Polygonal Meshes

COS 426
<table>
<thead>
<tr>
<th>Points</th>
<th>Solids</th>
<th>High-level structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>◦ Range image</td>
<td>◦ Voxels</td>
<td>◦ Scene graph</td>
</tr>
<tr>
<td>◦ Point cloud</td>
<td>◦ BSP tree</td>
<td>◦ Application specific</td>
</tr>
<tr>
<td></td>
<td>◦ CSG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>◦ Sweep</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surfaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td>◦ Polygonal mesh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>◦ Subdivision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>◦ Parametric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>◦ Implicit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3D Object Representations

Points
- Range image
- Point cloud

Surfaces
- Polygonal 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: \( v = 2832 \), \( p = 2834 \)
- galleon: \( v = 2372 \), \( p = 2384 \)
- cessna: \( v = 3745 \), \( p = 3927 \)
- beethoven: \( v = 2655 \), \( p = 2812 \)
- sandal: \( v = 2636 \), \( p = 2953 \)
- cow: \( v = 2904 \), \( p = 5804 \)
- cow_poly: \( v = 2904 \), \( p = 3263 \)
- shark: \( v = 2560 \), \( p = 2562 \)

(Shark is not shown; it is the same cow model, but not fully triangulated.)
3D Polygonal Mesh

Geometry & topology

Face
Edge
Vertex (x,y,z)

Zorin & Schroeder
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
typedef struct {
    Coordinate x;
    Coordinate y;
    Coordinate z;
} Point;
```

\[(x,y,z)\] Origin
3D Vector

Specifies a direction and a magnitude

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

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

(dx,dy,dz)
3D Vector

Dot product of two 3D vectors

\[ \mathbf{V}_1 \cdot \mathbf{V}_2 = ||\mathbf{V}_1|| \cdot ||\mathbf{V}_2|| \cos(\Theta) \]
3D Vector

Cross product of two 3D vectors

- $V_1 \cdot V_2 = (dy_1dx_2 - dz_1dy_2, dz_1dx_2 - dx_1dz_2, dx_1dy_2 - dy_1dx_2)$
- $V_1 \times V_2 = \text{vector perpendicular to both } V_1 \text{ and } V_2$
- $||V_1 \times V_2|| = ||V_1|| \cdot ||V_2|| \cdot \sin(\Theta)$
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
typedef struct {
    Point P1;
    Point P2;
} Segment;
```
3D Ray

Line segment with one endpoint at infinity

- Parametric representation:
  - \( P = P_1 + t \ V, \quad (0 \leq t < \infty) \)

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

Line segment with both endpoints at infinity

- Parametric representation:
  \[ P = P_1 + t \mathbf{V}, \quad (-\infty < t < \infty) \]

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

Defined by three points

$P_1$, $P_2$, $P_3$
3D Plane

Defined by three points

- Implicit representation:
  - \( ax + by + cz + d = 0 \)
  - \( P \cdot N + d = 0 \)

- \( N \) is the plane “normal”
  - Unit-length vector
  - Perpendicular to plane

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

Set of points “inside” a sequence of coplanar points

typedef struct {
    Point *points;
    int npoints;
} Polygon;

- 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
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.volumeographics.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
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 $P$ 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
Polygonal 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

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel

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

- Normals
- Curvature
- Warps
- Rotate
- Deform
- Filters
  - Smooth
  - Sharpen
  - Truncate
  - Bevel

<table>
<thead>
<tr>
<th>Polygon</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,3</td>
<td>(3.6.6)</td>
</tr>
<tr>
<td>4,3</td>
<td>(3.8.8)</td>
</tr>
<tr>
<td>5,3</td>
<td>(3.10.10)</td>
</tr>
<tr>
<td>3,3,3.3</td>
<td>(3.3.3.3)</td>
</tr>
<tr>
<td>3,3,3.3</td>
<td>(3.3.3.3)</td>
</tr>
<tr>
<td>3,3,3.3</td>
<td>(3.3.3.3)</td>
</tr>
<tr>
<td>3,3,3.3</td>
<td>(3.3.3.3)</td>
</tr>
<tr>
<td>3,3,3.3</td>
<td>(3.3.3.3)</td>
</tr>
</tbody>
</table>

Wikipedia
Polygonal Mesh Processing

Analysis
- Normals
- Curvature

Warps
- Rotate
- Deform

Filters
- Smooth
- Sharpen
- Truncate
- Bevel

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

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

Borodin
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
Polygonal Mesh Processing

Procedural generation
  - Surface of revolution
  - Sweep
Polygonal Mesh Processing

Procedural generation

- Surface of revolution
  - Sweep
Polygonal Mesh Processing

Procedural generation

- Surface of revolution
- Sweep

Fowler et al., 1992
Polyhedral 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
Most operations use a few low-level operations:

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

Subdivide face
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
Polygonal Mesh Processing

Most operations use a few low-level operations:

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

Remove Vertex
Outline

Acquisition

Processing

Representation
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?

Large Geometric Model Repository
Georgia Tech
Polygon Mesh Representation

Important properties of mesh representation?

- Efficient traversal of topology
- Efficient use of memory
- Efficient updates

Large Geometric Model Repository
Georgia Tech
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

**FACE TABLE**

<table>
<thead>
<tr>
<th>Face</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_1$</td>
<td>$(x_1, y_1, z_1)$ $(x_2, y_2, z_2)$ $(x_3, y_3, z_3)$</td>
</tr>
<tr>
<td>$F_2$</td>
<td>$(x_2, y_2, z_2)$ $(x_4, y_4, z_4)$ $(x_3, y_3, z_3)$</td>
</tr>
<tr>
<td>$F_3$</td>
<td>$(x_2, y_2, z_2)$ $(x_5, y_5, z_5)$ $(x_4, y_4, z_4)$</td>
</tr>
</tbody>
</table>
Vertex and Face Tables

Each face lists vertex references

- Shared vertices
- Still no adjacency information

```
<table>
<thead>
<tr>
<th>VERTEX TABLE</th>
<th>FACE TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_1 )</td>
<td>( F_1 )</td>
</tr>
<tr>
<td>( V_2 )</td>
<td>( V_1 \ V_2 \ V_3 )</td>
</tr>
<tr>
<td>( V_3 )</td>
<td>( V_2 \ V_4 \ V_3 )</td>
</tr>
<tr>
<td>( V_4 )</td>
<td>( V_3 \ V_4 \ V_5 )</td>
</tr>
<tr>
<td>( V_5 )</td>
<td>( V_3 \ V_4 \ V_5 )</td>
</tr>
</tbody>
</table>

(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)
```
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:

![Diagram of Winged Edge with vertex and edge tables]

**VERTEX TABLE**

<table>
<thead>
<tr>
<th>V1</th>
<th>X1</th>
<th>Y1</th>
<th>Z1</th>
<th>e1</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
<td>e6</td>
</tr>
<tr>
<td>V3</td>
<td>X3</td>
<td>Y3</td>
<td>Z3</td>
<td>e3</td>
</tr>
<tr>
<td>V4</td>
<td>X4</td>
<td>Y4</td>
<td>Z4</td>
<td>e5</td>
</tr>
<tr>
<td>V5</td>
<td>X5</td>
<td>Y5</td>
<td>Z5</td>
<td>e6</td>
</tr>
</tbody>
</table>

**EDGE TABLE**

<table>
<thead>
<tr>
<th>e1</th>
<th>V1</th>
<th>V3</th>
<th>F1</th>
<th>e2</th>
<th>e2</th>
<th>e4</th>
<th>e3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e2</td>
<td>V1</td>
<td>V2</td>
<td>F1</td>
<td>e1</td>
<td>e1</td>
<td>e3</td>
<td>e6</td>
</tr>
<tr>
<td>e3</td>
<td>V2</td>
<td>V3</td>
<td>F1</td>
<td>e2</td>
<td>e5</td>
<td>e1</td>
<td>e4</td>
</tr>
<tr>
<td>e4</td>
<td>V3</td>
<td>V4</td>
<td>F2</td>
<td>e1</td>
<td>e3</td>
<td>e7</td>
<td>e5</td>
</tr>
<tr>
<td>e5</td>
<td>V2</td>
<td>V4</td>
<td>F2</td>
<td>e3</td>
<td>e6</td>
<td>e4</td>
<td>e7</td>
</tr>
<tr>
<td>e6</td>
<td>V2</td>
<td>V5</td>
<td>F3</td>
<td>e5</td>
<td>e2</td>
<td>e7</td>
<td>e7</td>
</tr>
<tr>
<td>e7</td>
<td>V4</td>
<td>V5</td>
<td>F3</td>
<td>e4</td>
<td>e5</td>
<td>e6</td>
<td>e6</td>
</tr>
</tbody>
</table>

**FACE TABLE**

<table>
<thead>
<tr>
<th>F1</th>
<th>e1</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>e3</td>
</tr>
<tr>
<td>F3</td>
<td>e5</td>
</tr>
</tbody>
</table>
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