Time Critical Isosurface Refinement And Smoothing[*]

V. Pascucci
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

C. L. Bajaj
Center for Computational Visualization
CS & TICAM, University of Texas at Austin

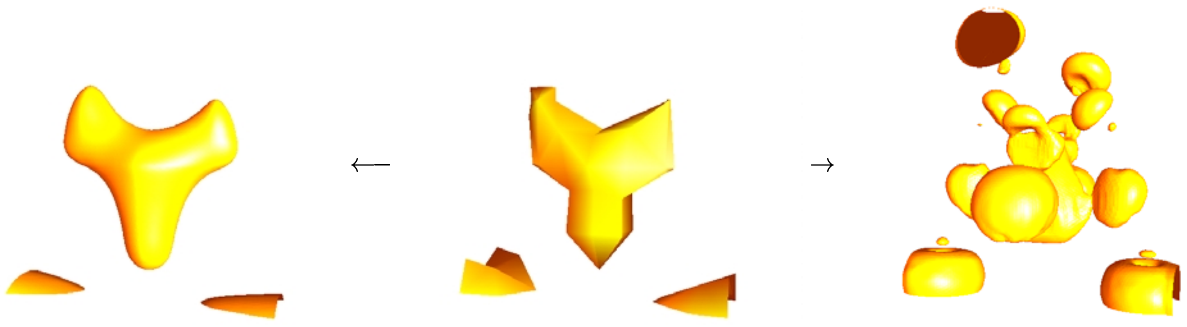


Figure 1: (left) Smoothing and (right) refinement with smoothing computed from a coarse representation (center) using the same progressive algorithm.

Abstract

Multi-resolution data-structures and algorithms are key in Visualization to achieve real-time interaction with large data-sets. Research has been primarily focused on the off-line construction of such representations mostly using decimation schemes. Drawbacks of this class of approaches include: (i) the inability to maintain interactivity when the displayed surface changes frequently, (ii) inability to control the global geometry of the embedding (no self-intersections) of any approximated level of detail of the output surface.

In this paper we introduce a technique for on-line construction and smoothing of progressive isosurfaces (see Figure 1). Our hybrid approach combines the flexibility of a progressive multi-resolution representation with the advantages of a recursive subdivision scheme. Our main contributions are: (i) a progressive algorithm that builds a multi-resolution surface by successive refinements so that a coarse representation of the output is generated as soon as a coarse representation of the input is provided, (ii) application of the same scheme to smooth the surface by means of a 3D recursive subdivision rule, (iii) a multi-resolution representation where any adaptively selected level of detail surface is guaranteed to be free of self-intersections.

[*]This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. Research supported in part by NSF grants CCR-9732306, DMS-9873326, ACI-9982297 and Sandia/LLNL DOE ASCI-BD4485

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surfaces and object representations

Keywords: Multi-resolution data-structures, progressive algorithms, recursive subdivision, smooth isosurfaces, consistent embedding.

1 INTRODUCTION

The current evolution of 3D scanning devices, computer simulations and modeling systems gives rise to surface and volumetric meshes of increasingly high complexity. The real-time display and transmission of such high resolution data is a challenging task requiring the fast generation of approximated representations. In recent years a great deal of research has been focused on the problem of constructing hierarchical representations with multiple levels of detail for any given surface [6, 8, 9, 11, 13, 17, 22, 24, 28, 29, 42]. The approaches developed have been successful in the off-line construction of quality multi-resolution surfaces. At run-time they allow fast selection of adaptive levels of detail for improved display speed. Unfortunately the time necessary to build such multi-resolution representations is too high to allow run-time modification of the surface. Moreover the simplification primitives used, like edge/triangle contraction or vertex removal, can easily lead to self-intersecting surfaces. The same problem of self-intersections can arise during the smoothing of the surface with a subdivision scheme.

In this paper we present one solution of such problems restricting our attention to the case of meshes computed as isocontours of 2D or 3D scalar fields [32]. An isocontour or isosurface $C(w)$ of a scalar field $\mathcal{F}(\mathbf{x})$ is defined as the locus of points where the field has value w, $C(w) \doteq \{\mathbf{x}|\mathcal{F}(\mathbf{x}) - w = 0\}$. Computation of isosurfaces is used for surface reconstructions from volumetric input. Scientific data (large physically based simulations, CT/MRI medical scans, etc.) are visualized using isosurfaces.

Adopting a multi-resolution data-structures for the output surface allows one to trade accuracy for speed and thereby achieve interactive response times in graphical display. In particular four predominant scenarios may arise in trying to achieve real time display:

Quality-Driven. Compute quickly the simplest surface satisfying a given error tolerance ϵ.

Time-Critical. Within a given time bound τ_1 compute the most accurate output surface. Within a second time interval τ_2 provide the best visual improvement of the output surface.

Asynchronous. Same as Time-Critical but with unknown τ_1 and τ_2. The computation is asynchronously interrupted at the deadline.

Dynamic. Solve either of the above while the isovalue w changes frequently (off-line hierarchy construction not feasible).

In this paper we introduce a progressive algorithm that solves all four scenarios above. The main feature of our scheme is the ability to extract in a multi-resolution fashion the input volumetric data to build a multi-resolution version of the output by successive refinements.

We use a refinement primitive that updates a local portion of the output so that a consistent output mesh is maintained at any given time. This allows for asynchronous termination of the computation within a small constant delay (termination of a critical section). Moreover the refinement operation is guaranteed to be performed within a small volumetric cell c. In particular the refinement primitive maintains a consistent portion of the surface inside c. This guarantees a correct embedding of any adaptive mesh extracted at run time. Finally we show how the same refinement scheme allows one to achieve smooth output when combined with a 3D subdivision rule used to super-sample the volumetric data with cubic precision.

We focus our attention on the case where the multi-resolution representation of the volumetric data is based on the edge bisection refinement rule widely used in mesh generation [38, 39, 40]. For rectilinear volumetric input this is an adaptive oct-tree like sampling.

The remainder of this paper is organized as follows. Section 2 discusses previous related work. Section 3 introduces the time-critical refinement algorithm for two-dimensional meshes. Section 4 presents the 3D extension of the algorithm. Asymptotic analysis of the algorithm complexity is reported in Section 5. Section 6 presents the coupled subdivision scheme for smoothing the output surfaces. Section 7 discusses the practical use of the scheme while conclusions are drawn Section 8.

2 RELATED WORK

A great deal of research has been devoted in the past to the development of multi-resolution geometric data-structures. Several contributions provided from different fields like Computer Graphics, Computational Geometry and Mesh Generation, have lead to the development of high quality surface simplification schemes, recursive subdivision methods for surface smoothing, wavelet analysis and decomposition, hierarchical geometric data-structures for efficient spatial indexing and mesh refinement techniques for multilevel finite element methods.

Simplification. The main simplification primitives proposed in literature are vertex removal [13, 43, 10], edge contraction [24], and triangle contraction [18]. They have been applied successfully to the construction of high quality simplified objects using several different error metrics [17, 42, 6, 8, 22, 3, 24]. Maintaining a history DAG of the decimation process applied to maximally independent sets of primitives allows us to build the required multi-resolution representation. The selective traversal of this representation allows fast construction of adaptive levels of detail [13, 10, 11, 12]. Higher quality representations can be achieved constructing correspondences among the different levels of detail so that mesh properties can enhance the display of the coarsest representations [29, 9]. This also enables continuous deformation between homeomorphic objects [28].

Wavelet Analysis. One recent trend in multi-resolution surface generation is the design of methods based on wavelet functions [33, 45]. One main advantage of the multi-resolution analysis at the basis of the wavelet approach is that it immediately gives a compact hierarchical multi-resolution data-structure with guaranteed error bounds. The basic ingredient needed for wavelet analysis is the construction of nested function spaces which are best associated with the connectivity of subdivision surfaces. This restricts the class of meshes that can be processed, requiring eventual remeshing of the input. Hybrid approaches can be designed to take advantage from the quality of wavelet analysis keeping the generality of a simplification scheme [27].

The general framework of wavelet analysis is formalized independently of the intrinsic/embedding dimension of the geometric object. This allows for example to achieve multi-resolution representation and analysis for volumetric data [41, 44, 34].

Mesh Refinement. Similar solutions have been designed in the meshing community for the adaptive refinement of triangular meshes using a fixed set of templates [4]. A simpler and more flexible approach is the edge bisection strategy by Rivara [40]. A unique subdivision template is used to recursively subdivide the cells of a 2D mesh until a given adaptivity constraint is achieved. This implicitly yields a multi-resolution data-structure built starting from a quality coarse representation. The approach generalizes immediately to 3D tetrahedralizations [40, 38] and to higher dimension by performing the refinement process from the lower dimensional simplices of the mesh to the higher dimensional. This is the scheme that we are currently assuming for the input tetrahedralizations.

Subdivision Schemes. Using a recursive subdivision scheme one automatically achieves a hierarchical multi-resolution representation. This enable for example multi-resolution editing techniques [50]. The quality of the generated meshes is dependent from the subdivision mask used. For triangular domains Loop [31] provides an approximating subdivision scheme converging to a surface that is C^2 almost everywhere. The exception is at extraordinary vertices, that do not have exactly six incident edges, where the continuity decreases to C^1. The butterfly subdivision scheme [15] converges to an interpolating surface that is C^1 everywhere except for extraordinary points with exactly three or more than seven incident edges. A modified version [51] has been proposed that converges to a C^1 surface everywhere. Similarly for subdivision of quadrilateral domains one can use the Catmull-Clark scheme [5] or the interpolatory scheme by Kobbelt [26] to build smooth approximations of a coarse meshes. Subdivision schemes have been also used to build smooth vector fields [47]. In the following we will consider an extension of the scheme in [26, 14] for smoothing of scalar fields.

Efficient Isocontouring. The basic isocontour computation algorithm [32] is know to be inefficient since it waste time exploring empty regions of the underlying volumetric data. Geometric space indexing [49] is sufficient to achieve a substantial speedup. Span space indexing techniques can provide further improvement with nearly optimal [30] or even optimal speedup [2, 7] even with minimal storage overhead [46]. The insufficient results achieved even with such optimal techniques require to use more flexible approaches [20, 16] that allow us to use the multi-resolution representation of the volumetric data to extract an adaptive level of detail for the output. The lack of an actual multi-resolution representation for the output limits the practical use of such schemes especially for large datasets since a substantial amount of computation may be required when adaptation between different levels of resolution needs to be recomputed frequently.

General solution to the problem of rendering in a time-critical environment have also been explored [19, 25]. Such optimization techniques solve the "hard deadline" problem for high quality hierarchies if the time available is known in advance but do not consider the asynchronous termination problem or the case of dynamic change of the represented object.

3 2D SPATIAL HIERARCHY AND ISOCONTOUR REFINEMENT

This section considers the 2D version of our progressive refinement algorithm and data-structure. The design of the algorithm is dependent on the kind of hierarchical data-structure assumed for the input spatial hierarchy. We focus on the case of the edge bisection refinement that is widely used in the meshing community. This scheme can also be used to build a multi-resolution data-structure for regular grids without preprocessing since it is equivalent to a pre-determined order (octree-like) for reading the vertices.

Figure 2 shows the 2D edge-bisection refinement. The coarse level is triangular mesh. Each refinement step inserts a new vertex on an edge and splits the triangles adjacent along such edge into two halves. The refinement can be applied locally to perform adaptive refinement (Figure 2a) or globally to increase uniformly the resolution of the mesh (Figure 2b). Figure 2c shows a sequence of edge-bisection refinements for a regular grid. We use this scheme to perform progressive extraction of boundary curves from a picture as in Color Plate 1.

The 2D mesh partitions the region of space of interest in triangles. Each vertex is associated with an input function value. Inside each triangular cell a linear function is used to interpolate the function values at the vertices. In this way we have a piecewise linear representation of the scalar field $\mathcal{F}(\mathbf{x})$ necessary to compute an isocontour. As the edge-bisection algorithm makes progress new function values are introduced and a more detailed definition of the function $\mathcal{F}(\mathbf{x})$ is obtained. We show how this refinement procedure can be mapped directly into a refinement of the output isocontour. That is, instead of recomputing portions of the contour, we augment its representation generating directly a progressive data-structure. This progressive representation of the isocontour can be traversed adaptively independently from the underlying mesh.

3.1 Vertex Coloring

We color the vertices of the mesh to classify all possible 2D cases of isocontour deformation induced by the edge-bisection refinement. A vertex is *black* if its function value is greater than the current isovalue. A vertex is *white* if its function value is smaller than or equal to the current isovalue.

Figure 3 shows the bisection of the edge $\overline{CD}$ and the insertion the new vertex E. The two triangles $\widehat{ACD}$ and $\widehat{BCD}$ are divided into four triangles $\widehat{AED}$,$\widehat{AEC}$,$\widehat{BED}$ and $\widehat{BEC}$. Note that if $\overline{CD}$ is a boundary edge only one triangle (either $\widehat{ACD}$ or $\widehat{BCD}$) needs to be considered. The isocontour (in thick line) is updated consequently.

Since for isocontouring purposes symmetric configurations, where *white* vertices become *black* and vice versa, are equivalent

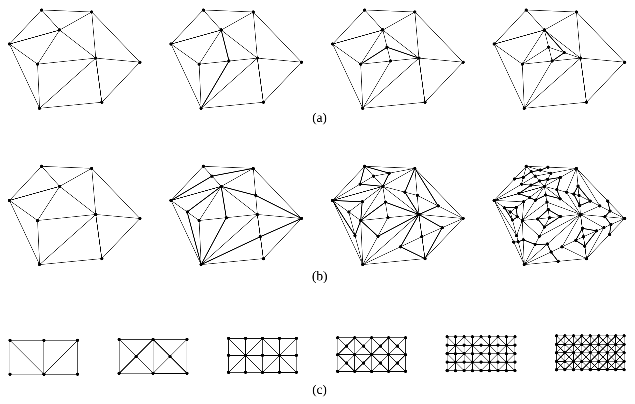


Figure 2: 2D Mesh refinement by edge-bisection. (a) Local adaptive refinement. (b) Global uniform refinement. (d) Sequence of edge-bisection refinements for a regular grid.

we consider only non-symmetric cases. Consequently we enumerate the configurations only in terms of "equal" colors or "different" colors. In this way we avoid enumerating twice equivalent cases. The basic observation driving the analysis below is that one edge of the 2D mesh intersects the isocontour iff its vertices have different colors.

Denote with the lower case letters a, b, c, d, e the colors of the vertices A, B, C, D, E respectively. Three main cases arise depending on the colors c, d and e. For each case two sub-cases may occur depending on the colors a and b.

$c \neq d$**.** The colors of the two extremes of $\overline{CD}$ are different (Figure 3a-c). Initially the isocontour does intersect the edge $\overline{CD}$. After the bisection the isocontour intersects either $\overline{CE}$ or $\overline{ED}$ (but not both).

$a = b$. If $a = b = e$ after the bisection only two triangles intersect the isocontour as in Figure 3a. Otherwise after the bisection all four triangles intersect the isocontour as in Figure 3b.

$a \neq b$. Two symmetric cases, both equivalent to Figure 3c, can occur.

$c = d = e$**.** The coloring of C, D, E is the same (Figure 3d-e). The isocontour does not intersect the edge $\overline{CD}$ neither before nor after the bisection.

$a = b$. If $a = b = c = d = e$ we have the trivial case where both the two initial triangles and the four final triangles do not intersect the isocontour. No action needs to be taken. If instead $a = b \neq c = d = e$ we have the case of Figure 3d (or symmetric).

$a \neq b$. Two cases equivalent to Figure 3e can occur.

$c = d \neq e$**.** The coloring of the vertices of the bisection edge is the same but different from the coloring of the new vertex E (Figure 3f-h). The isocontour initially does not intersect the edge $\overline{CD}$. After the bisection the isocontour intersects both $\overline{CE}$ and $\overline{DE}$.

$a = b$. This is the most interesting configuration where the topology of the contour is modified. In particular if $a = b = c = d$ the quadrilateral polygon $\widehat{ABCD}$ does not intersect the contour. Hence a new connected component is created inside (Figure 3g). If instead $a = b \neq c = d$ the connectivity of the contour flips from the configuration Figure 3h(top) to the configuration of Figure 3h(bottom).

$a \neq b$. Either a or b is equal to e so that the configuration of Figure 3f (or symmetric) is generated.

Note that in all cases the intersection between the boundary of the quadrilateral $\widehat{ABCD}$ and the isocontour does not change. Only the interior of $\widehat{ABCD}$ is affected by the refinement. Hence to maintain the topology and geometry of the isocontour we only need to update its data-structure corresponding to the eight deformations shown in Figure 3.

3.2 Local Isocontour Update

For contour refinement we use a set of six primitives that extends the basic vertex split used in progressive meshes [23]. The six primitives (illustrated in Figure 4) are: (i) VertexMove(A) the vertex A is moved to a new position, (ii) VertexSplit(A,B) a new edge $\overline{AB}$ is introduced at A, (iii) VertexOpen(A,B) similar to VertexSplit(A,B) but no edge connects A to B, (iv) EdgeFlip(A, B, C, D) the edges $\overline{AB}$ and $\overline{CD}$ are replaced by $\overline{AC}$ and $\overline{BD}$, (v) NewEdge(A, B) a new edge of vertices A and B is created, and (vi) NewLoop(A, B, C, D) a new closed polygon with four vertices A, B, C, D is created.

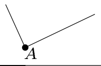
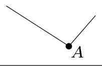
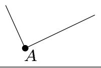
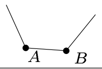
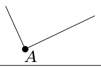
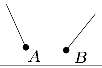
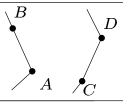
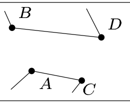
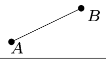
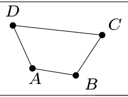
Primitive	Bifore	After
VertexMove(A)		
VertexSplit(A,B)		
VertexOpen(A,B)		
Edge Flip(A, B, C, D)		
NewEdge(A, B)		
NewLoop(A, B, C, D)		

Figure 4: Isocontour refinement primitives.

It is easy to see that the six refinement primitives above are sufficient to perform the eight updates shown in Figure 3. In particular each of the cases requires the following refinements:

case (a) One VertexMove().

case (b) One VertexMove() and two VertexSplit() operations. If $\overline{CD}$ is a boundary edge the update reduces to one VertexMove() and one VertexSplit().

case (c) One VertexMove() and one VertexSplit(). If $\overline{CD}$ is a boundary edge $\widehat{BCD}$ is the same as in case (a) while $\widehat{ACD}$ is the same as (b).

case (d) Two VertexSplit() operations. If $\overline{CD}$ is a boundary edge one VertexSplit() only.

case (e) One VertexSplit().

case (f) One VertexMove() and two VertexSplit() operations. If $\overline{CD}$ is a boundary edge $\widehat{ACD}$ would require one VertexMove() and one VertexOpen(). $\widehat{BCD}$ would instead require one NewEdge() and one VertexSplit().

case (g) One NewLoop(). If $\overline{CD}$ is a boundary edge one NewEdge() and one VertexSplit().

case (h) One EdgeFlip() and two VertexSplit() operations. If $\overline{CD}$ is a boundary edge one VertexOpen() and one VertexSplit().

The application of the rules above allows one to build progressively a multi-resolution representation of the isocontour. Note that, by construction, independent refinements occur in non-overlapping regions of space so that any adaptive level of detail extracted from the multi-resolution data-structure is guaranteed to be without self-intersections even if adaptive traversal is performed on the isocontour hierarchy instead of the mesh hierarchy.

Moreover, we execute each refinement step in Figure 4 within a critical section as an atomic transaction. We therefore maintain a consistent data-structure that can be safely accessed at any time within a small constant delay. In this way we solve the "hard deadline" problem for time critical computations even if the deadline is not known in advance or if the isosurface is changed dynamically. The quality may be degraded depending on the frequency of the modifications but the response time can be guaranteed. At the same time the classical quality-driven traversal can also be performed.

In the following section we show how this scheme can be implemented for the computation of isosurfaces of scalar fields defined on 3D volumetric meshes.

4 3D SPATIAL HIERARCHY AND ISOSURFACE REFINEMENT

The edge bisection refinement apply to 3D tetrahedral meshes in the same way it applies to 2D triangular meshes. Color Plate 2 shows the corresponding progressive refinement of an isosurface. For simplicity of analysis we first consider the update of the isosurface within a single tetrahedron. Then we show how to compose such deformation to update the isosurface within the set $\mathcal{C}$ of all tetrahedra around the bisection edge.

4.1 Vertex Coloring

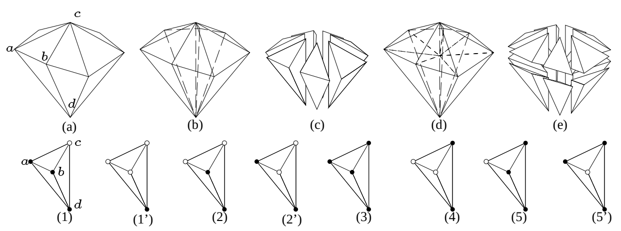


Figure 5: (a-c) A set of seven tetrahedra with a common edge. (d-e) The fourteen tetrahedra obtained by bisection of the common edge. The bottom raw shows the eight different coloring configurations with respect to the bisection edge $\overline{cd}$ (modulo black and white complementarity). Configurations (1),(2) and (5) are also equivalent (by symmetry) to (1'),(2') and (5') respectively.

Similarly to the 2D case one can classify the vertices of the tetrahedra around the split edge depending of the value of the function values compared with the current isovalue. The main difference from the 2D cases that the number of tetrahedra incident to the edge

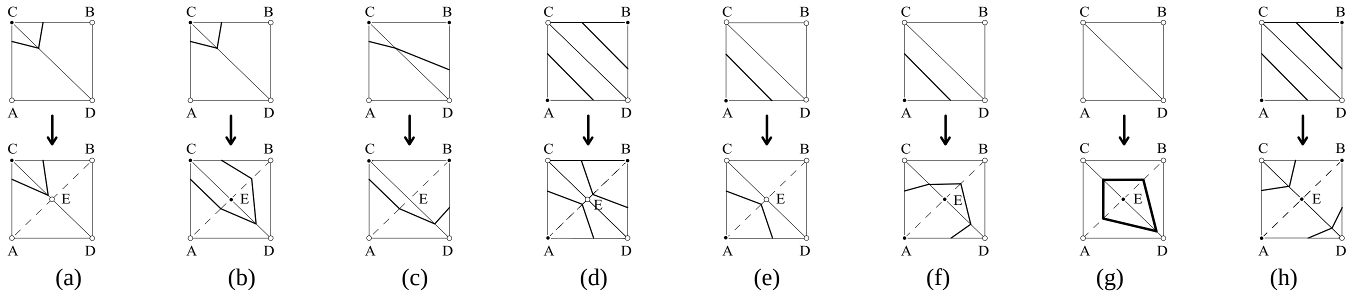


Figure 3: List of all possible type of updates of isocontour configurations for 2D edge bisection. The 2D mesh edges are drawn in thin lines. The isocontour is drawn in thick line. The new edges created by the edge bisection are drawn in dashed lines. The top row shows the initial configurations before the bisection. The bottom row shows the final configurations after bisection. The function values at the initial vertices A, B, C, D and at the new vertex E are marked black if greater than the current isovalue, white if smaller or equal).

bisected cannot be bounded (it is a local set but not necessarily a small set). Consequently the analysis cannot be done exhaustively on the entire set of tetrahedra. One needs instead to classify all the possibilities for a single tetrahedron and show how they can be combined in a unique result. Figure 5(a-c) shows three different views of seven tetrahedra all incident to a common edge $\overline{cd}$. Figure 5(d-e) shows two views of the fourteen tetrahedra resulting from the bisection of $\overline{cd}$. The bottom row of Figure 5 shows the eight different coloring for a single tetrahedron where vertex d (at the bottom) is fixed to be *black* and the edge being bisected is $\overline{cd}$. Note that the edges and facets shared among several tetrahedra must have the same color. For example cases (1),(2),(1') and (2') can all occur around the same edge but none of them can appear together with one of (3),(4),(5) or (4') because the bisected edge would have different coloring. Moreover adjacent tetrahedra must have the same coloring at the shared facet so that if cases (1) and (1') appear in the same configuration they cannot be adjacent.

Similarly to the 2D case the analysis of the isosurface refinement is divided into three main cases:

$\hat{c} \neq \hat{d}$ **(Figure 7).** The bisected edge $\overline{cd}$ is intersected by the isosurface.

$\hat{c} = \hat{d} = \hat{e}$ **(Figure 8).** The bisected edge $\overline{cd}$ is not intersected by the isosurface and neither are the two halves $\overline{ce}$ and $\overline{ed}$.

$\hat{c} = \hat{d} \neq \hat{e}$ **(Figure 9).** The bisected edge $\overline{cd}$ is not intersected by the isosurface but both $\overline{ce}$ and $\overline{ed}$ are intersected.

4.2 Refinement Primitives

Eight refinement primitives (see Figure 6) are used for the local update of the isosurface currently computed.

1. **VertexMove(**a**).** Vertex a is moved to a new position.

2. **VertexSplit(**$a, b, c; d$**).** The vertex b is split into two vertices b and d connected by an edge. The two edges $\overline{ab}$ and $\overline{bc}$ are split into the triangles $\overline{abd}$ and $\overline{bcd}$.

3. **VertexOpen(**$a, b, c; d; a', c'$**).** The same as VertexSplit if the two booleans a', c' are both $True$. If $a' = False$ the triangle $\overline{abd}$ is not created. If $c' = False$ the triangle $\overline{bcd}$ is not created.

4. **EdgeFlip(**a, b, c, d**).** The triangles $\overline{acd}$ and $\overline{bcd}$ are replaced by the triangles $\overline{abc}$ and $\overline{abd}$.

5. **NewLoop(**$a, b, c_1, \ldots, c_k$**).** $2k$ triangles are added. The first k triangles are $\overline{ac_1c_2}, \overline{ac_2c_3}, \ldots, \overline{ac_{k-1}c_k}, \overline{ac_kc_1}$. The remaining k triangles are $\overline{bc_1c_2}, \overline{bc_2c_3}, \ldots, \overline{bc_{k-1}c_k}, \overline{bc_kc_1}$. Overall the $2k$ triangles form a closed surface with the topology of a sphere.

6. **NewSurf(**$a, b, c_1, \ldots, c_k$**).** $(2k-2)$ triangles are added. The first $k-1$ triangles are $\overline{ac_1c_2}, \overline{ac_2c_3}, \ldots, \overline{ac_{k-1}c_k}$. The remaining k triangles are $\overline{bc_1c_2}, \overline{bc_2c_3}, \ldots, \overline{bc_{k-1}c_k}$. Overall the $2(k-1)$ triangles form a simple open surface.

7. **CylinderFlip(**$a_1, \ldots, a_k, b_1, \ldots, b_k; c, d$**).** The initial cylinder is formed by the $2k$ triangles $\overline{a_1, a_2, b_1}, \overline{b_1, b_2, a_2}, \ldots, \overline{a_{k-1}, a_k, b_{k-1}}, \overline{b_{k-1}, b_k, a_k}$, $\overline{a_k, a_1, b_k}$, $\overline{b_k, b_1, a_1}$. They are replaced by two groups of k triangles. The first k triangles are $\overline{a_1, a_2, c}, \overline{a_2, a_3, c}, \ldots, \overline{a_{k-1}, a_k, c}, \overline{a_k, a_1, c}$ (top of the cylinder). The second k triangles are $\overline{b_1, b_2, d}, \overline{b_2, b_3, d}, \ldots, \overline{b_{k-1}, b_k, d}, \overline{b_k, b_1, d}$ (bottom of the cylinder).

8. **ConnectivityChange(**a, b, c, d, e, f**).** This primitive changes connectivity of the surface without changing its geometry (that is the set of points that belong to the surface). This because the change converts the surface between two possible manifold representations of the same non-manifold geometry. The surface is made of four triangles $\overline{acd}, \overline{bcd}, \overline{ecd}$ and $\overline{fcd}$. Initially $\overline{acd}$ is adjacent to $\overline{bcd}$ while $\overline{ecd}$ is adjacent to $\overline{fcd}$. The primitive modifies such connectivity information to make $\overline{acd}$ adjacent to $\overline{fcd}$ and $\overline{bcd}$ adjacent to $\overline{ecd}$.

4.3 Simple intersection with the edge $\overline{cd}$.

Consider the case where $\hat{c} \neq \hat{d}$, the isosurface has a single intersection with $\overline{cd}$. With reference to Figure 7, the tetrahedra around $\overline{cd}$ are either of type (1a) or of type (2a). The tetrahedron of type (1a) can be adjacent to another tetrahedron of type (1a) or to facet $\overline{bcd}$ of a tetrahedron of type (2a). Similarly (2a) can be adjacent to its symmetric with respect to $\overline{cd}$ where the colors $\hat{a}$ and $\hat{b}$ are exchanged. Moreover at $\overline{bcd}$ it can adjacent to (1a) while at $\overline{acd}$ it can be adjacent to the upside down symmetric of (1a) where $\hat{a} = \hat{b} = \hat{c} \neq \hat{d}$. The update rules for each case are as follows.

(1a)→(1b). One VertexMove is needed to adjust the position of the isosurface vertex along $\overline{cd}$. Two VertexSplit operation are necessary to create the two new vertices on $\overline{ae}$ and $\overline{be}$ with the two additional triangles on the isosurface inside the tetrahedron.

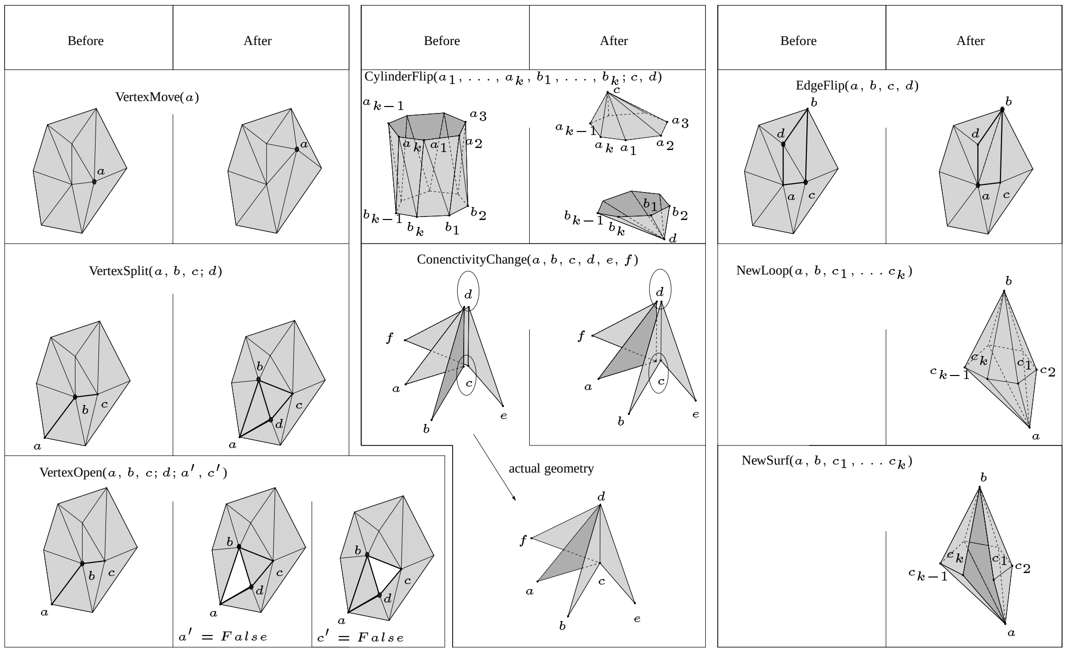

Before
After
VertexMove(a)
VertexSplit($a, b, c; d$)
VertexOpen($a, b, c; d; a', c'$)
$a' = False$
$c' = False$
CylinderFlip($a_1, \ldots, a_k, b_1, \ldots, b_k; c, d$)
ConenctivityChange(a, b, c, d, e, f)
actual geometry
EdgeFlip(a, b, c, d)
NewLoop($a, b, c_1, \ldots c_k$)
NewSurf($a, b, c_1, \ldots c_k$)

Figure 6: Isosurface refinement primitives.

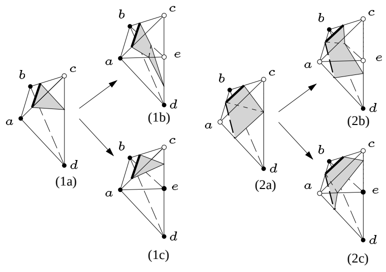


Figure 7: (1a) and (2a) are the type of tetrahedra around $\overline{cd}$ for an isosurface with a single intersection with the bisection edge. The portion of isosurface inside the tetrahedra is shown as a gray polygon. (1b-c) and (2b-c) are the possible configurations after the bisection operation. The bold lines mark the boundary of the region where the isosurface is not modified.

(1a)→(1c). Only one VertexMove is needed to adjust the position of the isosurface vertex along $\overline{cd}$.

(2a)→(2b). One VertexMove is needed to adjust the position of the isosurface vertex along $\overline{cd}$. One VertexSplit operation is necessary to create the new vertex on $\overline{be}$ and the additional triangle on the isosurface inside the tetrahedron.

(2a)→(2c). One VertexMove is needed to adjust the position of the isosurface vertex along $\overline{cd}$. One VertexSplit operation is necessary to create the new vertex on $\overline{ae}$ and the additional triangle on the isosurface inside the tetrahedron.

Note that one should perform the VertexMove along $\overline{cd}$ only once for all k tetrahedra around $\overline{cd}$. Similarly each VertexSplit refinement is done once for each facet $\overline{acd}$ or $\overline{bcd}$ on which a new vertex is created.

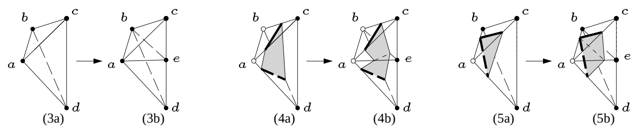


Figure 8: (3a),(4a) and (5a) are the type of tetrahedra around $\overline{cd}$ for an isosurface with no intersection with the bisection edge. The portion of isosurface inside the tetrahedra is shows as a gray polygon. (3b),(4b) and (5b) are the possible configurations after the bisection operation for $\hat{e} = \hat{c}$. The bold lines mark the boundary of the region where the isosurface is not modified.

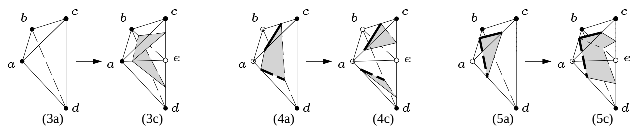


Figure 9: (3a),(4a) and (5a) are the type of tetrahedra around $\overline{cd}$ for an isosurface with no intersection with the bisection edge. The portion of isosurface inside the tetrahedra is shows as a gray polygon. (3c),(4c) and (5c) are the possible configurations after the bisection operation for $\hat{e} \neq \hat{c}$. The bold lines mark the boundary of the region where the isosurface is not modified.

4.4 No intersection with the edge $\overline{cd}$.

Consider the case of Figure 8 where $\hat{c} = \hat{d} = \hat{e}$ and the isosurface has no intersection with $\overline{cd}$ before the bisection nor with $\overline{ce}$, $\overline{ed}$ after the bisection. The tetrahedra around $\overline{cd}$ are of type (3a),(4a) or (5a). The tetrahedron of type (3a) can be adjacent to another tetrahedron of type (3a) or to facet $\overline{bcd}$ of tetrahedron (5a). The tetrahedron (4a) can be adjacent to another tetrahedron of type (4a) or to facet $\overline{acd}$ of tetrahedron (5a). Tetrahedron (5a) can be adjacent to its symmetric with respect to $\overline{cd}$ where the colors $\hat{a}$ and $\hat{b}$ are exchanged. On facet $\overline{bcd}$ tetrahedron (5a) can be adjacent to (3a) while on $\overline{acd}$ it can be adjacent to(4a). There are three refinement cases to consider:

(3a)→(3b). There is no portion of isosurface so nothing needs to be done.

(4a)→(4b). Two VertexSplit operation are necessary to create the two new vertices on $\overline{ae}$ and $\overline{be}$ on the isosurface inside the tetrahedron.

(5a)→(5b). One VertexSplit operation is necessary to create the new vertex on $\overline{ae}$ and one additional triangle on the isosurface inside the tetrahedron.

Note that if out of the k facets incident to $\overline{cd}$ only h intersect the isosurface then h VertexSplit refinement operations are sufficient in total to perform the refinement induced by the edge bisection of the 3D mesh.

4.5 Intersection with the edges $\overline{ce}$ and $\overline{ed}$ but not of $\overline{cd}$.

Consider the case where $\hat{c} = \hat{d} \neq \hat{e}$. The isosurface has no intersection with $\overline{cd}$ before the bisection but intersects both $\overline{ce}$ and $\overline{ed}$ after the bisection. The adjacency conditions among the tetrahedra are the same as the previous case. Differently from the previous two cases, it is not possible to simply derive the transformations on each tetrahedron separately. The three main cases to be considered are:

(3a)→(3c). Initially there is no portion of isosurface in the tetrahedron. The bisection induces the creation of two triangles inside the tetrahedron.

(4a)→(4c). There are two triangles before and after the bisection. Their connectivity need to be changed depending on the neighboring tetrahedra.

(5a)→(5c). One new vertex is added along the edge $\overline{cd}$ and three additional triangles must be created to refine the isosurface.

The above refinements can be combined in multiple ways which are dependent on the facet adjacency constraints due to compatible vertex coloring. Among the different tetrahedra sequences around $\overline{cd}$ the following four global configurations need to be considered.

Empty Region. All tetrahedra are of type (3a). Initially there is no surface. A NewLoop primitive creates all the $2k$ triangles (two per tetrahedron) of the new isosurface component. If there are boundary facets then a NewSurf is used instead.

Cylindrical Surface. All the tetrahedra are of type (4a). Initially the isosurface is roughly like a cylinder of axis $\overline{cd}$. A CylinderFlip primitive flips the connectivity to be the top and bottom of the cylinder.

Single Component. This is the case where the tetrahedra are concatenated as follows: one tetrahedron of type (5a), h tetrahedra of type (4a), one tetrahedron of type (5'a) (the axial symmetric of (5a)) followed by l tetrahedra of type (3a). In short this is written $\{$(5a)-(4a)^h-(5'a)-(3a)$^l\}$. Naturally for a loop of k tetrahedra, with $k = l + h + 2$. Note also that the loop may be interrupted at any point by two boundary facets.

Multiple Components. The loop of k simplicies is divided in m sequences of **Single Component** type. This can be written as $\{$(5a)-(4a)^{h_i}-(5'a)-(3a)$^{l_i}\}^m$, where $\sum_{j=0}^{m}(h_i + l_i + 2) = k$. Again such sequence may be interrupted by boundary facets.

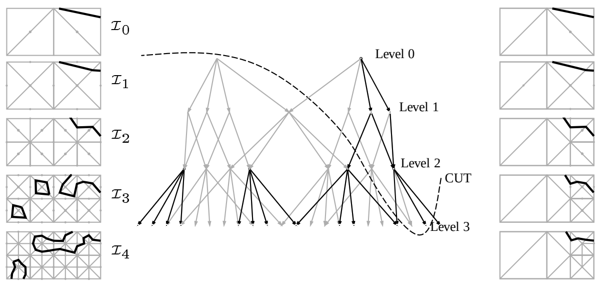


Figure 10: (a) Sequence of levels of resolution of an isocontour $\mathcal{I}$ (bold black line) of a 2D scalar fieled (light gray liens). (b) The hierarchical representation of $\mathcal{I}$ (black arrows) is a subgraph of the mesh hierarchy (gray arrows). (c) An adaptive refinement of $\mathcal{I}$.

The resolution of the **Single Component** $\{$(5a)-(4a)^h-(5'a)-(3a)$^l\}$ case requires an appropriate sequence of VertexSplit and EdgeFlip primitives which number depends on the specific values of h and l. This can be extended to the **Multiple Components** case with the use of one ConnectivityChange primitive for each eddiational connected component. More details regarding this two complicated refinements can be found in [37].

5 ANALYSIS

In this section we analyze the characteristics of the algorithm and its asymptotic behavior for input size n and output size k. The fact that any adaptive resolution isocontour has correct embedding derives from the fact that the output hierarchy is a subgraph of the input hierarchy. In fact any node in the output hierarchy is built by mapping one refinement node in the input into some refinements (merged in a single node) in the output hierarchy. Figure 10 shows the case of a 2D regular grid which hierarchy (in gray) is a graph that contains as subgraph the hierarchy of any contour (in black). Each node in the input hierarchy represents the (one or two) triangles incident on the split edge. As a consequence any adaptive traversal of the output graph (nodes above some given cut) corresponds both to an adaptive mesh in the input and in the output. The adaptive traversal of the input does not need to be computed[1] but the fact that it exists proves that the adaptive output is the isocontour of some actual mesh. Hence the contour itself does not have any self intersection.

We show an optimal behavior for the computation of large isocontours while the overhead for the computation of smaller isocontours is kept within a reasonable logarithmic factor.

Consider an isocontour of output size k (k line segments in 2D or k triangles in 3D). We assume that the input mesh is organized in a binary tree hierarchy with the coarse level having size $O(1)$ and lowest level (finest resolution) of size $\Theta(n)$. Note that under this condition no isocontour can have size larger than $O(n)$ because each cell in the input mesh intersects any isocontour in at most a constant number of simplices (line segments in 2D or triangles in 3D). In particular in our asymptotic analysis we call *large* an isocontour of output size exactly $\Theta(n)$ and *small* an isocontour that is $o(n^h)$ for any constant $h > 0$.

Theorem 1 *Given an isocontour of output size $k = \Theta(n^h)$, for some constant $h > 0$, the corresponding hierarchy generated by the progressive isocontouring algorithm has size $\Theta(k)$ for $h = 1$ and has size $O(k \log k)$ for $h < 1$.*

[1]Note that the adaptive traversal of the isocontour may be performed at a client side where the 3D mesh is too large to be available locally.

Proof The input hierarchy is a complete binary tree where the finest resolution has size $\Theta(n)$. Hence the height of the input hierarchy is $\log n + O(1)$. Since the progressive isocontouring algorithm produces one level in the output hierarchy for each level in the input hierarchy the height of the output hierarchy is also $\log n + O(1) = \frac{1}{h} \log k + O(1)$. For $h = 1$ this implies that the overall output hierarchy has size $\Theta(k)$. For $h < 1$ the overall output hierarchy has size $O(k \log k)$. $\diamond$

In other words for non-small contours the output is a balanced tree which is optimal for large contours.

Theorem 2 *Given an isocontour of size $k = \Theta(n^h)$, for some constant $h > 0$, the time necessary to compute the isocontour is $\Theta(k)$ for $h = 1$ and $O(k \log k)$ for $h < 1$.*

Proof Follows immediately from the previous theorem and from the fact that the progressive isocontouring algorithm generates an output tree of l nodes in l steps of constant time each. $\diamond$

For large contours the computation time is linear in the size of the finest output resolution and hence optimal. Otherwise a logarithmic penalty factor is introduced by the traversal of the input hierarchy.

Theorem 3 *Given an isocontour of output size k the corresponding hierarchy size and computation time is $O(k \log n)$.*

Proof From the same observations of theorem 1 it derives that the height of the output tree is $\log n + O(1)$. Hence the hierarchy's overall size is $O(k \log n)$. As shown in Theorem2 the computation time is just proportional to the overall size of the output. $\diamond$

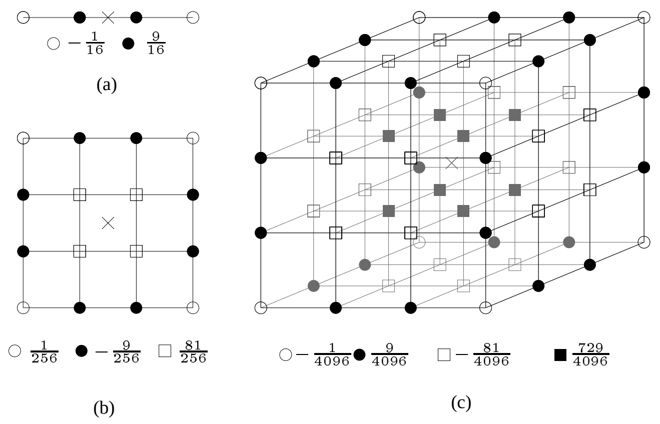

$\bigcirc\ -\frac{1}{16}$ $\bullet\ \frac{9}{16}$
(a)
$\bigcirc\ \frac{1}{256}$ $\bullet\ -\frac{9}{256}$ $\square\ \frac{81}{256}$
(b)
$\bigcirc\ -\frac{1}{4096}$ $\bullet\ \frac{9}{4096}$ $\square\ -\frac{81}{4096}$ $\blacksquare\ \frac{729}{4096}$
(c)

Figure 11: Coefficients of the 3D subdivision rule. (a) Computation for the central point of an edge. (b) Computation fro the central point of a face. (c) Computation for the central point of a cube. The cross marks the position of the new vertex computed as linear combination of the neighbors. The circles and squares (empty or full) symbols classify the vertices depending on their coefficient in the linear combination. The actual coefficient is reported at the bottom of each configuration near the relative symbol.

6 SMOOTHING

In this section we consider the problem of generating smooth approximations of isosurfaces from image data. In particular we use the same algorithm and data-structure presented in the previous section, applied now to a derived scalar field.

The key feature of our progressive algorithm is the ability to add more detail to the surface representation as new function values are added to the input field. This gives us the opportunity to solve the smoothing problem by adding appropriate function values to the scalar field to smooth down the field and hence the isosurface. To achieve this goal we use an interpolatory subdivision scheme, that generalized to the 3D case the 1D approach of [14] already extended to the 2D case in [26]. The subdivision mask of choice is the order three tensor $M_{i,j,k} = m_i * m_j * m_k$ with

$$m = \left[\frac{-\alpha}{16}, \frac{8+\alpha}{16}, \frac{8+\alpha}{16}, \frac{-\alpha}{16}\right].$$

The subdivision scheme is guaranteed to converge to a smooth limit function for $0 < \alpha < 2(\sqrt{5} - 1)$ [14]. The choice of $\alpha = 1$ yields a B-spline interpolant of cubic precision. Figure 11 shows the topology of the subdivision mask where the vertices are marked with different symbols if the have different averaging coefficients.

Since the gradient of the field is normal to the isosurface, the smoothness of the field in general yields smooth isocontours. Important exceptions to this rule are the criticalities of the field where the gradient vanishes and hence sharp features may be obtained. Figure 12 shows a smooth drop with a sharp tip obtained at a saddle point of a scalar field.

Since we can interleave refinement and smoothing steps within a unified framework we can combine these operations in several imaginative spatio-temporal ways. For example Figure 12b shows a surface extracted from the HIPIP dataset with the right half refined (with low error) and the left half smoothed directly from a coarse (high error) approximation. In practice we combine smoothing with refinement to improve the currently best approximation of the input model (see Color Plate 3).

Note also that for surfaces with continuous changes in curvature, like the brain model in Color Plate 4, even the smoothing process applied directly to the surface could lead to self-intersecting surfaces. With our approach we avoid this problem since we smooth the surface using the progressive algorithm that always guarantees no self-intersections.

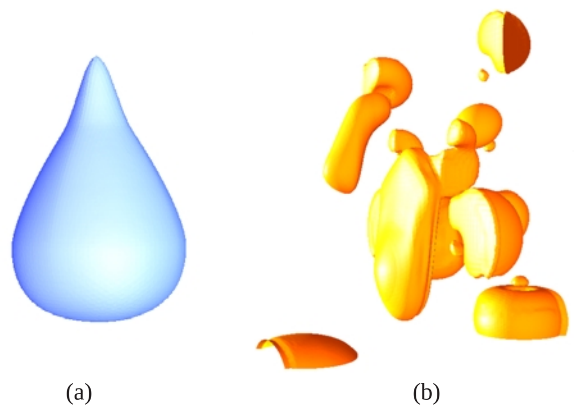


Figure 12: (a) A smooth drop with a sharp tip obtained at a singular point of the embedding function. (b) The model of Figure 1c with the left half smoothed from a coarse level and right half refined and smoothed.

7 DISCUSSION

The approach presented in this paper enables a new type of tradeoff between speed and accuracy. In our prototype implementation we uncoupled the isosurface construction from its display. The isocontour hierarchy is built by one process that traverses the input 3D mesh. A second process that never access the 3D mesh performs the isocontour traversal and display. The computation time is slower than an optimal scheme like [2]. Experimentally it was found to be roughly 1.5 times slower if the same finest resolution needs to be achieved. The optimal scheme is clearly preferable if the finest resolution isocontour is known to be small enough to be rendered in real time. If this not the case our new scheme allows us to render at any given time partial results while the computation of complete hierarchy makes progress.

The hierarchical surface data-structure generated by our algorithm is comparable to the one generated by a decimation scheme. In this case we compromise the quality of the hierarchy whose coarse levels are not as good as those generated by a decimation scheme, but still good enough to provide the user with an initial understanding of the shape that will be obtained. The problem with the decimation schemes is that even those that are most efficient like the quadratic error norm [17] are inherently off-line processes since they require one to build the fine resolution data first and then to start the decimation process. This off-line process does not satisfies the requirement typical of the interactive isocontouring query where the isovalue is changed continuously even before the finest resolution is ever computed. Moreover the coarse levels generated by a decimation scheme may have self intersecting portions since the coarse representations of the surfaces are not isosurfaces of a decimated 3D mesh.

Previous adaptive schemes overcome both these problems by performing an adaptive traversal of the input 3D mesh. This avoids the delay due to the construction of the fine resolution and guarantees no self intersections at any level of resolution since any approximation is the isosurface of some scalar field. This class of solutions provide no hierarchy for the output isocontour making it difficult to uncouple the adaptive traversal of the isosurface from its construction. Moreover if the high resolution information is required the lack of an output hierarchy makes the scheme incur in the same delays as a single resolution approach.

The key novelty of the present scheme is that providing a set of local rules for continuous geometric transitions (geomorphs) of one level of resolution into the next we bridge the gap between adaptive techniques and multi-resolution decimation-based techniques. Different adaptive traversal strategies can be applied concurrently at the isosurface hierarchy construction side and at the isosurface hierarchy traversal side. The discussion of the different adaptivity strategies is beyond the scope of the present paper. Different refinement strategies can be integrated [48, 1, 35, 21, 36]. The focus here is on the construction of an appropriate data-structure that allows on-line construction of an hierarchical multi-resolution representation of the output isosurfaces.

8 CONCLUSIONS

We have introduced a progressive algorithm and data-structure for time-critical computation of isocontours. The approach also provide one solution for the problem of maintaining non-self-intersecting surfaces while extracting adaptive levels of details from the multi-resolution surface representation. We coupled the scheme with a subdivision scheme that yields cubic precision B-splines for the embedding scalar field to obtain correspondingly smooth surfaces.

The approach presented shows the viability of the idea of developing fully progressive algorithms to exploit at their best the available multi-resolution data-structures and algorithms.

Research supported in part by NSF grants CCR-9732306, DMS-9873326, ACI-9982297 and Sandia/LLNL DOE ASCI-BD4485

References

[1] F. Allamandri, P. Cignoni, C. Montani, and R. Scopigno. Adaptively adjusting marching cubes output to fit a trilinear reconstruction filter. In *Visualization in Scientific Computing '98*. Eurographics, 1998.

[2] C. L. Bajaj, V. Pascucci, and D. R. Schikore. Fast isocontouring for improved interactivity. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 39–46, October 1996.

[3] C. L. Bajaj and D. R. Schikore. Topology preserving data simplification with error bounds. *Computers and Graphics*, 22(1):3–12, 1998.

[4] R. E. Bank, A. H. Sherman, and A. Weiser. *Scientific Computing (Applications of Mathematics and Computing to the Physical Science)*, chapter Refinement Algorithms and data structures for regular local mesh refinement, pages 3–17. North Holland, 1983.

[5] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–355, September 1978.

[6] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Comp.*, 13(5):228–246, 1997.

[7] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Optimal isosurface extraction from irregular volume data. In *Proceedings of 1996 Symposium on Volume Visualization*, pages 31–38, 1996.

[8] J. Cohen, D. Manocha, and M. Olano. Simplifying polygonal models using successive mappings. In *IEEE Visualization '97*, pages 395–402, 1997.

[9] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *SIGGRAPH 98 Proc.*

[10] M. De Berg and K. T. G. Dobrindt. On levels of detail in terrains. *Graphical models and image processing: GMIP*, 60(1):1–12, 1998.

[11] L. De Floriani, P. Magillo, and E. Puppo. Visualizing parametric surfaces at variable resolution. *Lecture Notes in Computer Science*, 1311:308–321, 1997.

[12] Leila De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 9(2):67–78, March 1989.

[13] D. Dobkin and D. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, 6:381–392, 1985.

[14] N. Dyn and D. Levin. Interpolating subdivision schemes for the generation of curves and surfaces. In *Multivar. Approx. and Interp.*, pages 91–106, 1990.

[15] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. on Graphics*, 9(2):160–169, 1990.

[16] K. Engel, R. Westermann, and T. Ertl. Isosurface extraction techniques for web-based volume visualization. In *IEEE Visualization 1999*, pages 139–146, 1999.

[17] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH 97 Proc.*, pages 209–216, 1997.

[18] T. S. Gieng, B. Hamann, K. I. Joy, G. L. Schussman, and I. J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, April 1998.

[19] E. Gobetti and E. Bouvier. Time-critical multiresolution scene rendering. In *IEEE Visualization 1999*, pages 123–130. IEEE, 1999.

[20] R. Grosso and T. Ertl. Progressive iso-surface extraction from hierarchical 3d meshes. In *Proceedings EUROGRAPHICS '98*, 1998.

[21] R. Grosso, C. Lürig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *IEEE Visualization ´97*, pages 387–394, 1997.

[22] A. Gueziec. Surface simplification inside a tolerance volume. Technical report, Yorktown Heights, 1996. IBM Research Report RC 20440.

[23] H. Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Proc.*, pages 99–108, 1996.

[24] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH 97 Proc.*, pages 189–198, 1997.

[25] J. T. Klosowski and C. Silva. Rendering on a budget: A framework for time-critical rendering. In *IEEE Visualization 1999*, pages 115–122. IEEE, 1999.

[26] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Computer Graphics Forum*, 15(3):C409–C420, C485, September 1996.

[27] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Proc.*, pages 105–114, 1998.

[28] A. Lee, D. Dobkin, W. Sweldens, and P. Schröder. Multiresolution mesh morphing. *Proceedings of SIGGRAPH 99*, pages 343–350, 1999.

[29] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parametrization of surfaces. In *SIGGRAPH'98 Proc.*, pages 95–104, 1998.

[30] Y. Livnat, H. W. Shen, and C. R. Johnson. A near optimal isosurface extraction algorithm for unstructured grids. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[31] C. Loop. Smooth spline surfaces over irregular meshes. In *SIGGRAPH '94 Proc.*, pages 303–310, 1994.

[32] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH '87 Proc.*, pages 163–169, 1987.

[33] M. Lounsbery, T. D. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[34] Shigeru Muraki. Multiscale volume representation by a doG wavelet. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):109–116, June 1995.

[35] P. J. Neugebauer and K. Klein. Adaptive triangulation of objects reconstructed from multiple range images. In *Vis. '97, Late Breaking Hot Topics*, pages 20–24, 1997.

[36] M. Ohlberger and M. Rumpf. Adaptive projection operators in multiresolution scientific visualization. *IEEE Trans. on Vis. and Comp. Graph.*, 4(4), 1998.

[37] V. Pascucci. *Multi-resolution and Multi-dimensional Geometric Data-structures for Scientific Visualization*. PhD thesis, Purdue University, Lafayette, IN, 2000.

[38] A. Plaza and G. F. Carey. About local refinement of tetrahedral grids based on local bisection. In *5th International Meshing Roundtable*, pages 123–136, 1996.

[39] M. C. Rivara. Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *J. Numer. Meth. Engrg.*, 20:745–756, 1984.

[40] M.-C. Rivara and C. Levin. A 3-d refinement algorithm suitable for adaptive and multi-grid techniques. *Comm. in Appl. Numer. Meth.*, 8:281–290, 1992.

[41] R. Sánchez and M. Carvajal. Wavelet based adaptive interpolation for volume rendering. In *IEEE Symposium on Volume Visualization*, pages 127–134, 1998.

[42] W. J. Schroeder. A topology modifying progressive decimation algorithm. In *IEEE Vis. 97*, pages 205–212.

[43] W. J. Schroeder, J. A. Zarge, and William E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, July 1992.

[44] O. G. Staadt, M. Gross, and R. Weber. Multiresolution compression and reconstruction. In *Proceedings of IEEE Visualization 1997*. IEEE, 1997.

[45] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgann Kaufmann, San Francisco, CA, 1996.

[46] M. van Kreveld, R. van Oostrum, C. L. Bajaj, V. Pascucci, and D. R. Schikore. Contour trees and small seed sets for isosurface traversal. In *13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.

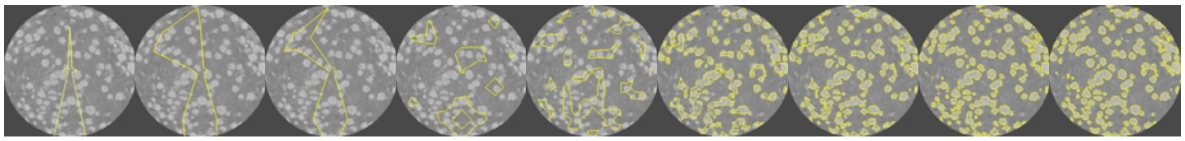
[47] H. Weimer and J. Warren. Subdivision schemes for fluid flow. *Proc. of SIGGRAPH 99*, pages 111–120.

[48] R. Westermann, L. Kobbelt, and T. Ertl. Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. *The Visual Computer*, 15(2):100–111, Apr 1999.

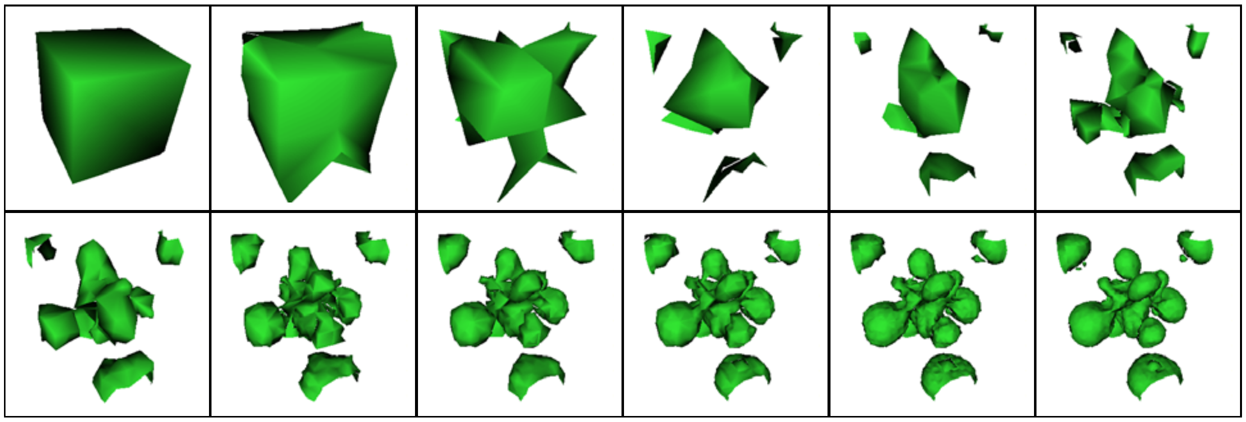
[49] Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, 1992.

[50] D. Zorin, P. Schroder, and W. Sweldens. Interactive multiresolution mesh editing. *Computer Graphics*, 31(3A):259–268, August 1997.

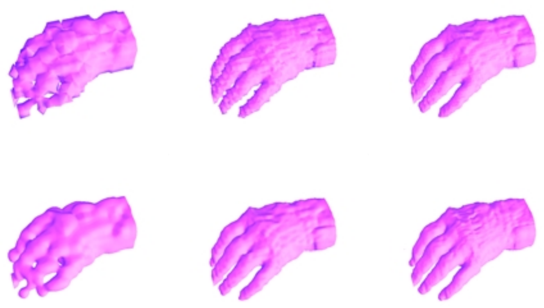
[51] D. Zorin, P. Schroeder, and W. Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *SIGGRAPH 96 Proc.*, pages 189–192, 1996.



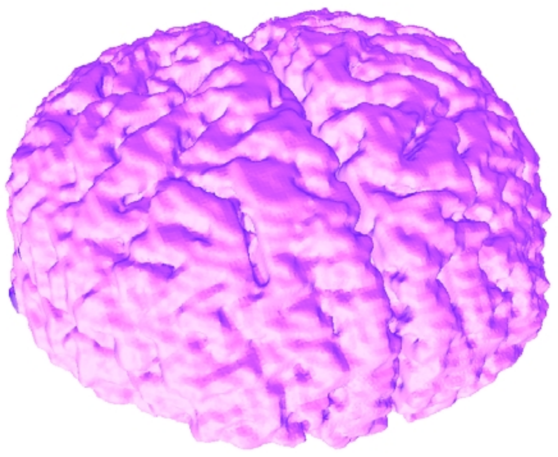
Color Plate 1: Progressive extraction of isocontours from a 2D image. The isocontours are drawn as yellow lines.



Color Plate 2: Steps in a progressive isosurface computation from the volumetric HIPIP dataset, left to right and top to bottom.



Color Plate 3: Refinement sequence of three approximations obtained during the reconstruction the hand model from a CT scan. (top row) Original sequence of approximations. (bottom row) Sequence of smooth approximations.



Color Plate 4: Successively refined and smoothed brain model reconstructed from an MRI scan with small error approximation.