Mesh Data Structures and CSG
Representing Surface Data

point cloud

triangle soup

manifold mesh

watertight mesh

more structure
Point Clouds
Point Clouds

List of points

- may or may not include normals
Point Clouds

List of points

• may or may not include normals
• normals estimated by plane-fitting
Point Clouds

List of points
- may or may not include normals
- normals estimated by plane-fitting

Raw data from depth sensors
Triangle Soup

List of triangles
- each triangle has own verts
Triangle Soup

List of triangles
• each triangle has own verts

No notion of triangle neighbors
• (but can find nearby triangles)
Triangle Soup

List of triangles
• each triangle has own verts

No notion of triangle neighbors
• (but can find nearby triangles)

Why do we need neighbors anyway?
Manifold Mesh

Must satisfy three properties:
1. Every edge shared by one/two faces
Manifold Mesh

Must satisfy three properties:
1. Every edge shared by one/two faces

\textbf{not} manifold: has T junction
Manifold Mesh

Must satisfy three properties:
1. Every edge shared by one/two faces
2. Faces around verts are triangle fans

not manifold
Manifold Mesh

Must satisfy three properties:
1. Every edge shared by one/two faces
2. Faces around verts are triangle fans

Intuitively: mesh locally “looks like a single surface”
Manifold Mesh

Must satisfy three properties:
1. Every edge shared by one/two faces
2. Faces around verts are triangle fans
3. Faces have consistent orientation
Watertight Mesh

Manifold mesh that
• is single piece
• has no boundary
**Watertight Mesh**

Manifold mesh that

- is single piece
- has no boundary

Splits space into well-defined inside/out

- can be “filled with water” without leaks
Representing Surface Data

Graphics Grand Challenge

real-word data often **unstructured**

graphics algorithms usually need **structured** data

watertight mesh
Representing Surface Data

point cloud

triangle soup

manifold mesh

watertight mesh

“mesh repair” algorithms
Representing Surface Data

- point cloud
- triangle soup
- manifold mesh
- watertight mesh

“mesh repair” algorithms (rarely works)
Representing Surface Data

- point cloud
- triangle soup
- manifold mesh
- watertight mesh

Poisson surface reconstruction
Poisson Surface Reconstruction

Interpolates point cloud and normals
Triangle Mesh Data Structures

List of points & triangle indices

<table>
<thead>
<tr>
<th>Vertex List</th>
<th>Triangle List</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Triangle Mesh Data Structures

List of points & triangle indices
Pros:
• lightweight, compact
• native GPU data structure
• very common file data structure
Cons:
Triangle Mesh Data Structures

List of points & triangle indices

Pros:
• lightweight, compact
• native GPU data structure
• very common file data structure

Cons:
• neighbor queries **slow**
• finding boundaries **slow**
Half-edge Data Structure

Store mesh as set of half-edges
Half-edge Data Structure

- Half-Edge
- Face
- Vertex
- Next
- Half-Edge face

http://coderender.blogspot.com
Half-edge Data Structure

Store mesh as set of half-edges

Pros:

- easy to “walk around” faces, vertices
- all kinds of neighbor queries easy
Half-edge Data Structure

Store mesh as set of half-edges

Pros:
• easy to “walk around” faces, vertices
• all kinds of neighbor queries easy

Cons:
• large memory footprint
• tricky to implement (tons of pointers!)
Half-edge Data Structure

Store mesh as set of **half-edges**

Pros:
- easy to “walk around” faces, vertices
- all kinds of neighbor queries easy

Cons:
- large memory footprint
- tricky to implement (tons of pointers!)
- use existing libraries (e.g. OpenMesh)
Types of Manifold Meshes

triangle

quad-dominant

quad

exotic (hexagonal, etc)
Triangles vs Quads

Triangles simpler
Quads more natural for flat & cylindrical geometry
Arbitrary Quads Are Not Planar!

Arbitrarily triangulate to render them…
Subdivision

Input: coarse control mesh

Output: finer mesh with smoother details
Linear Subdivision

Split faces 1:4

Insert verts at edge midpts

Adds faces, but doesn’t change shape
Nonlinear Subdivision

Everybody has pet subdivision method
Most popular:

• triangle meshes: **Loop** subdivision
Nonlinear Subdivision

Everybody has pet subdivision code
Most popular:
• triangle meshes: Loop subdivision
• quad meshes: Catmull-Clark
Catmull-Clark Subdivision

Rules for adding new points and replacing old points

Works best for regular quad meshes
Regular vs Irregular Vertices

Regular vertices have four edges

Irregular vertices have 3 or 5+ edges

• also called extraordinary vertices
Catmull-Clark Subdivision

Rules for adding new points and replacing old points

Many schemes for handling irregular verts
Subdivision: Complications

Dealing with irregular vertices
Dealing with creases

• some edges shouldn’t be smoothed...
Subdivision: Complications

Dealing with irregular vertices
Dealing with creases
  • some edges shouldn’t be smoothed…
Dealing with boundaries

In general, allowing finer-grained control of subdivision process
Subdivision Surface

Smooth surface at **limit** of subdivision

Fundamental building block of Pixar’s Renderman engine
Other Mesh Operations

Decimation

Remeshing

Quadrangulation

Smoothing
Other Mesh Operations

Graphics subfield: geometry processing

• uses sophisticated theory from linear algebra, differential geometry, etc.

In practice: several good packages

• CGAL – general purpose

• OpenMesh – halfedge, subdivision
Volume Data

Who cares about volumes?
• just render outer skin?
Volume Data

Who cares about volumes?
• just render outer skin?

Translucent object (colored glass, fog, …)
Volume Data

Who cares about volumes?
• just render outer skin?

Translucent object (colored glass, fog,...)
Physical simulations & deformations
• fracture
Volume Data

Who cares about volumes?
• just render outer skin?

Translucent object (colored glass, fog,…)
Physical simulations & deformations
• fracture

Modeling primitive for cutting & sculpting
Representing Volumes

Voxelization: 3D rasterization
Representing Volumes

Voxelization: 3D rasterization

really easy… but boundary chunky
Representing Volumes

**Tetrahedral** (tet) mesh: 3D analogue of triangle mesh

- each tet “hyperface” has 4 faces, 6 edges, 4 verts
Representing Volumes

**Tetrahedral** (tet) mesh: 3D analogue of triangle mesh

- each tet “hyperface” has 4 faces, 6 edges, 4 verts
- bdry is watertight triangle mesh
Generating 2D Mesh From Curve

Can be done by triangulation
Generating 2D Mesh From Curve

Can be done by triangulation

Greedy “ear-cutting” always works
Generating 3D Mesh From Surface

“Tetrahedralization” not always possible!

Shoenhardt Polytope
Generating 3D Mesh From Surface

“Tetrahedralization” not always possible!

Must add extra inner points
• “Steiner pts”
Generating 3D Mesh From Surface

“Tetrahedralization” not always possible!

Must add extra inner points

• “Steiner pts”

Algorithm is complex and bug-prone

• popular library: TetGen
Representing Volumes

**Hexahedral** mesh: 3D version of quad mesh

- “hexahedron” = 6 sides (cube)
Representing Volumes

**Hexahedral** mesh: 3D version of quad mesh
- “hexahedron” = 6 sides (cube)

No general hexahedralization algorithm exists (!)
Constructive Solid Geometry

Start with simple buildings blocks

- sphere, cubes, cylinders, …

Build complicated objects using tree of operations
CSG Operations

union

$A \cup B$
CSG Operations

union

$A \cup B$

intersection

$A \cap B$
CSG Operations

union

\[ A \cup B \]

intersection

\[ A \cap B \]

set difference

\[ A \setminus B \]
CSG Tree

Pros:
• geometry is exact
• simple representation for visually-complex shapes
CSG Tree

Pros:
• geometry is **exact**
• simple representation for visually-complex shapes

Cons:
• some shapes difficult to model

Used in many game engines (e.g. Unreal)