# Computer Graphics

**Prof. Jibum Kim**

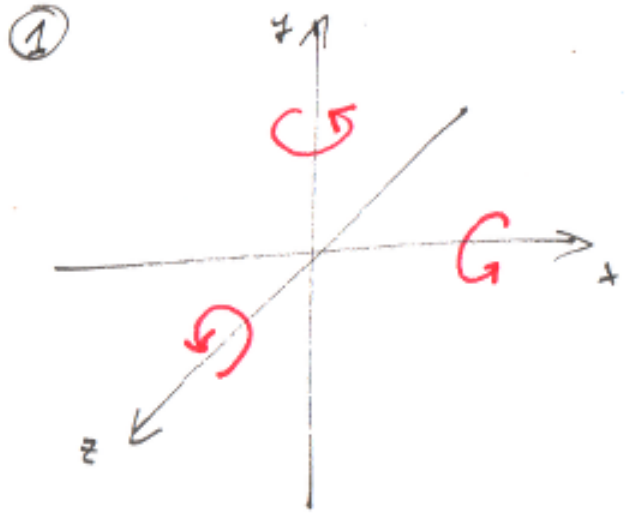
**Department of Computer Science & Engineering**

**Incheon National University**

- **3D rotation**

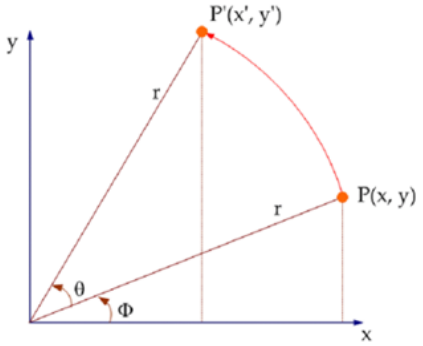
- We consider a rotation that is specified by (a) a directed line l through the origin, which is the axis of rotation and (b) by the angle $\theta$ of the rotation
- If a point P lies on the axis l itself, then it does not move
- We only consider a rotation about the Coordinate axes (x-axis, y-axis, and z-axis)
- Sometimes it is called x-roll, y-roll, z-roll

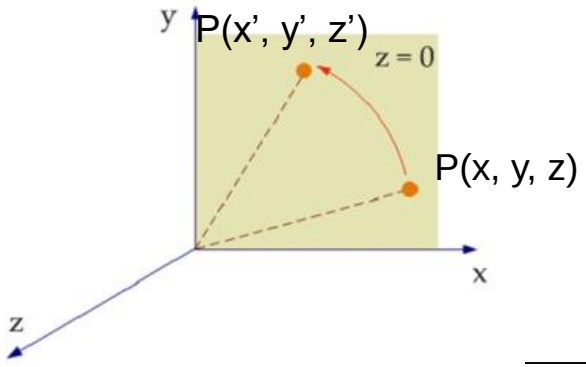
- **Right hand rule: 반시계 방향 회전 (Counter clock wise, CCW)**
- **Axis of rotation: 오른손 엄지 방향**
- **회전 방향: 오른손 나머지 손가락 방향 (반시계 방향 회전)**
- **예: 오른쪽 그림**
- **X축을 기준으로 CCW 회전**
- **Y축을 기준으로 CCW 회전**
- **z축을 기준으로 CCW 회전**



- **1. Rotation about the Z-axis**
- **예전에 배웠던 2D rotation (반시계 방향 회전)은 3D에서 생각하면 사실 회전축이 z축이면서 반시계 방향으로 회전한 것과 같은 것이다 (the x-axis rotates to the y-axis)**

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



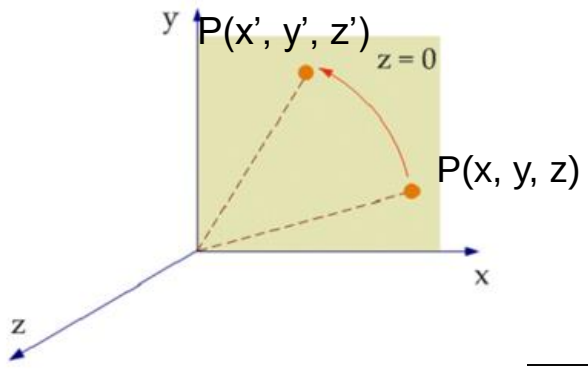



- **전개해보면**

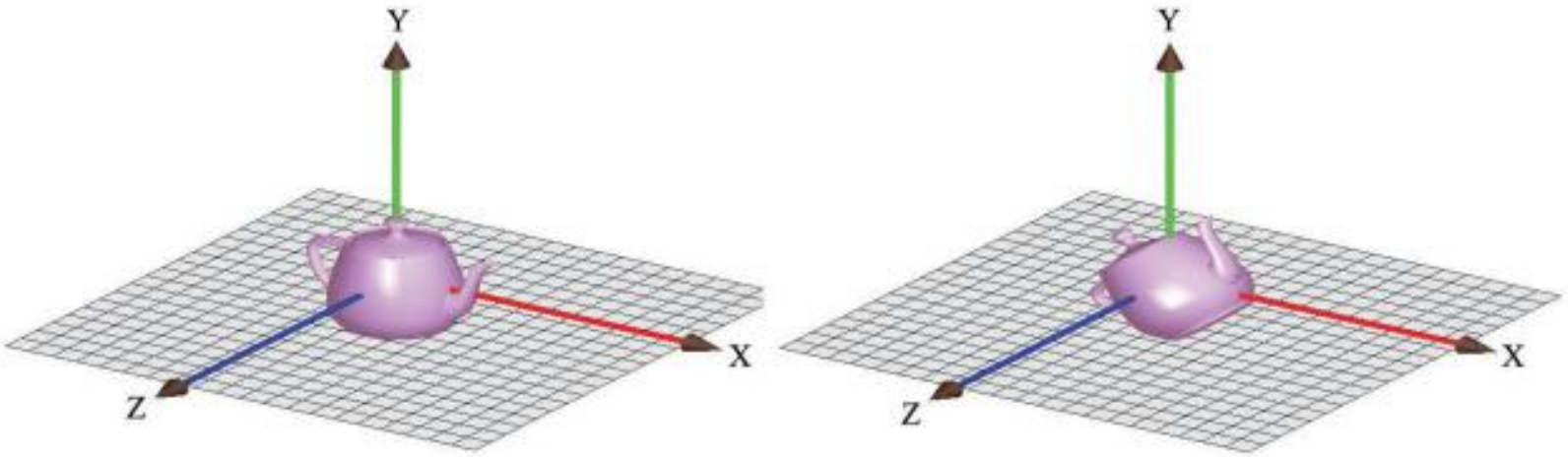
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- **z=z’ , x’=xcos$\theta$ -ysin$\theta$, y’=xsin$\theta$+ycos$\theta$**

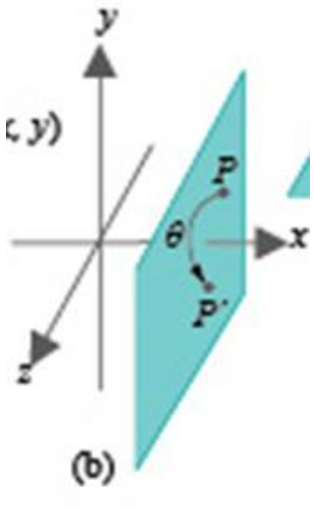
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- **Rotation about the z-axis by 45 degree**



- 2. Rotation about the x-axis, P(x, y, z) => P'(x', y', z')
- The y-axis rotates to the z-axis



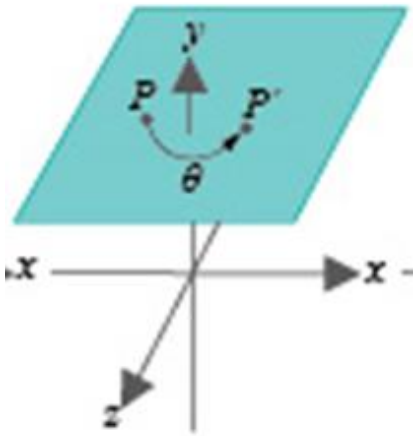

- 오른손가락 4개 (엄지 제외)를 회전시키면 엄지 방향이 회전축 방향
- Q) 1에서 x축에 해당하는 축은? Y축

  y축에 해당하는 축은? z축

- **Rotation about the x-axis by $\theta$**
- **P(x, y, z) => P'(x', y', z')**
- **Homogeneous coordinates으로 표현, P' = R P**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **3. Rotation about the y-axis, P(x, y, z) => P'(x', y', z')**
- **The z-axis rotates to the x-axis**



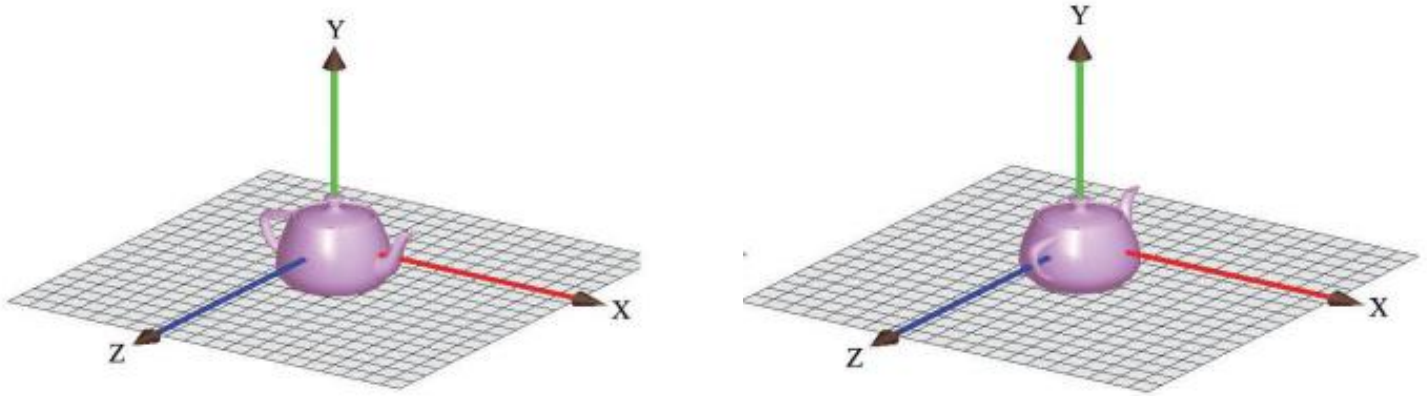

- **Q) 1에서 x축에 해당하는 축은? Z축**

  **y축에 해당하는 축은? X축**

- **Rotation about the x-axis by $\theta$**
- **P(x, y, z) => P'(x', y', z')**
- **Homogeneous coordinates으로 표현, P' = R P**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **Rotation about the x-axis by 45 degree**



- **In summary,**
- **x-roll:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **y-roll:**

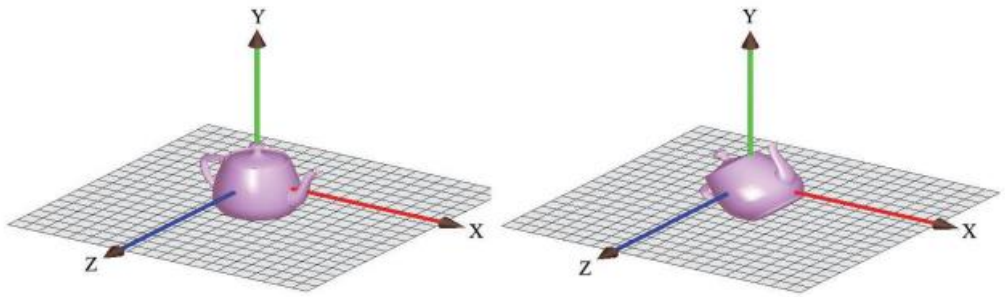
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **z-roll:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

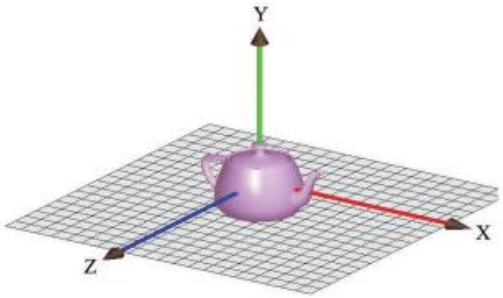
- **Rotation in OpenGL**

- **glRotatef(angle, x, y, z)**
- **The glRotatef function computes a matrix that performs a counterclockwise rotation of *angle* degrees about the vector from the origin through the point (*x*, *y*, *z*).**
- **예) glRotatef(45, 0, 0, 1)**

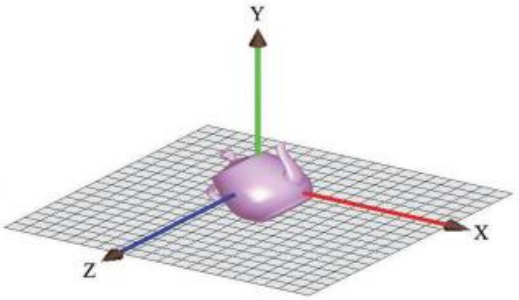



(A) glRotatef(0.0, 0.0, 0.0, 0.0); (B) glRotatef(45.0, 0.0, 0.0, 1.0);

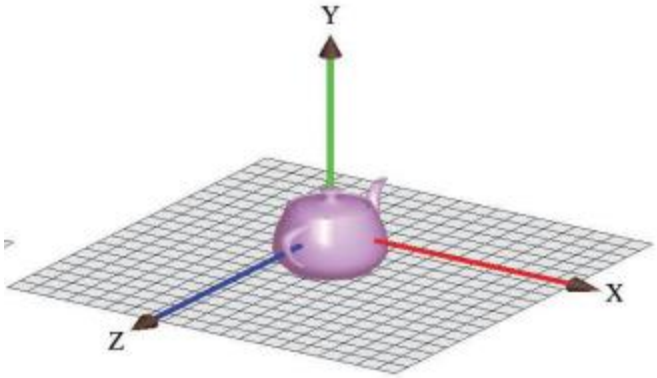
# 3D 공간에서 강체의 회전(Rotation) 구현 결과




(A) glRotatef(0.0, 0.0, 0.0, 0.0);




(B) glRotatef(45.0, 0.0, 0.0, 1.0);




(D) glRotatef(90.0, 0.0, 1.0, 0.0);

- // box.cpp에서
- glTranslatef(0.0, 0.0, -15.0);
- glRotatef(60.0, 0.0, 0.0, 1.0);
- glutWireTeapot(5.0);

Figure 4.11: Screenshot for Experiment 4.6.

- **glm library를 사용한 rotation**
- **OpenGL – Transformations**

```
glm::mat4 trans = glm::mat4(1.0f);
trans = glm::rotate(trans, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f));
```

The first line creates a new 4-by-4 matrix and initializes it to the identity matrix. The `glm::rotate` function multiplies this matrix by a rotation transformation of 180 degrees around the Z axis. Remember that since the screen lies in the XY plane, the Z axis is the axis you want to rotate points around.

To see if it works, let's try to rotate a vector with this transformation:

```
glm::vec4 result = trans * glm::vec4(1.0f, 0.0f, 0.0f, 1.0f);
printf("%f, %f, %f\n", result.x, result.y, result.z);
```

- **Opengl shear transformation**

- 이전 OpenGL 2.x에는 ‘glMultMatrix’ 함수가 있다

| void glMultMatrixf( | const GLfloat * *m*); |
|---|---|

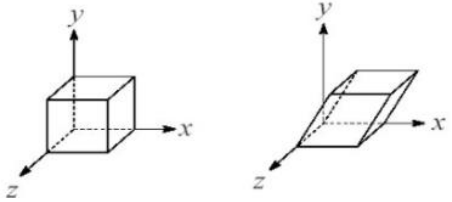
- ‘glMultiMatrix’ multiplies the current matrix with the one specified using m, and replaces the current matrix with the product.
- The current matrix is determined by the current matrix mode (i.e., glMatrixMode). It is either the projection matrix, modelview matrix, or the texture matrix

- 원래 정사각형 보양의 quad를 x-shear 변환 적용
- https://www.dropbox.com/s/sin5vb1j1trwf5m/shear.txt?dl=0

- 주의: OpenGL uses 4x4 matrix for transformations. Note that 16 elements in the matrix are stored as 1D array in column-major order
- static float matrixdata[16] = // x-shear transformation.
- {
- 1.0, 0.0 0.0, 0.0, 0.2, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0
- };

Shearing in 3D

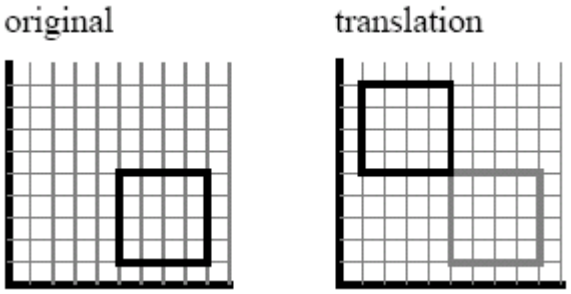
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & b & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

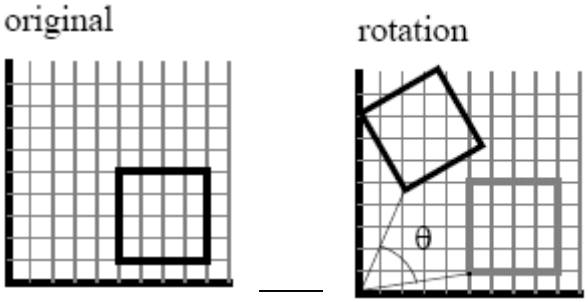


- 최신 OpenGL버전에서도 glm library를 사용하여 shear transformation가능

- **Classification of transformations**

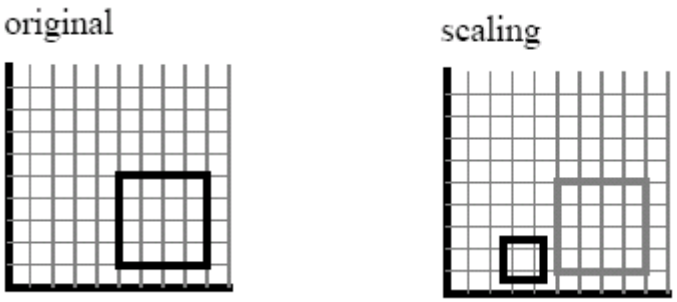
- **1. Euclidean transformation (rigid body transformation)**
- **It preserves size and shape**
- **Only allows Translation  and rotation**



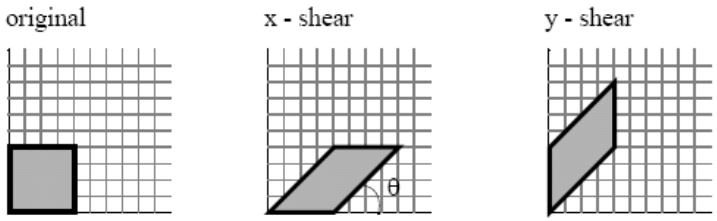



- **2. Similarity transformation (닮은 변환)**
- **Euclidean transformation + scale change**
- **Translation + rotation + scaling**

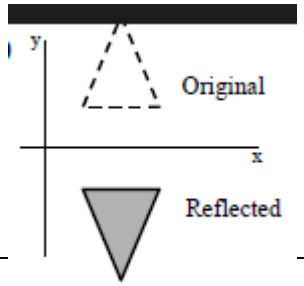
- **Example of similarity transformation**

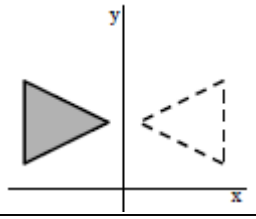


- **3. Affine transformation**
- **Allows Translation, rotation, scaling, shearing, reflection**
- **Shearing:**




- **Reflection**



- **Affine transformation의 특징**
- **Affine transformations preserve straightness, planarity, parallelism and convexity**
- **An affine transformation maps straight lines to straight lines and planes to planes. Moreover, it maps parallel straight lines to parallel straight lines, intersecting straight lines to intersecting straight lines, parallel planes to parallel planes and intersecting planes to intersecting planes**

- **Straightness, planarity, parallelism and convexity preserves**

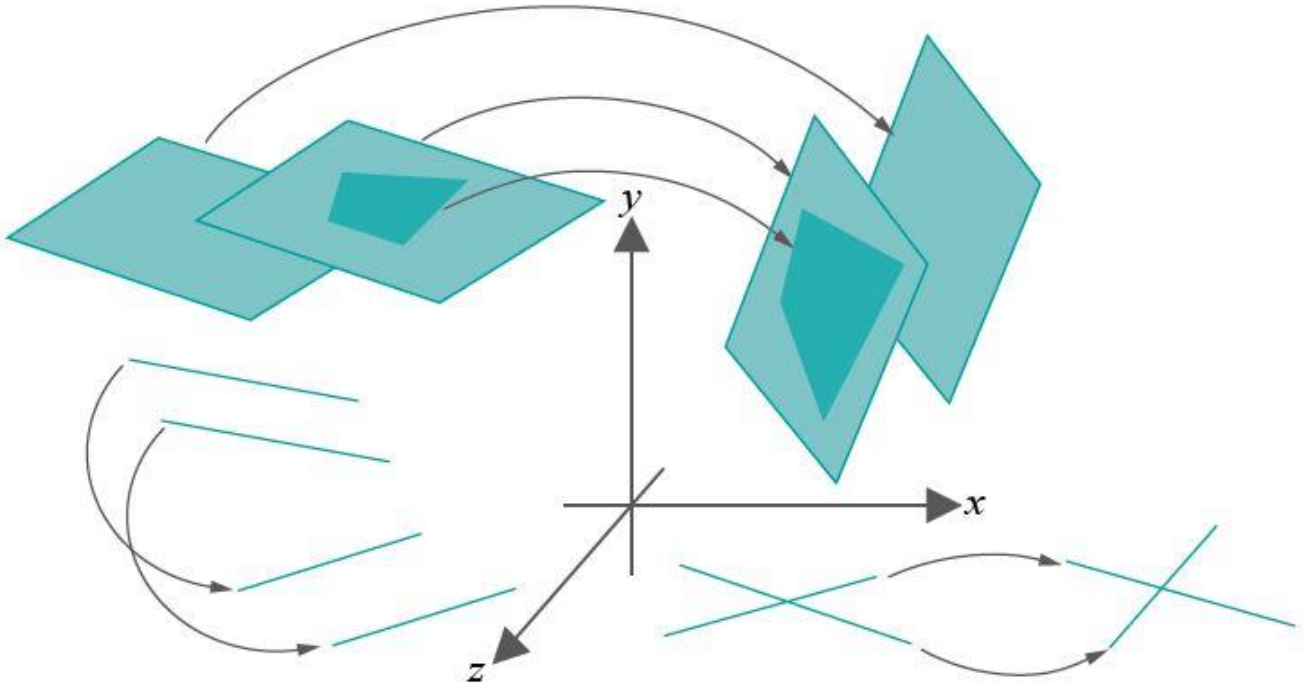



Figure 5.8: Affine transformation in $\mathbb{R}^3$.

- **Non-affine transformation의 예**

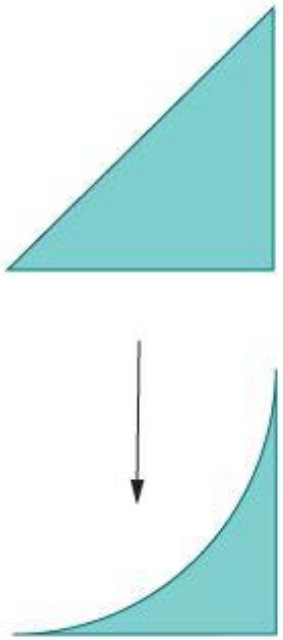


Figure 5.10: Non-affine transformation of a triangle.

- **Composing (concatenating) affine transformations**

- **Recall that composition of two affine transformations is another affine transformation, whose matrix is the product of the matrices of the individual affine transformations**
- **The matrices appear in reverse order to that in which the transformations are applied**
- **If we first apply $T_1$ with matrix $M_1$ and then apply $T_2$ with matrix $M_2$ to the result, the overall transformation has matrix $M_2M_1$**

- 예: Point (1, 2)에 대하여 다음 2D transformation이 다음 순서대로 수행되었을 때 변환 후의 위치를 구해보자
- 1. Rotate through 45 degrees about the origin
- 2. scale in x-axis by 1.5 and y-axis by -2
- 3. translate in x-axis by 3 and y-axis by 5

- 복합 변환시 변환의 순서의 중요성

- 다음을 생각해보자. 어떤 점 P에
- 1. T1 변환 후 T2 변환 적용
- 2. T2 변환 후 T1 변환 적용

1과 2는 같은 결과일까 행렬 곱을 생각해 보자

- $T2(T1 \cdot P),\ T1(T2 \cdot P)$
- 즉, 복합 변환 시에는 변환의 순서에 따라 완전히 다른 결과를 낳을 수 있다

- 다음의 두 변환이 있다고 하자
- 1) Rotate by 45$^{o}$ CCW in z-axis
- 2) Translate by 10 in x-axis

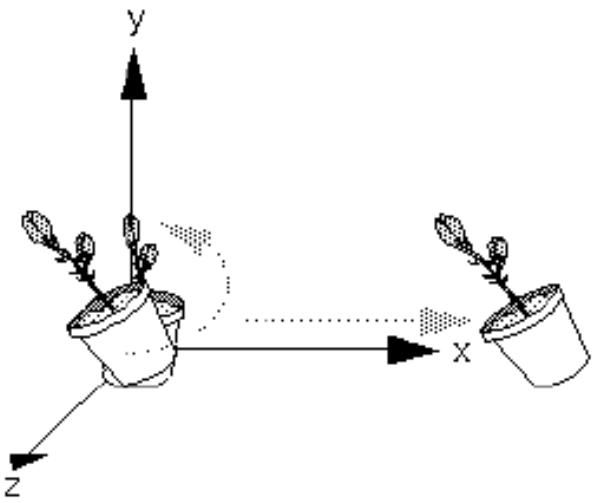
두 변환을 두 가지 서로 다른 방법으로 순차적으로 적용하였다

- Case 1: 변환 1 적용 후 변환 2 적용
- Case 2: 변환 2 적용 후 변환 1 적용

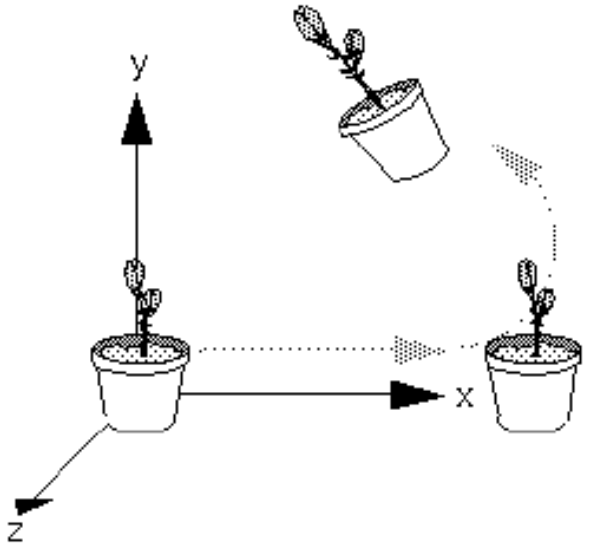
**Case 1 (left): z축으로 CCW 방향으로 45도 회전후 x축으로 10만큼 이동 (rotation first)**

**Case 2 (right): x축으로 10만큼 이동후 z축으로 CCW 방향으로 45도 회전 (translation first)**

**∴ 어떤 차이가 있는가?**




Rotation first

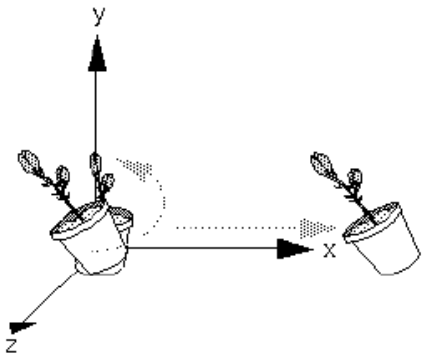



Translation first

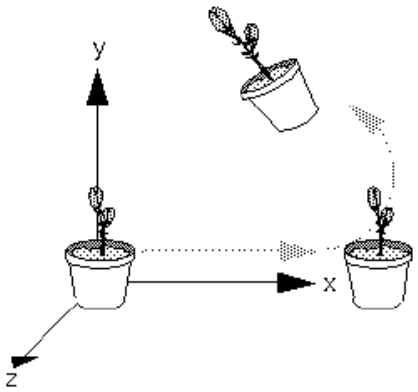
- glMatrixMode(GL_MODELVIEW);

  glLoadIdentity();  // modelview 행렬 초기화 , CTM=I
- glTranslatef(0, 0, -15);  // 무엇때문에 필요한가?
- glTranslatef(10, 0, 0);
- glRotatef (45, 0, 0, 1);
- glutWireTeapot(3.0);

- 이번엔 순서를 바꾸어 보자
- Q) 회전 후 이동? 이동 후 회전?
- glRotatef (45, 0, 0, 1);
- glTranslatef(10, 0, 0);
- glutWireTeapot(3.0);



Rotation first



Translation first

- 앞의 box.cpp를 앞의 코드를 이용하여 바꾼후에 결과를 비교해보자

- 여러 개의 물체에 대한 변환

- 지금까지는 하나의 물체에 대하여 변환, 특히 affine 변환이 이루어지는 것에 대하여 알아보았다

- 여러 개의 물체에 대하여 변환이 이루어지는 경우 변환이 각 물체에 어떻게 적용되는지 살펴보자

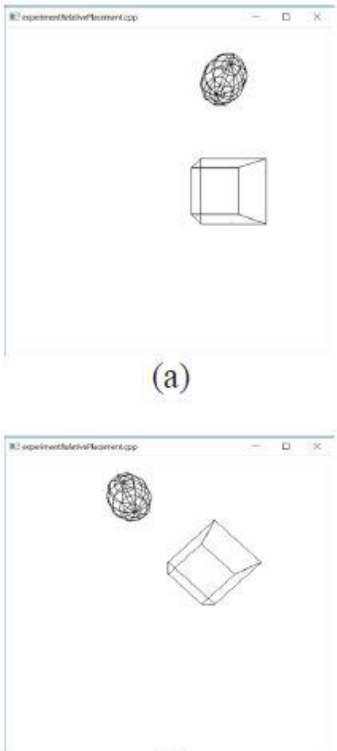
다음의 예는 3D cube와 sphere 두 물체에 대하여 복합 변환이 일어나는 경우이다. CTM을 이용하여 각 물체에 어떤 변환들이 적용되는지 살펴보자

- glMatrixmode(GL_MODEL_VIEW);  // 4x4 modelview 행렬
- glLoadIdentity();  CTM= I (단위 행렬)
- glTranslatef(0.0, 0.0, -15.0); // T1,  CTM= I ·T1
- glTranslatef(5 , 0 , 0);  // T2,  CTM=T1·T2
- glutWireCube(5.0);  // V1,  T1·(T2 · V1)
- glTranslatef( 0, 10 , 0);  // T3  CTM=T1·T2·T3
- glutWireSphere( 2.0, 10, 8); //V2  T1·T2 · (T3 · V2)

- 1. sphere : T1·(T2 · V1)
- 2. cube : T1·T2 · (T3 · V2)
- 즉, 처음 두 변환은 Cube와 Sphere 공통으로 적용

- **Box.cpp에서 이부분 변경**
- **실행후 glRotatef 부분 주석 없앰**

```
void drawScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0.0, 0.0, 0.0);
   glLoadIdentity();

   // Modeling transformations.
   glTranslatef(0.0, 0.0, -15.0);
   // glRotatef(45.0, 0.0, 0.0, 1.0);
   glTranslatef(5.0, 0.0, 0.0);

   glutWireCube(5.0); // Box.

   //More modeling transformations.
   glTranslatef(0.0, 10.0, 0.0);

   glutWireSphere(2.0, 10, 8); // Sphere.

   glFlush();
}
```



(a)

(b)

Figure 4.21:
Screenshots:
(a) Experiments 4.11 and
(b) 4.12.