Lecture 10/11:

Mesh Representations & Geometry Processing

Computer Graphics and Imaging
UC Berkeley CS184/284A
A Small Triangle Mesh

8 vertices, 12 triangles
A Large Triangle Mesh

David
Digital Michelangelo Project
28,184,526 vertices
56,230,343 triangles
A Very Large Triangle Mesh

Google Earth
Meshes reconstructed from satellite and aerial photography
Trillions of triangles
Digital Geometry Processing

3D Scanning

3D Printing
Geometry Processing Pipeline

Scan → Process → Print
Geometry Processing
Tasks: 3 Examples
Mesh Upsampling – Subdivision

Increase resolution via interpolation
Mesh Downsampling – Simplification

Decrease resolution; try to preserve shape/appearance
Mesh Regularization

Modify sample distribution to improve quality
This Lecture

Study how to represent meshes (data structures)
Study how to process meshes (geometry processing)
Mesh Representations
List of Triangles

\[
\begin{array}{c|c|c|c}
\text{tris[0]} & \text{[0]} & \text{[1]} & \text{[2]} \\
\hline
x_0, y_0, z_0 & x_2, y_2, z_2 & x_1, y_1, z_1 \\
x_0, y_0, z_0 & x_3, y_3, z_3 & x_2, y_2, z_2 \\
\vdots & \vdots & \vdots \\
\end{array}
\]

\[
(x_0, y_0, z_0) \\
(x_1, y_1, z_1) \\
(x_2, y_2, z_2) \\
(x_3, y_3, z_3)
\]
Lists of Points / Indexed Triangle

vert[0]
vert[1]
x₀, y₀, z₀
x₁, y₁, z₁
x₂, y₂, z₂
x₃, y₃, z₃
⋮

tInd[0]
tInd[1]
0, 2, 1
0, 3, 2
⋮
Comparison

Triangles

+ Simple
– Redundant information

Points + Triangles

+ Sharing vertices reduces memory usage
+ Ensure integrity of the mesh (moving a vertex causes that vertex in all the polygons to move)
Topology vs Geometry

Same geometry, different mesh topology

Same mesh topology, different geometry
Topological Mesh Information

Applications:

• Constant time access to neighbors
e.g. surface normal calculation, subdivision

• Editing the geometry
e.g. adding/removing vertices, faces, edges, etc.

Solution: Topological data structures
Definition: a 2D manifold is a surface that when cut with a small sphere always yields a disk.

Mesh manifolds have the following properties:

- An edge connects exactly two faces
- An edge connects exactly two vertices
- A face consists of a ring of edges and vertices
- A vertex consists of a ring of edges and faces
- Euler's formula $f - e + v = 2$ (for a surface topologically equivalent to a sphere)

(For a cube: $6 - 12 + 8 = 2$)

Topological Validity: Manifold

Manifold

- With border

Not manifold

- With border
Topological Validity: Manifold

Definition: a 2D manifold is a surface that when cut with a small sphere always yields a disk.

If a mesh is manifold we can rely on these useful properties:

• An edge connects exactly two faces
• An edge connects exactly two vertices
• A face consists of a ring of edges and vertices
• A vertex consists of a ring of edges and faces
• Euler’s polyhedron formula holds: \( \#f - \#e + \#v = 2 \) (for a surface topologically equivalent to a sphere) (Check for a cube: \( 6 - 12 + 8 = 2 \))
Topological Validity: Orientation Consistency

Both facing front

Inconsistent orientations

Non-orientable
Triangle-Neighbor Data Structure

```c
struct Tri {
    Vert  * v[3];
    Tri   * t[3];
};

struct Vert {
    Point  pt;
    Tri   * t;
};
```
Triangle-Neighbor – Mesh Traversal

Find next triangle counter-clockwise around vertex v from triangle t

Tri *tccwvt(Vert *v, Tri *t) {
    if (v == t->v[0])
        return t[0];
    if (v == t->v[1])
        return t[1];
    if (v == t->v[2])
        return t[2];
}

Diagram: Triangle with vertices v[0], v[1], v[2], and triangles t[0], t[1], t[2].
Half-Edge Data Structure

```
struct Halfedge {
    Halfedge *twin,
    Halfedge *next;
    Vertex *vertex;
    Edge *edge;
    Face *face;
}
struct Vertex {
    Point pt;
    Halfedge *halfedge;
}
struct Edge {
    Halfedge *halfedge;
}
struct Face {
    Halfedge *halfedge;
}
```

Key idea: two half-edges act as "glue" between mesh elements

Each vertex, edge and face points to one of its half edges
Half-Edge Facilitates Mesh Traversal

Use twin and next pointers to move around mesh

Process vertex, edge and/or face pointers

Example 1: process all vertices of a face

Halfedge* h = f->halfedge;
do {
    process(h->vertex);
    h = h->next;
} 
while( h != f->halfedge );
Half-Edge Facilitates Mesh Traversal

Example 2: process all edges around a vertex

Halfedge* h = v->halfedge;

do {
    process(h->edge);
    h = h->twin->next;
}
while( h != v->halfedge );
Local Mesh Operations
Half-Edge – Local Mesh Editing

Basic operations for linked list: insert, delete

Basic ops for half-edge mesh: flip, split, collapse edges

Allocate / delete elements; reassign pointers
(Care needed to preserve mesh manifold property)
Half-Edge – Edge Flip

• Triangles \((a,b,c), (b,d,c)\) become \((a,d,c), (a,b,d)\):

• Long list of pointer reassignments
• However, no elements created/destroyed.
Half-Edge – Edge Split

• Insert midpoint m of edge (c,b), connect to get four triangles:

• This time have to add elements

• Again, many pointer reassignments
Half-Edge – Edge Collapse

• Replace edge (c,d) with a single vertex m:

• This time have to delete elements

• Again, many pointer reassignments
Global Mesh Operations: Geometry Processing

- Mesh subdivision
- Mesh simplification
- Mesh regularization
Subdivision Surfaces
Subdivision Surfaces (Reminder)

Start with coarse polygon mesh ("control cage")

- Subdivide each element
- Update vertices via local averaging

Many possible rules:

- Catmull-Clark (quads)
- Loop (triangles)
- ...

Common issues:

- interpolating or approximating?
- continuity at vertices?

Relatively easy for modeling; harder to guarantee continuity
Core Idea: Let Subdivision Define The Surface

In Bezier curves, we saw:

- Evaluation by subdivision (de Casteljau algorithm)
- Or evaluation by algebra (Bernstein polynomials)

Insight that leads to subdivision surfaces:

- Free ourselves from the algebraic evaluation
- Let subdivision fully define the surface

Many possible subdivision rules – different surfaces

- Technical challenge shifts to designing rules and proving properties (e.g. convergence and continuity)
- Applying rules to compute surface is procedural
To Be Continued
Acknowledgments

Thanks to Keenan Crane, Pat Hanrahan, Steve Marschner and James O’Brien for presentation resources.