OpenGL Transformations

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

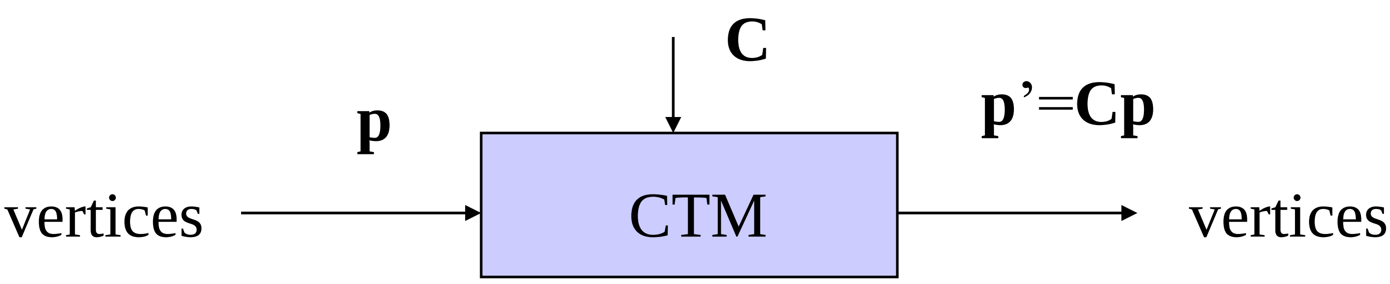
- Learn how to carry out transformations in OpenGL
  - Rotation
  - Translation
  - Scaling
- Introduce mat,h and vec.h transformations
  - Model-view
  - Projection

Pre 3.1OpenGL Matrices

- In OpenGL matrices were part of the state
- Multiple types
  - Model-View (**`GL_MODELVIEW`**)
  - Projection (**`GL_PROJECTION`**)
  - Texture (**`GL_TEXTURE`**)
  - Color(**`GL_COLOR`**)
- 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 postmutiplication

  Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
  Load an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{M}$

  Load a translation matrix: $\mathbf{C} \leftarrow \mathbf{T}$
  Load a rotation matrix: $\mathbf{C} \leftarrow \mathbf{R}$
  Load a scaling matrix: $\mathbf{C} \leftarrow \mathbf{S}$

  Postmultiply by an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{CM}$
  Postmultiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$
  Postmultiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{C}\ \mathbf{R}$
  Postmultiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{C}\ \mathbf{S}$

Rotation about a Fixed Point

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

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

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

Reversing the Order

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

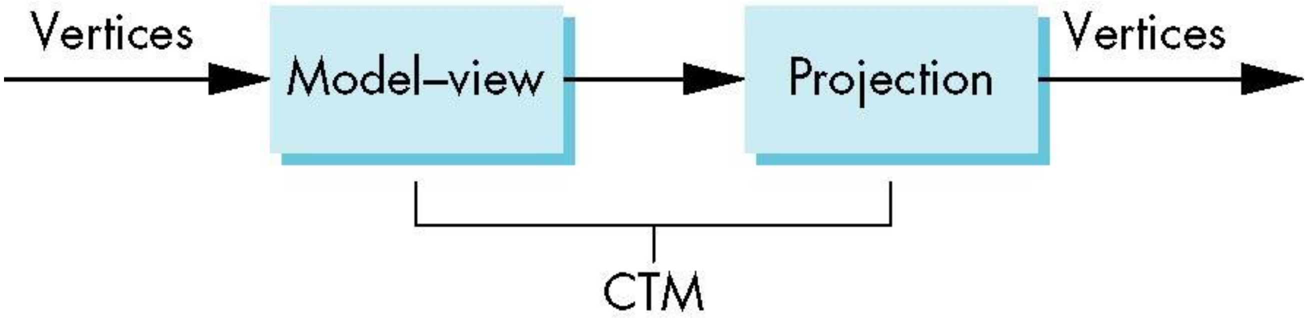
$\mathbf{C} \leftarrow \mathbf{I}$
$\mathbf{C} \leftarrow \mathbf{CT}^{-1}$
$\mathbf{C} \leftarrow \mathbf{CR}$
$\mathbf{C} \leftarrow \mathbf{CT}$

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 had a model-view and a projection matrix in the pipeline which were concatenated together to form the CTM
- We will emulate this process



Rotation, Translation, Scaling

Create an identity matrix:

```
mat4 m = Identity();
```

Multiply on right by rotation matrix of **`theta`** in degrees where (**`vx, vy, vz`**) define axis of rotation

```
mat4 r = Rotate(theta, vx, vy, vz)
m = m*r;
```

Do same with translation and scaling:

```
mat4 s = Scale( sx, sy, sz)
mat4 t = Transalate(dx, dy, dz);
m = m*s*t;
```

Example

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

```
mat 4 m = Identity();
m = Translate(1.0, 2.0, 3.0)*
    Rotate(30.0, 0.0, 0.0, 1.0)*
    Translate(-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
- Matrices are stored as one dimensional array of 16 elements which are the components of the desired 4 x 4 matrix stored by <u>columns</u>
- OpenGL functions that have matrices as parameters allow the application to send the matrix or its transpose

Matrix Stacks

- In many situations we want to save transformation matrices for use later
  - Traversing hierarchical data structures (Chapter 8)
  - Avoiding state changes when executing display lists
- Pre 3.1 OpenGL maintained stacks for each type of matrix
- Easy to create the same functionality with a simple stack class

Reading Back State

- Can also access OpenGL variables (and other parts of the state) by *query* functions

```
glGetIntegerv
glGetFloatv
glGetBooleanv
glGetDoublev
glIsEnabled
```

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 in a standard way
    - Centered at origin
    - Sides aligned with axes
    - Will discuss modeling in next lecture

main.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();
}
```

Idle and Mouse callbacks

```
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;
}
```

Display callback

We can form matrix in application and send to shader and let shader do the rotation or we can send the angle and axis to the shader and let the shader form the transformation matrix and then do the rotation

More efficient than transforming data in application and resending the data

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glUniform(…); //or glUniformMatrix
   glDrawArrays(…);
   glutSwapBuffers();
}
```

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 a LookAt function
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens
- Although these matrices are no longer part of the OpenGL state, it is usually a good strategy to create them in our own applications

Smooth Rotation

- From a practical standpoint, we are often want to use transformations to move and reorient an object smoothly
  - Problem: find a sequence of model-view matrices $\mathbf{M}_0,\mathbf{M}_1,\ldots..,\mathbf{M}_n$ 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 $\mathbf{R}_0,\mathbf{R}_1,\ldots..,\mathbf{R}_n$, find the Euler angles for each and use $\mathbf{R}_i = \mathbf{R}_{iz}\,\mathbf{R}_{iy}\,\mathbf{R}_{ix}$
    - 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 $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

- Quaternions can express rotations on sphere smoothly and efficiently. Process:
  - Model-view matrix $\rightarrow$ quaternion
  - Carry out operations with quaternions
  - Quaternion $\rightarrow$ 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 obejcts
- 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