
CIS 454 Computer Graphics

Lecture 13, 10/19/2006

Li Shen
Computer and Information Science
UMass Dartmouth

Notes

- Sunday: Review materials
 - Tuesday: Midterm review
 - Thursday: Midterm exam
-

Representation

- Introduce concepts such as dimension and basis
 - Introduce coordinate systems for representing vectors spaces and frames for representing affine spaces
 - Discuss change of frames and bases
 - Introduce homogeneous coordinates
-

Dimension

- In a vector space, the maximum number of linearly independent vectors is fixed and is called the *dimension* of the space
- In an n -dimensional space, any set of n linearly independent vectors form a *basis* for the space
- Given a basis v_1, v_2, \dots, v_n , any vector v can be written as

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$$

where the $\{\alpha_i\}$ are unique

Representation in a Coordinate System

- Consider a basis v_1, v_2, \dots, v_n
- A vector is written $v = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$
- The list of scalars $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ is the *representation* of v with respect to the given basis
- We can write the representation as a row or column array of scalars

$$\mathbf{a} = [\alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_n]^T = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \cdot \\ \alpha_n \end{bmatrix}$$

Representation in a Frame

- Frame determined by (P_0, v_1, v_2, v_3)
- Within this frame, every vector can be written as

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n$$

- Every point can be written as

$$P = P_0 + \beta_1 v_1 + \beta_2 v_2 + \dots + \beta_n v_n$$

Homogeneous Representation

If we define $0 \cdot \mathbf{P} = \mathbf{0}$ and $1 \cdot \mathbf{P} = \mathbf{P}$ then we can write

$$\mathbf{v} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 0] [v_1 \ v_2 \ v_3 \ \mathbf{P}_0]^T$$

$$\mathbf{P} = \mathbf{P}_0 + \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 = [\beta_1 \ \beta_2 \ \beta_3 \ 1] [v_1 \ v_2 \ v_3 \ \mathbf{P}_0]^T$$

Thus we obtain the four-dimensional *homogeneous coordinate* representation

$$\mathbf{v} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 0]^T$$

$$\mathbf{p} = [\beta_1 \ \beta_2 \ \beta_3 \ 1]^T$$

Change of Coordinate Systems

The coefficients define a 3 x 3 matrix

$$\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

and the bases can be related by

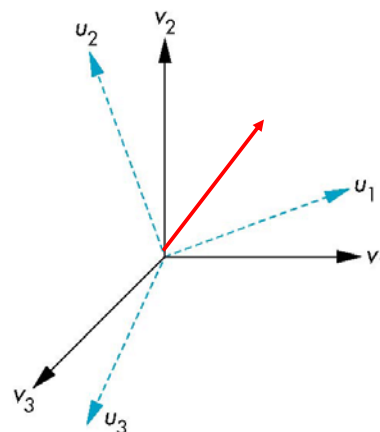
$$\mathbf{a} = \mathbf{M}^T \mathbf{b}$$

$$\mathbf{a} = [\alpha_1 \ \alpha_2 \ \alpha_3]^T \quad \mathbf{b} = [\beta_1 \ \beta_2 \ \beta_3]^T$$

$$\mathbf{u}_1 = \gamma_{11} \mathbf{v}_1 + \gamma_{12} \mathbf{v}_2 + \gamma_{13} \mathbf{v}_3$$

$$\mathbf{u}_2 = \gamma_{21} \mathbf{v}_1 + \gamma_{22} \mathbf{v}_2 + \gamma_{23} \mathbf{v}_3$$

$$\mathbf{u}_3 = \gamma_{31} \mathbf{v}_1 + \gamma_{32} \mathbf{v}_2 + \gamma_{33} \mathbf{v}_3$$



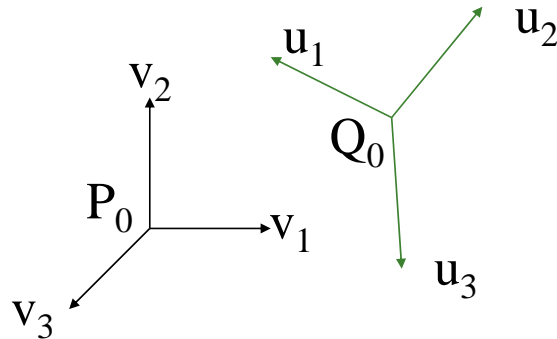
Change of Frames

- We can apply a similar process in homogeneous coordinates to the representations of both points and vectors

Consider two frames:

$$(P_0, v_1, v_2, v_3)$$

$$(Q_0, u_1, u_2, u_3)$$



- Any point or vector can be represented in either frame
- We can represent Q_0, u_1, u_2, u_3 in terms of P_0, v_1, v_2, v_3

Representing One Frame in Terms of the Other

Extending what we did with change of bases

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$

$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$

$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$

$$Q_0 = \gamma_{41}v_1 + \gamma_{42}v_2 + \gamma_{43}v_3 + \gamma_{44}P_0$$

defining a 4 x 4 matrix

$$\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}$$

Working with Representations

Within the two frames any point or vector has a representation of the same form

$\mathbf{a} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ \alpha_4]$ in the first frame

$\mathbf{b} = [\beta_1 \ \beta_2 \ \beta_3 \ \beta_4]$ in the second frame

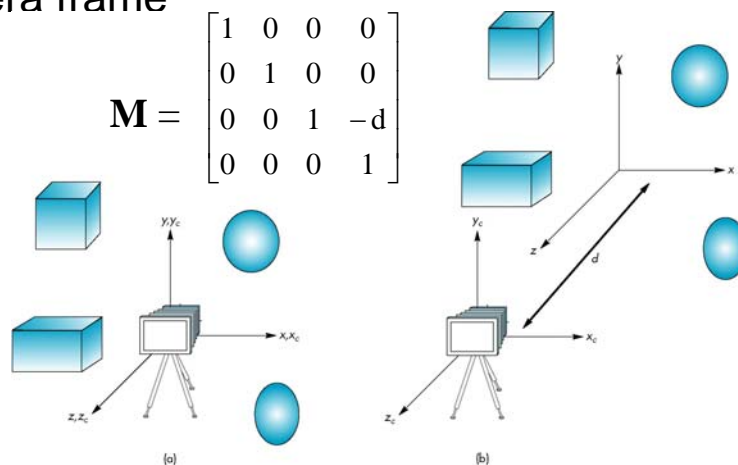
where $\alpha_4 = \beta_4 = 1$ for points and $\alpha_4 = \beta_4 = 0$ for vectors and

$$\mathbf{a} = \mathbf{M}^T \mathbf{b}$$

The matrix \mathbf{M} is 4 x 4 and specifies an affine transformation in homogeneous coordinates

Moving the Camera

If objects are on both sides of $z=0$, we must move camera frame



Transformations

Objectives

- Introduce standard transformations
 - Rotation
 - Translation
 - Scaling
 - Shear
 - Derive homogeneous coordinate transformation matrices
 - Learn to build arbitrary transformation matrices from simple transformations
-

Notation

We will be working with both coordinate-free representations of transformations and representations within a particular frame

P, Q, R : points in an affine space

u, v, w : vectors in an affine space

α, β, γ : scalars

$\mathbf{p}, \mathbf{q}, \mathbf{r}$: representations of points

-array of 4 scalars in homogeneous coordinates

$\mathbf{u}, \mathbf{v}, \mathbf{w}$: representations of points

-array of 4 scalars in homogeneous coordinates

Translation Using Representations

Using the homogeneous coordinate representation in some frame

$$\mathbf{p} = [x \ y \ z \ 1]^T$$

$$\mathbf{p}' = [x' \ y' \ z' \ 1]^T$$

$$\mathbf{d} = [dx \ dy \ dz \ 0]^T$$

Hence $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ or

$$x' = x + d_x$$

$$y' = y + d_y$$

$$z' = z + d_z$$

note that this expression is in four dimensions and expresses point = vector + point

Translation Matrix

We can also express translation using a 4 x 4 matrix \mathbf{T} in homogeneous coordinates

$\mathbf{p}' = \mathbf{T}\mathbf{p}$ where

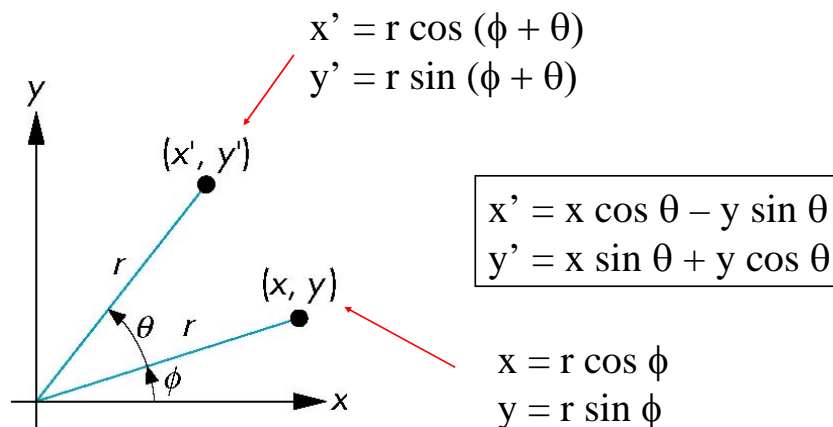
$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

Rotation (2D)

Consider rotation about the origin by θ degrees

- radius stays the same, angle increases by θ



Rotation about the z axis

- Rotation about z axis in three dimensions leaves all points with the same z
 - Equivalent to rotation in two dimensions in planes of constant z

$$x' = x \cos \theta - y \sin \theta$$

$$y' = x \sin \theta + y \cos \theta$$

$$z' = z$$

- or in homogeneous coordinates

$$\mathbf{p}' = \mathbf{R}_z(\theta)\mathbf{p}$$

Rotation Matrix

$$\mathbf{R} = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about x and y axes

- Same argument as for rotation about z axis

- For rotation about x axis, x is unchanged
- For rotation about y axis, y is unchanged

$$\mathbf{R} = \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R} = \mathbf{R}_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

Expand or contract along each axis (fixed point of origin)

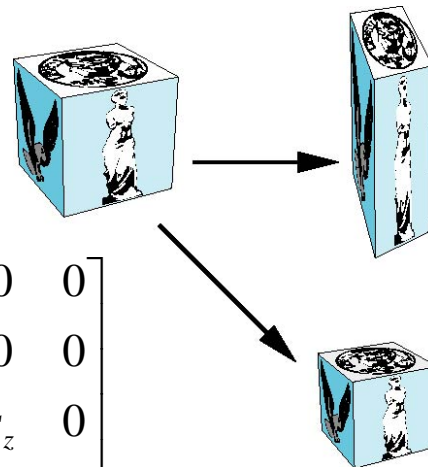
$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

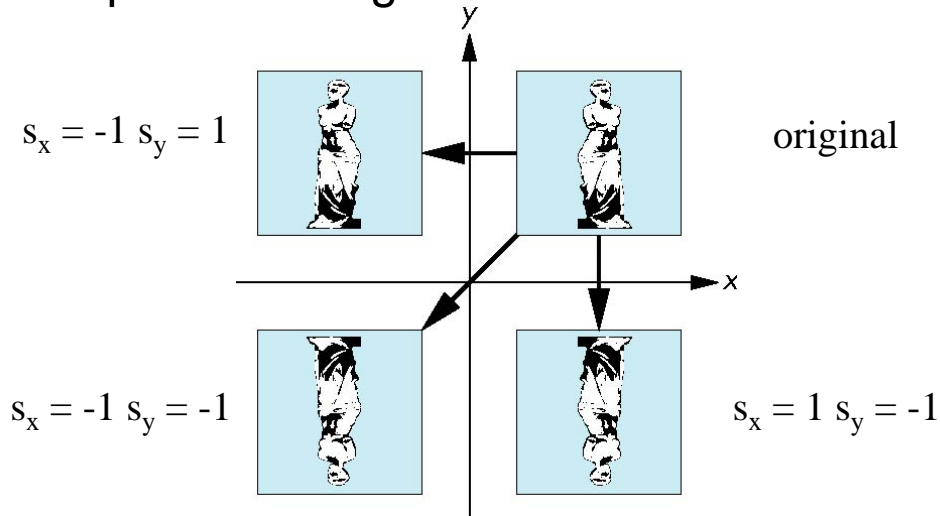
$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Reflection

corresponds to negative scale factors



Inverses

- Although we could compute inverse matrices by general formulas, we can use simple geometric observations
 - Translation: $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
 - Rotation: $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$
 - Holds for any rotation matrix
 - Note that since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$
$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^T(\theta)$$
 - Scaling: $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

Concatenation

- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices
 - Because the same transformation is applied to many vertices, the cost of forming a matrix $\mathbf{M}=\mathbf{ABCD}$ is not significant compared to the cost of computing \mathbf{Mp} for many vertices \mathbf{p}
 - The difficult part is how to form a desired transformation from the specifications in the application
-

Order of Transformations

- Note that matrix on the right is the first applied
- Mathematically, the following are equivalent
$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A}(\mathbf{B}(\mathbf{Cp}))$$
- Note many references use column matrices to represent points. In terms of column matrices

$$\mathbf{p}'^T = \mathbf{p}^T \mathbf{C}^T \mathbf{B}^T \mathbf{A}^T$$

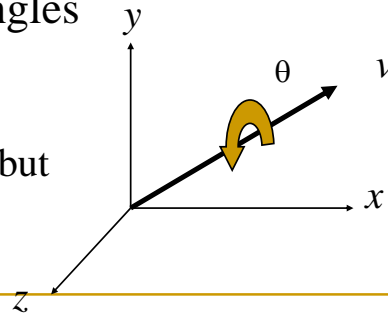
General Rotation About the Origin

A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the x , y , and z axes

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta_z) \mathbf{R}_y(\theta_y) \mathbf{R}_x(\theta_x)$$

θ_x θ_y θ_z are called the Euler angles

Note that rotations do not commute
We can use rotations in another order but with different angles



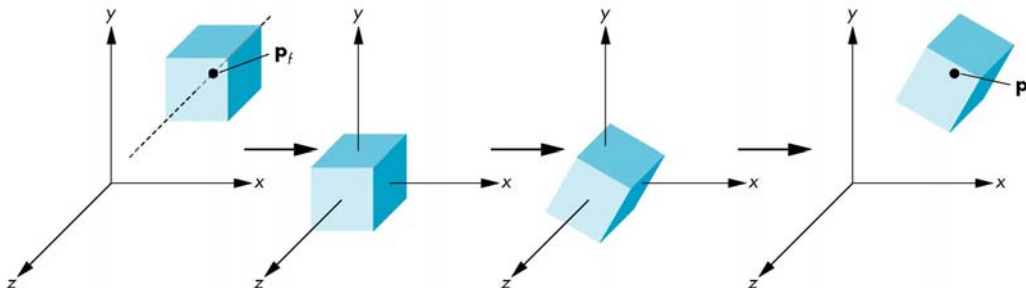
Rotation About a Fixed Point

Move fixed point to origin

Rotate

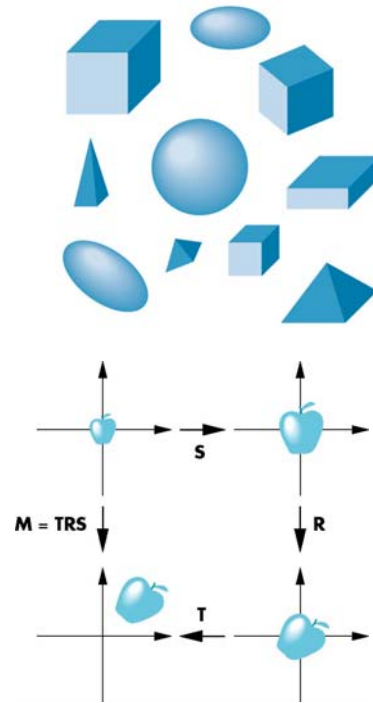
Move fixed point back

$$\mathbf{M} = \mathbf{T}(p_f) \mathbf{R}(\theta) \mathbf{T}(-p_f)$$



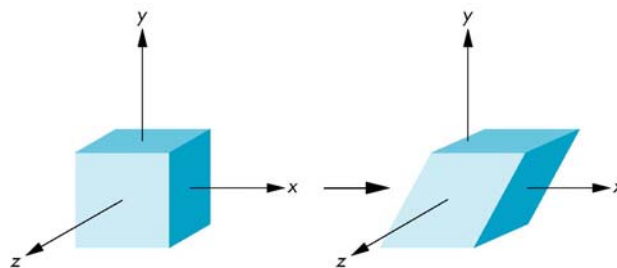
Instancing

- In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size
- We apply an *instance transformation* to its vertices to
 - Scale
 - Orient
 - Locate



Shear

- Helpful to add one more basic transformation
- Equivalent to pulling faces in opposite directions

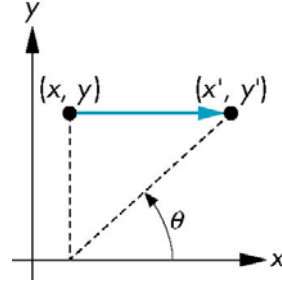


Shear Matrix

Consider simple shear along x axis

$$\begin{aligned}x' &= x + y \cot \theta \\y' &= y \\z' &= z\end{aligned}$$

$$\mathbf{H}(\theta) = \begin{bmatrix} 1 & \cot \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Summary: Transformations

- Introduce standard transformations
 - Rotation
 - Translation
 - Scaling
 - Shear
- Derive homogeneous coordinate transformation matrices
- Learn to build arbitrary transformation matrices from simple transformations (see book for more details)

OpenGL Transformations

Objectives

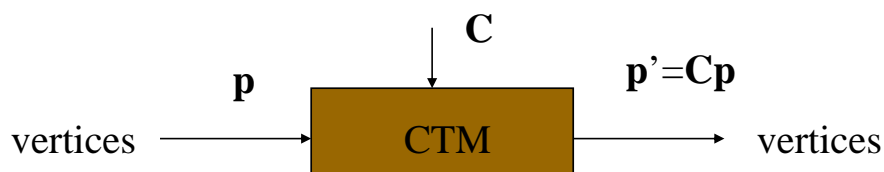
- Learn how to carry out transformations in OpenGL
 - Rotation
 - Translation
 - Scaling
 - Introduce OpenGL matrix modes
 - Model-view
 - Projection
-

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 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: $C \leftarrow CS$

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: $C = TRT^{-1}$ which is **backwards**.

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

Reversing the Order

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

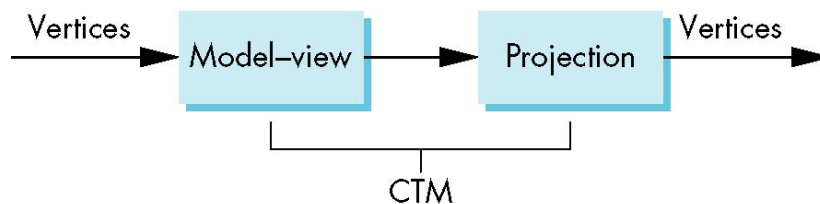
$C \leftarrow I$
 $C \leftarrow C T^{-1}$
 $C \leftarrow C R$
 $C \leftarrow C T$

Each operation corresponds to one function call in the program.

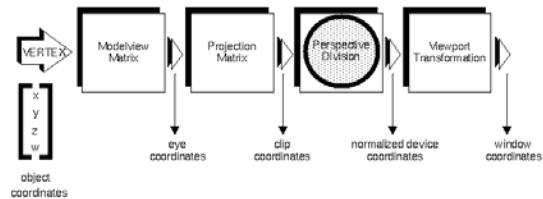
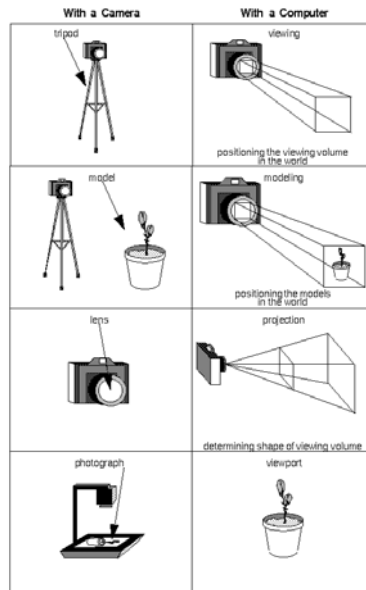
Note that the last operation specified is the first executed in the program

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



Transformation and Viewing



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**)

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
 - Access 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
-

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

```
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

After Class

- Read
 - Chapter 4
 - Review book materials and prepare for midterm
-