Lecture 10:
Mesh Representations & Geometry Processing

Computer Graphics and Imaging
UC Berkeley CS184/284A, Spring 2017
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
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

\[
\begin{array}{ccc}
 0 & x_0, y_0, z_0 & x_2, y_2, z_2 & x_1, y_1, z_1 \\
 1 & x_0, y_0, z_0 & x_3, y_3, z_3 & x_2, y_2, z_2 \\
 \vdots & \vdots & \vdots & \vdots \\
\end{array}
\]
Lists of Points / Indexed Triangle

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

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

Triangle $T₀$ and $T₁$
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, etc.

Solution: Topological data structures
struct Tri {
    Vert  * v[3];
    Tri   * t[3];
}

struct Vert {
    Point  pt;
    Tri    *t;
}
Find next triangle counter-clockwise around vertex v from triangle t

```c
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];
}
```
Topological Validity: Manifold

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

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)
  (Check for a cube: \( 6 - 12 + 8 = 2 \))
Topological Validity: Orientation Consistency

Both facing front

Inconsistent orientations

Non-orientable
Half-Edge Data Structure

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

```c
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 \(m\):

![](image)

• This time have to delete elements
• Again, many pointer reassignments
Global Mesh Operations: Geometry Processing

- Mesh subdivision
- Mesh simplification
- Mesh regularization
To Be Continued
Acknowledgments

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