Discussion Session 2

Sikun LIN
sikun@ucsb.edu

Today’s Topic

- Rendering Pipeline
  - Modeling transformation
  - Viewing transformation
  - Projection transformation
- Library hierarchy

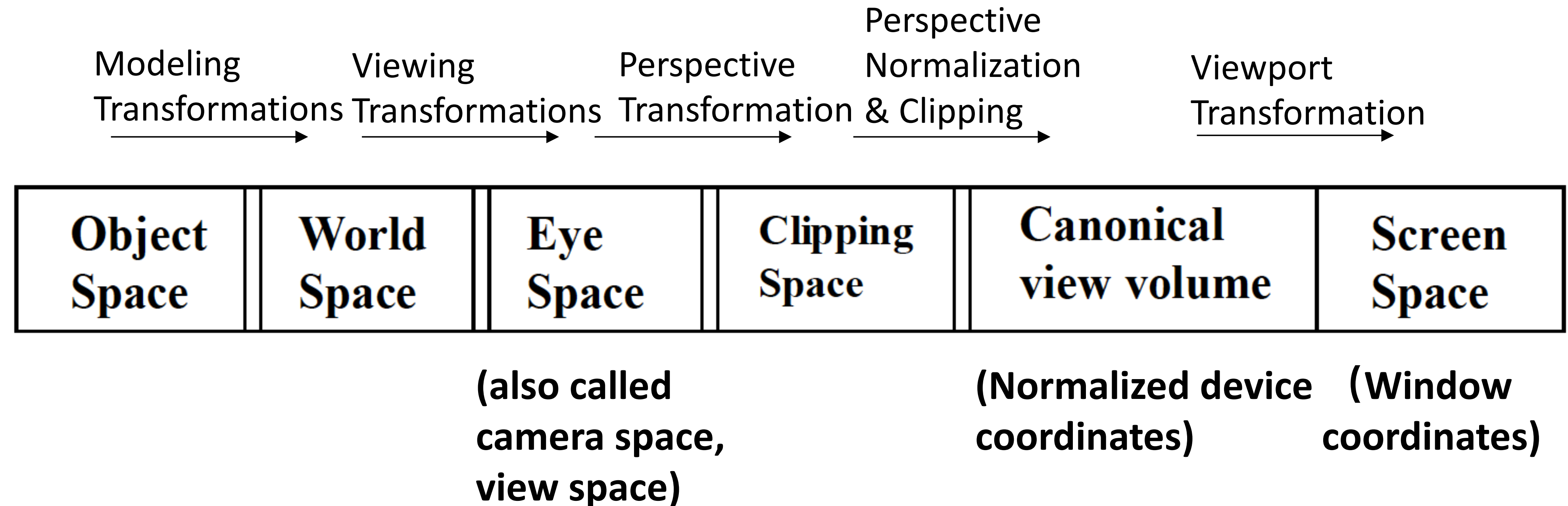
Rendering Pipeline

Object Space	World Space	Eye Space	Clipping Space	Canonical view volume	Screen Space

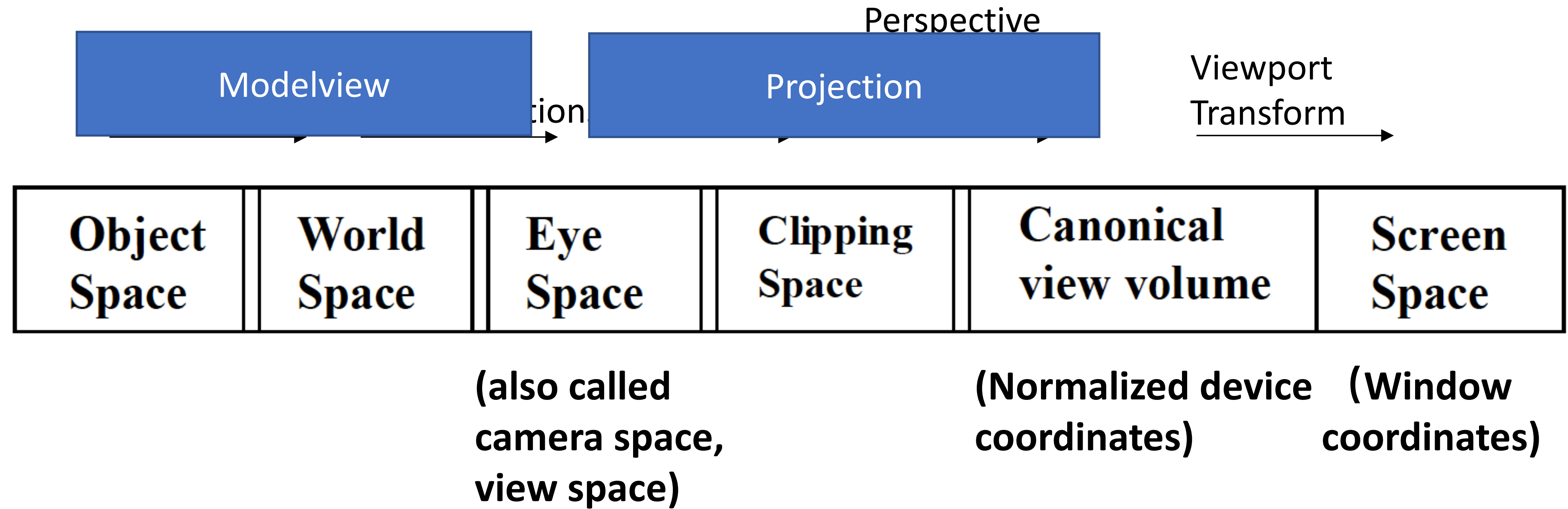
- **Object space**: coordinate space where each object is defined
- **World space**: all objects put together into the same 3D scene via affine transformations. (camera, lighting defined in this space)
- **Eye space**: camera at the origin, view direction coincides with the z axis. Near and far planes perpendicular to the z axis
- **Clipping space**: apply perspective transformation, *but before division*. All points are in homogeneous coordinate, i.e., each point is represented by (x,y,z,w)
- **Canonical view volume** (3D image space): A parallelpiped shape. *Obtained after perspective division*. Objects in this space are distorted (farther are smaller)
- **Screen space**: x and y coordinates are pixel coordinates, z coordinate used for screen-space hidden surface removal

Rendering Pipeline (cont.)

What are the transformations for each step?



Rendering Pipeline (cont.)

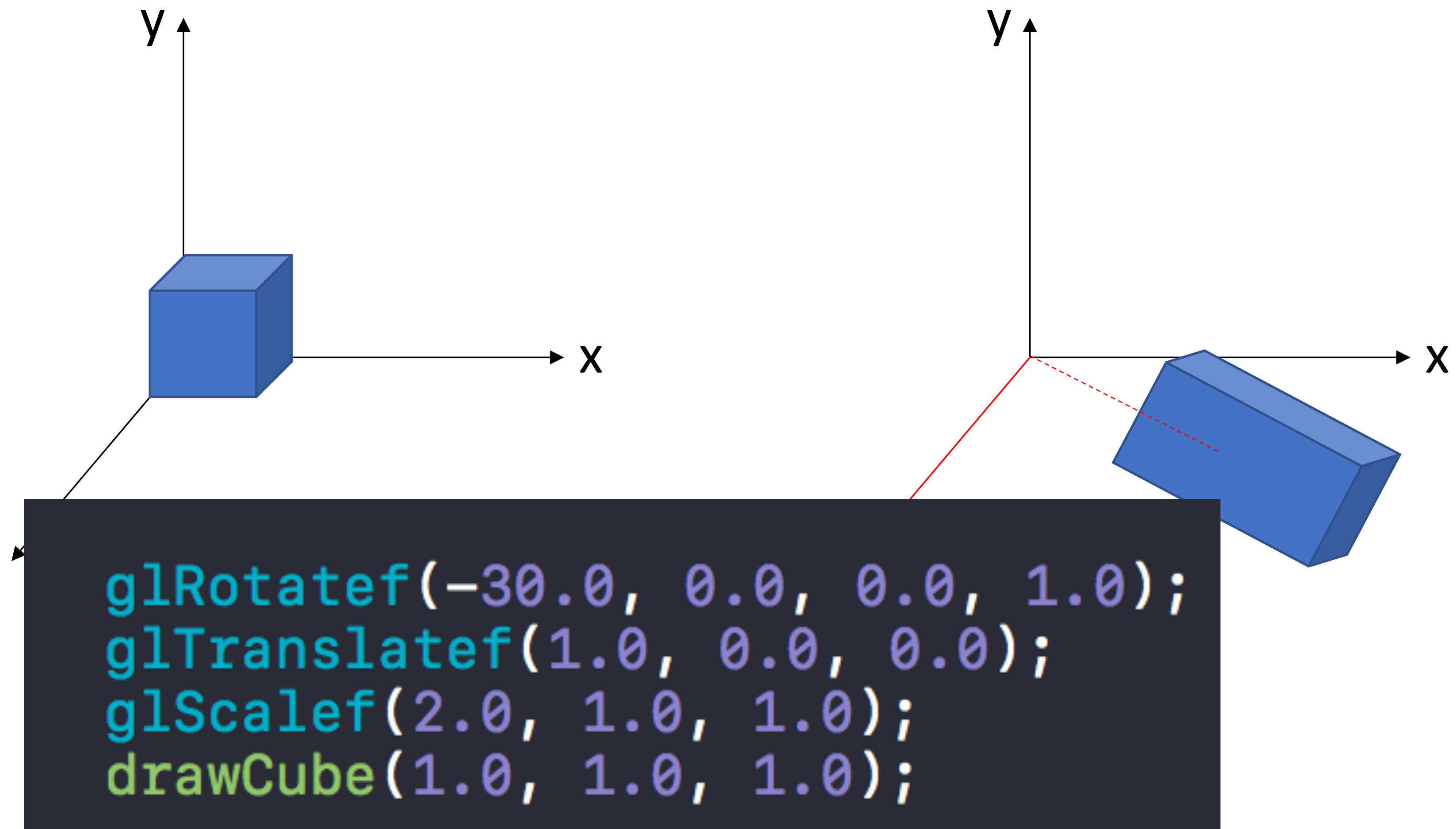


Modeling transformations:
Function choices

- Use OpenGL
  - glTranslate[f,d](x,y,z)
  - glRotate[f,d](angle,x,y,z)
  - glScale[f,d](x,y,z)
- Write your own M: T, R, S
  - glLoadMatrix[f,d](M)
  - glMultiMatrix[f,d](M)

Use OpenGL functions

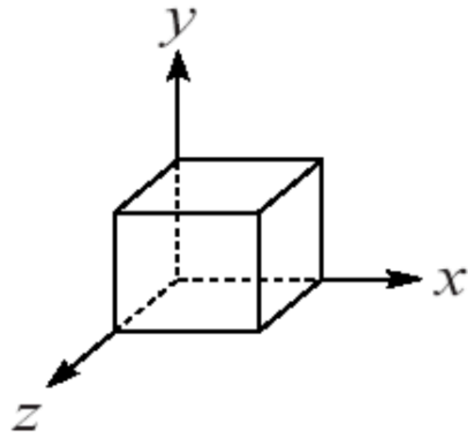
- Where?
- How?

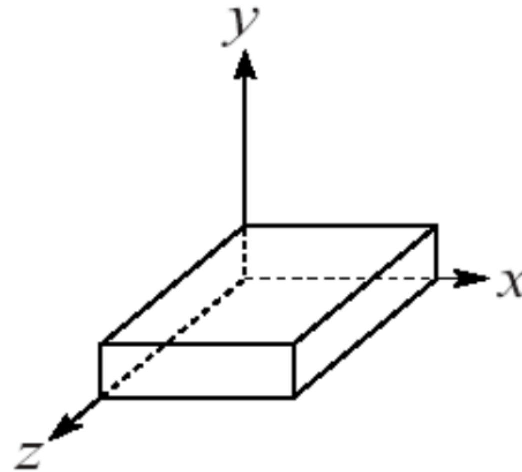


Write your own S, T, R

- Scaling (S)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

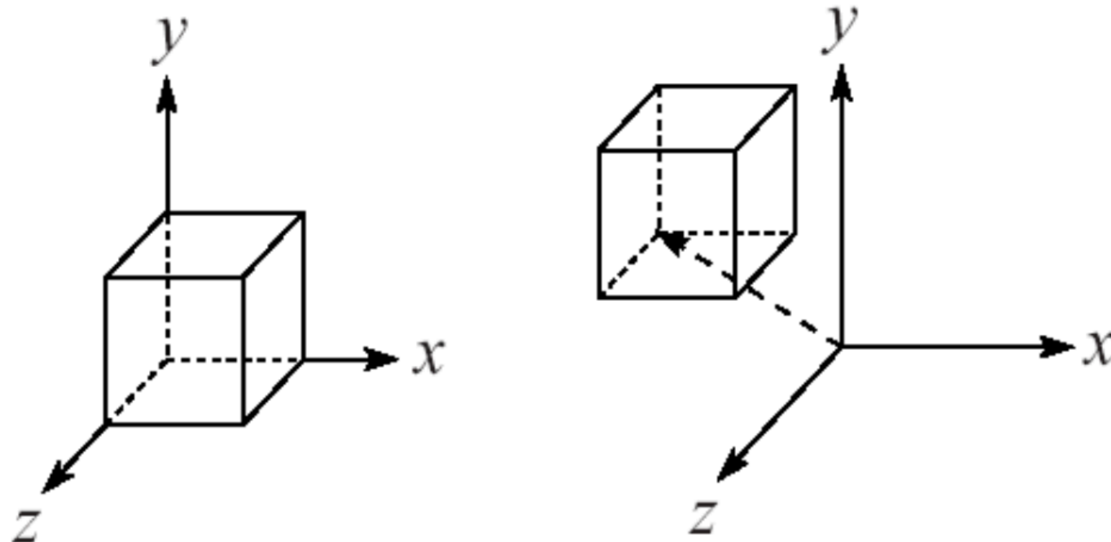




Write your own S, T, R (cont.)

- Translation (T)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Write your own S, T, R (cont.)

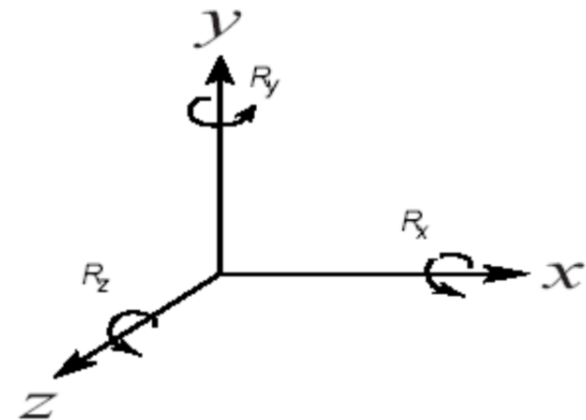
- Rotation (R)

Rotation now has more possibilities in 3D:

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

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

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



Use right hand rule

What are the corresponding OpenGL functions?

Write your own S, T, R (cont.)

- Rotation (R)

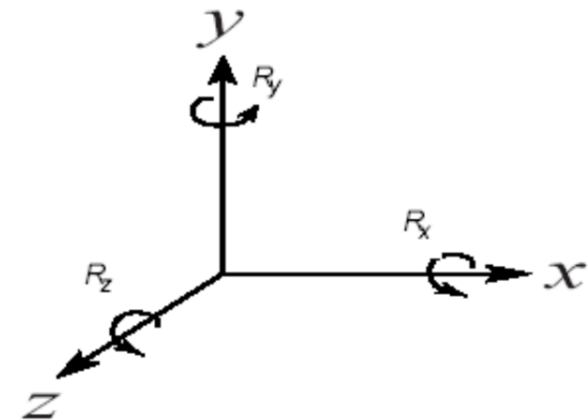
Rotation now has more possibilities in 3D:

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

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

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \cdots \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

glRotatef$(\theta \cdot 180/\pi, 0, 0, 1)$



Use right hand rule

What are the corresponding OpenGL functions?

Other affine transformations

- Reflection

$$F_x = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The cases for the other two coordinate frames are similar.

- Shearing

$$H_{yz}(h_y, h_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ h_y & 1 & 0 & 0 \\ h_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad H_{zx}(h_z, h_x) = \begin{pmatrix} 1 & h_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & h_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad H_{xy}(h_x, h_y) = \begin{pmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Other affine transformations

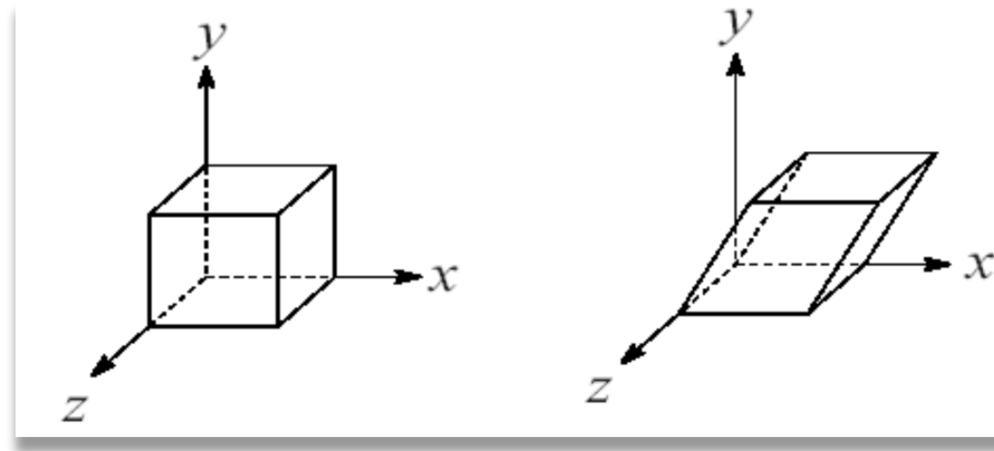
- Reflection

$$F_x = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

The cases for the other two coordinate frames are similar.

- Shearing

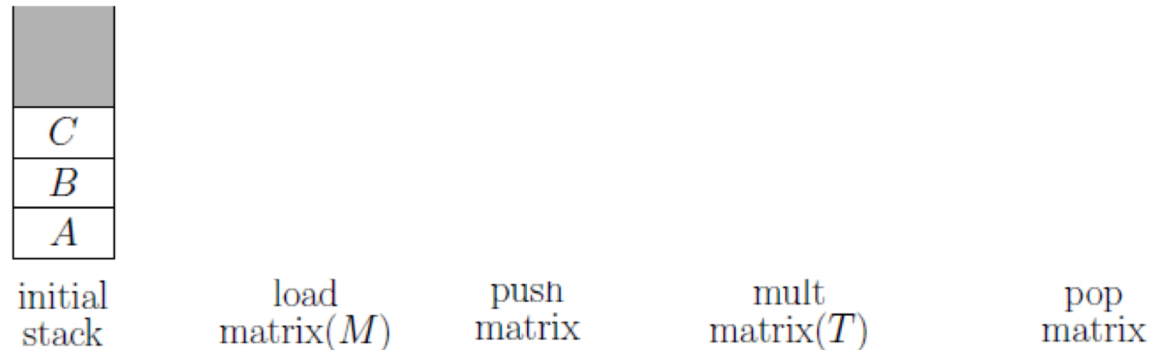
$$H_{yz}(h_y, h_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ h_y & 1 & 0 & 0 \\ h_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



$$H_{xy}(h_x, h_y) = \begin{pmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Matrix stack: load, push, and pop

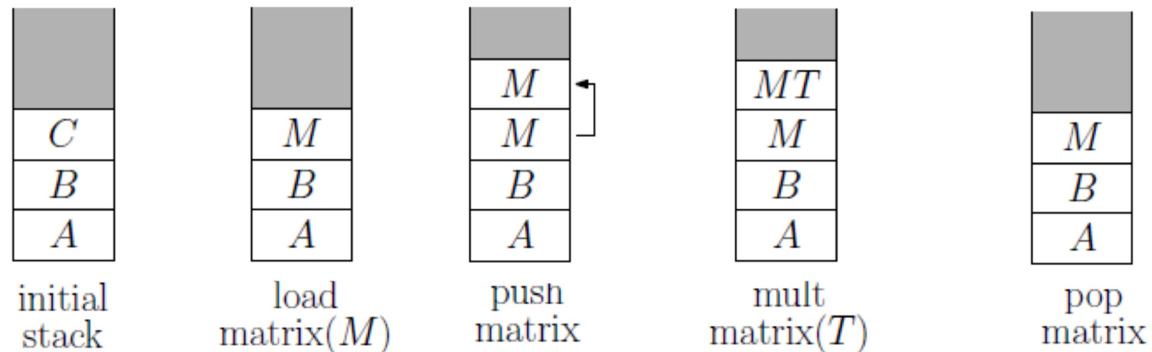
- glLoadMatrix(M) replaces the current matrix with the one whose elements are specified by M. The current matrix is the projection matrix, modelview matrix, or texture matrix, depending on the current matrix mode



Whenever you draw (e.g., using glRectf()), points are automatically transformed using the **top matrix** in Modelview matrix stack.

Matrix stack: load, push, and pop

- glLoadMatrix(*M*) replaces the current matrix with the one whose elements are specified by *M*. The current matrix is the projection matrix, modelview matrix, or texture matrix, depending on the current matrix mode

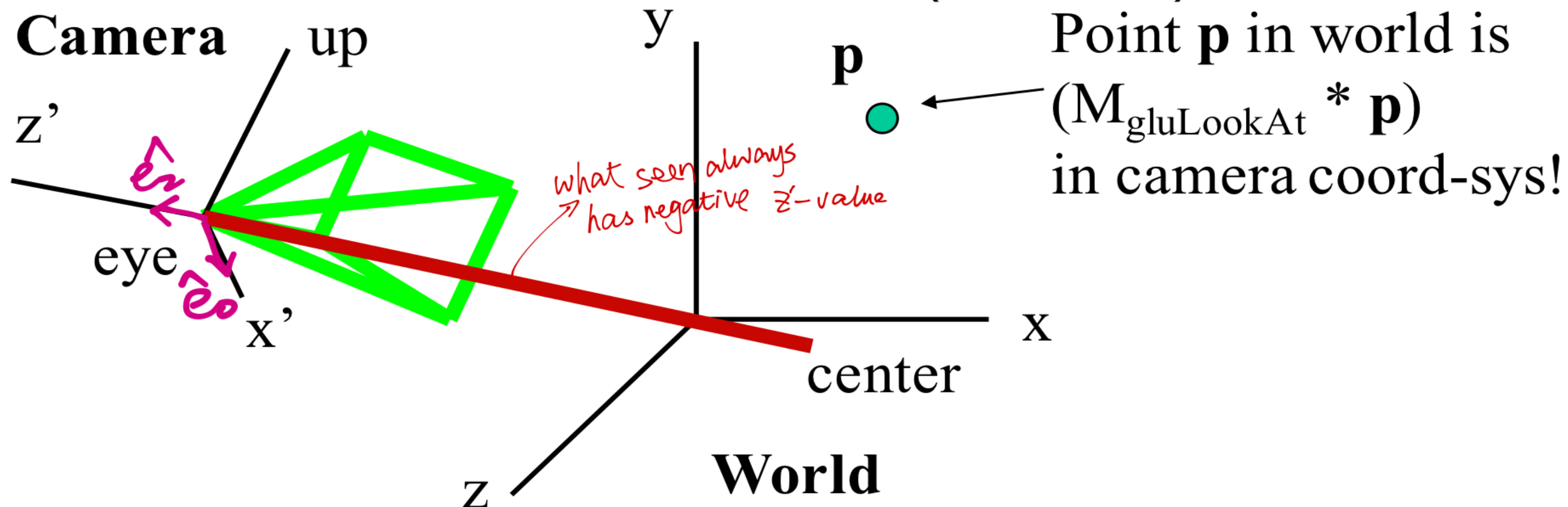


Whenever you draw (e.g., using glRectf()), points are automatically transformed using the **top matrix** in Modelview matrix stack.

Viewing Transformation: gluLookAt()

- **gluLookAt**(eye.x, eye.y, eye.z, center.x, center.y, center.z, up.x, up.y, up.z)
  - Viewing direction: center – eye
  - Up vector specifies orientation of camera

- These parameters define the **eye coordinate system**
  - Origin is at eye location
  - Z axis is opposite direction of viewing vector (e2 = normalize(eye – center))
  - X axis is normal to the plane spanned by view vector and up vector, pointing to the right of viewer (e0 = normalize(up x e2))
  - Y axis is orthonormal to x axis and z axis (e1 = e2 x e0)



Point **p** in world is ($M_{gluLookAt}$ * **p**) in camera coord-sys!

If no gluLookAt is specified …

- The viewing transformation matrix is identity matrix
  (i.e. eye coordinate system == world coordinate system)
  - Eye is at ______ of world space (origin)
  - Looking down the ______________ of world space (negative z axis)
  - Up vector is ______________ (positive y axis)

Summary of Modelview Transformation

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glRotatef(-30.0, 0.0, 0.0, 1.0);
glTranslatef(1.0, 0.0, 0.0);
glScalef(2.0, 1.0, 1.0);
drawCube(1.0, 1.0, 1.0);
```

$M_{world \to eye}$ R T S P ← local coord.

R T S P: world coord.

$M_{world \to eye}$ R T S P: eye coord.

calling order:
gluLookAt()
R
T
S

Projection Transformation

- Specified by the OpenGL commands such as gluOrtho2D, glOrtho, glFrustum, and gluPerspective.

- Perspective projection: glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-2.0, 2.0, -2.0, 2.0, 1.5, 20.0);
```

It's symmetric, so equivalently we can use …?

Projection Transformation

- Specified by the OpenGL commands such as gluOrtho2D, glOrtho, glFrustum, and gluPerspective.
- Perspective projection: glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)

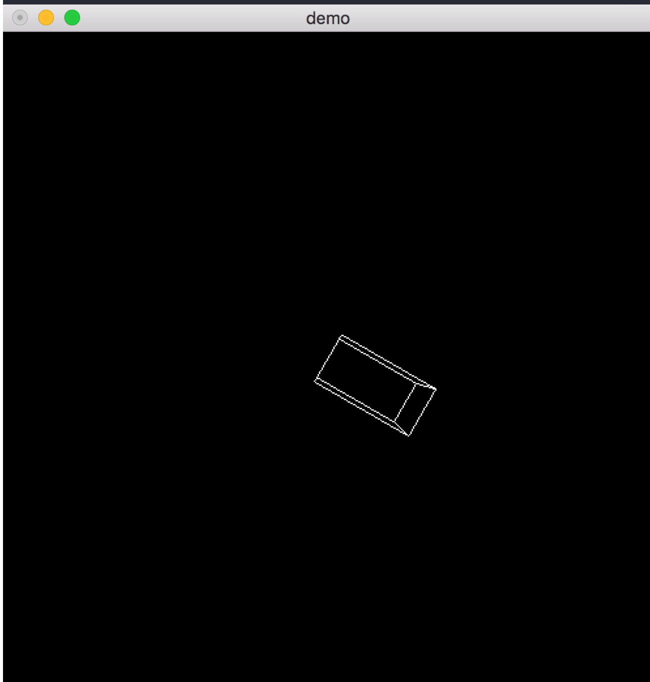
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-2.0, 2.0, -2.0, 2.0, 1.5, 20.0);
```

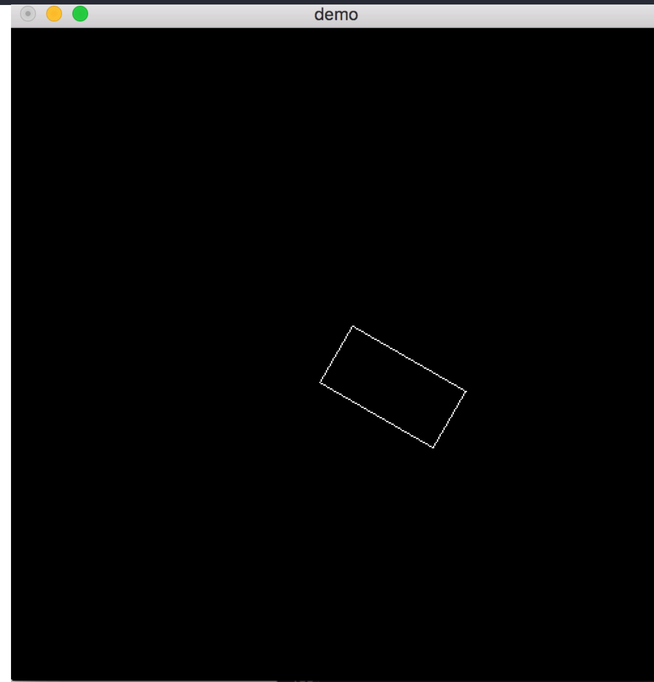
```
gluPerspective(106.26, 1, 1.5, 20.0);
```

Projection Transformation (cont.)

- Parallel Projection: glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-5.0, 5.0, -5.0, 5.0, 1.5, 20.0);
```





OpenGL Hierarchy

- Several levels of abstraction are provided
- GL
  - Lowest level: vertex, matrix manipulation
  - e.g., glVertex3f(point.x, point.y, point.z)
- GLU
  - Helper functions for shapes, transformations
  - e.g., gluPerspective(fovy, aspect, near, far)
- GLUT
  - Highest level: Window and interface management
  - e.g., glutSwapBuffers()

Q&A