OpenGL Transformations

Objectives
- Learn how to carry out transformations in OpenGL
  - Rotation
  - Translation
  - Scaling
- Introduce OpenGL matrix modes
  - Model-view
  - Projection

OpenGL Matrices
- In OpenGL matrices are part of the state
- Multiple types
  - Model-View (GL_MODELVIEW)
  - Projection (GL_PROJECTION)
  - Texture (GL_TEXTURE) (ignore for now)
  - Color(GL_COLOR) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulated by
  - glMatrixMode(GL_MODELVIEW);
  - glMatrixMode(GL_PROJECTION);

Current Transformation Matrix (CTM)
- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit

CTM operations
- The CTM can be altered either by loading a new CTM or by post-multiplication
  - Load an identity matrix: C ← I
  - Load an arbitrary matrix: C ← M
  - Load a translation matrix: C ← T
  - Load a rotation matrix: C ← R
  - Load a scaling matrix: C ← S
  - Postmultiply by an arbitrary matrix: C ← CM
  - Postmultiply by a translation matrix: C ← CT
  - Postmultiply by a rotation matrix: C ← CR
  - Postmultiply by a scaling matrix: C ← CS
Rotation about a Fixed Point

Start with identity matrix: $C \leftarrow I$
Move fixed point to origin: $C \leftarrow CT$
Rotate: $C \leftarrow CR$
Move fixed point back: $C \leftarrow CT^{-1}$

Result: $C = TR T^{-1}$ which is backwards.

This result is a consequence of doing postmultiplications. Let’s try again.

Reversing the Order

We want $C = T^{-1}RT$
so we must do the operations in the following order

$C \leftarrow I$
$C \leftarrow CT^{-1}$
$C \leftarrow CR$
$C \leftarrow CT$

Each operation corresponds to one function call in the program.

Note that the last operation specified is the first executed in the program.

CTM in OpenGL

• OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
• Can manipulate each by first setting the correct matrix mode

Rotation, Translation, Scaling

Load an identity matrix:

```glLoadIdentity();```

Multiply on right:

```glRotatef(theta, vx, vy, vz)`
```glTranslatef(dx, dy, dz)`
```glScalef(sx, sy, sz)`

Each has a float (f) and double (d) format (glScaled)
Example

• Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

  glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);

• Remember that last matrix specified in the program is the first applied

Arbitrary Matrices

• Can load and multiply by matrices defined in the application program

  glLoadMatrixf(m)
glMultMatrixf(m)

• The matrix \( m \) is a one dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns

• In glMultMatrixf, \( m \) multiplies the existing matrix on the right

Matrix Stacks

• In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 10)
  - Avoiding state changes when executing display lists

• OpenGL maintains stacks for each type of matrix
  - Push/Pop present type (as set by glMatrixMode) by
    
    glPushMatrix()
    glPopMatrix()

Reading Back Matrices

• Can also access matrices (and other parts of the state) by query functions

  glGetIntegerv
  glGetFloatv
  glGetBooleanv
  glGetDoublev
  glIsEnabled

• For matrices, we use as

  double m[16];
glGetFloatv(GL_MODELVIEW, m);
Using Transformations

- Example: use idle function to rotate a cube and mouse function to change direction of rotation
- Start with a program that draws a cube (colorcube.c) in a standard way
  - Centered at origin
  - Sides aligned with axes
  - Will discuss modeling in next lecture

 Idle and Mouse callbacks

```c
void spinCube() {
    theta[axis] += 2.0;
    if (theta[axis] > 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y) {
    if (btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
        axis = 0;
    if (btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
        axis = 1;
    if (btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
        axis = 2;
}
```

main.c

```c
void main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Display callback

```c
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    colorcube();
    glutSwapBuffers();
}
```

Note that because of fixed from of callbacks, variables such as `theta` and `axis` must be defined as globals.

Camera information is in standard reshape callback.
Using the Model-view Matrix

• In OpenGL the model-view matrix is used to
  - Position the camera
    • Can be done by rotations and translations but is often easier to use `gluLookAt`
  - Build models of objects
• The projection matrix is used to define the view volume and to select a camera lens

Model-view and Projection Matrices

• Although both are manipulated by the same functions, we have to be careful because incremental changes are always made by postmultiplication
  - For example, rotating model-view and projection matrices by the same matrix are not equivalent operations.
  - Postmultiplication of the model-view matrix is equivalent to premultiplication of the projection matrix

Smooth Rotation

• From a practical standpoint, we often want to use transformations to move and reorient an object smoothly
  - Problem: find a sequence of model-view matrices $M_0, M_1, \ldots, M_n$ so that when they are applied successively to one or more objects we see a smooth transition
• For orientating an object, we can use the fact that every rotation corresponds to part of a great circle on a sphere
  - Find the axis of rotation and angle
  - Virtual trackball (see text)

Incremental Rotation

• Consider the two approaches
  - For a sequence of rotation matrices $R_0, R_1, \ldots, R_n$, find the Euler angles for each and use $R_i = R_{i-1} R_0$ $R_n$
    • Not very efficient
  - Use the final positions to determine the axis and angle of rotation, then increment only the angle
• Quaternions can be more efficient than either
Quaternions

- Extension of imaginary numbers from two to three dimensions
- Requires one real and three imaginary components i, j, k
  \[ q = q_0 + q_1 i + q_2 j + q_3 k \]
- Quaternions can express rotations on sphere smoothly and efficiently. Process:
  - Model-view matrix → quaternion
  - Carry out operations with quaternions
  - Quaternion → Model-view matrix

Interfaces

- One of the major problems in interactive computer graphics is how to use two-dimensional devices such as a mouse to interface with three dimensional objects
- Example: how to form an instance matrix?
- Some alternatives
  - Virtual trackball
  - 3D input devices such as the spaceball
  - Use areas of the screen
    - Distance from center controls angle, position, scale depending on mouse button depressed