Transformations IV

Week 3, Wed Jan 23

Readings for Jan 16-25

- FCG Chap 6 Transformation Matrices
  - except 6.1.6, 6.3.1
- FCG Sect 13.3 Scene Graphs
- RB Chap Viewing
  - Viewing and Modeling Transforms until Viewing Transformations
  - Examples of Composing Several Transformations through Building an Articulated Robot Arm
- RB Appendix Homogeneous Coordinates and Transformation Matrices
  - until Perspective Projection
- RB Chap Display Lists
Review: General Transform Composition

- transformation of geometry into coordinate system where operation becomes simpler
  - typically translate to origin

- perform operation

- transform geometry back to original coordinate system
Review: Arbitrary Rotation

- arbitrary rotation: change of basis
  - given two orthonormal coordinate systems \( XYZ \) and \( ABC \)
  - transformation from one to the other is matrix \( R \) whose columns are \( A, B, C \):

\[
R(X) = \begin{bmatrix}
a_x & b_x & c_x & 0 \\
a_y & b_y & c_y & 0 \\
a_z & b_z & c_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} = (a_x, a_y, a_z, 1) = A
\]
Review: 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
Review: Transformation Hierarchies

- demo:
  - 1. all scene graph parts would be on top of each other if translation set to 0 everywhere
  - 2. composition of transformations can be surprising and tricky even with just a few simple building blocks
  - 3. negative scale is a reflection

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

- challenge of avoiding unnecessary computation
  - using inverse to return to origin
  - computing incremental $T_1 \rightarrow T_2$

![Diagram showing object coordinates and world coordinates with transformations $T_1(x)$, $T_2(x)$, and $T_3(x)$]
Matrix Stacks

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

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

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

\[ \text{glPushMatrix} \]
\[ \text{glPopMatrix} \]

\[ \text{DrawSquare}() \]
\[ \text{glPushMatrix}() \]
\[ \text{glScale3f}(2,2,2) \]
\[ \text{glTranslate3f}(1,0,0) \]
\[ \text{DrawSquare}() \]
\[ \text{glPopMatrix}() \]
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(0,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,0,0);
  glVertex3f(1,1,0);
  glVertex3f(1,1,0);
  glVertex3f(0,1,0);
  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
GLfloat glLoadIdentity();
GLfloat glTranslatef(4,1,0);
GLfloat glPushMatrix();
GLfloat glRotatef(45,0,0,1);
GLfloat glTranslatef(0,2,0);
GLfloat glScalef(2,1,1);
GLfloat glTranslate(1,0,0);
GLfloat glPopMatrix();
```
Transformation Hierarchy Example 4

glTranslate3f(x,y,0);
glRotatef(\theta_1,0,0,1);
DrawBody();
glPushMatrix();
  glTranslate3f(0,7,0);
  DrawHead();
  glPopMatrix();
glPushMatrix();
  glTranslate(2.5,5.5,0);
  glRotatef(\theta_2,0,0,1);
  DrawUArm();
  glTranslate(0,-3.5,0);
  glRotatef(\theta_3,0,0,1);
  DrawLArm();
  glPopMatrix();
... (draw other arm)
Hierarchical Modelling

• 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
  • can’t do other constraints
    • collision detection
      • self-intersection
      • walk through walls
Display Lists
Display Lists

• precompile/cache block of OpenGL code for reuse
  • usually more efficient than immediate mode
    • exact optimizations depend on driver
  • good for multiple instances of same object
    • but cannot change contents, not parametrizable
  • good for static objects redrawn often
    • display lists persist across multiple frames
    • interactive graphics: objects redrawn every frame from new viewpoint from moving camera
• can be nested hierarchically
• snowman example
  http://www.lighthouse3d.com/opengl/displaylists
void drawSnowMan() {
    
    glColor3f(1.0f, 1.0f, 1.0f);

    // Draw Body
    glTranslatef(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f, 20, 20);

    // Draw Head
    glTranslatef(0.0f, 1.0f, 0.0f);
    glutSolidSphere(0.25f, 20, 20);

    // Draw Nose
    glPushMatrix();
    glColor3f(1.0f, 0.5f, 0.5f);
    glRotatef(0.0f, 1.0f, 0.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10, 2);
    glPopMatrix();

    // Draw Eyes
    glPushMatrix();
    glColor3f(0.0f, 0.0f, 0.0f);
    glTranslatef(0.05f, 0.10f, 0.18f);
    glutSolidSphere(0.05f, 10, 10);
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f, 10, 10);
    glPopMatrix();
}
Instantiate Many Snowmen

// Draw 36 Snowmen
for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
        glPushMatrix();
        glTranslatef(i*10.0, 0, j * 10.0);
        // Call the function to draw a snowman
        drawSnowMan();
        glPopMatrix();
    }

36K polygons, 55 FPS
Making Display Lists

```c
GLuint createDL() {
    GLuint snowManDL;
    // Create the id for the list
    snowManDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    drawSnowMan();
    glEndList();
    return(snowManDL); }

snowmanDL = createDL();
for(int i = -3; i < 3; i++)
    for(int j=-3; j < 3; j++) {
        glPushMatrix();
        glTranslatef(i*10.0, 0, j * 10.0);
        glCallList(Dlid);
        glPopMatrix(); }
```

36K polygons, 153 FPS
Transforming Normals
Transforming Geometric Objects

- lines, polygons made up of vertices
  - transform the vertices
  - interpolate between
- does this work for everything? no!
  - normals are trickier
Computing Normals

- normal
  - direction specifying orientation of polygon
    - $w=0$ means direction with homogeneous coords
    - vs. $w=1$ for points/vectors of object vertices
  - used for lighting
    - must be normalized to unit length
  - can compute if not supplied with object

\[ N = (P_2 - P_1) \times (P_3 - P_1) \]
Transforming Normals

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  0
\end{bmatrix}
= \begin{bmatrix}
  m_{11} & m_{12} & m_{13} & T_x \\
  m_{21} & m_{22} & m_{23} & T_y \\
  m_{31} & m_{32} & m_{33} & T_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  0
\end{bmatrix}
\]

- so if points transformed by matrix \( \mathbf{M} \), can we just transform normal vector by \( \mathbf{M} \) too?
  - translations OK: \( w=0 \) means unaffected
  - rotations OK
  - uniform scaling OK

- these all maintain direction
Transforming Normals

- nonuniform scaling does not work
- $x-y=0$ plane
  - line $x=y$
  - normal: $[1, -1, 0]$
    - direction of line $x=-y$
    - (ignore normalization for now)
Transforming Normals

• apply nonuniform scale: stretch along x by 2
  • new plane x = 2y
• transformed normal: [2,-1,0]

\[
\begin{bmatrix}
2 \\
-1 \\
0 \\
0
\end{bmatrix}
=\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 \\
-1 \\
0 \\
0
\end{bmatrix}
\]

• normal is direction of line x = -2y or x+2y=0
• not perpendicular to plane!
• should be direction of 2x = -y
Planes and Normals

- plane is all points perpendicular to normal
  - $N \cdot P = 0$ (with dot product)
  - $N^T \cdot P = 0$ (matrix multiply requires transpose)

\[
N = \begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix},\quad P = \begin{bmatrix}
x \\
y \\
z \\
w \\
\end{bmatrix}
\]

- explicit form: plane = $ax + by + cz + d$
Finding Correct Normal Transform

• transform a plane

\[
\begin{align*}
P, & \\
N, & \\
N' = QN, & \\
N^T P' = 0, & \\
(QN)^T (MP) = 0, & \\
N^T Q^T MP = 0, & \\
Q^T M = I & \\
Q = (M^{-1})^T & \\
\end{align*}
\]

given M, what should Q be?

stay perpendicular

substitute from above

thus the normal to any surface can be transformed by the inverse transpose of the modelling transformation

\[
(AB)^T = B^T A^T
\]

\[
N^T P = 0 \text{ if } Q^T M = I
\]