1.2 Surface Representation

& Data Structures
Administrative

• No class next Tuesday, due to **Siggraph deadline**

• Introduction to first programming exercise next Thursday

Siggraph Deadline 2013@ILM, Ewww!
After Siggraph Deadline @ILM
Last Time

Capture → Geometry Processing
   → Reconstruction
   → Analysis
   → Manipulation

→ Rendering
→ Reproduction
Geometric Representations

- Point based
- Quad mesh
- Triangle mesh
- Implicit surfaces / particles
- Volumetric
- Tetrahedrons
Geometric Representations

Surface Representations

- Point based
- Quad mesh
- Triangle mesh

Implicit surfaces / particles
Volumetric
Tetrahedrons
High Resolution
Large scenes
Outline

• Parametric Approximations
• Polygonal Meshes
• Data Structures
Parametric Representation

Surface is the range of a function

\[ f : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad \mathcal{S}_\Omega = f(\Omega) \]

2D example: A Circle

\[ f : [0, 2\pi] \rightarrow \mathbb{R}^2 \]

\[ f(t) = \begin{pmatrix} r \cos(t) \\ r \sin(t) \end{pmatrix} \]
Parametric Representation

Surface is the range of a function

\[ f : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad \mathcal{S}_\Omega = f(\Omega) \]

2D example: Island coast line

\[ f : [0, 2\pi] \rightarrow \mathbb{R}^2 \]

\[ f(t) = \left( \begin{array}{c} ? \\ ? \end{array} \right) \]
Surface is the range of a function

\[ f : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad \mathcal{S}_\Omega = f(\Omega) \]

2D example: Island coast line

\[ f : [0, 2\pi] \rightarrow \mathbb{R}^2 \]

\[ f(t) = (\ ? \ ) \]
Polynomial Approximation

Polynomials are computable functions

\[ f(t) = \sum_{i=0}^{p} c_i t^i = \sum_{i=0}^{p} \tilde{c}_i \phi_i(t) \]

Taylor expansion up to degree \( p \)

\[ g(h) = \sum_{i=0}^{p} \frac{1}{i!} g^{(i)}(0) h^i + O(h^{p+1}) \]

Error for approximation \( g \) by polynomial \( f \)

\[ f(t_i) = g(t_i), \quad 0 \leq t_0 < \cdots < t_p \leq h \]

\[ |f(t) - g(t)| \leq \frac{1}{(p + 1)!} \max f^{(p+1)} \prod_{i=0}^{p} (t - t_i) = O(h^{p+1}) \]
Polynomial Approximation

Approximation error is $O(h^{p+1})$

Improve approximation quality by

- increasing $p$ ... higher order polynomials
- decreasing $h$ ... shorter / more segments

Issues

- smoothness of the target data ($\max_t f^{(p+1)}(t)$)
- smoothness condition between segments
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
- Error inversely proportional to #faces
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
Polygonal meshes are a good compromise

- Piecewise linear approximation $\rightarrow$ error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
Polygonal meshes are a good compromise

- Piecewise linear approximation → error is $O(h^2)$
- Error inversely proportional to #faces
- Arbitrary topology surfaces
- Piecewise smooth surfaces
- Adaptive sampling
- Efficient GPU-based rendering/processing
Outline

• Parametric Approximations

• Polygonal Meshes

• Data Structures
• Graph \( \{ V, E \} \)
Graph Definitions

- Graph \( \{ V,E \} \)
- Vertices \( V = \{ A,B,C,\ldots,K \} \)
Graph Definitions

- Graph \( \{V,E\} \)
- Vertices \( V = \{A,B,C,\ldots,K\} \)
- Edges \( E = \{(AB),(AE),(CD),\ldots\} \)
Graph Definitions

- Graph \( \{V,E\} \)
- Vertices \( V = \{A,B,C,\ldots,K\} \)
- Edges \( E = \{(AB),(AE),(CD),\ldots\} \)
- Faces \( F = \{(ABE),(EBF),(EFIH),\ldots\} \)
Graph Definitions

Vertex degree or valence: number of incident edges

- $\text{deg}(A) = 4$
- $\text{deg}(E) = 5$
Connectivity

Connected:
Path of edges connecting every two vertices
Connectivity

Connected:
Path of edges connecting every two vertices

Subgraph:
Graph \{V',E'\} is a subgraph of graph \{V,E\} if \(V'\) is a subset of \(V\) and \(E'\) is a subset of \(E\) incident on \(V'\).
Connectivity

Connected:
Path of edges connecting every two vertices

Subgraph:
Graph \( \{V',E'\} \) is a subgraph of graph \( \{V,E\} \) if \( V' \) is a subset of \( V \) and \( E' \) is a subset of \( E \) incident on \( V' \).
Connectivity

Connected:
Path of edges connecting every two vertices

Subgraph:
Graph \( \{V', E'\} \) is a subgraph of graph \( \{V, E\} \) if \( V' \) is a subset of \( V \) and \( E' \) is a subset of \( E \) incident on \( V' \).

Connected Components:
Maximally connected subgraph
Connectivity

Connected:
Path of edges connecting every two vertices

Subgraph:
Graph \( \{V',E'\} \) is a subgraph of graph \( \{V,E\} \) if \( V' \) is a subset of \( V \) and \( E' \) is a subset of \( E \) incident on \( V' \).

Connected Components:
Maximally connected subgraph
Graph Embedding

**Embedding:** Graph is **embedded** in $\mathbb{R}^d$, if each vertex is assigned a position in $\mathbb{R}^d$. 

Embedding in $\mathbb{R}^2$ 

Embedding in $\mathbb{R}^3$
**Embedding**: Graph is **embedded** in $\mathbb{R}^d$, if each vertex is assigned a position in $\mathbb{R}^d$. 

Embedding in $\mathbb{R}^3$
Planar Graph

Graph whose vertices and edges can be embedded in $\mathbb{R}^2$ such that its edges do not intersect.
Triangulation:

*Straight line plane* graph where every face is a triangle

**Why?**

- simple homogenous data structure
- efficient rendering
- simplifies algorithms
- by definition, triangle is planar
- any polygon can be triangulated
• **Mesh**: straight-line graph embedded in $\mathbb{R}^3$

• **Boundary edge**: adjacent to exactly 1 face

• **Regular edge**: adjacent to exactly 2 faces

• **Singular edge**: adjacent to more than 2 faces

• **Closed mesh**: mesh with no boundary edges
A geometric graph $Q = (V, E)$
with $V = \{p_0, p_1, \ldots, p_{n-1}\}$ in $\mathbb{R}^d$, $d \geq 2$
and $E = \{(p_0, p_1) \ldots (p_{n-2}, p_{n-1})\}$
is called a **polygon**

A polygon is called
- flat, if all edges are on a plane
- closed, if $p_0 = p_{n-1}$
While digital artists call it Wireframe, ...
A set $M$ of finite number of closed polygons $Q_i$ if:

- Intersection of inner polygonal areas is empty
- Intersection of 2 polygons from $M$ is either empty, a point or an edge $e \in E$
- Every edge $e \in E$ belongs to at least one polygon
- The set of all edges which belong only to one polygon are called edges of the polygonal mesh and are either empty or form a single closed polygon
Polygonal Mesh Notation

\[ \mathcal{M} = (\{v_i\}, \{e_j\}, \{f_k\}) \]

geometry \( v_i \in \mathbb{R}^3 \)
A polygonal mesh notation is given as:

\[ \mathcal{M} = (\{v_i\}, \{e_j\}, \{f_k\}) \]

**Geometry:** \[ v_i \in \mathbb{R}^3 \]

**Topology:** \[ e_i, f_i \subset \mathbb{R}^3 \]
**Global Topology: Genus**

- **Genus**: Maximal number of closed simple cutting curves that do not disconnect the graph into multiple components.
- Or half the maximal number of closed paths that do not disconnect the mesh.
- Informally, the number of **holes** or **handles**.

Genus 0  Genus 1  Genus 2  Genus 3
• For a closed polygonal mesh of genus $g$, the relation of the number $V$ of vertices, $E$ of edges, and $F$ of faces is given by Euler’s formula:

$$V - E + F = 2(1 - g)$$

• The term $2(1 - g)$ is called the Euler characteristic $\chi$. 
Euler Poincaré Formula

\[ V - E + F = 2(1 - g) \]

\[ 4 - 6 + 4 = 2(1 - 0) \]
Euler Poincaré Formula

\[ V - E + F = 2(1 - g) \]

\[ 16 - 32 + 16 = 2(1 - 1) \]
**Theorem:** Average vertex degree in a closed manifold triangle mesh is $\sim 6$

**Proof:** Each face has 3 edges and each edge is counted twice: $3F = 2E$

by Euler’s formula: $V+F-E = V+2E/3-E = 2-2g$

Thus $E = 3(V-2+2g)$

So average degree $= 2E/V = 6(V-2+2g) \sim 6$ for large $V$
Euler Consequences

Triangle mesh statistics

- $F \approx 2V$
- $E \approx 3V$
- Average valence = 6

Quad mesh statistics

- $F \approx V$
- $E \approx 2V$
- Average valence = 4
Euler Characteristic

Sphere: \( \chi = 2 \)
Torus: \( \chi = 0 \)
Moebius Strip: \( \chi = 0 \)
Klein Bottle: \( \chi = 0 \)
How many pentagons?
Any \textbf{closed surface} of \textbf{genus 0} consisting only of \textbf{hexagons} and \textbf{pentagons} and where every \textbf{vertex} has \textbf{valence 3} must have exactly \textbf{12 pentagons}
Local neighborhoods are disk-shaped

\[ f(D_{\epsilon}[u, v]) = D_{\delta}[f(u, v)] \]

Guarantees meaningful neighbor enumeration

- required by most algorithms

Non-manifold Examples:
Outline

• Parametric Approximations
• Polygonal Meshes
• Data Structures
Mesh Data Structures

• How to store geometry & connectivity?
• compact storage and file formats
• Efficient algorithms on meshes
  • Time-critical operations
  • All vertices/edges of a face
  • All incident vertices/edges/faces of a vertex
What should be stored?

- Geometry: 3D vertex coordinates
- Connectivity: Vertex adjacency
- Attributes:
  - normals, color, texture coordinates, etc.
  - Per Vertex, per face, per edge
What should it support?

- Rendering
- Queries
  - What are the vertices of face #3?
  - Is vertex #6 adjacent to vertex #12?
  - Which faces are adjacent to face #7?
- Modifications
  - Remove/add a vertex/face
  - Vertex split, edge collapse
Different Data Structures:

- Time to construct (preprocessing)
- Time to answer a query
  - Random access to vertices/edges/faces
  - Fast mesh traversal
  - Fast Neighborhood query
- Time to perform an operation
  - split/merge
- Space complexity
- Redundancy
Different Data Structures:

- Different topological data storage
- Most important ones are face and edge-based (since they encode connectivity)
- Design decision ~ memory/speed trade-off
Face:

- 3 vertex positions

<table>
<thead>
<tr>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

$9 \times 4 = 36 \text{ B/f}$ (single precision)

$72 \text{ B/v}$ (Euler Poincaré)

No explicit connectivity
Indexed Face List:

- Vertex: position
- Face: Vertex Indices

<table>
<thead>
<tr>
<th>Vertices</th>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>i</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>x</td>
<td>...</td>
</tr>
</tbody>
</table>

$12 \text{ B/v} + 12 \text{ B/f} = 36\text{B/v}$

No explicit adjacency info
Face-Based Connectivity

**Vertex:**
- position
- 1 face

**Face:**
- 3 vertices
- 3 face neighbors

64 B/v

No edges: Special case handling for arbitrary polygons
Edges always have the same topological structure

Efficient handling of polygons with variable valence
(Winged) **Edge-Based Connectivity**

**Vertex:**
- position
- 1 edge

**Edge:**
- 2 vertices
- 2 faces
- 4 edges

**Face:**
- 1 edges

120 B/v

Edges have no orientation: special case handling for neighbors
Halfedge-Based Connectivity

Vertex:
- position
- 1 halfedge

Edge:
- 1 vertex
- 1 face
- 1, 2, or 3 halfedges

Face:
- 1 halfedge

96 to 144 B/v

Edges have orientation: No-run time overhead due to arbitrary faces
Arbitrary Faces during Modeling
1. Start at vertex
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
One-Ring Traversal

1. Start at vertex
2. Outgoing halfedge
3. Opposite halfedge
4. Next halfedge
5. Opposite
6. Next
7. ...
Halfedge datastructure Libraries

**CGAL**
- www.cgal.org
- Computational Geometry
- Free for non-commercial use

**OpenMesh**
- www.openmesh.org
- Mesh processing
- Free, LGPL license
Why *Open*mesh?  

**Flexible / Lightweight**  
- Random access to vertices/edges/faces  
- Arbitrary scalar types  
- Arrays or lists as underlying kernels

**Efficient in space and time**  
- Dynamic memory management for array-based meshes (not in CGAL)  
- Extendable to specialized kernels for non-manifold meshes (not in CGAL)

**Easy to Use**
• Textbook: Chapter
• http://www.openmesh.org
• Kettner, Using generic programming for designing a data structure for polyhedral surfaces, Symp. on Comp. Geom., 1998
• Botsch et al., OpenMesh - A generic and efficient polygon mesh data structure, OpenSG Symp. 2002
Learn the terms and notations
Next Time

• **Explicit & Implicit Surfaces**

• **Exercise 1**: Getting Started with Mesh Processing