

OpenGL Matrices

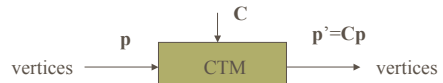
- In OpenGL matrices are part of the state
- Three types
 - Model-View (`GL_MODEL_VIEW`)
 - Projection (`GL_PROJECTION`)
 - Texture (`GL_TEXTURE`) (ignore for now)
- Single set of functions for manipulation
- Select which to be manipulated by
 - `glMatrixMode(GL_MODEL_VIEW);`
 - `glMatrixMode(GL_PROJECTION);`



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

CTM operations

- The CTM can be altered either by loading a new CTM or by postmultiplication

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

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

Postmultiply by an arbitrary matrix: $C \leftarrow CM$
Postmultiply by a translation matrix: $C \leftarrow CT$
Postmultiply by a rotation matrix: $C \leftarrow CR$
Postmultiply by a scaling matrix: $C \leftarrow CS$



Slides by Edward Angel © 2002

Rotation about a Fixed Point

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

Result: $C = T^{-1}RT$

Each operation corresponds to one function call in the program.

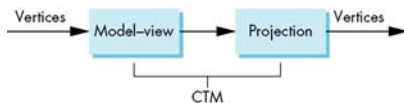
Note that the last operation specified is the first executed in the program



Slides by Edward Angel © 2002

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 matrix mode



Slides by Edward Angel © 2002

Rotation, Translation, Scaling

Load an identity matrix:

```
glLoadIdentity()
```

Multiply on right:

```
glRotatef(theta, vx, vy, vz)
```

`theta` in degrees, (`vx`, `vy`, `vz`) define axis of rotation

```
glTranslatef(dx, dy, dz)
```

```
glScalef(sx, sy, sz)
```

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



Slides by Edward Angel © 2002

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, .10);
glTranslatef(-1.0, -2.0, -3.0);
```

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



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

Matrix Stacks

- In many situations we want to save transformation matrices for use later
 - Traversing hierarchical data structures (Chapter 9)
 - Avoiding state changes when executing display lists
- OpenGL maintains stacks for each type of matrix

- Access present type (as set by `glMatrixMode`) by

```
glPushMatrix()
glPopMatrix()
```



Slides by Edward Angel © 2002

Reading Back Matrices

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

```
glGetIntegerv
glGetFloatv
glGetBooleanv
glGetDoublev
glIsEnabled
```

- For matrices, we use as
`double m[16];`
`glGetFloatv(GL_MODELVIEW, m);`



Slides by Edward Angel © 2002

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



Slides by Edward Angel © 2002

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();
}
```



Slides by Edward Angel © 2002

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;
}
```

Slides by Edward Angel © 2002

Display callback

```
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 form of callbacks, variables such as `theta` and `axis` must be defined as globals

Camera information is in standard reshape callback

Slides by Edward Angel © 2002

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` (Chapter 5)
 - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens
- 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

Slides by Edward Angel © 2002

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 M_0, M_1, \dots, 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)

Slides by Edward Angel © 2002

Incremental Rotation

- Consider the two approaches
 - For a sequence of rotation matrices R_0, R_1, \dots, R_n , find the Euler angles for each and use $R_i = R_z R_y R_x$
 - 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

Slides by Edward Angel © 2002

Quaternions

- Extension of imaginary numbers from two to three dimensions
- Requires one real and three imaginary components i, j, k
$$q = q_0 + q_1i + q_2j + q_3k$$
- 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

Slides by Edward Angel © 2002