geometry*. Science Press of China, Beijing, 2008. In Chinese.

[252] G. Xu, Q. Pan, and C. Bajaj. Discrete surface modeling using geometric flows. *ICES Technical Report 03-38, the University of Texas at Austin*, 2003.

[253] G. Xu, Q. Pan, and C. Bajaj. Discrete Surface Modeling using Geometric Flows. *Technical Report TR-03-36, Department of Computer Sciences, University of Texas at Austin*, 2003.

[254] G. Xu, Q. Pan, and C. L. Bajaj. Discrete surface modelling using partial differential equations. 23(2):125–145, 2006.

[255] G. Xu and Q. Zhang. Construction of geometric partial differential equations in computational geometry. *Mathematica Numerica Sinica*, 28(4):337–356, 2006.

[256] G. Xu, Q. Zhang, and D. Liu. Smooth surface reconstruction from noisy scattered data — variational level set methods. *Journal of Computer-Aided Design & Computer Graphics*, 19(7):840–849, 2007.

[257] Z. Yang, J. Deng, and F. Chen. Dynamic implicit curve reconstruction based on approximate geometric distance. *Journal of Software*, 15(6):960–969, 2004.

[258] Z. Yang, J. Deng, and F. Chen. Fitting unorganized point clouds with active implicit b-spline curves. *Visual Computer*, 21(8):831–839, 2005.

[259] Z. Yang, C. Wu, J. Deng, and F. Chen. Specification of initial shapes for dynamic implicit curve/surface reconstruction. *J. Comput. Sci. Technol.*, 21(2):249–254, 2006.

[260] Z. Yu and C. Bajaj. Automatic ultra-structure segmentation of reconstructed cryo-em maps of icosahedral viruses. *IEEE Transactions on Image Processing*, 14, 9:1324–1337, 2005.

[261] Z. Yu and C. Bajaj. Computational approaches for automatic structural analysis of large bio-molecular complexes. Technical Report TR-06-05, Department of Computer Sciences, UT Austin, 2006.

[262] Z. Yu and C. L. Bajaj. Automatic ultrastructure segmentation of reconstructed Cryo-EM maps of icosahedral viruses. *IEEE Transactions on Image Processing: Special Issue on Molecular and Cellular Bioimaging.*, 14(9):1324–1337, 2005.

[263] Q. Zhang, B. Subramanian, G. Xu, and C. Bajaj. Quality multi-domain meshing for volumetric data. Obtained from Zhang Qin, 2010.

[264] Y. Zhang and C. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric data. In *13th International Meshing Roundtable*, pages 365–376, 2004.

[265] Y. Zhang, C. Bajaj, and B. Sohn. 3D finite element meshing from imaging data. *Computer methods in applied mechanics and engineering*, 194(48-49):5083–5106, 2005.

[266] Y. Zhang, C. Bajaj, and B.-S. Sohn. Adaptive and quality 3d meshing from imaging data. In *Proc. of 8th ACM Symposium on Solid Modeling and Applications*, pages 286–291, June 2003.

[267] Y. Zhang, C. Bajaj, and B.-S. Sohn. 3d finite element meshing from imaging data. *the special issue of Computer Methods in Applied Mechanics and Engineering (CMAME) on Unstructured Mesh Generation, in press, www.ices.utexas.edu/∼jessica/meshing*, 2004.

[268] Y. Zhang, C. Bajaj, and B.-S. Sohn. 3D finite element meshing from imaging data. *The special issue of Computer*

Methods in Applied Mechanics and Engineering (CMAME) on Unstructured Mesh Generation, 194(48-49):5083–5106, 2005.

[269] Y. Zhang, C. Bajaj, and G. Xu. Surface smoothing and quality improvement of quadrilateral/hexahedral meshes with geometric flow. *Communications in numerical methods in engineering*, 25(1):1–18, 2009.

[270] Y. Zhang, T. J. R. Hughes, and C. L. Bajaj. An automatic 3D mesh generation method for domains with multiple materials. *Comput. Methods Appl. Mech. Engrg.*, 199(5–8):405–415, 2009.

[271] Y. Zhang, G. Xu, and C. Bajaj. Quality meshing of implicit solvation models of biomolecular structures. *Computer Aided Geometric Design*, 23(6):510–530, 2006.

[272] H. Zhao, T. Chan, B. Merriman, and S. Osher. A variational level set approach to multi-phase motion. *J. Comput. Phys.*, 122:179, 1996.

[273] H. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method, 2001. 1st IEEE Workshop on Variational and Level Set Methods, 8th ICCV, Vancouver, 194-202, 2001.

[274] H. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit nonparametric shape reconstruction from unorganized points using a variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.

[275] H. K. Zhao, S. Osher, and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proceedings of the IEEE Workshop on Variational and Level Set Methods in Computer Vision (VLSM2001)*, pages 194–201, 2001.

[276] H. K. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit nonparametric shape reconstruction from unorganized points using a variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.

[277] T. Zhou and K. Shimada. An angle-based approach to two-dimensional mesh smoothing. In *9th International Meshing Roundtable*, pages 373–384, 2000.

[278] Z. H. Zhou, M. L. Baker, W. Jiang, M. Dougherty, J. Jakana, G. Dong, G. Lu, and W. Chiu. Electron cryomicroscopy and bioinformatics suggest protein fold models for rice dwarf virus. *Nature Structural Biology*, 8(10):868–873, 2001.

[279] Z. H. Zhou, M. Dougherty, J. Jakana, J. He, F. J. Rixon, and W. Chiu. Seeing the herpesvirus capsid at 8.5 å. *Science*, Vol. 288(5467):pp. 877 – 880, 2000.

Exercises

Chapter 5

Appendix

5.1 Mesh Decompositions

Polynomial, piecewise polynomials, factoring, gcd

5.2 Interpolants

equations, root finding, subdivision, evaluation, interpolation singularity, arithmetic genus, rationality normals, curvatures,

5.3 Topology

spaces and their classification (open, closed, compact etc.) , equivalence of spaces (Homeomorphism, Isotopy, Homotopy etc.).

5.4 Analysis

integral moments, quadrature, cubature. Fourier Series Norm, Functional Spaces

5.5 Statistics

Sampling Theory, Nyquist theorem Probability, random variable, expectation, variance, conditional probability.

Chapter 6

Appendix

6.1 Computer Graphics

6.2 User Interfaces

6.3 GPU