Transformations III

Week 3, Wed Jan 19

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005
Corrections to Previous Slides
Composing Transformations

- translation

\[
T_1 = T(dx_1, dy_1) = \begin{bmatrix} 1 & dx_1 \\ 1 & dy_1 \\ 1 & 1 \end{bmatrix} \quad T_2 = T(dx_2, dy_2) = \begin{bmatrix} 1 & dx_2 \\ 1 & dy_2 \\ 1 & 1 \end{bmatrix}
\]

\[
P'' = T_2 \cdot P' = T_2 \cdot [T_1 \cdot P] = [T_2 \cdot T_1] \cdot P, \text{ where}
\]

\[
T_2 \cdot T_1 = \begin{bmatrix} 1 & dx_1 + dx_2 \\ 1 & dy_1 + dy_2 \\ 1 & 1 \end{bmatrix}
\]

so translations add
Arbitrary Rotation

- **problem:**
  - given two orthonormal coordinate systems $XYZ$ and $UVW$
  - find transformation from one to the other
- **answer:**
  - transformation matrix $R$ whose columns are $U, V, W$:

\[
R = \begin{bmatrix}
  u_x & v_x & w_x \\
  u_y & v_y & w_y \\
  u_z & v_z & w_z 
\end{bmatrix}
\]
Arbitrary Rotation

- why?

$$R(X) = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = (u_x, u_y, u_z) = U$$

- similarly $$R(Y) = V \& R(Z) = W$$
Project 1

- out today, due 6pm Thu Jan 27
  - you should start very soon!
- labs are for help with programming projects, written assignments
  - no separate materials to be handed in
- build kangaroo out of cubes and 4x4 matrices
  - think cartoon, not beauty
  - build then animate one section at a time
    - helps make sure you’re constructing hierarchy correctly
- template code gives you program shell
Real Kangaroos
Articulated Kangaroo
Articulated Kangaroo
Review: Composing Transforms

- order matters
  - 4x4 matrix multiplication not commutative!

- moving to origin
  - transformation of geometry into coordinate system where operation becomes simpler
  - perform operation
  - transform geometry back to original coordinate system
Review: Interpreting Transformations

\[ p' = TRp \]

translate by (-1,0)

right to left: moving object

left to right: changing coordinate system

intuitive?

OpenGL
Review: Transforming Normals

- nonuniform scale makes normal nonperpendicular
  - need to use inverse transpose matrix instead
Transformation Hierarchies

- scene graph

```
+---+     +---+     +---+
|   | -->  |   | -->  |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```

```
+---+     +---+     +---+
|   |     |   |     |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```

```
+---+     +---+     +---+
|   |     |   |     |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```

```
+---+     +---+     +---+
|   |     |   |     |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```

```
+---+     +---+     +---+
|   |     |   |     |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```

```
+---+     +---+     +---+
|   |     |   |     |   |
+---+     +---+     +---+
    |     |     |     |
    |     |     |     |
    v     v     v     v
```
Transformation Hierarchies

trans(0.30,0,0) rot(z,\theta)
Matrix Stacks

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

![Diagram showing matrix stacks and transformations between object and world coordinates.](image)
Transformation Hierarchies

- matrix stack

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

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

\[ \text{glTranslate3f}(1,0,0) \]
\[ \text{glScale3f}(2,2,2) \]
\[ \text{glPushMatrix}() \]
\[ \text{glPopMatrix}() \]
\[ \text{DrawSquare}() \]
\[ \text{glPushMatrix}() \]
\[ \text{glScale3f}(2,2,2) \]
\[ \text{glTranslate3f}(1,0,0) \]
\[ \text{DrawSquare}() \]
\[ \text{glPopMatrix}() \]
Composing Transformations

- OpenGL example

```c
glLoadIdentity();
glTranslatef(4,1,0);
glPushMatrix();
glRotatef(45,0,0,1);
glTranslatef(0,2,0);
glScalef(2,1,1);
glTranslate(1,0,0);
glPopMatrix();
glPopMatrix();
```

```c
F
F
W
W
F
F
h
h
F
F
h
h
```
Transformation Hierarchies

- example

```c
// Example code for drawing a robot arm

#define PI 3.14159265358979323846

void drawRobotArm() {
    // Translate the body to the origin
    glTranslate3f(0,0,0);
    DrawBody();
    glPopMatrix();

    // Translate the head
    glTranslate3f(0,7,0);
    DrawHead();
    glPopMatrix();

    // Translate the upper arm
    glTranslate3f(2.5,5.5,0);
    glRotatef(θ2,0,0,1);
    DrawUArm();
    glTranslate3f(0,-3.5,0);
    glRotatef(θ1,0,0,1);
    DrawLArm();
    glPopMatrix();
}
```

... (draw other arm)
Modularization

- drawing a scaled square

```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();
}
```
Demo: Brown Applets

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

- Hierarchies don’t fall apart when changed
- Transforms apply to graph nodes beneath
Hierarchical Modeling

- advantages:
  - matrix stacks make it feasible to have multiple copies of one object
  - no need to compute inverse matrices all the time
  - avoids incremental changes to coordinate systems
    - accumulation of numerical errors
  - modularize changes to pipeline state

- practical issues:
  - in graphics hardware, depth of matrix stacks is limited
    - (typically 16 for model/view and about 4 for projective matrix)
Hierarchies Limits

- advantages
  - often good control knobs
  - maintain structural constraints

- limitations
  - expressivity: not always the best controls
  - can’t do closed kinematic chains
    - keep hand on hip
  - can’t do other constraints
    - e.g. don’t walk through walls
Single Parameter: simple

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

\[
m = s/60, \quad h = m/60,
\]
\[
\theta_s = \frac{2\pi s}{60},
\]
\[
\theta_m = \frac{2\pi m}{60},
\]
\[
\theta_h = \frac{2\pi h}{60}
\]
Single Parameter: complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk
Single Parameter: complex

- mechanisms not easily expressible with affine transforms

http://www.flying-pig.co.uk/mechanisms/pages/irregular.html
Snowmen Example

http://www.lighthouse3d.com/opengl/displaylists

- hierarchical modelling
- display lists
  - OpenGL efficiency trick
  - make list of OpenGL commands for later use
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
    glColor3f(1.0f, 0.5f , 0.5f);
    glutSolidSphere(0.25f,20,20);

    // 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();

}
Snowmen: No Lists

// 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

```cpp
GLuint createDL() {
    GLuint snowManDL;
    // Create the id for the list
    snowManDL = glGenLists(1);
    // start list
    glNewList(snowManDL, GL_COMPILE);
    // call the function that contains the rendering commands
    drawSnowMan();
    // endList
    glEndList();
    return(snowManDL); }
```
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
Snowmen: Display Lists

// 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
        glCallList(Dlid);
        glPopMatrix();
    }
Performance

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36x
Snowmen: One Big List

```c
GLuint createDL() {
    GLuint snowManDL;
    snowManDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    for(int i = -3; i < 3; i++)
        for(int j=-3; j < 3; j++) {
            glPushMatrix();
            glTranslatef(i*10.0,0,j * 10.0);
            drawSnowMan();
            glPopMatrix();
        }
    glEndList();
    return(snowManDL);
}
```
Performance

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36x
- 108 FPS: single 36 snowman display list
Snowmen: Nested Lists

GLuint createDL() {
    GLuint snowManDL,loopDL;
    snowManDL = glGenLists(1);
    loopDL = glGenLists(1);
    glNewList(snowManDL, GL_COMPILE);
    drawSnowMan();
    glEndList();
    glNewList(loopDL, GL_COMPILE);
    for(int i = -3; i < 3; i++)
        for(int j=-3; j < 3; j++) {
            glPushMatrix();
            glTranslatef(i*10.0,0,j * 10.0);
            glCallList(snowManDL);
            glPopMatrix();
        }
    glEndList();
    return(loopDL); }


Performance

- 55 FPS: no display list
- 153 FPS: 1 snowman display list, called 36x
- 108 FPS: single 36 snowman display list
- 153 FPS: nested display lists
  - exploit hierarchical structure