Adjacency Data Structures

material from Justin Legakis

Assignment 1

• Questions?
• Creative Ideas?
• Having Fun?
• Mailing list

Last Time?

• Simple Transformations
• Classes of Transformations
• Representation
  – homogeneous coordinates
• Composition
  – not commutative

How do we compute Average Normals?

• Illusion of smooth surfaces by using the average normal
  (Gouraud Shading & Phong Normal Interpolation)

How do we compute Average Normals?

• Illusion of smooth surfaces by using the average normal
  (Gouraud Shading & Phong Normal Interpolation)

glShadeModel (GL_SMOOTH);

• From OpenGL Reference Manual:
  – Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized, typically assigning different colors to each resulting pixel fragment.
  – Flat shading selects the computed color of just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.
  – In either case, the computed color of a vertex is the result of lighting if lighting is enabled, or it is the current color at the time the vertex was specified if lighting is disabled.

Today

• Definitions
• Simple Data Structures
• Winged Edge Data Structure (Baumgart, 1975)
• HalfEdge Data Structure (Eastman, 1982)
• QuadEdge Data Structure (Guibas and Stolfi, 1985)
• FacetEdge Data Structure (Dobkin and Laszlo, 1987)
• SplitEdge Data Structure
• Corner Data Structure
Today

- Definitions
  - Well-Formed Surfaces
  - Orientable Surfaces
  - Computational Complexity
- Simple Data Structures
- Winged Edge Data Structure (Baumgart, 1975)
- HalfEdge Data Structure (Eastman, 1982)
- SplitEdge Data Structure
- Corner Data Structure
- QuadEdge Data Structure (Guibas and Stolfi, 1985)
- FacetEdge Data Structure (Dobkin and Laszlo, 1987)

Well-Formed Surfaces

- Components Intersect "Properly"
  - Faces are: disjoint, share single Vertex, or share 2 Vertices and the Edge joining them
  - Every edge is incident to exactly 2 vertices
  - Every edge is incident to exactly 2 faces
- Local Topology is "Proper"
  - Neighborhood of a vertex is homeomorphic to a disk (permits stretching and bending, but not tearing)
  - Called a 2-manifold
  - Boundaries: half-disk, "manifold with boundaries"
- Global Topology is "Proper"
  - Connected
  - Closed
  - Bounded

Orientable Surfaces?

Closed Surfaces and Refraction

- Original Teapot model is not "watertight": intersecting surfaces at spout & handle, no bottom, a hole at the spout tip, a gap between lid & base
- Requires repair before ray tracing with refraction

Computational Complexity

- Access Time
  - linear, constant time average case, or constant time?
  - requires loops/recursion/if?
- Memory
  - variable size arrays or constant size?
- Maintenance
  - ease of editing
  - ensuring consistency

Questions?
Today

• Definitions
• Simple Data Structures
  – List of Polygons
  – List of Edges
  – List of Unique Vertices & Indexed Faces:
    – Simple Adjacency Data Structure
• Winged Edge Data Structure (Baumgart, 1975)
• HalfEdge Data Structure (Eastman, 1982)
• SplitEdge Data Structure
• Corner Data Structure
• QuadEdge Data Structure (Guibas and Stolfi, 1985)
• FacetEdge Data Structure (Dobkin and Laszlo, 1987)

List of Polygons:

(3, -2, 5), (3, 6, 2), (-6, 2, 4)
(2, 2, 4), (0, -1, -2), (9, 4, 0), (4, 2, 9)
(1, 2, -2), (8, 8, 7), (-4, -5, 1)
(-8, 2, 7), (-2, 3, 9), (11, 2, -7)

List of Edges:

(3, 6, 2), (-6, 2, 4)
(2, 2, 4), (0, -1, -2)
(9, 4, 0), (4, 2, 9)
(3, 6, 2), (-6, 2, 4)
(-3, 0, -4), (7, -3, -4)

List of Unique Vertices & Indexed Faces:

Vertices:
-1, -1, -1
-1, -1, 1
-1, 1, -1
-1, 1, 1
1, 1, -1
1, 1, 1

Faces:
1 2 4 3
5 7 8 6
1 5 6 2
3 4 8 7
1 3 7 5
2 6 8 4

Problems with Simple Data Structures

• No Adjacency Information
• Linear-time Searches

Mesh Data

• So, in addition to:
  – Geometric Information (position)
  – Attribute Information (color, texture, temperature, population density, etc.)
• Let’s store:
  – Topological Information (adjacency, connectivity)
Simple Adjacency

- Each element (vertex, edge, and face) has a list of pointers to all incident elements
- Queries depend only on local complexity of mesh
- Data structures do not have fixed size
- Slow! Big! Too much work to maintain!

Today

- Definitions
- Simple Data Structures
- Winged Edge Data Structure (Baumgart, 1975)
- HalfEdge Data Structure (Eastman, 1982)
- SplitEdge Data Structure
- Corner Data Structure
- QuadEdge Data Structure (Guibas and Stolfi, 1985)
- FacetEdge Data Structure (Dobkin and Laszlo, 1987)

Winged Edge (Baumgart, 1975)

- Each edge stores pointers to 4 Adjacent Edges
- Vertices and Faces have a single pointer to one incident Edge
- Data Structure Size? Fixed
- How do we gather all faces surrounding one vertex? Messy, because there is no consistent way to order pointers

Questions?

Today

- Definitions
- Simple Data Structures
- Winged Edge Data Structure (Baumgart, 1975)
- HalfEdge Data Structure (Eastman, 1982)
- SplitEdge Data Structure
- Corner Data Structure
- QuadEdge Data Structure (Guibas and Stolfi, 1985)
- FacetEdge Data Structure (Dobkin and Laszlo, 1987)
HalfEdge (Eastman, 1982)

- Every edge is represented by two directed HalfEdge structures
- Each HalfEdge stores:
  - vertex at end of directed edge
  - symmetric half edge
  - face to left of edge
  - next points to the HalfEdge counter-clockwise around face on left
- Orientation is essential, but can be done consistently!

HalfEdge (Eastman, 1982)

- Starting at half edge HE, how do we find:
  - the other vertex of the edge?
  - the other face of the edge?
  - the clockwise edge around the face at the left?
  - all the edges surrounding the face at the left?
  - all the faces surrounding the vertex?

HalfEdge (Eastman, 1982)

- Loop around a Face:
  ```
  SplitEdgeMesh::FaceLoop(SplitEdge *HE) {
    SplitEdge *loop = HE;
    do {
      loop = loop->Next->Sym;
      } while (loop != HE);
  }
  ```

- Loop around a Vertex:
  ```
  SplitEdgeMesh::VertexLoop(SplitEdge *HE) {
    SplitEdge *loop = HE;
    do {
      loop = loop->Next;
      } while (loop != HE);
  }
  ```

Questions?

Today

- Definitions
- Simple Data Structures
- Winged Edge Data Structure (Baumgart, 1975)
- HalfEdge Data Structure (Eastman, 1982)
- SplitEdge Data Structure
- Corner Data Structure
- QuadEdge Data Structure (Guibas and Stolfi, 1985)
- FacetEdge Data Structure (Dobkin and Laszlo, 1987)
SplitEdge Data Structure:

• HalfEdge and SplitEdge are dual structures!

\[
\text{SplitEdgeMesh::FaceLoop() = HalfEdgeMesh::VertexLoop()}
\]
\[
\text{SplitEdgeMesh::VertexLoop() = HalfEdgeMesh::FaceLoop()}
\]

Corner Data Structure:

• The Corner data structure is its own dual!

Questions?

Today

• Definitions
• Simple Data Structures
• Winged Edge Data Structure (Baumgart, 1975)
• HalfEdge Data Structure (Eastman, 1982)
• SplitEdge Data Structure
• Corner Data Structure
• QuadEdge Data Structure (Guibas and Stolfi, 1985)
• FacetEdge Data Structure (Dobkin and Laszlo, 1987)

QuadEdge (Guibas and Stolfi, 1985)

• Consider the Mesh and its Dual simultaneously
  – Vertices and Faces switch roles, we just re-label them
  – Edges remain Edges
• Now there are eight ways to look at each edge
  – Four ways to look at primal edge
  – Four ways to look at dual edge

QuadEdge (Guibas and Stolfi, 1985)

• Relations Between Edges: Edge Algebra
• Elements in Edge Algebra:
  – Each of 8 ways to look at each edge

• Operators in Edge Algebra:
  – Rot: Bug rotates 90 degrees to its left
  – Sym: Bug turns around 180 degrees
  – Flip: Bug flips up-side down
  – Onext: Bug rotates CCW about its origin (either Vertex or Face)
QuadEdge (Guibas and Stolfi, 1985)

- Some Properties of Flip, Sym, Rot, and Onext:
  - $e \cdot \text{Rot}^4 = e$
  - $e \cdot \text{Rot}^2 \neq e$
  - $e \cdot \text{Flip} = e$
  - $e \cdot \text{Flip} \cdot \text{Rot} \cdot \text{Flip} = e$
  - $e \cdot \text{Rot} \cdot \text{Onext} = e$
  - $e \cdot \text{Onext} \cdot \text{Flip} = e$
  - $e \cdot \text{Flip}^{-1} = e \cdot \text{Flip}$
  - $e \cdot \text{Sym} = e \cdot \text{Rot}^2$
  - $e \cdot \text{Rot}^{-1} = e \cdot \text{Rot}^3$
  - $e \cdot \text{Onext}^{-1} = e \cdot \text{Onext} \cdot \text{Rot}$
  - $e \cdot \text{Onext}^{-1} = e \cdot \text{Flip}$

- Other Useful Definitions:
  - $e \cdot \text{Lnext} = e \cdot \text{Rot}^{-1} \cdot \text{Onext} \cdot \text{Rot}$
  - $e \cdot \text{Rnext} = e \cdot \text{Rot} \cdot \text{Onext} \cdot \text{Rot}^{-1}$
  - $e \cdot \text{Dnext} = e \cdot \text{Sym} \cdot \text{Onext} \cdot \text{Sym}^{-1}$
  - $e \cdot \text{Onext}^{-1} = e \cdot \text{Rot} \cdot \text{Onext} \cdot \text{Rot}$
  - $e \cdot \text{Dnext}^{-1} = e \cdot \text{Rot}^{-1} \cdot \text{Onext} \cdot \text{Rot}$

All of these functions can be expressed as a constant number of Rot, Sym, Flip, and Onext operations independent of the local topology and the global size and complexity of the mesh.

FacetEdge (Dobkin and Laszlo, 1987)

- QuadEdge (2D, surface) → FacetEdge (3D, volume)
- Faces → Polyhedra / Cells
- Edge → Polygon & Edge pair

Questions?

For Next Time:

- Read Hugues Hoppe “Progressive Meshes” SIGGRAPH 1996