Display Lists, Viewing

Week 3, Fri Jan 21

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005

Project 1 Clarifications, Hints

- finish all required parts before
  - going for extra credit
  - playing with lighting or viewing
- ok to use glRotate, glTranslate, glScale
- ok to use glutSolidCube, or build your own
  - where to put origin? your choice
    - center of object, range - .5 to +.5
    - corner of object, range 0 to 1

Project 1 Clarifications, Hints

- visual debugging
  - color cube faces differently
  - colored lines sticking out of glutSolidCube faces
- thinking about transformations
  - move physical objects around
  - play with demos
  - Brown scenegraph applets

Project 1 Clarifications, Hints

- transitions
  - safe to linearly interpolate parameters for glRotate/glTranslate/glScale
  - do not interpolate individual elements of 4x4 matrix!
Review: Matrix Stacks

- OpenGL matrix calls postmultiply matrix M onto current matrix P, overwrite it to be PM
- or can save intermediate states with stack

```cpp
glPushMatrix
glPopMatrix
```

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

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)

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

Single Parameter: Simple

- parameters as functions of other parameters
  - 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

Display Lists

http://www.flying-pig.co.uk
http://www.flying-pig.co.uk/mechanisms/pages/irregular.html
Display Lists

- reuse block of OpenGL code
- more efficient than immediate mode
  - avoid function calls for every vertex/attribute, driver optimization, graphics board cache (bandwidth!)
- 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

drawSnowMan

```c
void drawSnowMan() {
    // Draw Eyes
    glColor3f(1.0f, 1.0f, 1.0f);
    glPushMatrix();
    glVertex3f(0.0f, 1.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10, 2);
    glVertex3f(0.0f, 1.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10, 2);
    glPopMatrix();
    // Draw Body
    gColor3f(0.0f,0.75f,0.0f);
    glPushMatrix();
    glVertex3f(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f,20,20);
    glVertex3f(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f,20,20);
    glPopMatrix();
    // Draw Head
    gColor3f(1.0f, 0.5f, 0.5f);
    glPushMatrix();
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f,10,10);
    glVertex3f(0.25f,20,20);
    glPopMatrix();
    // Draw Nose
    gColor3f(0.8f,0.5f,10,2);
    glPushMatrix();
    glTranslatef(0.0f,1.0f, 0.0f);
    glutSolidCone(0.08f,0.0f, 0.0f);
    glPopMatrix();
}
```

Making Display Lists

```c
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();
    // end list
    glEndList();
    return(snowManDL);
}
```

Snowmen: Display Lists

```c
// 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

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);
            // Call the function to draw a snowman
            glCallList(Dlid);
            glPopMatrix();
        }
    }
}
```

`return(36K polygons, 55 FPS)`
Snowmen: Hierarchical Lists

```c
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);
}
```

Display Lists

- example: 36 snowmen
  - small display list with 36x reuse
    - 3x faster
  - big display list with 1x reuse
    - 2x faster
  - nested display lists, 1x * 36x reuse:
    - 3x faster, high-level block available
    - exploit hierarchical structure

Using Transformations

- three ways
  - modelling transforms
    - place objects within scene (shared world)
  - viewing transforms
    - place camera
  - projection transforms
    - change type of camera

Viewing and Projection

- need to get from 3D world to 2D image
- projection: geometric abstraction
  - what eyes or cameras do
- two pieces
  - viewing transform:
    - where is the camera, what is it pointing at?
  - perspective transform: 3D to 2D
    - flatten to image

Rendering Pipeline
Rendering Pipeline

- **Geometry Database**
- **Model/View Transform.**
- **Lighting**
- **Perspective Transform.**
- **Clipping**
- **Scan Conversion**
- **Texturing**
- **Depth Test**
- **Blending**
- **Frame-buffer**

**Result**

- All vertices of scene in shared 3D world coordinate system

**Coordinate Systems**

- Result of a transformation
- Names
  - Convenience
  - Kangaroo: neck, head, tail
  - Standard conventions in graphics pipeline
  - Object/modelling
  - World
  - Camera/viewing/eye
  - Screen/window
  - Raster/device
Basic Viewing
- starting spot - OpenGL
  - camera at world origin
    - probably inside an object
  - y axis is up
  - looking down negative z axis
    - why? RHS with x horizontal, y vertical, z out of screen
- translate backward so scene is visible
  - move distance \( d = \text{focal length} \)
- can use rotate-translate-scale to move camera
  - demo: Nate Robins tutorial \textit{transformations}

Viewing in Project 1
- where is camera in template code?
  - 5 units back, looking down -z axis

Convenient Camera Motion
- rotate/translate/scale not intuitive
- arbitrary viewing position
  - eye point, gaze/lookat direction, up vector

From World to View Coordinates
- translate \textit{eye} to origin
- rotate \textit{view} vector (\textit{lookat} – \textit{eye}) to \textit{w} axis
- rotate around \textit{w} to bring \textit{up} into \textit{vw}-plane
Deriving World-to-View Transformation

- rotate view vector (lookat – eye) to w axis
  - w is just opposite of view/gaze vector \( \mathbf{g} \)
  
  \[
  \mathbf{w} = -\hat{\mathbf{g}} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}
  \]

- rotate from WCS \( \mathbf{xyz} \) into \( \mathbf{uvw} \) coordinate system with matrix that has rows \( \mathbf{u}, \mathbf{v}, \mathbf{w} \)
  
  \[
  \mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|}
  \]
  
  \[
  \mathbf{v} = \mathbf{w} \times \mathbf{u}
  \]
  
  \[
  \mathbf{w} = -\hat{\mathbf{g}} = -\frac{\mathbf{g}}{\|\mathbf{g}\|}
  \]

- reminder: rotate from \( \mathbf{uvw} \) to \( \mathbf{xyz} \) coord sys with matrix \( \mathbf{M} \) that has columns \( \mathbf{u}, \mathbf{v}, \mathbf{w} \)

- rotate from \( \mathbf{xyz} \) coord sys to \( \mathbf{uvw} \) coord sys with matrix \( \mathbf{M}^T \) that has rows \( \mathbf{u}, \mathbf{v}, \mathbf{w} \)

OpenGL Viewing Transformation

\[ \text{gluLookAt(} \text{ex, ey, ez, lx, ly, lz, ux, uy, uz)} \]

- postmultiplies current matrix, so to be safe:

\[ \text{glMatrixMode(GL_MODELVIEW)}; \]
\[ \text{glLoadIdentity}(); \]
\[ \text{glMatrixMode(GL_MODELVIEW)}; \]

// now ok to do model transformations

reminder:

\[ \text{glLoadIdentity}(); \]
\[ \text{glMatrixMode(GL_MODELVIEW)}; \]
\[ \text{glLoadIdentity}(); \]
\[ \text{glMatrixMode(GL_MODELVIEW)}; \]

// now ok to do model transformations

demo: Nate Robins tutorial projection

- translate eye to origin

\[
\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]
Moving the Camera or the World?

- two equivalent operations
  - move camera one way vs. move world other way
- example
  - initial OpenGL camera: at origin, looking along -z axis
  - create a unit square parallel to camera at z = -10
  - translate in z by 3 possible in two ways
    - camera moves to z = -3
      - Note OpenGL models viewing in left-hand coordinates
    - camera stays put, but square moves to -7
  - resulting image same either way
    - possible difference: are lights specified in world or view coordinates?