Adjacency Data Structures

6.838J Meeting 3: Adjacency Data Structures

Justin Legakis

- **Introduction**
  1. Definitions
  2. Well-Formed Surfaces
  3. Subdividing a Domain
  4. Data
  5. Applications

- **Simple Data Structures for Meshes**
  1. Examples
  2. Problems

- **Adjacency - Navigating a Mesh**
  1. Simple Adjacency
  2. Third Kind of Data
  3. *Winged Edge* Data Structure (Baumgart, 1975)
  4. Another Look at Adjacency

- **HalfEdge** Data Structure (Eastman,
1982)

1. For Orientable Surfaces Only
2. Every Edge Represented by Two HalfEdge Structures
3. Three Pointers
4. Data
5. What can we do with this?
6. Time Complexity

- **QuadEdge Data Structure (Guibas and Stolfi, 1985)**
  1. Consider Mesh and its Dual at the Same Time
  2. Eight Ways to Look at an Edge
  3. Relations Between Elements
  4. One QuadEdge Structure per Edge, with 4 Parts
  5. "Data" Pointer
  6. "Onext" Pointer
  7. Four Parts to an Edge -> Four Loops
  8. Duality

- **FacetEdge Data Structure (Dobkin and Laszlo 1987)**
  - Demo

- **Queries**
  - Demo
● **SplitEdge and Corner Data Structures**

  ● Diagrams

● **Degeneracies**

  1. Boundaries
  2. Infinite Edges, Faces, and Cells
  3. Loops

This page created and maintained by Justin Legakis
legakis@graphics.lcs.mit.edu
Last updated: 2/15/98
I. Introduction

1. Definitions

- 1D, 2D, 3D spaces
- Curve, Surface Mesh, Spatial Decomposition
- Vertices, Edges, Faces, Cells

2. Well-Formed Surfaces

- Components Intersect "Properly"
  1. Faces are: disjoint, share single Vertex, or share 2 Vertices and the Edge joining them
  2. Every edge is incident to exactly 2 vertices
  3. Every edge is incident to exactly 2 faces

- Local Topology is "Proper"
  1. Neighborhood of a Vertex is homeomorphic to a disk (permits stretching and bending, but not tearing)
  2. Called a 2-manifold
  3. Boundaries: half-disk, "manifold with boundaries"
  4. Link of Vertex is a simple closed path

- Global Topology is "Proper"
  1. Connected
  2. Closed
  3. Bounded
3. **Subdividing a Domain**

   - Structured Subdivision
   - Unstructured Subdivision

4. **Data**

   - Geometric Information
Adjacency Data Structures - Introduction

1. Vertex Locations
2. Others?

- Attribute Information
  1. Color (at vertex or face)
  2. Temperature/Pressure (at vertex, face, or cell)
  3. Vegetation, Population, other statistics
  4. Others?

5. Applications

- Polygon Mesh for Efficient Object Representation
- Scientific Computation (fluid dynamics)
- Subdivision Surfaces
- Partitioning: Voronoi Diagram / Delaunay Triangulation

This page created and maintained by Justin Legakis
legakis@graphics.lcs.mit.edu
Last modified: 2/14/98
II. Simple Data Structures for Meshes

1. Data Structures

- List of Polygons:

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

- List of Edges:

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

- List of Unique Vertices + Indexed Faces:
Adjacency Data Structures - Simple Data Structures for Meshes

**Vertices:**
1. \((-1, -1, -1)\)
2. \((-1, -1, 1)\)
3. \((-1, 1, -1)\)
4. \((-1, 1, 1)\)
5. \((1, -1, -1)\)
6. \((1, -1, 1)\)
7. \((1, 1, -1)\)
8. \((1, 1, 1)\)

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

2. **Problems**
   - No Adjacency Information
   - Linear-Time Searches
III. Adjacency - Navigating a Mesh

1. Simple Adjacency
   - Vertex, Edge, and Face Structures
   - Each element has list of pointers to all incident elements
   - Queries depend only on local complexity of mesh!
   - Slow! Big! Too much work to maintain!
   - Data structures do not have fixed size

2. We've Added a Third Kind of Data
   - Geometric Information
   - Attribute Information
   - Topological Information

3. Winged Edge Data Structure (Baumgart, 1975)
   - Main Data Structure: Edge
   - 4 Adjacent Edges -> 4 Pointers
### Adjacency Data Structures - Navigating a Mesh

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Edge</th>
<th>Face</th>
</tr>
</thead>
</table>

- Vertices and Faces have a single pointer to one incident Edge
- Fixed size data structures
- No consistent way to order pointers

### 4. Another Look at Adjacency

- Four ways to look at an Edge (demo)

This page created and maintained by [Justin Legakis](mailto:legakis@graphics.lcs.mit.edu)
Last modified: 2/14/98
IV. HalfEdge Data Structure (Eastman, 1982)
"Doubly-Connected Edge List"

1. Orientable Surfaces Only (no Mobius Strips!)
   - Consider only 2 of the 4 ways to look at an edge
   - Pick the two on the "outside" of the mesh
   - Consistency is the key!
   - Local consistency everywhere implies global consistency

2. Every edge is represented by two HalfEdge structures
   - Each HalfEdge is a "directed edge"
   - Each HalfEdge is associated with exactly one vertex, edge, and face
     1. Vertex and end of directed edge
     2. Edge is obvious
     3. Face to left of edge
     4. Orientation is essential, but can be done consistently!
3. Three Pointers: Vertex, Sym, and Next

- **Vertex** points to Vertex at end of HalfEdge
- **Sym** points to symmetric HalfEdge
  1. Same Edge
  2. Opposite Vertex
  3. Opposite Face
  4. \( HE \rightarrow Sym \rightarrow Sym = HE \)
- **Next** points to HalfEdge Counter-Clockwise around Face on left
  1. Same Face
  2. CCW Vertex around Face on left
  3. CCW Edge around Face on left
  4. \( HE \rightarrow Next \rightarrow Face = HE \rightarrow Face \)
  5. \( HE \rightarrow Next \rightarrow Sym \rightarrow Vertex = HE \rightarrow Vertex \)

4. Data

- Geometric Information in Vertices only!
- Attribute Information in Vertices, HalfEdges, and/or Faces
- Topological Information in HalfEdges only!

5. What can we do with this?

- Loop around a Face:

```cpp
HalfEdgeMesh::FaceLoop(HalfEdge *HE) {
    HalfEdge *loop = HE;
    do {
        loop = loop->Next;
    } while (loop != HE);
}
```

- Loop around a Vertex:

```cpp
HalfEdgeMesh::VertexLoop(HalfEdge *HE) {
    HalfEdge *loop = HE;
```
do {
    loop = loop->Next->Sym;
} while (loop != HE);

6. **Time Complexity**

- Time is linear in the *amount of information gathered*
- Independent of global complexity!
- We will re-visit queries later
IV. *QuadEdge* Data Structure (Guibas and Stolfi, 1985)

1. 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
- Bug Demo (reprise)

**Relations Between Edges: Edge Algebra**

**Elements** in Edge Algebra:
1. Each of 8 ways to look at each edge

**Operators** in Edge Algebra:
1. Rot: Bug rotates 90 degrees to its right
2. Sym: Bug turns around 180 degrees
3. Flip: Bug flips up-side down
4. Onext: Bug rotates CCW about its origin (either Vertex or Face)

**Some Properties of Flip, Rot, and Onext:**
1. $e \text{ Rot}^4 = e$
2. $e \text{ Rot}^2 \neq e$
3. $e \text{ Flip}^2 = e$
4. $e \text{ Flip Rot Flip Rot} = e$
5. $e \text{ Rot Flip Rot Flip} = e$
6. $e \text{ Rot Onext Rot Onext} = e$
7. $e \text{ Flip Onext Flip Onext} = e$

**Properties** of Edge Algebra deduced from those above:
1. $e \text{ Flip}^{-1} = e \text{ Flip}$
2. $e \text{ Sym} = e \text{ Rot}^2$
3. $e \text{ Rot}^{-1} = e \text{ Rot}^3$
4. $e \text{ Rot}^{-1} = e \text{ Flip Rot Flip}$
5. \( e \text{ Onext}^{-1} = e \text{ Rot} \text{ Onext} \text{ Rot} \)
6. \( e \text{ Onext}^{-1} = e \text{ Flip} \text{ Onext} \text{ Flip} \)

- Other Useful Definitions:
  1. \( e \text{ Lnext} = e \text{ Rot}^{-1} \text{ Onext} \text{ Rot} \)
  2. \( e \text{ Rnext} = e \text{ Rot} \text{ Onext} \text{ Rot}^{-1} \)
  3. \( e \text{ Dnext} = e \text{ Sym} \text{ Onext} \text{ Sym}^{-1} \)

- Even More Useful Definitions:
  1. \( e \text{ Oprev} = e \text{ Onext}^{-1} = e \text{ Rot} \text{ Onext} \text{ Rot} \)
  2. \( e \text{ Lprev} = e \text{ Lnext}^{-1} = e \text{ Onext} \text{ Sym} \)
  3. \( e \text{ Rprev} = e \text{ Rnext}^{-1} = e \text{ Sym} \text{ Onext} \)
  4. \( e \text{ Dprev} = e \text{ Dnext}^{-1} = e \text{ Rot}^{-1} \text{ Onext} \text{ Rot}^{-1} \)

**NOTE:** Every function defined so far can be expressed as a constant number of Rot, Flip, and Onext operations! This is independent of the local topology and the global size and complexity of the mesh.

- One QuadEdge Structure per Edge, with 4 Parts
  - Pick 4 of the 8 Edge elements on a single side
  - Can represent flipped versions with an extra bit
  - Each part has 2 pointers, "Data" and "Onext"
  - Demo

- "Data" Pointer
  - Geometric and Attribute Information
  - Application Specific
  - No Effect on Adjacency Operations

- "Onext" Pointer
• Pointer to \textit{Onext} Edge element
• CCW element around \textit{Origin} of edge (either a Vertex or a Face)

\textbf{. Four Parts to an Edge -> Four Loops}

• Each element is a node in 1 of 4 linked lists
• Loop around Face on both sides of Edge
• Loop around Vertex on both ends of Edge

\textbf{. Duality}

• The QuadEdge structure represents both the mesh and its dual simultaneously
• Therefore, computing the dual is easy:

\begin{verbatim}
QuadEdgeMesh::ComputeDual()
{
}
\end{verbatim}
VIII. *SplitEdge* and *Corner* Data Structures

1. **HalfEdge** Data Structure:

2. **SplitEdge** Data Structure:
3. Face and Vertex Loops:

- 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);

4. **HalfEdge/SplitEdge Duality:**
   - HalfEdgeMesh::VertexLoop() = SplitEdgeMesh::FaceLoop()
   - HalfEdgeMesh::FaceLoop() = SplitEdgeMesh::VertexLoop()
   - HalfEdge and SplitEdge are dual structures!

5. **Corner Data Structure:**

![Corner Data Structure Diagram]
6. **Corner Duality:**

   - The Corner data structure is its own dual!
Index of /legakis

<table>
<thead>
<tr>
<th>Name</th>
<th>Last_modified</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Directory</td>
<td>14-Sep-2005 21:17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Apache/1.3.26 Server at people.csail.mit.edu Port 80