CS418
Discussion Section
(II)
MP1 & Matrix Transformation

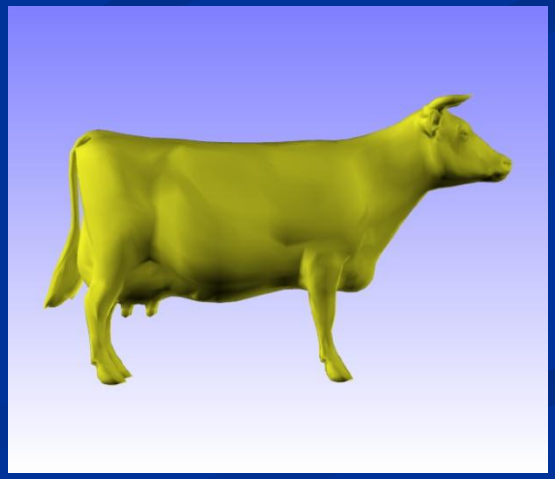
Presented by : Wei-Wen Feng
2/4/2009

MP1 : Mesh Rendering

- Due on February 16 (Mon)
  - Object display (30%)
  - Object coloring (10%)
  - **Interactive viewing (40%)**
  - Efficiency (10%)
  - Documentation (10%)

Load Your Mesh

- Customized .obj 3D models **with colors.**
- Won't work with a obj reader code.
- You should have skills to write a simple parser loading the files.



Mesh File Format

Position v1
Position v2

Color v1
Color v2

```
v 0.0 0.0 0.0
v 1.0 0.0 0.0
…..

vc 1 0 0
vc 0 0 1
…..

f 1 2 3
f 1 3 4
…..
```

- You will have a list of vertex attributes : Positions (v), Colors (vc), etc.
- Vertices are indexed according to their orders in file.
- Another indexed list for triangles (f)
  - Ex : Triangle 1 is formed by vertex v1,v2 and v3

Exercise

```
v 0.0 0.0 0.0
v 1.0 0.0 0.0
v 1.0 1.0 0.0
v 0.0 1.0 0.0

vc 1 0 0
vc 0 0 1
vc 1 0 0
vc 0 1 0

f 1 2 3
f 1 3 4
```

Draw the object from the given vertex/face list :

Mesh Structure

- Face-index List :
  - Recommend to use/implement basic matrix/vector structure and operations. (ex : **libgfx**)
  - Store vertex attributes in one array
  - Store face-vertex indices in another array.
  - When rendering, iterate through each face, and grab vertex attributes based on Indices.
  - More complicated structure is possible → Half-Edge, etc.

Display Your Mesh

- Assuming you've set up the view/projection transformation.
- Display one triangles

```
glBegin(GL_TRIANGLES);
glVertex3f(x1,y1,z1);
glVertex3f(x2,y2,z2);
glVertex3f(x3,y3,z3);
glEnd();
```

- glBegin→Decide which primitive you will display.
  - GL_POINTS, GL_LINES, GL_TRIANGLES, etc.
- Display a mesh is similar, just go through each triangle in the mesh. (Put loop between glBegin/glEnd)

Color Your Mesh

- glColor3f→Set R,G,B color
  - Range from 0.0~1.0. (1.0,1.0,1.0) is white.
- Use the provided colors, or generate your own.

- Ex : Color one triangle with Red, Green, Blue at each vertex

```
glBegin(GL_TRIANGLES);
glColor3f(1.0,0.0,0.0); //red
glVertex3f(x1,y1,z1);
glColor3f(0.0,1.0,0.0); // green
glVertex3f(x2,y2,z2);
glColor3f(0.0,0.0,1.0); // blue
glVertex3f(x3,y3,z3);
glEnd();
```

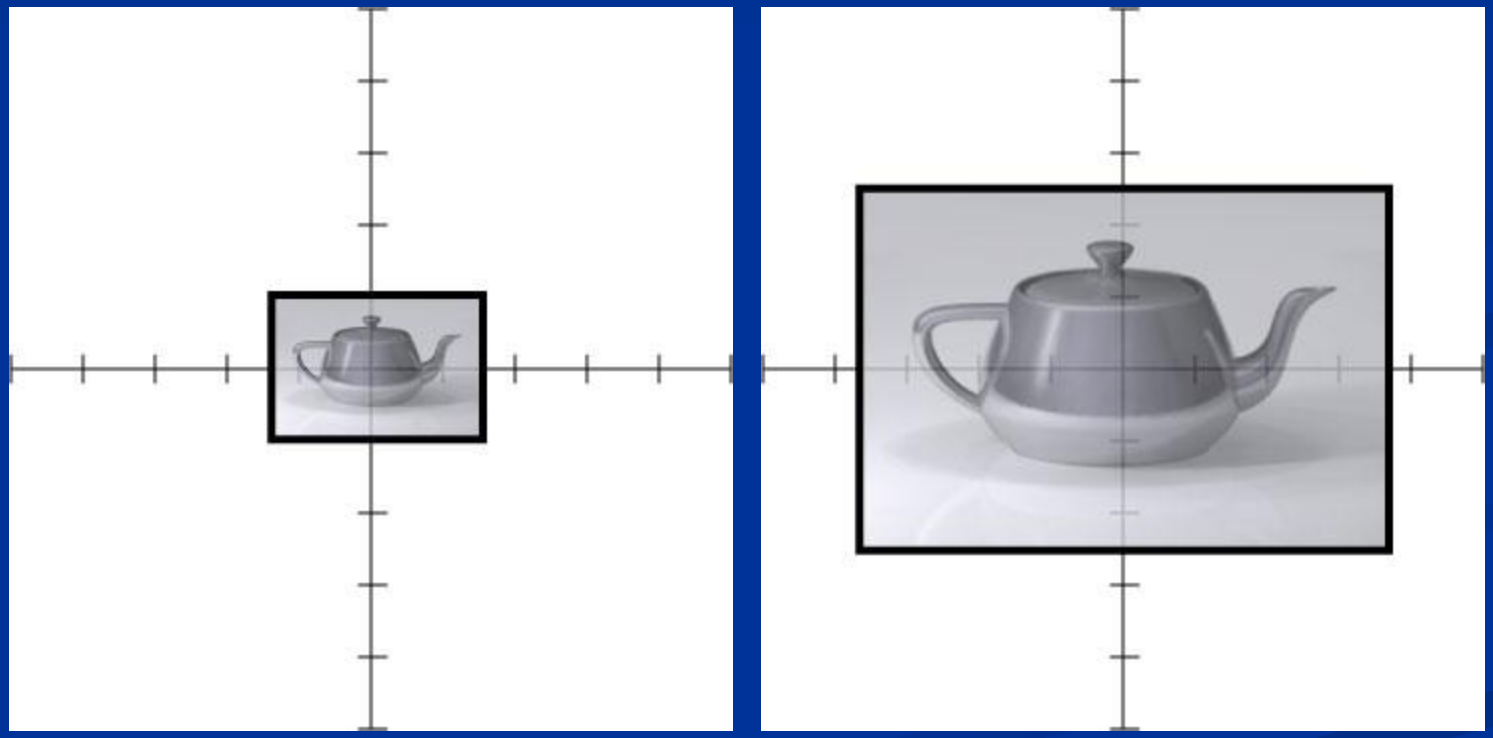
OpenGL Matrix Transformation

- Essential for interactive viewing/animation.

- Things to Take Home
  - #1 : You are modifying a global "current matrix"
  - #2 : The "last" transformation gets applied "first".
  - #3 : OpenGL store matrix in "Column Major"

Review of Matrix Ops.

- glScalef (2.5, 2.5, 1.0);

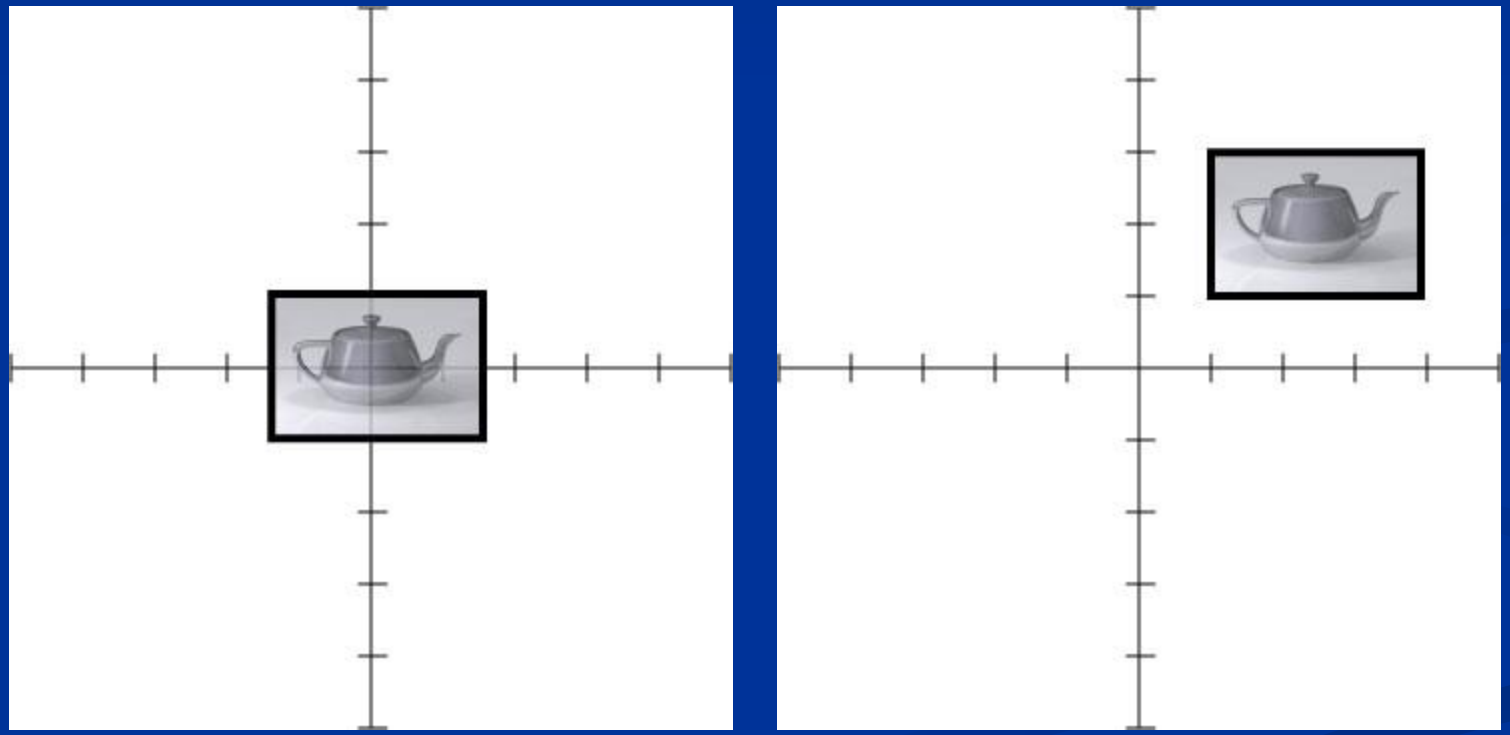


Scaling

$$\begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax \\ by \\ cz \\ 1 \end{bmatrix}$$

Translation

- glTranslatef(2.5,2.0,0.0);

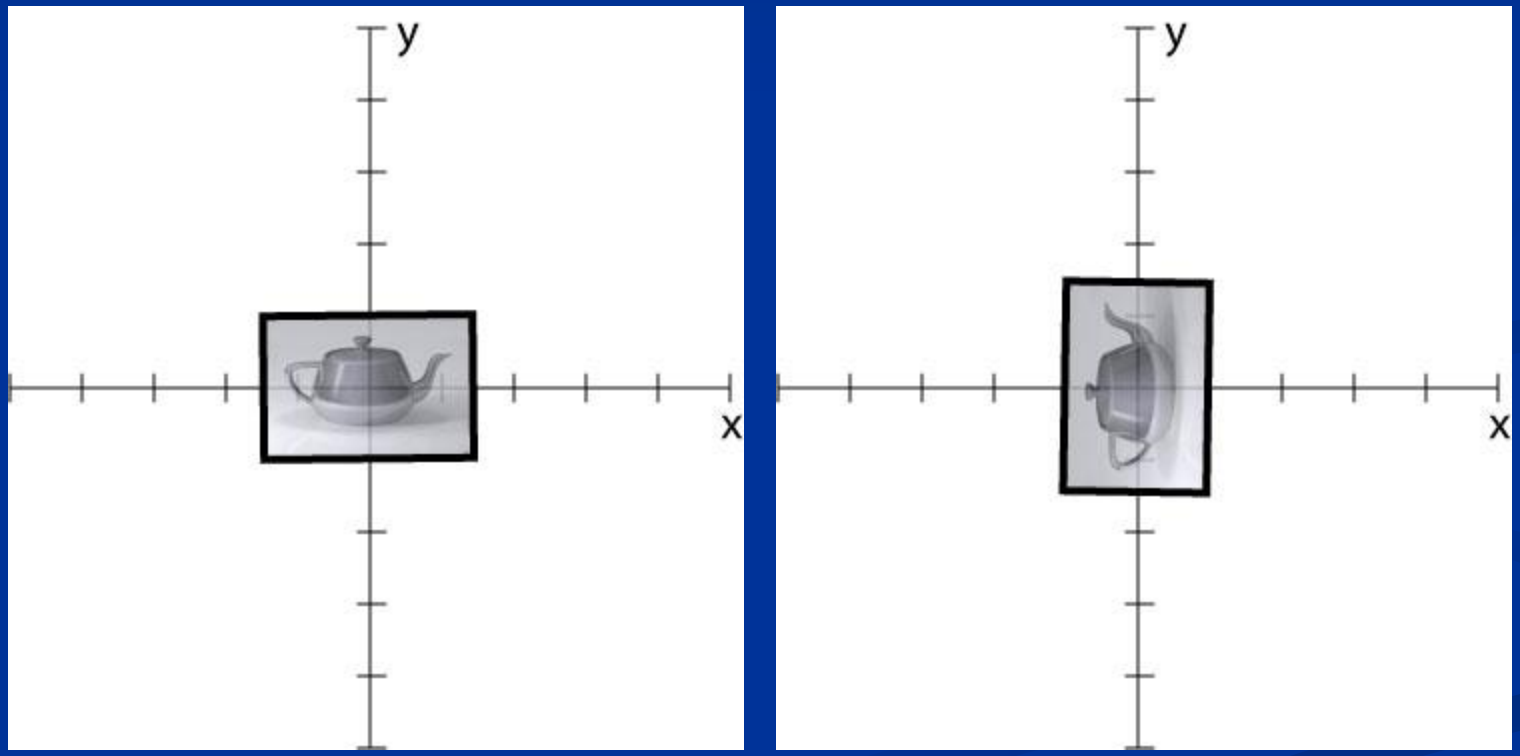


Translation

$$\begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+a \\ y+b \\ z+c \\ 1 \end{bmatrix}$$

Rotation

- glRotatef(90.0, 0.0, 0.0, 1.0)



Rotation

- About x-axis
  - rotates $y \rightarrow z$

$$\begin{bmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{bmatrix}$$

- About y-axis
  - rotates $z \rightarrow x$

$$\begin{bmatrix} \cos\theta & & \sin\theta & \\ & 1 & & \\ -\sin\theta & & \cos\theta & \\ & & & 1 \end{bmatrix}$$

- About z-axis
  - rotates $x \rightarrow y$

$$\begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

- You may also specify rotation about an arbitrary axis.

#1 Current Matrix

- An OpenGL matrix operation affects a **global 4x4 matrix.**
- It is the top matrix in the matrix stack you are currently working on. → **glMatrixMode**

```
glMatrixMode(GL_MODEL_VIEW)
glRotatef(1.0,0.0,0.0,1.0);
```

Current Matrix →

Model View Matrix

M1=M1*R

```
glMatrixMode(GL_PROJECTION)
gluPerspective(…);
```

Projection Matrix

M2=M2*P

#1 Current Matrix

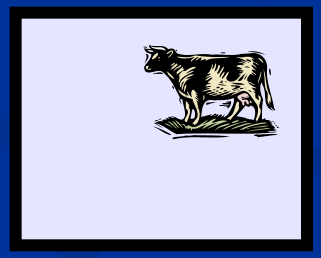
- When rendering, both of them are combined to transform the object.

Object V

MVP

Transform

V_Transform



Projection Matrix

Model View Matrix

MVP = (Projection)*(Model View)

V_Transform = MVP * V

#2 Last Transform Applied First

- OpenGL Post-multiply new transformation with current matrix when we call glRotate, glTranslate, or glScale.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The last transformation is applied first to the object.

```
glLoadIdentity();
glRotatef(1.0,0.5,0.5,0.5);
glTranslatef(0.5,0.5,0.5);
glRotatef(1.0,0.5,0.5,0.5);
```

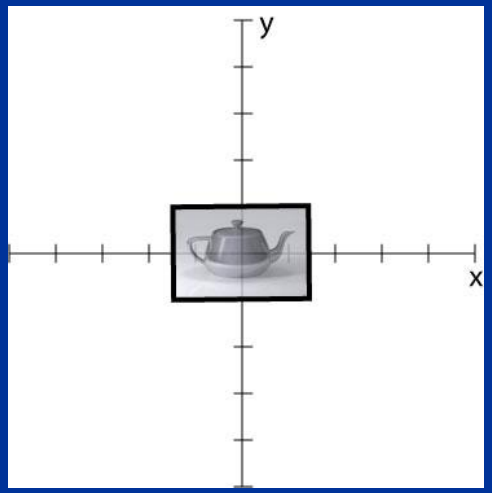
Transformation :

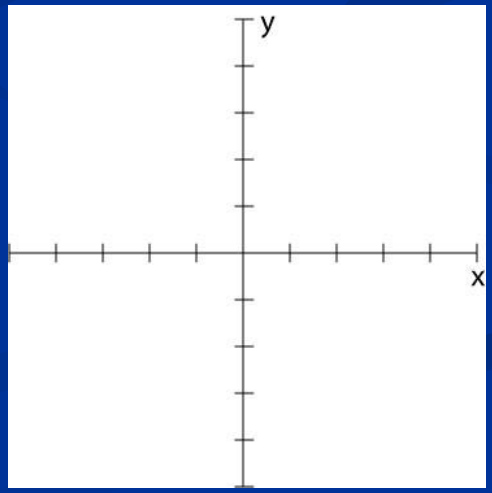
M=IRTR

Exercise

Draw the result of the following OpenGL transformation code.

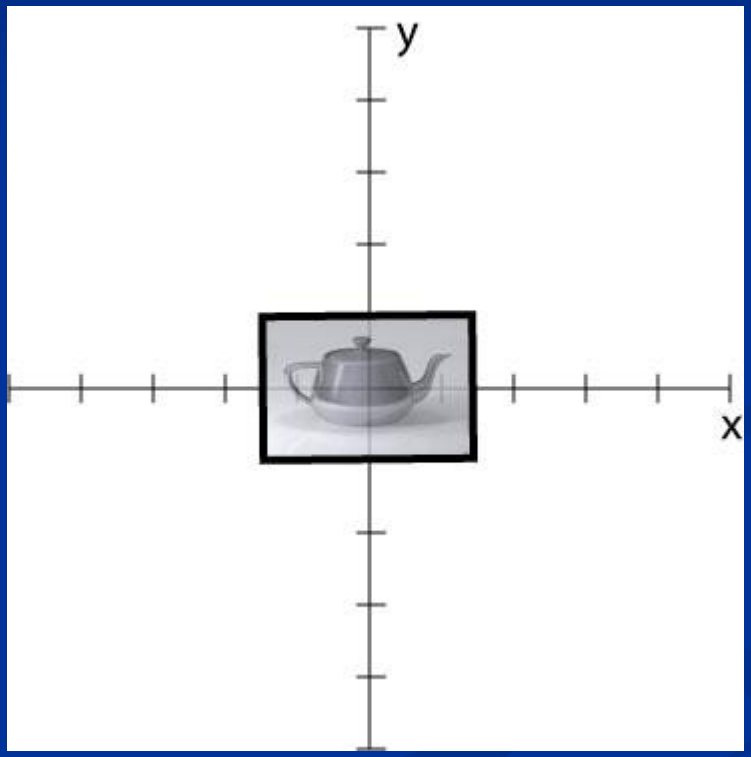
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();      // draws the teapot
```





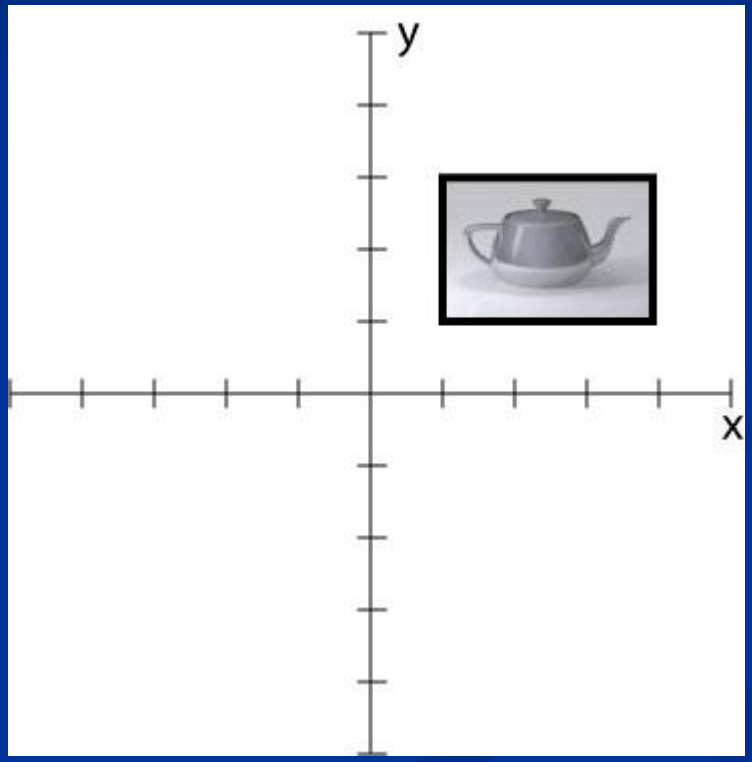
Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



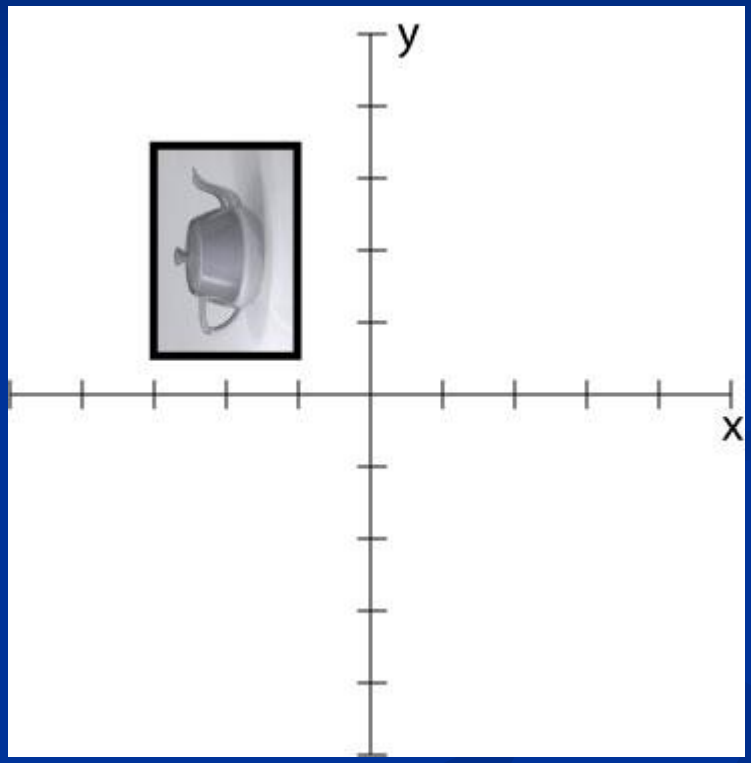
Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



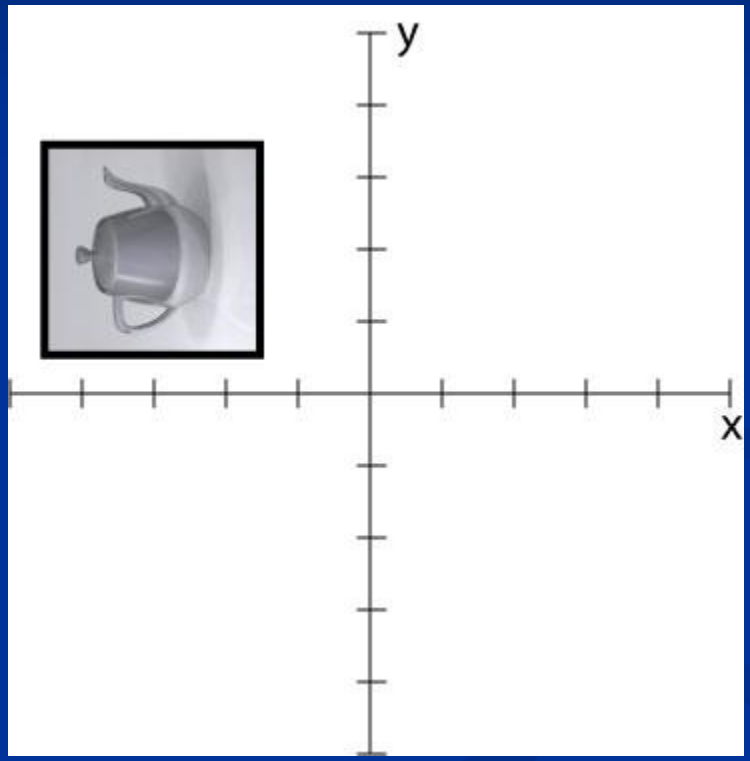
Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



Exercise

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glScalef(1.5, 1.0, 1.0);
glRotatef(90.0, 0.0, 0.0, 1.0);
glTranslatef(2.0, 2.0, 0.0);
draw_teapot_image();
```



Useful OpenGL Matrix Ops.

- glLoadIdentity : M = I

- glScale : M = MS
- glTranslate : M = MT
- glRotate : Specify rotation axis, angle. M = MR

Useful OpenGL Matrix Ops.

- glLoadMatrix(**M0**) : M = M0
- glGetFloat(**MatrixMode,M0**) : Get values of M and store them in M0.
- glMultMatrix(**M0**) : M = M*M0

- Caveat : OpenGL store matrix in **"Column Major"** instead of **"Row Major"**

#3 Column Major

- glLoadMatrix(**M0**) : M = M0
- glGetFloat(**MatrixMode,M0**) : Get values of M and store them in M0.
- glMultMatrix(**M0**) : M = M*M0

- Caveat : OpenGL store matrix in **"Column Major"** instead of **"Row Major"**

#3 Column Major

Given a 1D array of 16 floats :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

2D array in C :

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Matrix in OpenGL :

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Pre-multiply ?

- What to do if you want to pre-multiply the matrix ?

M=RM ?

- Make use of "glGetFloat" & "glMultMatrix".

```
glLoadIdentity();
glTranslatef(0.3,0.3,0.2);
glGetFloat(MODEL_VIEW,tempM);
glLoadIdentity();
glRotatef(1.0,0.0,0.0,1.0);
glMultMatrix(tempM);
```

tempM=M=I T

M=IRtempM

- Useful for updating transformation with UI control.

Q&A

- Demo
  - Massive Model Rendering