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

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Scan Conversion → Texturing → Depth Test → Blending → Frame-buffer

Rasterization → Fragment Processing

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: transformation of coordinate frame
glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );

// bottom-to-top: transformation of object
glBegin( GL_TRIANGLES );
// specify object geometry...
```
Transformation Hierarchies

Scene may have a hierarchy of coordinate systems

Stores matrix at each level with incremental transform from parent's coordinate system

Scene graph

Transformation Hierarchy Example 1

world

head

trans(0.3,0,0) rot(z,θ)
**Transformation Hierarchies**

- Hierarchies don't fall apart when changed
- Transforms apply to graph nodes beneath

**Brown Applets**

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

- Have a look later
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 $T_1 \rightarrow T_2$
Matrix Stacks

```
glPushMatrix()
glPopMatrix()
```

```
D = C scale(2,2,2) trans(1,0,0)
```

```
DrawSquare()
glPushMatrix()
gScale3f(2,2,2)
gTranslate3f(1,0,0)
DrawSquare()
glPopMatrix()
```

Modularization

**Drawing a scaled square**

- Push/pop ensures no coord system change

```c
void drawBlock(float k) {
    glPushMatrix();
    glScalef(k,k,k);
    glBegin(GL_LINE_LOOP);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(1,1,0);
    glVertex3f(0,1,0);
    glEnd();
    glPopMatrix();
}
```

Wolfgang Heidrich
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 3

```c
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glTranslatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslatef(1,0,0);
glPopMatrix();
```
Transformation Hierarchy
Example 4

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

\[ m = s/60, \quad h = m/60, \]
\[ \theta_s = (2 \pi s) / 60, \]
\[ \theta_m = (2 \pi m) / 60, \]
\[ \theta_h = (2 \pi h) / 60 \]

---

Single Parameter: complex

Mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk
Representing Complex Geometry

Wolfgang Heidrich

Display Lists

Concept:
- If multiple copies of an object are required, it can be compiled into a display list:
  
  ```
  glNewList( listId, GL_COMPILE );
  glBegin( ... );
  ... // geometry goes here
  glEndList();
  // render two copies of geometry offset by 1 in z-direction:
  glCallList( listId );
  glTranslatef( 0.0, 0.0, 1.0 );
  glCallList( listId );
  ```
**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*
**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

![Diagram showing the idea of triangle strips]

---

**Triangle Strips**

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

![Diagram showing the orientation of triangle strips]
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

Stripification by Dana Sharon
Vertex Arrays

Concept:
- Store array of vertex data for meshes with arbitrary connectivity (topology)
  
  ```c
  GLfloat *points[3*nvertices];
  GLfloat *colors[3*nvertices];
  Glint *tris[numtris]=
    {0,1,3, 3,2,4, ...};
  glVertexPointer( ..., points );
  glColorPointer( ...,colors );
  glDrawElements( GL_TRIANGLES,...,tris );
  ```

Vertex Arrays

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
  - Cache only parts of the vertex array on board (may lead to cache trashing!)
  - 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. -> Lighting -> Perspective Transform. -> Clipping

Geometry Processing

Scan Conversion -> Texturing -> Depth Test -> Blending -> Frame-buffer

Rasterization -> Fragment Processing

Coming Up:

This Week:

- Perspective projection