Polygonal Meshes

COS 426
3D Object Representations

Points
- Range image
- Point cloud

Surfaces
- Polyhedral mesh
  - Subdivision
  - Parametric
  - Implicit

Solids
- Voxels
- BSP tree
- CSG
- Sweep

High-level structures
- Scene graph
- Application specific
3D Polygonal Mesh

Set of polygons representing a 2D surface embedded in 3D

- Triceratops: $v = 2832$, $p = 2834$
- Beethoven: $v = 2655$, $p = 2812$
- Schooner: $v = 2372$, $p = 2384$
- Cessna: $v = 3745$, $p = 3927$
- Sandal: $v = 2636$, $p = 2953$
- Cow: $v = 2904$, $p = 5804$
- Cow_poly (the polygonal cow is not shown; it is the same cow model, but not fully triangulated): $v = 2904$, $p = 3263$
- Shark: $v = 2560$, $p = 2562$
3D Polygonal Mesh

Geometry & topology
Geometry background

Scene is usually approximated by 3D primitives

- Point
- Vector
- Line segment
- Ray
- Line
- Plane
- Polygon
3D Point

Specifies a location

- Represented by three coordinates
- Infinitely small

```c
struct Point {
    Coordinate x;
    Coordinate y;
    Coordinate z;
};
```
3D Vector

Specifies a direction and a magnitude

- Represented by three coordinates
- Magnitude $||V|| = \sqrt{dx^2 + dy^2 + dz^2}$
- Has no location

```c
struct Vector {
    Coordinate dx;
    Coordinate dy;
    Coordinate dz;
};
```

(dx,dy,dz)
Scalar / dot product of two 3D vectors

\[ V_1 \cdot V_2 = dx_1 * dx_2 + dy_1 * dy_2 + dz_1 * dz_2 = ||V_1|| \ ||V_2|| \cos(\Theta) \]
3D Vector

Cross product of two 3D vectors

\[ V_1 \times V_2 = (dy_1dx_2 - dz_1dy_2, dz_1dx_2 - dx_1dz_2, dx_1dy_2 - dy_1dx_2) \]

- vector perpendicular to both \( V_1 \) and \( V_2 \)
- magnitude is \( ||V_1|| \ ||V_2|| \ \sin(\Theta) \)
3D Line Segment

Linear path between two points

- Parametric representation:
  \[ p = P_1 + t (P_2 - P_1), \quad (0 \leq t \leq 1) \]

```c
struct Segment {
    Point P1;
    Point P2;
};
```
3D Ray

Line segment with one endpoint at infinity

- Parametric representation:
  » $p = P_1 + tV, \quad (0 \leq t < \infty)$

```c
struct Ray {
    Point P1;
    Vector V;
};
```
3D Line

Line segment with both endpoints at infinity

- Parametric representation:
  \[ p = P_1 + t V, \quad (-\infty < t < \infty) \]

```
struct Line {
    Point P1;
    Vector V;
};
```
3D Plane

Defined by three points
3D Plane

Defined by three points

- Implicit representation:
  - $ax + by + cz + d = 0$
  - OR
  - $P \cdot N + d = 0$

- $N$ is the plane normal
  - Unit-length
  - Perpendicular to plane

```
struct Plane {
    Vector N;
    float d;
};
```
3D Polygon

Region “inside” a sequence of coplanar points

struct Polygon {
    vector<Point> points;
};

• Points in counter-clockwise order
  (defines normal)

• Winding rule determines inside/outside
3D Polygonal Mesh

Set of polygons representing a 2D surface embedded in 3D
3D Polygonal Meshes

Why are they of interest?

- Simple, common representation
- Rendering with hardware support
- Output of many acquisition tools
- Input to many simulation/analysis tools
3D Polygonal Meshes

Properties

- Efficient display
- Easy acquisition
- Accurate
- Concise
- Intuitive editing
- Efficient editing
- Efficient intersections
- Guaranteed validity
- Guaranteed smoothness
- etc.
Outline

Acquisition
Processing
Representation
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
- Interchange formats

Scanners
- Laser range scanners
- Geological survey
- CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
  - Interchange formats

Scanners
- Laser range scanners
- Geological survey
- CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
  - Interchange formats

Scanners
- Laser range scanners
- Geological survey
- CAT, MRI, etc.

Simulations
- Physical processes

Jose Maria De Espona
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
- Interchange formats

Scanners
- Laser range scanners
- Geological survey
- CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
- Interchange formats

Scanners
- Laser range scanners
  - Geological survey
- CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
- Interchange formats

Scanners
- Laser range scanners
- Geological survey
  - CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes

www.volumegraphics.com

SUNY Stony Brook
Polygonal Mesh Acquisition

Interactive modeling
- Polygon editors
- Interchange formats

Scanners
- Laser range scanners
- Geological survey
- CAT, MRI, etc. (isosurfaces)

Simulations
- Physical processes
Outline

Acquisition

Processing

Representation
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis

- Normals
  - Curvature

Warps

- Rotate
- Deform

Filters

- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
  - Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel

Figure 32: Curvature of curve at point P is $1/k$
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel

Thouis "Ray" Jones

Weighted Average of Neighbor Vertices

Olga Sorkine
Polygons Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
  - Sharpen
- Truncate
- Bevel

Desbrun

Weighted Average of Neighbor Vertices

Olga Sorkine
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warp
- Rotate
- Deform

Filters
- Smooth
- Sharpen
  - Truncate
- Bevel

Cube $\rightarrow$ \(\frac{1}{4}\) truncated $\rightarrow$ uniform truncated $\rightarrow$ \(\frac{3}{4}\) truncated $\rightarrow$ Rectified

Conway

0.35
## Polygonal Mesh Processing

<table>
<thead>
<tr>
<th>Original</th>
<th>Truncation</th>
<th>Rectification</th>
<th>Bitruncation (truncated dual)</th>
<th>Birecification (dual)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tetrahedron</td>
<td>Truncated tetrahedron</td>
<td>Octahedron</td>
<td>Truncated tetrahedron</td>
<td>Tetrahedron</td>
</tr>
<tr>
<td>Cube</td>
<td>Truncated cube</td>
<td>Cuboctahedron</td>
<td>Truncated octahedron</td>
<td>Octahedron</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>Truncated dodecahedron</td>
<td>Icosidodecahedron</td>
<td>Truncated icosahedron</td>
<td>Icosahedron</td>
</tr>
</tbody>
</table>
## Polygonal Mesh Processing

<table>
<thead>
<tr>
<th>{3,3}</th>
<th>(3.6.6)</th>
<th>(3.3.3.3)</th>
<th>(4.6.6)</th>
<th>(3.3.3.3.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{4,3}</td>
<td>(3.8.8)</td>
<td>(3.4.3.4)</td>
<td>(4.6.8)</td>
<td>(3.3.3.3.4)</td>
</tr>
<tr>
<td>{5,3}</td>
<td>(3.10.10)</td>
<td>(3.5.3.5)</td>
<td>(4.6.10)</td>
<td>(3.3.3.3.5)</td>
</tr>
</tbody>
</table>

Wikipedia
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel

www.srcf.ucam.org
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
  - Resample
  - Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing

- **Subdivide**
  - Resample
  - Simplify

Topological fixup

- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations

- Crop
- Subtract

Dirk Balfanz, Igor Guskov, Sanjeev Kumar, & Rudro Samanta,
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract

Rossignac

Vertex Clustering
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
  - Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing

- Subdivide
- Resample
- Simplify

Topological fixup

- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations

- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygons Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract
Polygonal Mesh Processing

Remeshing
- Subdivide
- Resample
- Simplify

Topological fixup
- Fill holes
- Fix cracks
- Fix self-intersections

Boolean operations
- Crop
- Subtract

FvDFH Figure 12.27
Polygonal Mesh Processing

Procedural generation

- Surface of revolution
- Sweep
Polygonal Mesh Processing

Procedural generation

- Surface of revolution
  - Sweep
Procedural generation

- Surface of revolution
- Sweep

Fowler et al., 1992
Polygonal Mesh Processing

Procedural generation

- Surface of revolution
- Sweep

Fowler et al., 1992
Polygonal Mesh Processing

Most operations use a few low-level operations:

- Subdivide face
- Subdivide edge
- Collapse edge
- Merge vertices
- Remove vertex
Polygonal Mesh Processing

Most operations use a few low-level operations:

- Subdivide face
  - Subdivide edge
  - Collapse edge
  - Merge vertices
  - Remove vertex
Polygonal Mesh Processing

Most operations use a few low-level operations:

- Subdivide face
- Subdivide edge
- Collapse edge
- Merge vertices
- Remove vertex

Subdivide edge
Most operations use a few low-level operations:

- Subdivide face
- Subdivide edge
- **Collapse edge**
- Merge vertices
- Remove vertex

**Collapse edge**
Polygonal Mesh Processing

Most operations use a few low-level operations:

- Subdivide face
- Subdivide edge
- Collapse edge
- Merge vertices
- Remove vertex

Merge Vertices
Polygonal Mesh Processing

Most operations use a few low-level operations:

- Subdivide face
- Subdivide edge
- Collapse edge
- Merge vertices
- Remove vertex

Remove Vertex
Polygon Mesh Representation

Data structures determine algorithms
- Data structure must support key operations of algorithm efficiently

Examples:
- Drawing a mesh
- Removing a vertex
- Smoothing a region
- Intersecting polyhedra

Different data structures for different algorithms
Polygon Mesh Representation

Important properties of mesh representation?
Polygon Mesh Representation

Important properties of mesh representation?

- Efficient traversal of topology
- Efficient use of memory
- Efficient updates
Polygon Mesh Representation

Possible data structures

- List of independent faces
- Vertex and face tables
- Adjacency lists
- Winged edge
- Half edge
- etc.
Independent Faces

Each face lists vertex coordinates
- Redundant vertices
- No adjacency information

<table>
<thead>
<tr>
<th>FACE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>F₁</td>
</tr>
<tr>
<td>F₂</td>
</tr>
<tr>
<td>F₃</td>
</tr>
</tbody>
</table>
Vertex and Face Tables

Each face lists vertex references

- Shared vertices
- Still no adjacency information

**Vertex Table**

<table>
<thead>
<tr>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
<td>X2</td>
<td>Y2</td>
</tr>
<tr>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
<td>X3</td>
<td>Y3</td>
</tr>
<tr>
<td>X3</td>
<td>Y3</td>
<td>Z3</td>
<td>X4</td>
<td>Y4</td>
</tr>
<tr>
<td>X4</td>
<td>Y4</td>
<td>Z4</td>
<td>X5</td>
<td>Y5</td>
</tr>
</tbody>
</table>

**Face Table**

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>V2</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>V4</td>
<td>V3</td>
</tr>
<tr>
<td>V2</td>
<td>V5</td>
<td>V4</td>
</tr>
</tbody>
</table>
Adjacency Lists

Store all vertex, edge, and face adjacencies

- Efficient adjacency traversal
- Extra storage
Partial Adjacency Lists

Can we store only some adjacency relationships and derive others?
Winged Edge

Adjacency encoded in edges

- All adjacencies in $O(1)$ time
- Little extra storage (fixed records)
- Arbitrary polygons
Winged Edge

Example:

\[
\begin{align*}
(x_1, y_1, z_1) & \quad (x_2, y_2, z_2) & \quad (x_3, y_3, z_3) & \quad (x_4, y_4, z_4) & \quad (x_5, y_5, z_5) \\
\end{align*}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{VERTEX TABLE} & V_1 & V_2 & V_3 & V_4 & V_5 \\
\hline
X_1 & Y_1 & Z_1 & & & \\
X_2 & Y_2 & Z_2 & & & \\
X_3 & Y_3 & Z_3 & & & \\
X_4 & Y_4 & Z_4 & & & \\
X_5 & Y_5 & Z_5 & & & \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{EDGE TABLE} & e_1 & e_2 & e_3 \\
\hline
V_1 & V_2 & F_1 & e_1 \\
V_1 & V_3 & F_2 & e_2 \\
V_1 & V_4 & F_3 & e_3 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{FACE TABLE} & F_1 & F_2 & F_3 \\
\hline
e_1 & e_6 & e_5 \\
e_2 & e_3 & e_5 \\
e_3 & e_4 & e_2 \\
e_4 & e_1 & e_6 \\
e_5 & e_7 & e_3 \\
e_6 & e_2 & e_4 \\
e_7 & e_5 & e_6 \\
\hline
\end{array}
\]
Half Edge

Adjacency encoded in edges
- All adjacencies in $O(1)$ time
- Little extra storage (fixed records)
- Arbitrary polygons

Similar to winged-edge, except adjacency encoded in half-edges
Summary

Polygonal meshes
- Most common surface representation
- Fast rendering

Processing operations
- Must consider irregular vertex sampling
- Must handle/avoid topological degeneracies

Representation
- Which adjacency relationships to store depend on which operations must be efficient