Adjacency Data Structures

material from Justin Legakis

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

Today
• Orthographic & Perspective Projections
• OpenGL Basics
• Averaging Vertex Colors & Normals
• Surface Definitions
• Simple Data Structures
• Fixed Storage Data Structures
• Fixed Computation Data Structures

Orthographic vs. Perspective
• Orthographic
• Perspective

Simple Orthographic Projection
• Project all points along the $z$ axis to the $z = 0$ plane

\[
\begin{pmatrix}
  x \\
  y \\
  0 \\
  1
\end{pmatrix} = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
\]

Simple Perspective Projection
• Project all points along the $z$ axis to the $z = d$ plane, eyepoint at the origin:

\[
\begin{pmatrix}
  x' \\
  y' \\
  w
\end{pmatrix} = \begin{pmatrix}
  \frac{d \cdot x}{z} & \frac{d \cdot y}{z} & 0 \\
  z/d & 0 & 0 \\
  0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
  x \\
  y \\
  d \\
  1
\end{pmatrix}
\]

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix} = \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 1/d & 0
\end{pmatrix} \begin{pmatrix}
  x' \\
  y' \\
  w
\end{pmatrix}
\]
Alternate Perspective Projection

- Project all points along the z axis to the $z = 0$ plane, eyepoint at the $(0,0,-d)$:

$$\begin{pmatrix} x_p \frac{d - z}{z + d} - \frac{x}{u} \frac{d - z}{z + d} + 1 \\ y_p \frac{d - z}{z + d} - \frac{y}{v} \frac{d - z}{z + d} + 1 \\ (z + d)/d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

In the limit, as $d \to \infty$

this perspective projection matrix...

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

...is simply an orthographic projection

Questions?

Today

- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures

OpenGL Basics: GL_POINTS

```glDisable(GL_LIGHTING);
glBegin(GL_POINTS);
glColor3f(0.0,0.0,0.0);
glVertex3f(…);
glEnd();
```

- lighting should be disabled...

OpenGL Basics: GL_QUADS

```glEnable(GL_LIGHTING);
glBegin(GL_QUADS);
glNormal3f(…);
glColor3f(1.0,0.0,0.0);
glVertex3f(…);
glVertex3f(…);
glVertex3f(…);
glVertex3f(…);
glEnd();
```

- lighting should be enabled...
- an appropriate normal should be specified
OpenGL Basics: Transformations

- Useful commands:
  - `glMatrixMode(GL_MODELVIEW);`
  - `glPushMatrix();`
  - `glPopMatrix();`
  - `glMultMatrixf(...);`

From OpenGL Reference Manual

Questions?

Today

- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures

Color Interpolation

- Interpolate colors of the 3 vertices
- Linear interpolation, barycentric coordinates

```
glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(...);
glColor3f(0.0, 1.0, 0.0);
glVertex3f(...);
glColor3f(0.0, 0.0, 1.0);
glVertex3f(...);
glEnd();
```

```
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.

```
glBegin(GL_TRIANGLES);
glNormal3f(...);
glVertex3f(...);
glNormal3f(...);
glVertex3f(...);
glNormal3f(...);
glVertex3f(...);
glEnd();
```

Normal Interpolation
Gouraud Shading

- Instead of shading with the normal of the triangle, we’ll shade the vertices with the average normal and interpolate the shaded color across each face

- How do we compute Average Normals? Is it expensive??

Questions?

Today

- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
  - Well-Formed Surfaces
  - Orientable Surfaces
  - Computational Complexity
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures

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

- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- **Simple Data Structures**
  - List of Polygons
  - List of Edges
  - List of Unique Vertices & Indexed Faces:
    - Simple Adjacency Data Structure
- Fixed Storage Data Structures
- Fixed Computation Data Structures

List of Polygons:

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

List of Edges:

\[(3, 6, 2), (-6, 2, 4)\]
\[(2, 2, 4), (0, -1, -2)\]
\[(9, 4, 0), (4, 2, 9)\]
\[(-8, 2, 7), (1, 2, -7)\]
\[(3, 0, -3), (-7, 4, -3)\]
\[(9, 4, 0), (4, 2, 9)\]
\[(3, 6, 2), (-6, 2, 4)\]
\[(-3, 0, -4), (7, 3, -4)\]

List of Unique Vertices & Indexed Faces:

**Vertices:**
\[(-1, -1, 1)\]
\[(-1, -1, -1)\]
\[(1, 1, 1)\]
\[(-1, 1, 1)\]
\[(-1, -1, -1)\]

**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!

Questions?

Today
- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
  - Winged Edge (Baumgart, 1975)
- Fixed Computation Data Structures

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

- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
  - HalfEdge (Eastman, 1982)
  - SplitEdge
  - Corner
  - QuadEdge (Guibas and Stolfi, 1985)
  - FacetEdge (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!

• Loop around a Face:
  HalfEdgeMesh:FaceLoop(HalfEdge *HE) {
    HalfEdge *loop = HE;
    do {
      loop = loop->Next;
    } while (loop != HE);
  }

• Loop around a Vertex:
  HalfEdgeMesh:VertexLoop(HalfEdge *HE) {
    HalfEdge *loop = HE;
    do {
      loop = loop->Next->Sym;
    } while (loop != HE);
  }

HalfEdge (Eastman, 1982)

- Data Structure Size?
  Fixed

- Data:
  - geometric information stored at Vertices
  - attribute information in Vertices, HalfEdges, and/or Faces
  - topological information in HalfEdges only!
- Orientable surfaces only (no Mobius Strips!)
- Local consistency everywhere implies global consistency
- Time Complexity?
  linear in the amount of information gathered
**SplitEdge Data Structure:**
- HalfEdge and SplitEdge are dual structures!
- SplitEdge\texttt{Mesh::FaceLoop()} = HalfEdge\texttt{Mesh::VertexLoop()}
- SplitEdge\texttt{Mesh::VertexLoop()} = HalfEdge\texttt{Mesh::FaceLoop()}

**Corner Data Structure:**
- The Corner data structure is its own dual!

**Questions?**

**Today**
- Orthographic & Perspective Projections
- OpenGL Basics
- Averaging Vertex Colors & Normals
- Surface Definitions
- Simple Data Structures
- Fixed Storage Data Structures
- Fixed Computation Data Structures
  - HalfEdge (Eastman, 1982)
  - SplitEdge
  - Corner
  - QuadEdge (Guibas and Stolfi, 1985)
  - FacetEdge (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:
  - $e \cdot \text{Rot}^4 = e$
  - $e \cdot \text{Rot}^2 \neq e$
  - $e \cdot \text{Flip}^2 = e$
  - $e \cdot \text{Flip} \cdot \text{Rot} \cdot \text{Rot} = e$
  - $e \cdot \text{Rot} \cdot \text{Onext} \cdot \text{Rot} \cdot \text{Onext} = e$
  - $e \cdot \text{Flip} \cdot \text{Onext} \cdot \text{Flip} \cdot \text{Onext} = e$
  - $e \cdot \text{Flip}^{-1} = \text{Flip}$
  - $e \cdot \text{Sym} = e \cdot \text{Rot}^2$
  - $e \cdot \text{Rot}^{-1} = e \cdot \text{Rot}^3$
  - $e \cdot \text{Rot}^{-1} = e \cdot \text{Flip} \cdot \text{Rot} \cdot \text{Flip}$
  - $e \cdot \text{Onext}^{-1} = e \cdot \text{Rot} \cdot \text{Onext} \cdot \text{Rot}$
  - $e \cdot \text{Onext}^{-1} = e \cdot \text{Flip} \cdot \text{Onext} \cdot \text{Flip}$

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

- 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