

Computer Graphics

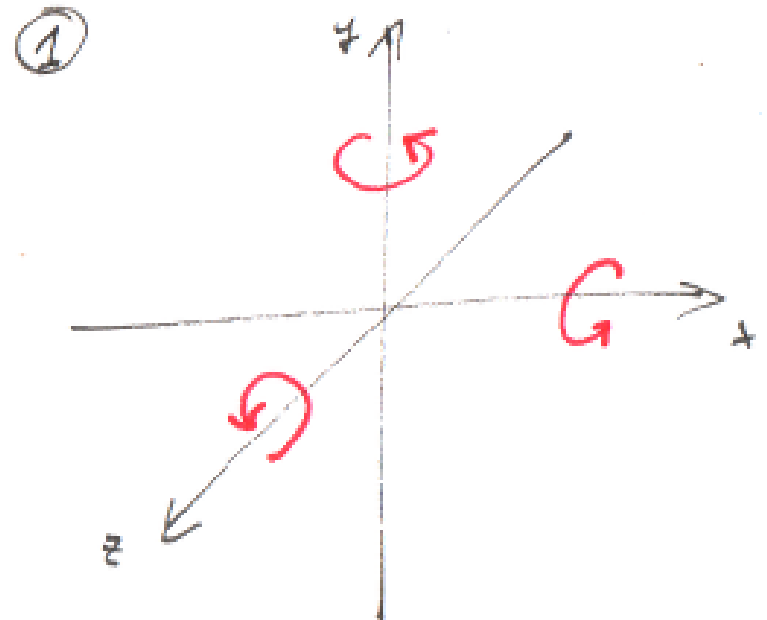
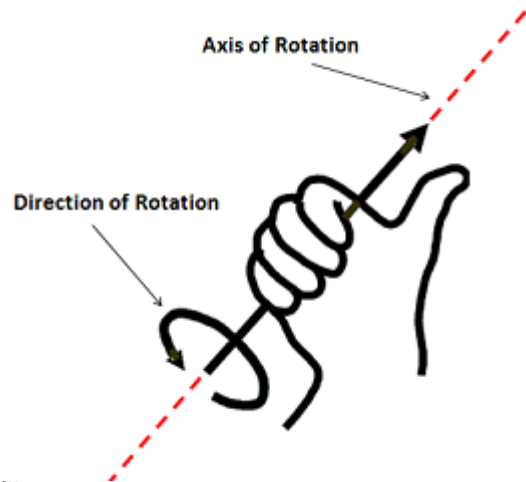
Prof. Jibum Kim

Department of Computer Science & Engineering

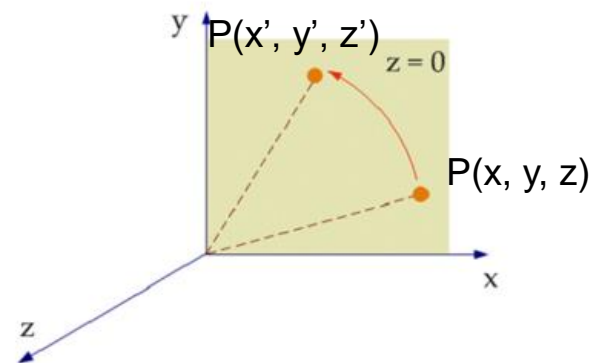
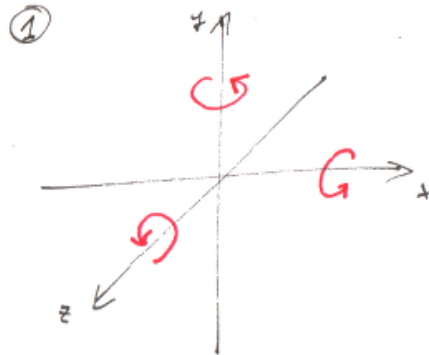
Incheon National University

■ 3D rotation

- 3D rotation: 어떤 축을 기준으로 회전.
- **Right hand rule: 반시계 방향 회전 (Counter clock wise, CCW)**
- X축을 기준으로 CCW 회전
- Y축을 기준으로 CCW 회전
- z축을 기준으로 CCW 회전



- 2D 회전을 3D로 그대로 확장하면 $P(x, y, z)$ 가 Z 축을 기준으로 CCW 방향으로 θ 만큼 회전한 것과 동일
- 이렇게 회전한 점을 $P'(x', y', z')$ 라 하면
- $z=z'$, $x'=x\cos\theta -y\sin\theta$, $y'=x\sin\theta+y\cos\theta$
- 어느 축을 기준으로 회전시 그 기준 축에 해당하는 좌표는 변화지 않는다
- 오른손가락 4개 (엄지 제외)를 회전시키면 엄지 방향이 회전축 방향

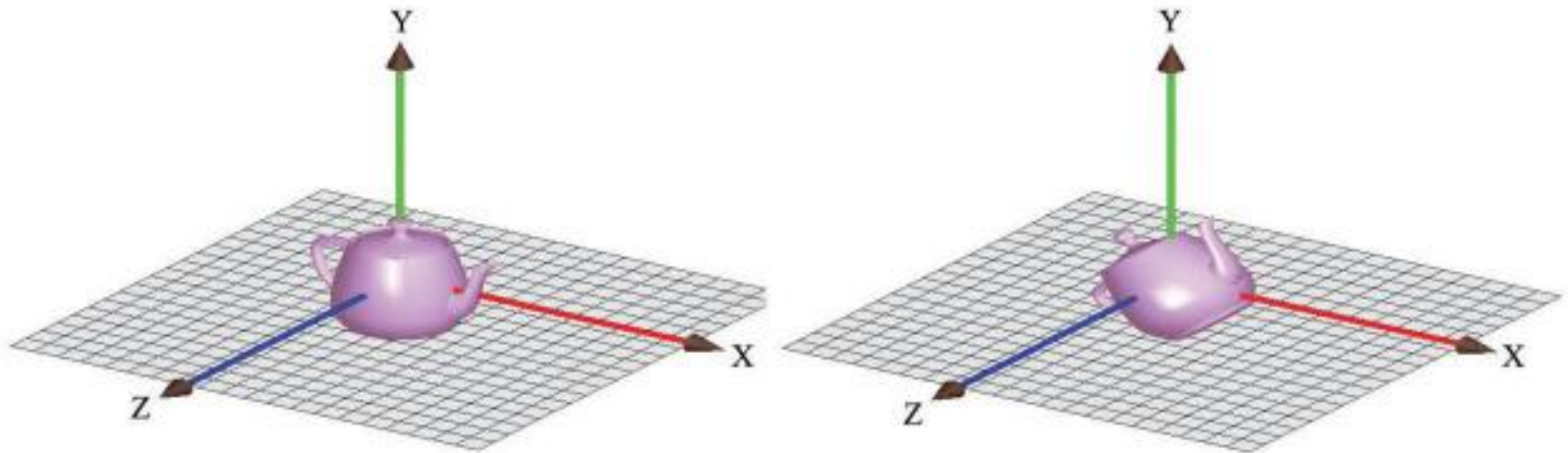


- $P(x, y, z)$ 가 z 축으로 θ 만큼 CCW으로 회전 후 이동한 점
- $P'(x', y', z')$
- $z'=z$, $x'=x\cos\theta -y\sin\theta$, $y'=x\sin\theta+y\cos\theta$
- Homogeneous coordinates으로 표현

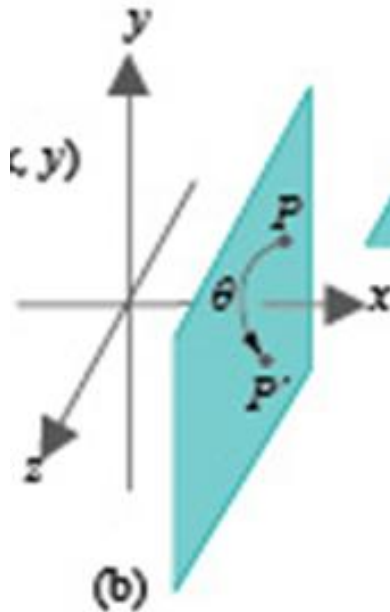
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$P' = R P$, R : rotation matrix

■ z 축 기준으로 45도 만큼 CCW 회전



- CASE 2: (b) x축 방향으로 CCW 회전 , $P(x, y, z) \Rightarrow P'(x', y', z')$



- 오른손가락 4개 (엄지 제외)를 회전시키면 엄지 방향이 회전축 방향

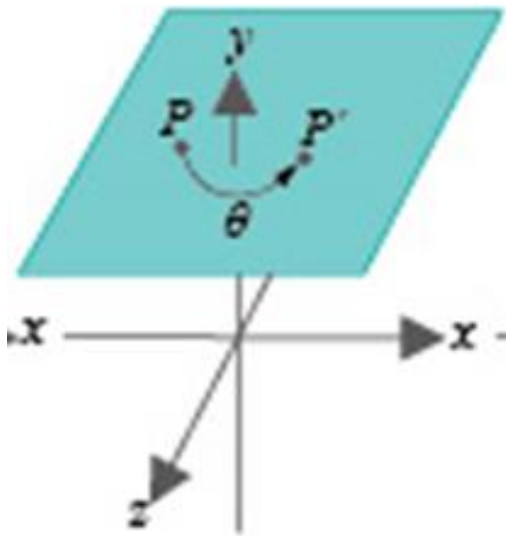
Q) CASE 1의 x축에 해당하는 축은? y축

y축에 해당하는 축은? z축

- $P(x, y, z)$ 가 x 축으로 θ 만큼 CCW 방향으로 회전 후 이동한 점
- $P'(x', y', z')$
- Homogeneous coordinates으로 표현, $P' = R P$

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

CASE 3:(c) y축 방향으로 CCW 회전 $P(x, y, z) \Rightarrow P'(x', y', z')$



- 오른손가락 4개 (엄지 제외)를 회전시키면 엄지 방향이 회전축 방향

Q) CASE 1의 x축에 해당하는 축은? Z축

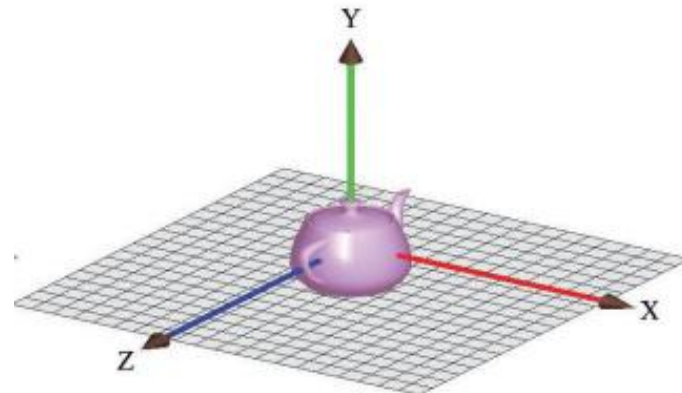
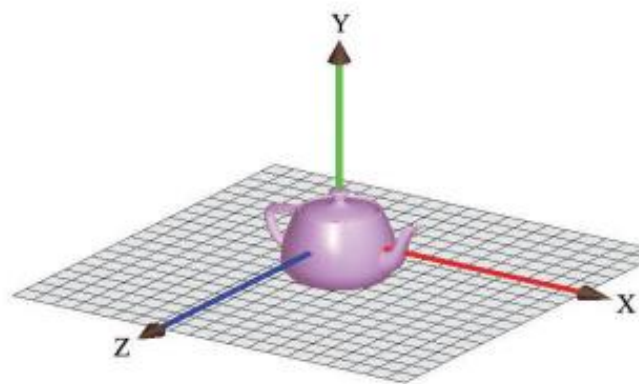
Y축에 해당하는 축은? X축

- $P(x,y,z)$ 가 y 축으로 θ 만큼 CCW 방향으로 회전후 이동한 점

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

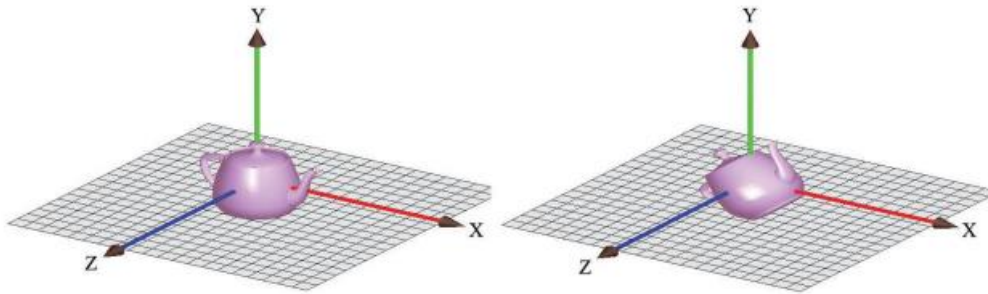
- $P' = RP$

■ y 축 기준으로 90도 만큼 CCW 회전



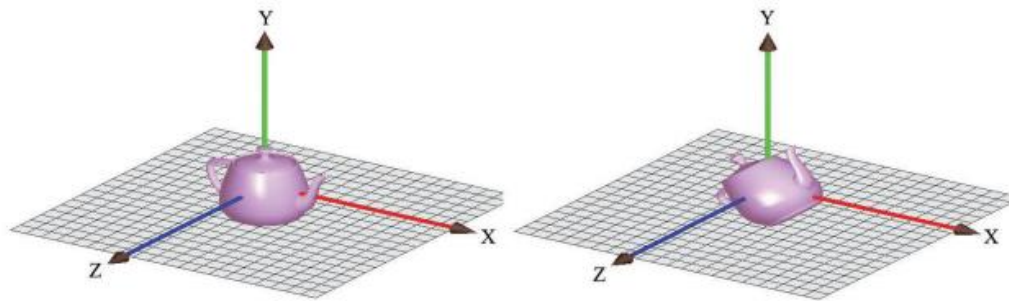
■ Rotation in OpenGL

- `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)`

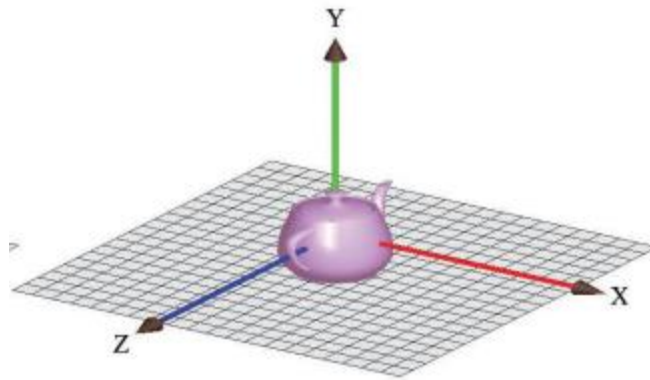


(A) `glRotatef(0.0, 0.0, 0.0, 0.0);` (B) `glRotatef(45.0, 0.0, 0.0, 1.0);`

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);`

-
- https://www.dropbox.com/s/ysx4u87ld2rb7m8/rotate_z.txt?dl=0
 - `// glRotatef(45, 0, 0, 1);`

-
- **glRotatef** 를 다음과 같이 바꾸어 보자
 - **glRotatef(90, 0, 1, 0);**

■ Shearing in OpenGL

-
- OpenGL에서 물체의 변환 시 OpenGL에서 제공하는 함수를 사용하지 않고 앞에서 본 행렬 곱을 이용하여 직접 물체의 위치를 변환시킬 수도 있다
 - Affine 변환 중에 shearing에 해당하는 OpenGL 함수가 없으므로 직접 affine 변환 행렬을 만들어 보고자 한다

-
- <https://www.dropbox.com/s/rbyb7q1mxiaz4r6/shearing.txt?dl=0>

- 주석을 없애보자
- 어떤 변환이 생기는가?



- 주의 이 변환 행렬은 (4행 4열은) 우리가 앞에서 쓰던 변환 행렬과 읽는 법이 다르다 (**column-wise**로 읽는다)
- `glMultMatrixf(matrixdata); // 행렬 곱`
- `static float matrixdata[16] = // Shear matrix.`
- `{`
- `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`
- `};`

-
- `static float matrixdata[16] = // Shear matrix.`
 - `{`
 - `1.0, 0.2, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,`
`0.0, 0.0, 0.0, 1.0`
 - `};`



-
- 이 변환행렬을 이용하여 다른 변환을 테스트해 볼 수 도 있음
 - **// translate in x-axis by 1**
 - **static float matrixdata[16] =**
 - **{**
 - **1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,**
1.0
 - **};**



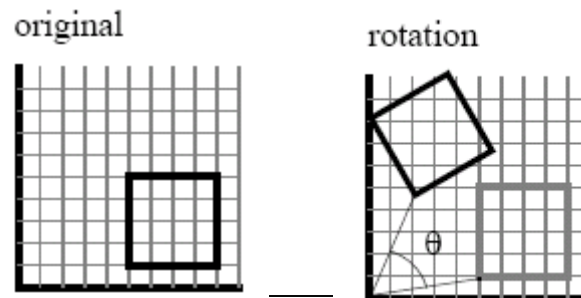
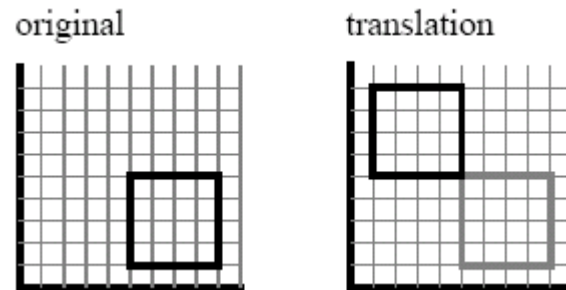
// translate in y-axis by 2

- **static float matrixdata[16] =**
- **{**
- **1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 2.0, 0.0,**
1.0
- **};**



■ Classification of transformations

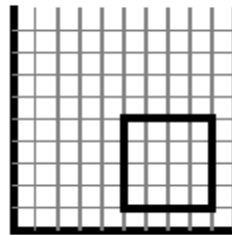
- **1. Euclidean transformation (rigid body transformation)**
- **It preserves size and shape**
- **Only allows Translation and rotation**



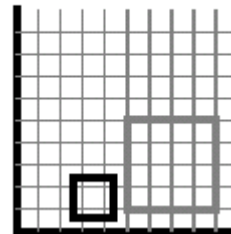
-
- **2. Similarity transformation (닮은 변환)**
 - **Euclidean transformation + scale change**
 - **Translation + rotation + scaling**
 - **(이동, 회전, 크기변화 허용)**

■ Example of similarity transformation

original



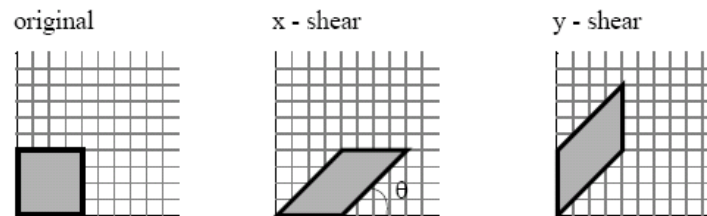
scaling



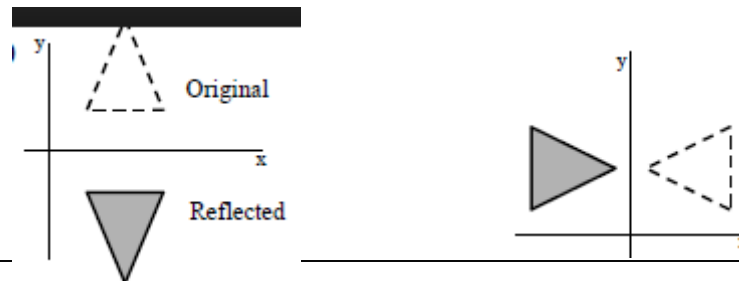
■ 3. Affine transformation

- Allows Translation, rotation, scaling, shearing, reflection

- Shearing:



- Reflection



- Ex) Example of affine transformation
- affine 변환의 특징을 만족하는가?

- $$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 & 1 \\ 0.5 & 0.5 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- static float matrixdata[16] =

- {

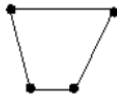
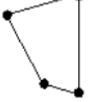

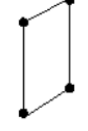
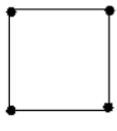
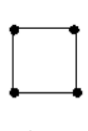


- 2.0, 0.5, 0.0, 0.0, 1.0, 0.5, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0

- }; // shearing+ translation+ scaling

- Degree of freedom (DOF)
- 3D Affine 변환의 경우, the 12 values can be set arbitrarily, and we say that this transformation has **12 degrees of freedom (DOF)**.

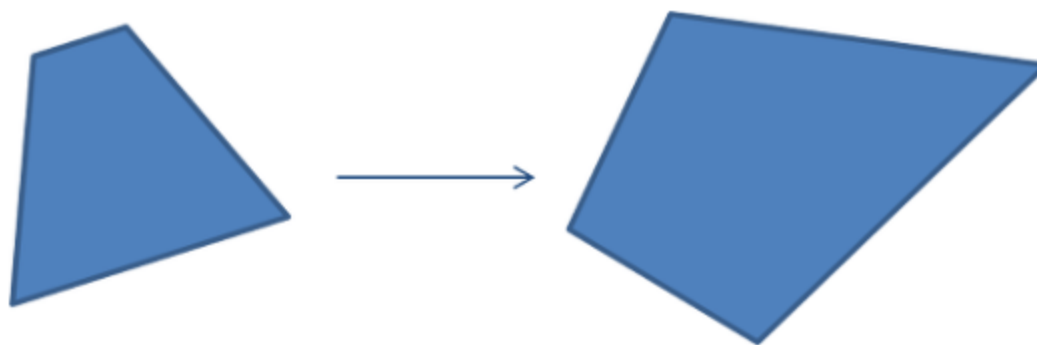
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Affine 변환이 아닌 변환에는 어떤 것이 있을까?
- 수업에서 자세히 다루지는 않지만 Non-affine 변환에 해당하는 변환은 projective transformation (homography) 변환이 있다. 변환 행렬이 어떻게 생겼을까? 이 경우 DOF (degree of freedom)는?

Transformation	Before	After
Projective		
Affine		
Similarity		
Euclidean		

■ 출처: 다크프로그래머 블로그

Homography를 직관적으로 이해하기 위한 한 좋은 방법은 2D 평면에서 임의의 사각형을 임의의 사각형으로 매핑시킬 수 있는 변환이 homography다 라고 생각하는 것입니다. 물론 사각형이 뒤집혀질 수도 있습니다.



■ <https://darkpgmr.tistory.com/79>

■ 복합 변환 (composite transformations)

-
- **Note: OpenGL에서의 모든 물체의 변환 (affine 변환)은 행렬 곱으로 표시 되었다, 3D의 경우 4x4 행렬로 모든 물체 변환이 표시되었다**
 - 원래 점 P , 변환 후의 점 P' , 변환 행렬, T 이면
 - $P' = T \cdot P$

- 일반적으로 변환은 연속적으로 가해 진다. 변환이 연속적으로 발생하는 것을 **복합 변환**이라 한다
- 예: 어떤 vertex (P)에 대해 크기 조절 ($T1$)을 가한 후 그 결과 물체를 z축으로 회전 ($T2$)하였다고 하자
- 이를 행렬 곱으로 나타내면 어떻게 될까?
- $P' = T1 \cdot P$
- $P'' = T2 \cdot P'$
- $P'' = T2 \cdot T1 \cdot P$

$$P'' = T2(T1 \cdot P)$$

-
- 변환 T_1, T_2, T_3 가 이 순서대로 vertex P 에 연속적으로 발생했다고 하자.
 - 그렇다면 변환후의 점은 , $T_3 \cdot T_2 \cdot T_1 \cdot P$
 - 연산 후 알아 낼 수 있다

- 행렬 곱시 **역순 (reverse order)**으로 곱해지는 것처럼 보인다
- 또한, 복합 변환은 행렬 곱으로 나타낼 수 있으므로 편리하다
- 수업시간에 다루는 affine 변환의 경우 두 affine 변환 행렬의 곱으로 생성된 변환행렬을 적용 시 역시 **affine 변환**이다

■ 복합 변환시 변환의 순서의 중요성

-
- 다음을 생각해 보자. 어떤 점 P 에

- 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 낼 수 있다

-
- 다음의 두 변환이 있다고 하자
 - 1) Rotate by 45° CCW in z-axis
 - 2) Translate by 10 in x-axis

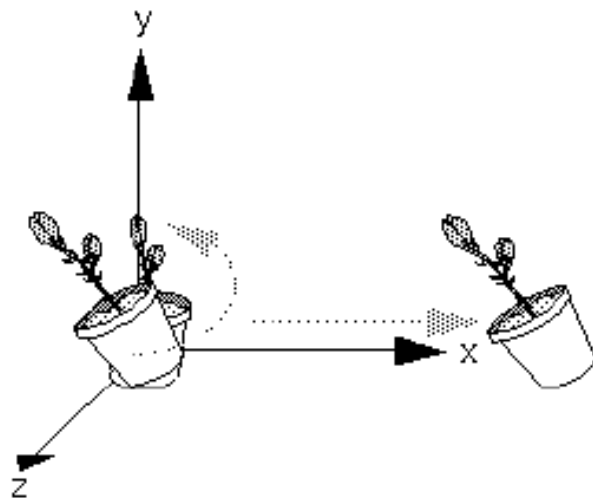
두 변환을 두 가지 서로 다른 방법으로 순차적으로 적용하였다

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

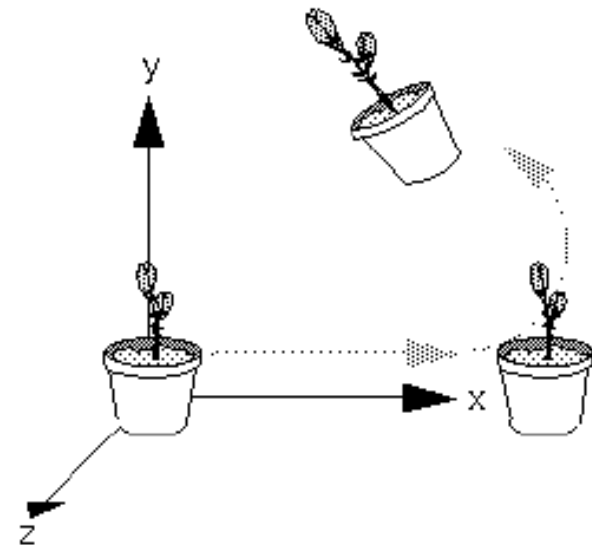
Case 1 (left): z축으로 CCW 방향으로 45도 회전후 x축으로 10만큼 이동 (rotation first)

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

∴ 어떤 차이가 있는가?



Rotation first



Translation first

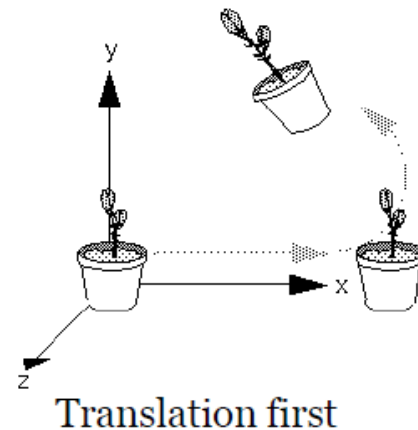
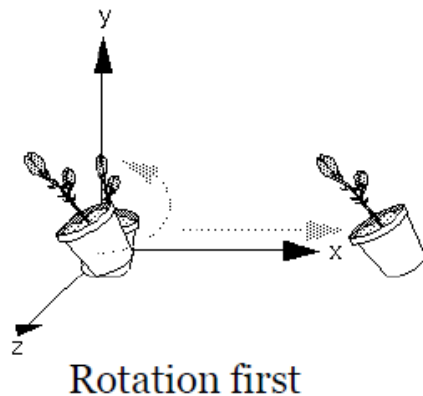
- OpenGL에서의 CTM을 이용한 복합 변환

- 아래 예제에 대해서 CTM이 무엇인지 살펴보자
- 어떤 변환부터 수행되는가?
- E.g.,

```
glScalef(sx, sy, sz);  
glRotatef(theta, vx, vy, vz);  
Object 1(v)
```

```
// CTM=I  
// CTM=I · S  
// CTM=I · S · R  
// I · S · (R · v)
```

- 다음 OpenGL코드의 경우 Teapot에 어떤 변환이 먼저 적용되는가?
아래 그림에서 어떤 경우에 해당될까?
- Q) 회전 후 이동? 이동 후 회전?
- `glTranslatef(10, 0, 0);`
- `glRotatef(45, 0, 0, 1);`
- `glutWireTeapot(1.0);`



```

■ #include <GL/glut.h>

■ void myInit(void)
■ {
■     glClearColor(1.0,1.0,1.0,0.0);    // the background color is white
■     glColor3f(1.0f, 0.0f, 0.0f);      // the drawing color is black
■         glMatrixMode(GL_PROJECTION);
■         glLoadIdentity();
■         glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0);
■ }

■ void Display()
■ {
■     glClear(GL_COLOR_BUFFER_BIT);
■     glMatrixMode(GL_MODELVIEW);
■     glLoadIdentity();
■     glTranslatef(0, 0, -15);
■     glTranslatef(10, 0, 0);
■     glRotatef (45, 0, 0, 1);
■     glutWireTeapot(3.0);
■     glFlush();
■ }
■ int main(){
■     glutInitWindowSize(400,400);
■     glutInitWindowPosition(100,100);
■     glutCreateWindow("OpenGL Hello World!");
■     glutDisplayFunc(Display);
■     myInit();
■     glutMainLoop();
■     return 0;

```

■ Modelview 행렬 (OpenGL 2.x)

-
- 이전 버전의 OpenGL 2.x 버전에서는 transformation과 projection시에 쓸 수 있는 modelview 행렬, projection 행렬, matrix stack 개념이 있다
 - 변환에서 CTM을 사용시 **modelview 행렬**이라는 4x4 행렬을 사용한다

- OpenGL에서는 modelview행렬을 사용시에 항등행렬로 초기화 한다
- Modelview 행렬의 초기화

- $M=I = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$

- OpenGL 코드
- `glMatrixMode(GL_MODELVIEW);`
- `// 행렬 선택, modelview? Projection?`

- `glLoadIdentity();`

`//4x4 modelview 행렬을 항등행렬로 초기화`

-
- **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);`

