Transformation Hierarchies

Wolfgang Heidrich

Course News

Assignment 1
- Due February 2

Homework 1
- Exercise problems for transformations
- Discussed in labs next week

Reading (this week)
- Chapter 5

Reading (next week)
- Chapter 6
Recap: Rendering Geometry in OpenGL

Example:
```
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();

glTranslatef( 4, 3 );
glRotatef( 30, 0.0, 0.0, 1.0 );
glTranslatef( -4, -3 );
glBegin( GL_TRIANGLES );
// specify object geometry...
```

Top-to-bottom: transformation of coordinate frame
Bottom-to-top: transformation of object

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:
```
road

stripe1 stripe2 ... car1 car2 ...

w1 w2 w3 w4
```
Transformation Hierarchy Example

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 $T_1 \rightarrow T_2$

![Diagram showing matrix transformations and world coordinates](Image)

Matrix Stacks

```cpp
glPushMatrix();
glPopMatrix();
```

$D = C \text{ scale}(2,2,2) \text{ trans}(1,0,0)$

```
DrawSquare();
glPushMatrix();
glScale3f(2,2,2);
glTranslatef3f(1,0,0);
DrawSquare();
glPopMatrix();
```

Wolfgang Heidrich
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();
}
```

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();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslatef(1,0,0);
glPopMatrix();
```

Transformation Hierarchy
Example 4

```c
glTranslatef(x,y,0);
glRotatef(\theta_1,0,0,1);
DrawBody();
glPushMatrix();
glTranslatef(0,7,0);
DrawHead();
glPopMatrix();
glPopMatrix();
glPushMatrix();
glTranslatef(2.5,5,5,0);
glRotatef(\theta_2,0,0,1);
DrawUArm();
glTranslatef(0,-3.5,0);
glRotatef(\theta_3,0,0,1);
DrawLArm();
glPopMatrix();
glPopMatrix();
... (draw other arm)
```
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 $s$

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

**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
**Triangle Strips**

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

![Triangle Strips Diagram](image)

---

**Triangle Fans**

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

![Triangle Fans Diagram](image)
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*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
- Then either:
  - 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

1. **Geometry Processing**
   - Geometry Database
   - Model/View Transform.
   - Lighting
   - Perspective Transform.
   - Clipping

2. **Fragment Processing**
   - Scan Conversion
   - Texturing
   - Depth Test
   - Blending
   - Frame-buffer

Wolfgang Heidrich
Coming Up:

Next Week:
- Perspective projection
- Lighting/shading