Computer Graphics
CS 543 – Lecture 5 (Part 2)
Implementing Transformations

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Objectives

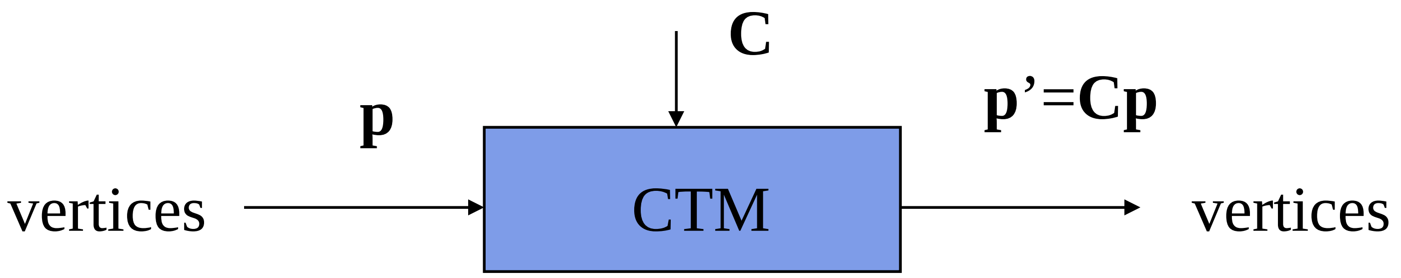
- Learn how to implement transformations in OpenGL
  - Rotation
  - Translation
  - Scaling
- Introduce mat,h and vec.h transformations
  - Model-view
  - Projection

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: $\mathbf{C} \leftarrow \mathbf{C\,R}$
Postmultiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{C\,S}$

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

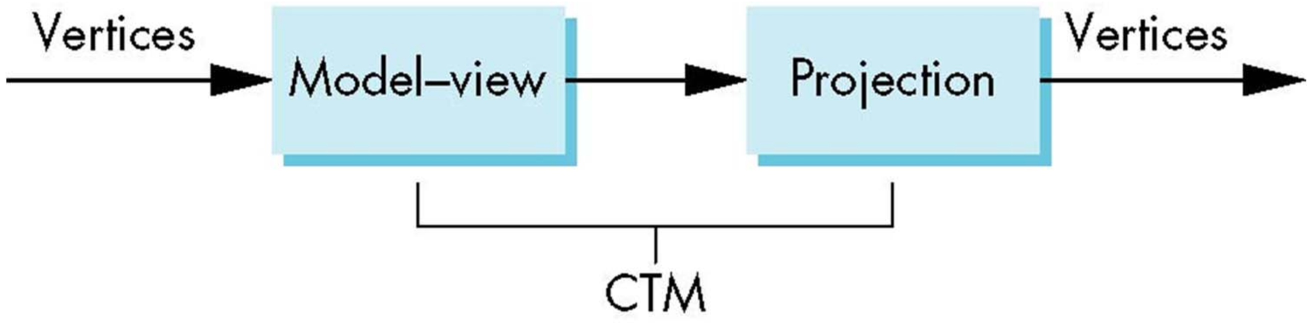
- We want $\mathbf{C} = \mathbf{T}^{-1}\,\mathbf{R}\,\mathbf{T}$
- So we must do operations in the following order

$$\mathbf{C} \leftarrow \mathbf{I}$$
$$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}^{-1}$$
$$\mathbf{C} \leftarrow \mathbf{C}\mathbf{R}$$
$$\mathbf{C} \leftarrow \mathbf{C}\mathbf{T}$$

- Each operation corresponds to one function call in the program.
- **Note:** last operation specified is first executed in program

CTM in OpenGL

- Previously, OpenGL had a **model-view** and a **projection matrix** in the pipeline that were concatenated together to form the **CTM**
- Useful!!
- So, we will emulate this process in our application



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;
```

Transformation matrices Formed?

- Converts all transforms (translate, scale, rotate) to 4x4 matrix
- Put 4x4 transform matrix into **modelview matrix**
- How? multiplies current modelview matrix by 4x4 matrix
- Example

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

$\Rightarrow$

CTM Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformation matrices Formed?

```
mat4 m = Identity();
mat4 t = Translate(3,6,4);
m = m*t;
```

Identity Matrix **Translation Matrix** **CTM Matrix**

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

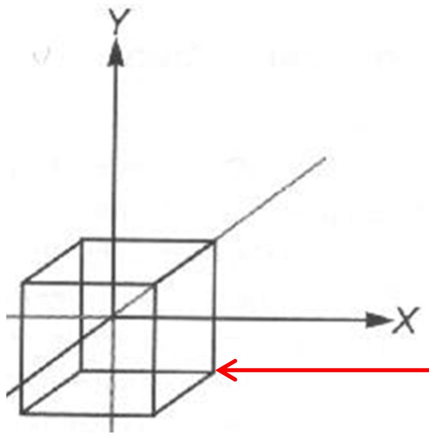
Transformation matrices Formed?

- Then what?

```
mat4 m = Identity();
mat4 t = Translate(3,6,4);
m = m*t;
colorcube( );
```

CTM Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{Original vertex}} = \underbrace{\begin{pmatrix} 4 \\ 7 \\ 5 \\ 1 \end{pmatrix}}_{\text{Transformed vertex}}$$



Each vertex of cube is multiplied by CTM matrix to get translated vertex

Transformation matrices Formed?

- Consider following code snipet

```
mat4 m = Identity();
mat4 s = Scale(1,2,3);
m = m*s;
```

Identity Matrix **Scaling Matrix** **CTM Matrix**

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

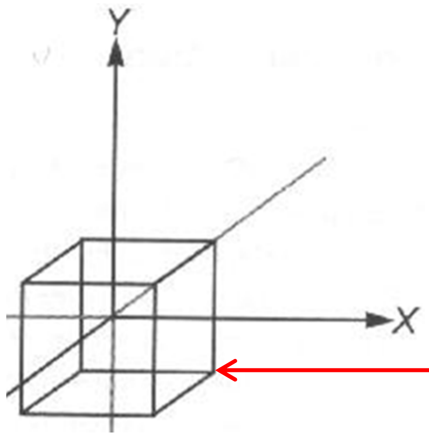
Transformation matrices Formed?

- Then what?

```
mat4 m = Identity();
mat4 s = Scale(1,2,3);
m = m*s;
colorcube( );
```

CTM Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \underbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}}_{\text{Original vertex}} = \underbrace{\begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \end{pmatrix}}_{\text{Transformed vertex}}$$



Each vertex of cube is multiplied by modelview matrix to get scaled vertex position

Transformation matrices Formed?

- What of gltranslate, then scale, then ….
- Just multiply them together. Evaluated in ***reverse order***!! E.g:

```
mat4 m = Identity();
mat4 s = Scale(1,2,3);
mat4 t = Translate(3,6,4);
m = m*s*t;
```

$$\underbrace{\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}}_{\textbf{Identity Matrix}}\times\underbrace{\begin{pmatrix}1&0&0&0\\0&2&0&0\\0&0&3&0\\0&0&0&1\end{pmatrix}}_{\textbf{Scale Matrix}}\times\underbrace{\begin{pmatrix}1&0&0&3\\0&1&0&6\\0&0&1&4\\0&0&0&1\end{pmatrix}}_{\textbf{Translate Matrix}}=\underbrace{\begin{pmatrix}1&0&0&3\\0&2&0&12\\0&0&3&12\\0&0&0&1\end{pmatrix}}_{\textbf{Final CTM Matrix}}$$

Transformation matrices Formed?

- Then what?

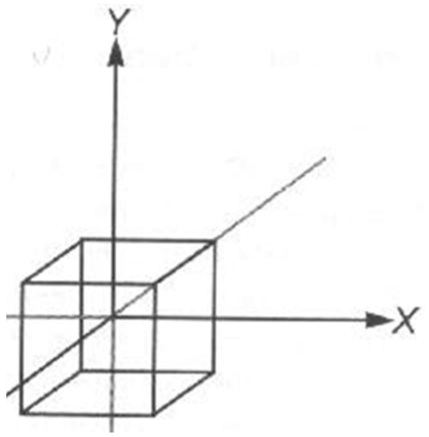
```
mat4 m = Identity();
mat4 s = Scale(1,2,3);
mat4 t = Translate(3,6,4);
m = m*s*t;
colorcube( );
```

Modelview Matrix

$$\begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 2 & 0 & 12 \\ 0 & 0 & 3 & 12 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 14 \\ 15 \\ 1 \end{pmatrix}$$

Original vertex

Transformed vertex



Each vertex of cube is multiplied by modelview matrix to get scaled vertex position

Example

- Rotation about z axis by 30 degrees about a fixed point (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 last matrix specified in program (i.e. translate matrix in example) is first applied

Arbitrary Matrices

- Can multiply by matrices from transformation commands (Translate, Rotate, Scale) defined in the application program
- Can also load CTM with arbitrary 4x4 matrices
- Matrices are stored as one dimensional array of 16 elements that are components of desired 4 x 4 matrix

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 program that draws cube as before
  - Centered at origin
  - Sides aligned with axes

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

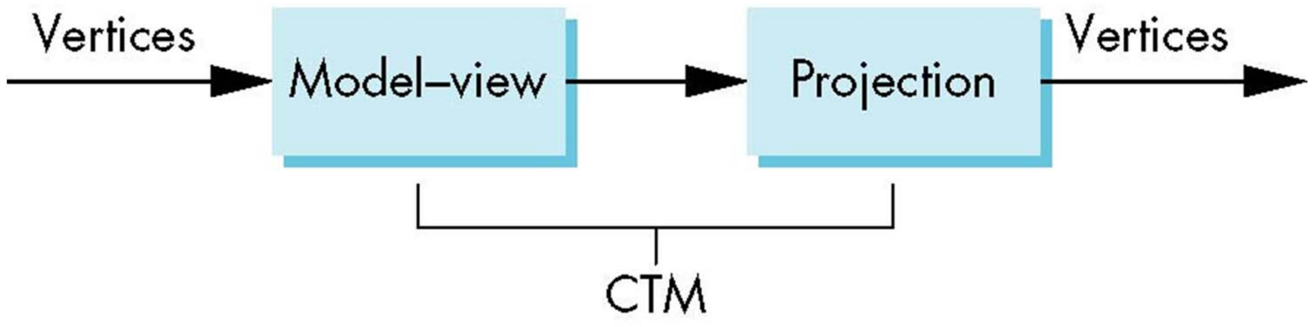
```
void spinCube()
{
  theta[axis] += 2.0;
  if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
  glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
   if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
          axis = 0;
   if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
          axis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

Display callback

- We can form matrix (CTM) 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



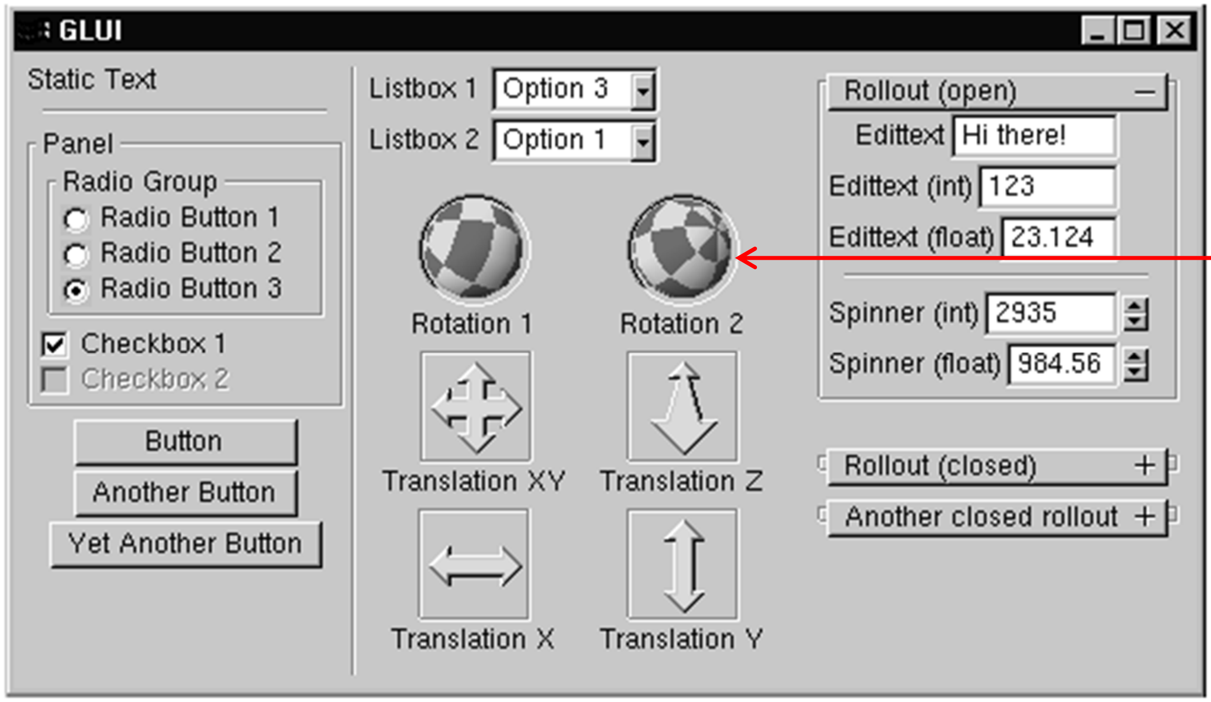
- In OpenGL the model-view matrix is used to
  - Position camera (using LookAt function)
  - Transform 3D models
- The projection matrix used to define view volume and select a camera lens
- Although these matrices no longer part of OpenGL state, good to create them in our applications

3D? 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

GLUI

- User Interface Library by Paul Rademacher
- Provides sophisticated controls and menus
- Not used in this class/optional



Virtual trackball

References

- Angel and Shreiner, Chapter 3
- Hill and Kelley, appendix 4