# Advanced Computer Graphics

## 4 - Homogeneous Coordinates, 3D Affine Transformations

Yoonsang Lee
Fall 2018

# Today's Topics

- Composing Transformations
- Homogeneous Coordinates
- 3D Cartesian Coordinate System
- Transformations in 3D world

# Composing Transformations & Homogeneous Coordinates

# Composing Transformations

- Move an object, then move it some more

$$\mathbf{p} \rightarrow T(\mathbf{p}) \rightarrow S(T(\mathbf{p})) = (S \circ T)(\mathbf{p})$$

- **Composing 2D linear transformations** just works by **2x2 matrix multiplication**

$$T(\mathbf{p}) = M_T\mathbf{p}; S(\mathbf{p}) = M_S\mathbf{p}$$

$$(S \circ T)(\mathbf{p}) = M_S M_T \mathbf{p} = (M_S M_T)\mathbf{p} = M_S(M_T\mathbf{p})$$

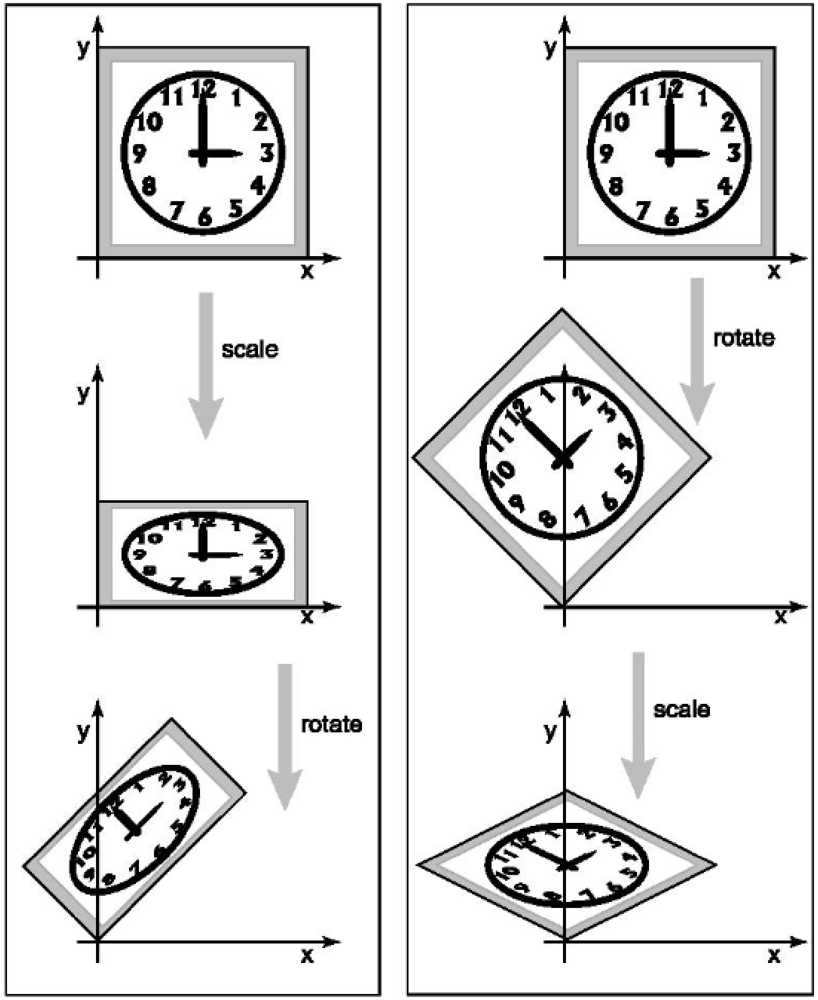
# Order Matters!

- Note that matrix multiplication is associative, but **not commutative**.

$$(AB)C = A(BC)$$

$$AB \neq BA$$

- As a result, the **order of transforms is very important.**



# [Practice] Composition

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        S = np.array([[1.,0.],
                      [0.,2.]])
        th = np.radians(60)
        R = np.array([[np.cos(th), -np.sin(th)],
                      [np.sin(th), np.cos(th)]])

        # compare results of these two lines
        render(R @ S)
        # render(S @ R)

        # ...
```

# Problems when handling Translation as Vector Addition

- Cannot treat linear transformation (rotation, scale,…) and translation in a consistent manner.

- Composing affine transformations is complicated

$$T(\mathbf{p}) = M_T\mathbf{p} + \mathbf{u}_T \qquad (S \circ T)(\mathbf{p}) = M_S(M_T\mathbf{p} + \mathbf{u}_T) + \mathbf{u}_S$$

$$S(\mathbf{p}) = M_S\mathbf{p} + \mathbf{u}_S \qquad \qquad = (M_S M_T)\mathbf{p} + (M_S\mathbf{u}_T + \mathbf{u}_S)$$

- We need a cleaner way!

➡ **Homogeneous coordinates**

# Homogeneous Coordinates

- Key idea: Represent 2D points in 3D coordinate space
- Extra component *w* for vectors, extra row/column for matrices
  - For points, can always keep $w = 1$
  - 2D point x, y -> 3D vector $[x, y, 1]^T$.
- Linear transformations are represented as:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \\ 1 \end{bmatrix}$$

# Homogeneous Coordinates

- Translations are represented as:

$$\begin{bmatrix} 1 & 0 & \boxed{t} \\ 0 & 1 & \boxed{s} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t \\ y + s \\ 1 \end{bmatrix}$$

- Affine transformations are represented as:

linear part

$$\begin{bmatrix} m_{11} & m_{12} & u_x \\ m_{21} & m_{22} & u_y \\ 0 & 0 & 1 \end{bmatrix}$$

translational part

# Homogeneous Coordinates

- **Composing affine transformations** just works by **3x3 matrix multiplication**

$$T(\mathbf{p}) = M_T\mathbf{p} + \mathbf{u}_T$$
$$S(\mathbf{p}) = M_S\mathbf{p} + \mathbf{u}_S$$

$$T(\mathbf{p}) = \begin{bmatrix} \overset{\text{2x2}}{M_S} & \overset{\text{2x1}}{\mathbf{u}_S} \\ 0 & 1 \end{bmatrix} \qquad S(\mathbf{p}) = \begin{bmatrix} \overset{\text{2x2}}{M_T} & \overset{\text{2x1}}{\mathbf{u}_T} \\ 0 & 1 \end{bmatrix}$$

# Homogeneous Coordinates

- **Composing affine transformations** just works by **3x3 matrix multiplication**

$$
\begin{aligned}
(S \circ T)(\mathbf{p}) &= \begin{bmatrix} \overset{\text{2x2}}{M_S} & \overset{\text{2x1}}{\mathbf{u}_S} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \overset{\text{2x2}}{M_T} & \overset{\text{2x1}}{\mathbf{u}_T} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \overset{\text{2x1}}{\mathbf{p}} \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} (M_S M_T)\mathbf{p} + (M_S \mathbf{u}_T + \mathbf{u}_S) \\ 1 \end{bmatrix}
\end{aligned}
$$

- Much cleaner

# [Practice] Homogeneous Coordinates

```
def render(T):
    # ...
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv( (T @ np.array([.0,.5,1.]))[:-1] )
    glVertex2fv( (T @ np.array([.0,.0,1.]))[:-1] )
    glVertex2fv( (T @ np.array([.5,.0,1.]))[:-1] )
    glEnd()
```

# [Practice] Homogeneous Coordinates

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        # rotate 60 deg about z axis
        th = np.radians(60)
        R = np.array([[np.cos(th), -np.sin(th),0.],
                      [np.sin(th), np.cos(th),0.],
                      [0.,         0.,        1.]])

        # translate by (.4, .1)
        T = np.array([[1.,0.,.4],
                      [0.,1.,.1],
                      [0.,0.,1.]])

        render(R)
        # render(T)
        # render(T @ R)
        # render(R @ T)
        # ...
```

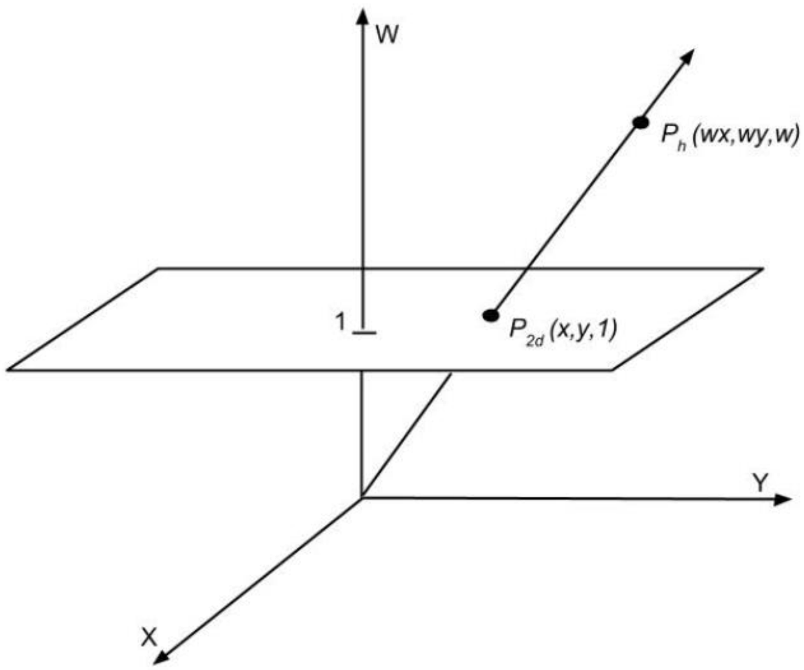
# Summary: Homogeneous Coordinates in 2D

- Use $\mathbf{(x,y,1)^T}$ instead of $(x,y)^T$ for **2D points**
- Use **3x3 matrices** instead of 2x2 matrices **for 2D linear transformations**
- Use **3x3 matrices** instead of vector additions **for 2D translations**

- -> We can treat linear transformations and translations **in a consistent manner!**

# Intuition for Homogeneous Coordinates

- Homogeneous coord.: 2D point x, y-> 3D vector $[x, y, 1]^T$.




- The plane w=1 is our xy plane.
- Translation in our xy plane is “shear” in this xyw space.

# Intuition for Homogeneous Coordinates

2x2 shear matrix in Cartesian coordinate

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$

Shear in x

3x3 translation matrix in homogeneous coordinate

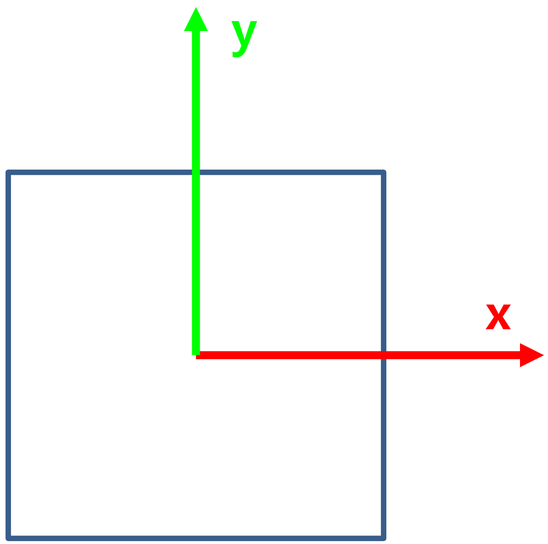
$$\begin{bmatrix} 1 & 0 & t \\ 0 & 1 & s \\ 0 & 0 & 1 \end{bmatrix}$$

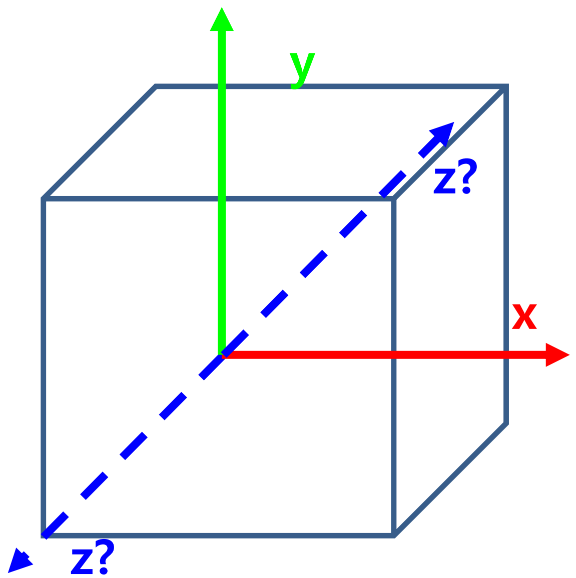
Shear in x, y
(in homogeneous coordinate)

- That's why both linear transformation and translation can be represented as "linear transformation" in homogeneous coordinate.

# 3D Affine Transformations

# Now, Let's go to the 3D world!

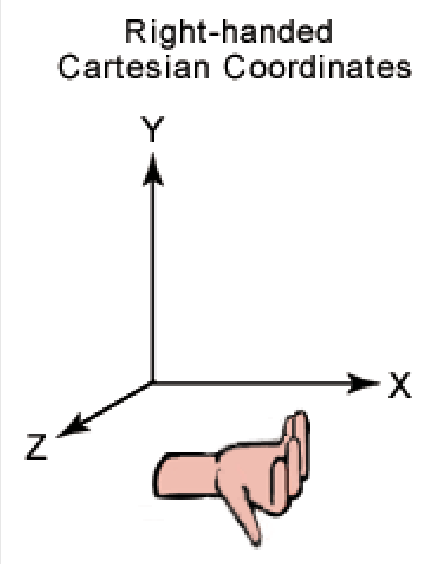
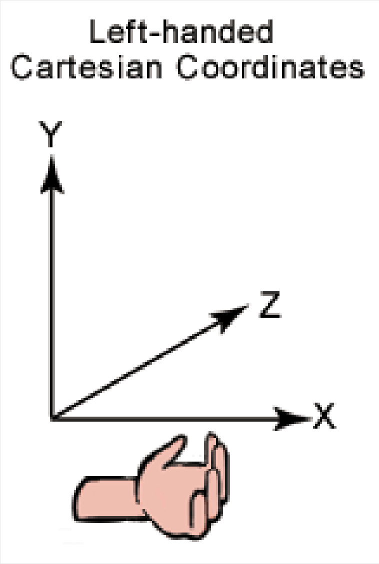



- Coordinate system (좌표계)
  - Cartesian coordinate system (직교좌표계)

# Two Types of 3D Cartesian Coordinate Systems

What we're using

| | | |
|---|---|---|
| |  |  |
| **Positive rotation** direction | **counterclockwise** about the axis of rotation  | **clockwise** about the axis of rotation  |
| Used in… | **OpenGL**, Maya, Houdini, AutoCAD, …<br>Standard for Physics & Math | DirectX, Unity, Unreal, … |

# Point Representation in Cartesian & Homogeneous Coordinate System

| | Cartesian coordinate system | Homogeneous coordinate system |
|---|---|---|
| A **2D point** is represented as… | $\begin{bmatrix} p_x \\ p_y \end{bmatrix}$ | $\begin{bmatrix} p_x \\ p_y \\ \boxed{1} \end{bmatrix}$ |
| A **3D point** is represented as… | $\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$ | $\begin{bmatrix} p_x \\ p_y \\ p_z \\ \boxed{1} \end{bmatrix}$ |

# Review of Linear Transform in 2D

- Linear transformation in **2D** can be represented as matrix multiplication of …

**2x2 matrix**
(in Cartesian coordinates)

or

**3x3 matrix**
(in homogeneous coordinates)

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & 0 \\ m_{21} & m_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

# Linear Transformation in 3D

- Linear transformation in **3D** can be represented as matrix multiplication of …

**3x3 matrix**
(in Cartesian coordinates)

or

**4x4 matrix**
(in homogeneous coordinates)

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

# Linear Transformation in 3D

**Scale:**

3D

$$\mathbf{S_s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 \\ 0 & \mathbf{S}_y & 0 \\ 0 & 0 & \mathbf{S}_z \end{bmatrix}$$

3D-H

$$\mathbf{S_s} = \begin{bmatrix} \mathbf{S}_x & 0 & 0 & 0 \\ 0 & \mathbf{S}_y & 0 & 0 \\ 0 & 0 & \mathbf{S}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Shear (in x, based on y,z position):**

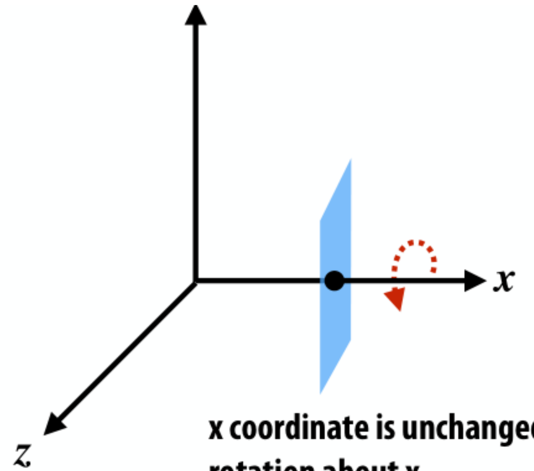
$$\mathbf{H}_{x,\mathbf{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_{x,\mathbf{d}} = \begin{bmatrix} 1 & \mathbf{d}_y & \mathbf{d}_z & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Linear Transformation in 3D

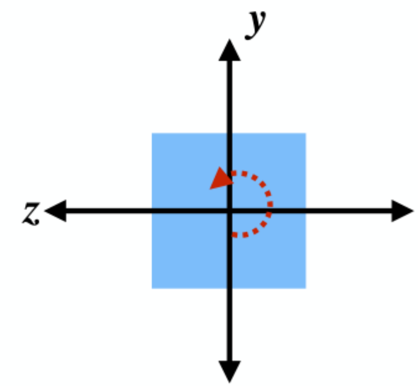
**Rotation about x axis:**

$$\mathbf{R}_{x,\theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$




x coordinate is unchanged by rotation about x

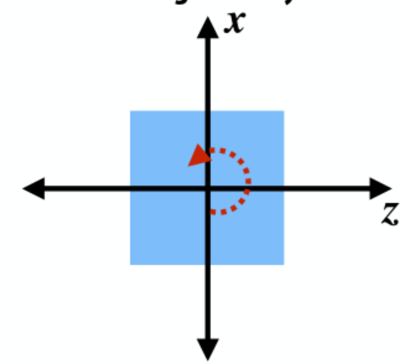
View looking down -x axis:




**Rotation about y axis:**

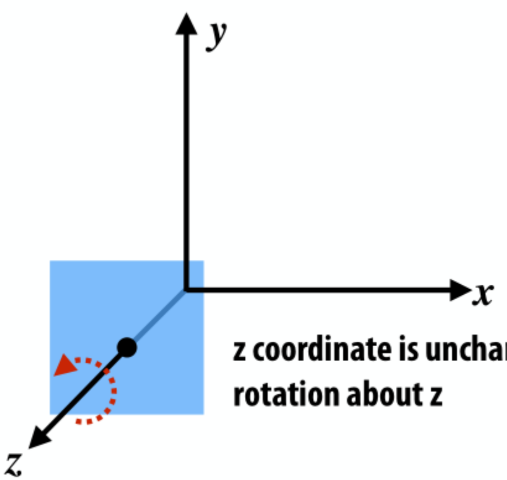
$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ \boxed{-}\sin\theta & 0 & \cos\theta \end{bmatrix}$$

View looking down -y axis:




**Rotation about z axis:**

$$\mathbf{R}_{z,\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$




z coordinate is unchanged by rotation about z

# Review of Translation in 2D

- Translation in **2D** can be represented as …

**Vector addition**
(in Cartesian coordinates)

$$T(\mathbf{p}) = \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} u_x \\ u_y \end{bmatrix}$$

Matrix multiplication of
**3x3 matrix**
(in homogeneous coordinates)

$$\begin{bmatrix} 1 & 0 & u_x \\ 0 & 1 & u_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

# Translation in 3D

- Translation in **3D** can be represented as …

**Vector addition**
(in Cartesian coordinates)

$$T(\mathbf{p}) = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$

Matrix multiplication of
**4x4 matrix**
(in homogeneous coordinates)

$$\begin{bmatrix} 1 & 0 & 0 & u_x \\ 0 & 1 & 0 & u_y \\ 0 & 0 & 1 & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

# Review of Affine Transformation in 2D

- In homogeneous coordinates, **2D** affine transformation can be represented as multiplication of **3x3 matrix**:

linear part

$$\begin{bmatrix} m_{11} & m_{12} & u_x \\ m_{21} & m_{22} & u_y \\ 0 & 0 & 1 \end{bmatrix}$$

translational part

# Affine Transformation in 3D

- In homogeneous coordinates, **3D** affine transformation can be represented as multiplication of **4x4 matrix**:

linear part

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translational part

# [Practice] 3D Transformations

```
import glfw
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np

def render(M, camAng):
    # enable depth test (we'll see details
later)
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glLoadIdentity()

    # use orthogonal projection (we'll see
details later)
    glOrtho(-1,1, -1,1, -1,1)

    # rotate "camera" position to see this
3D space better (we'll see details later)
    gluLookAt(.1*np.sin(camAng),.1,
.1*np.cos(camAng), 0,0,0, 0,1,0)
```

```
    # draw coordinate: x in red, y in
green, z in blue
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()

    # draw triangle
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex3fv((M @
np.array([.0,.5,0.,1.]))[:-1])
    glVertex3fv((M @
np.array([.0,.0,0.,1.]))[:-1])
    glVertex3fv((M @
np.array([.5,.0,0.,1.]))[:-1])
    glEnd()
```

```python
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640,“3D Trans", None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.swap_interval(1)
    count = 0
    while not glfw.window_should_close(window):
        glfw.poll_events()

        # rotate 60 deg about x axis
        th = np.radians(-60)
        R = np.array([[1.,0.,0.,0.],
          [0., np.cos(th), -np.sin(th),0.],
          [0., np.sin(th), np.cos(th),0.],
                      [0.,0.,0.,1.]])

        # translate by (.4, 0., .2)
        T = np.array([[1.,0.,0.,.4],
                      [0.,1.,0.,0.],
                      [0.,0.,1.,.2],
                      [0.,0.,0.,1.]])

        camAng = np.radians(count% 360)
        render(R, camAng)
        # render(T, camAng)
        # render(T @ R, camAng)
        # render(R @ T, camAng)
        count += 1

        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
```

```python
def main():
    # ...
    glfw.swap_interval(1)
    count = 0
    while not glfw.window_should_close(window):
        glfw.poll_events()

        # rotate 60 deg about x axis
        th = np.radians(-60)
        R = np.identity(4)
        R[:3,:3] = [[1.,0.,0.],
                    [0., np.cos(th), -np.sin(th)],
                    [0., np.sin(th), np.cos(th)]]

        # translate by (.4, 0., .2)
        T = np.identity(4)
        T[:3,3] = [.4, 0., .2]

        camAng = np.radians(count % 360)
        render(R, camAng)
        # render(T, camAng)
        # render(T @ R, camAng)
        # render(R @ T, camAng)
        count += 1

        # ...
```

You can use **slicing** for cleaner code (the behavior is the same as the previous page)

# Next Time

- Reference Frame & Composite Trans., OpenGL Transformation Functions

- Next week:
  - Reference Frame & Composite Trans., OpenGL Transformation Functions
  - Affine Matrix, Hierarchical Modeling

- Assignment 2 (Due date: Sep 18, 23:59)


- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Taesoo Kwon, Hanyang Univ., http://calab.hanyang.ac.kr/cgi-bin/cg.cgi
  - Prof. Steve Marschner, Cornell Univ., http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml
  - Prof. Kayvon Fatahalian and Prof. Keenan Crane, CMU, http://15462.courses.cs.cmu.edu/fall2015/