### Transformation Hierarchies

**Wolfgang Heidrich**

---

### Course News

**Assignment 1**
- Due January 31

**Homework 2**
- Exercise problems for perspective
- Discussed in labs next week

**Reading**
- Chapter 7 (new book) or 6 (old book)

---

### The Rendering Pipeline

---

### Rendering Geometry in OpenGL

**Example:**
```c
glBegin( GL_TRIANGLES );
glColor3f( 1.0, 0.0, 0.0 );
glVertex3f( 1.0, 0.0, 0.0 );
glColor3f( 0.0, 0.0, 1.0 );
glVertex3f( 0.0, 1.0, 0.0 );
glVertex3f( 0.0, 0.0, 0.0 );
glEnd();
```

---

### Recap: Rendering Geometry in OpenGL

**Additional attributes**
- `glColor3f`: RGB color value (0...1 per component)
- `glNormal3f`: normal vector
- `glTexCoord2f`: texture coordinate (explained later)

**OpenGL is state machine:**
- Every vertex gets color, normal etc. that corresponds to last specified value

---

### Recap: Interpreting Composite OpenGL Transformations

**Example for earlier lectures:**
- Rotation around arbitrary center
  - In OpenGL:
    ```c
    // initialization of matrix
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    
    // Top-to-bottom:
    glTranslatef ( 4, 3 );
    glRotatef ( 30, 0.0, 0.0, 1.0 );
    glTranslatef ( -4, -3 );
    
    glBegin( GL_TRIANGLES );
    // specify object geometry...
    
    // Bottom-to-top:
    
    glEnd();
    ```

---
Transformation Hierarchies

Scene may have a hierarchy of coordinate systems
Stores matrix at each level with incremental transform from parent’s coordinate system

Transformation Hierarchy Example 1

Transformation Hierarchies

Hierarchies don’t fall apart when changed transforms apply to graph nodes beneath

Transformation Hierarchy Example 2

Draw same 3D data with different transformations: instancing

Matrix Stacks

Challenge of avoiding unnecessary computation
- Using inverse to return to origin
- Computing incremental $T_1 \rightarrow T_2$

Brown Applets

http://www.cs.brown.edu/exploratories/freeSoftware/catalogs/scenegraphs.html

- Have a look later
Matrix Stacks

<table>
<thead>
<tr>
<th>glPushMatrix()</th>
<th>D = C scale(3,2,2) translate(1,2,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>glPopMatrix()</td>
<td>D = C scale(3,2,2) translate(1,2,3)</td>
</tr>
</tbody>
</table>

DrawSquare()

---

Modularization

Drawing a scaled square

- Push/pop ensures no coord system change

```cpp
void drawBlock(float k) {
    glPushMatrix();
    glScalef(k, k, k);
    glBegin(GL_LINES); loop;
    glEnd();
    glPopMatrix();
}
```

Transformation Hierarchy

Example 3

```cpp
glLoadIdentity();
glTranslatef(4, 1, 0);  // Translate to (4, 1, 0)
glPushMatrix();
    glTranslatef(2, 1, 1);  // Translate to (6, 2, 1)
    glScalef(2, 1, 1);  // Scale by 2 in x-axis
    glTranslatef(1, 0, 0);  // Translate to (7, 2, 1)
    glPopMatrix();
glPopMatrix();
```

---

Matrix Stacks

- Advantages
  - No need to compute inverse matrices all the time
  - Modularize changes to pipeline state
  - Avoids incremental changes to coordinate systems
  - Accumulation of numerical errors

- Practical issues
  - In graphics hardware, depth of matrix stacks is limited
    - Typically 16 for model/view and about 4 for projective matrix

---

Transformation Hierarchy

Example 4

```cpp
// Draw square
// Draw filled square
// Draw filled triangle
// Draw filled polygon
```

---

Hierarchical Modeling

- Advantages
  - Define object once, instantiate multiple copies
  - Transformation parameters often good control knobs
  - Maintain structural constraints if well-designed

- Limitations
  - Expressivity: not always the best controls
  - Can't do closed kinematic chains
  - Keep hand on hip
Single Parameter: simple

Parameters as functions of other params
- Clock: control all hands with seconds s

\[
\begin{align*}
m &= s / 60, \\
h &= m / 60, \\
\theta_{s} &= (2 \pi s) / 60, \\
\theta_{m} &= (2 \pi m) / 60, \\
\theta_{h} &= (2 \pi h) / 60
\end{align*}
\]

Single Parameter: complex

Mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk

Display Lists

Representing Complex Geometry

Wolfgang Heidrich

Display Lists

Advantages:
- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth!)
- Display lists exist across multiple frames
  - Represent static objects in an interactive application

Shared Vertices

Triangle Meshes

- Multiple triangles share vertices
- If individual triangles are sent to graphics board, every vertex is sent and transformed multiple times!
  - Computational expense
  - Bandwidth

\[
\begin{align*}
&1, 3, 4, 7, 9, 10, 13, 15, 16, 19, 21, 22 \\
&2, 5, 6, 8, 11, 12, 14, 17, 18, 20, 23, 25
\end{align*}
\]
**Triangle Strips**

**Idea:**
- Encode neighboring triangles that share vertices
- Use an encoding that requires only a constant-sized part of the whole geometry to determine a single triangle

N triangles need n+2 vertices

**Orientation:**
- Strip starts with a counter-clockwise triangle
- Then alternates between clockwise and counter-clockwise

---

**Triangle Fans**

**Similar concept:**
- All triangles share one center vertex
- All other vertices are specified in CCW order

---

**Triangle Strips and Fans**

**Transformations:**
- n+2 for n triangles
- Only requires 3 vertices to be stored according to simple access scheme
- Ideal for pipeline (local knowledge)

**Generation**
- E.g. from directed edge data structure
- Optimize for longest strips/fans

---

**Vertex Arrays**

**Concept:**
- Store array of vertex data for meshes with arbitrary connectivity (topology)

```c
GLfloat *points[3*vertices];
GLfloat *colors[3*vertices];
GLuint *tris[numtris] =
{0,1,3, 1,2,4, ...};
glVertexPointer(3, points);
glColorPointer(3, colors);
glDrawElements(
    GL_TRIANGLES, ..., tris);
```

**Benefits:**
- Ideally, vertex array fits into memory on GPU
- Then all vertices are transformed exactly once

**In practice:**
- Graphics memory may not be sufficient to hold model
- Then either:
  - Cache only parts of the vertex array on board (may lead to cache thrashing)
  - Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex!
**The Rendering Pipeline**

- Geometry Database
- Model/View Transform
- Perspective Transform
- Clipping
- Lighting
- Frame-buffer
- Scan Conversion
- Texturing
- Depth Test
- Blending
- Rasterization
- Fragment Processing

---

**Coming Up:**

**This Week:**
- Perspective projection