

# Computer Graphics

---

**Prof. Jibum Kim**

**Department of Computer Science & Engineering**

**Incheon National University**

---

## ■ 동차 좌표 (homogeneous coordinate)

- 정의: **Point (점)** is a location in space
- 특징: Points have position, but neither length nor direction
- 정의: A point  $P=[x_1 \ x_2 \ x_3 \dots x_m]^T$  in  $R^m$  is represented in homogeneous coordinates (동차 좌표) by any  $m+1$  tuple of the form  $[cx_1 \ cx_2 \ cx_3 \dots cx_m \ c]^T$ , where  $c$  is a non-zero scalar
- 전치 행렬?          Tuple의 의미?
- 예: 점  $P = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$ ,  $P \in R^2$
- $P$  의 가능한 동차 좌표 예 (차원을 하나 올림)

$$\begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 6 \\ 12 \\ 2 \end{bmatrix}, \dots$$

- 동차 좌표의 물리적 의미
- <http://darkpgmr.tistory.com/78>
- Point에 대해서는 **c=1로 고정하고 사용** (vector는 c=0)
- **$P=[x_1 \ x_2 \ x_3 \dots x_m]^T$**
- 동차좌표
- **$[x_1 \ x_2 \ x_3 \ \dots \ x_m \ 1]^T$**
- 예: 점  $P = \begin{bmatrix} 3 \\ 7 \end{bmatrix}$  의 동차 좌표  $\begin{bmatrix} 3 \\ 7 \\ 1 \end{bmatrix}$

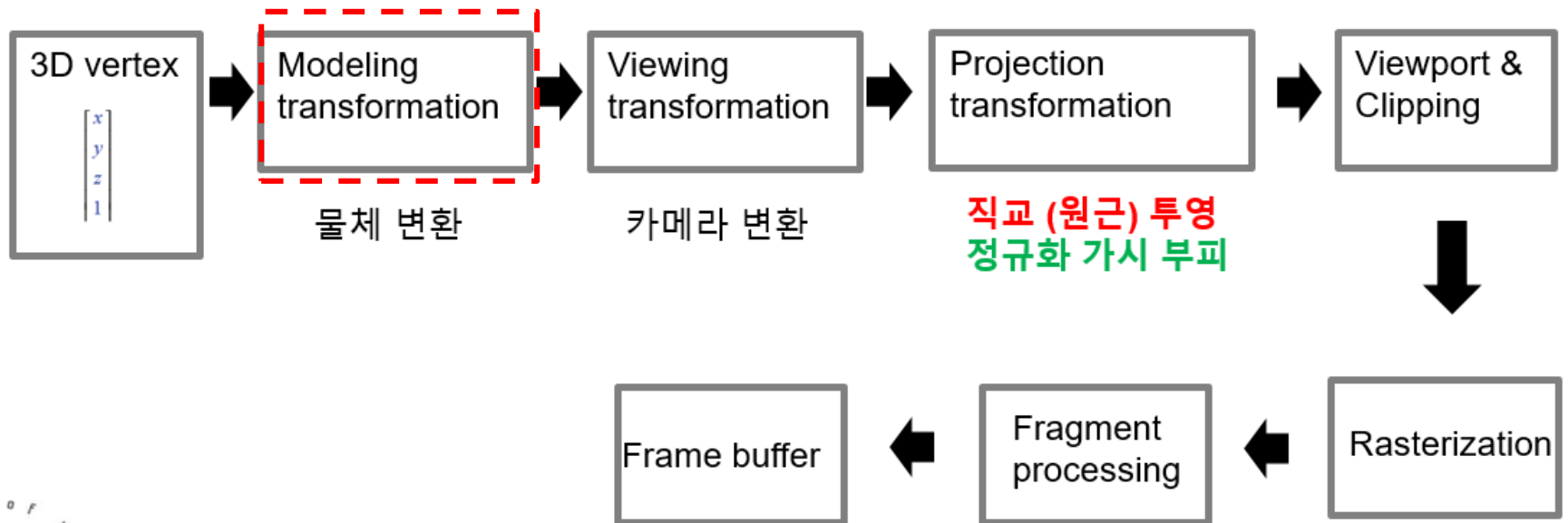
- 왜 컴퓨터 그래픽스에서는 동차좌표를 사용할까?
- 1. 기본적으로 다음에 배울 변환 (transformation)시에 동차좌표를 사용하면 **변환을 행렬 곱, 형태로 표현 가능하다**
- 2. 여러 개의 변환이 연속으로 적용되는 복합 변환의 경우에는 이러한 **행렬 곱을 하나의 행렬 곱 형태로** 바꿀 수 있다 (이것은 계산 속도 면에서도 유리하다)
- OpenGL에서도 동차좌표를 사용함
- 3. 또한, Point와 vector를 구별할 수 있다

$$\begin{array}{ll} \mathbf{v} = [a_1, a_2, a_3] & \mathbf{v} = [a_1, a_2, a_3, 0]^T \\ \mathbf{p} = [b_1, b_2, b_3] & \mathbf{p} = [b_1, b_2, b_3, 1]^T \end{array}$$

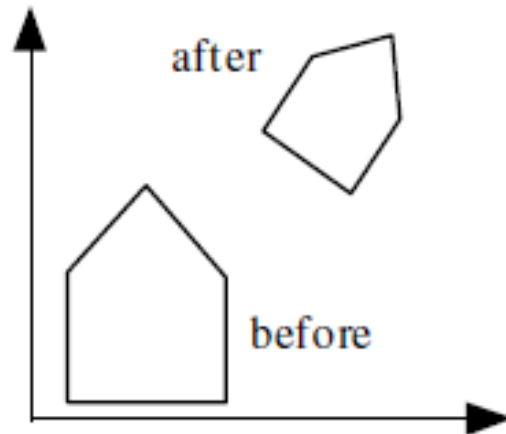
---

## ■ Object transformations (물체의 변환)

## ■ Graphics pipeline (OpenGL 2.x 기준)

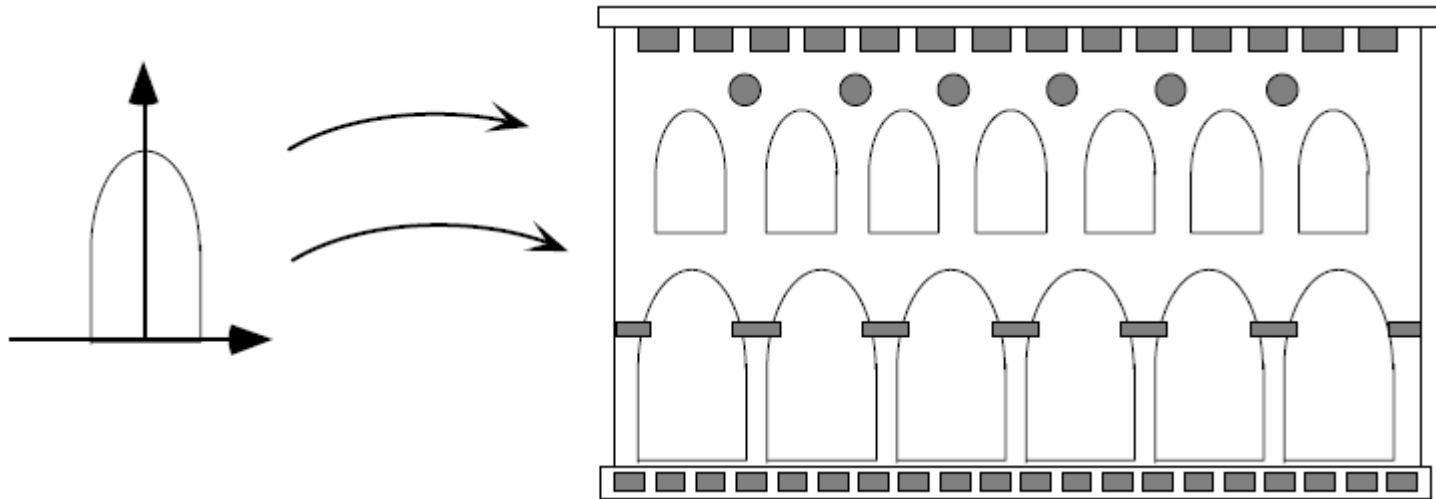


- 물체의 변환 (transformation of objects)이란 간단히 얘기하면 물체의 크기가 변화되거나 물체의 위치가 이동하는 것들을 말한다.
- 물체의 좌표  $P(x, y, z)$ 가 어떤 함수를 통하여  $P'(x', y', z')$ 로 mapping 된 것을 의미 한다.  $P'=f(P)$
- 먼저 물체의 변환의 필요성에 대해서 살펴보자

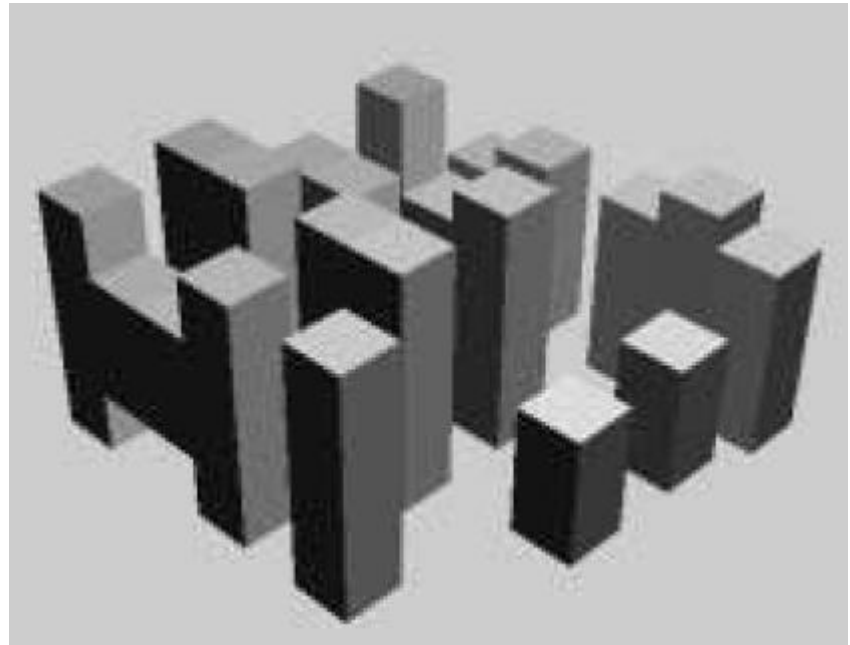
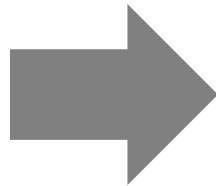




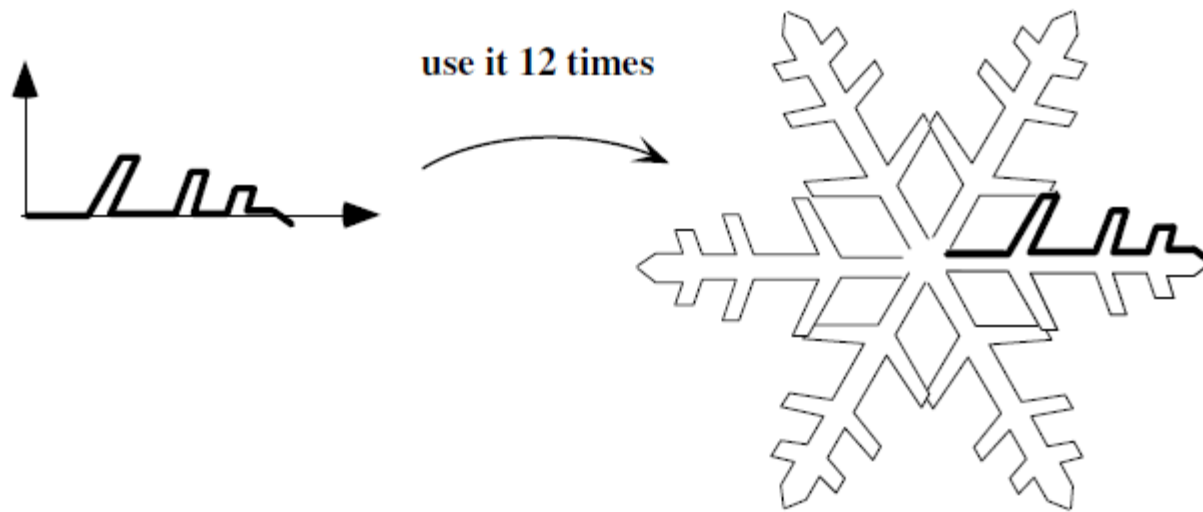
- 물체의 변환의 필요성
- 1) 2D 예: 기본적인 형태의 물체를 만든 후 이를 변환 (이동, 크기 변환 등..)시켜서 다양한 형태를 손쉽게 구성할 수 있다



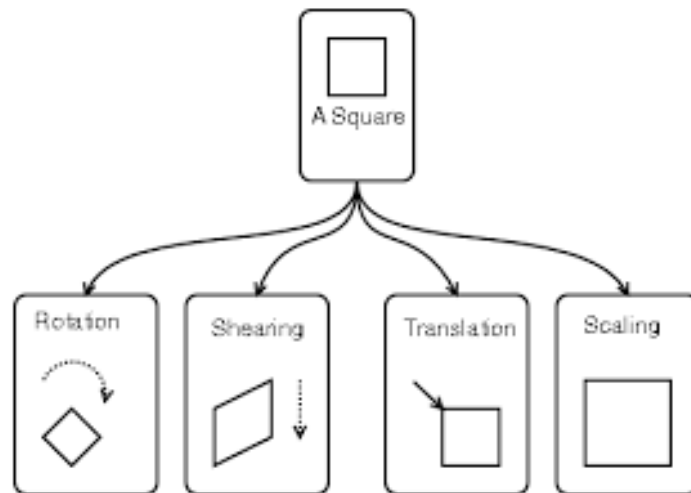
## ■ 2) 3D 예: 위치 이동 및 크기 변환



- 3) 눈송이 예: 아래의 눈송이 (snowflake)와 같은 몇몇 물체들은 기본 형태를 회전, 이동, 반사와 같은 여러 개의 변환을 반복 적용한 결과이다.



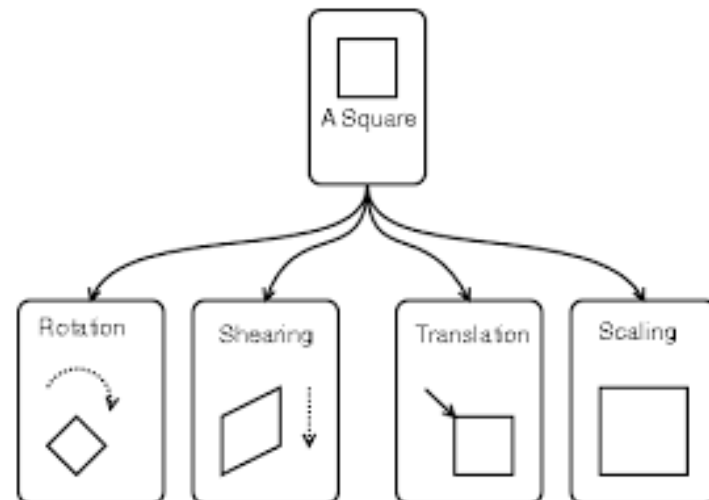
- 물체의 변환 중 컴퓨터 그래픽스에서 가장 중요하면서 가장 흔한 변환은 **affine transformation (어파인 변환)**이다
- Affine 변환에는 translation (천이), scaling (크기 변화), rotation (회전), shearing (전단)이 있다



- 
- Affine 변환의 중요한 특징
  - 1) Affine 변환은 간단한 행렬로 표현 가능하다
  - 2) Affine 변환을 여러 번 실시 한 것을 하나의 Affine 변환 (행렬)으로 표현 가능하다는 것이다

- 2D affine 변환
- 동차좌표를 사용시 아래와 같이 2D에서 행렬 곱으로 정의
- Translation, scaling, rotation, shearing 모두 affine 변환의 일종

- $$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



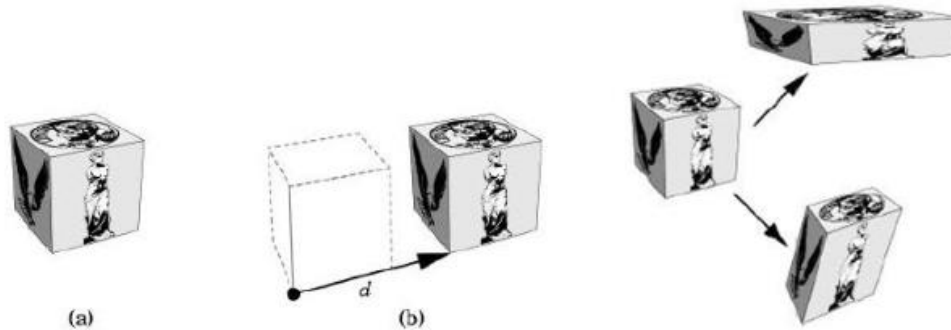
---

## ■ Translation

## ■ Translation 이란?

- 탄력이 없는 단단한 물체 (rigid body)를 이동하면 물체를 구성하는 vertex들이 동일한 양만큼 움직이는데 이를 **translation**이라 한다

- rigid-body transformation : object 의 크기가 불변
  - 예: translation, rotation
- non-rigid-body transformation : object 크기에 변화
  - 예: scaling



rigid-body transformation

non-rigid-body transformation

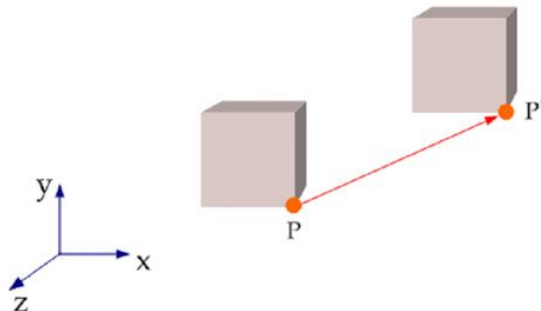


- 
- 1. 2D Translation (천이, 이동)
  - 점  $P(x,y)$ 가 점  $P'(x',y')$ 로  $x$ 축으로  $T_x$ ,  $y$ 축으로  $T_y$ 만큼 이동
  - $x'=x+T_x$ ,  $y'=y+T_y$
  - 동차좌표표 표현시 : 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

## ■ 3D Translation (천이, 이동)

- $P(x, y, z)$ 가  $P'(x', y', z')$ 로 x축으로  $T_x$ , y축으로  $T_y$ , z축으로  $T_z$ 만큼 이동
- $x'=x+T_x$ ,  $y'=y+T_y$ ,  $z'=z+T_z$

- 동차좌표로 표현시: 
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



## ■ 2. 2D Scaling (크기 조절)

- 점  $P(x, y)$ 가 점  $P'(x', y')$ 로  $x$ 축으로  $S_x$ 배,  $y$ 축으로  $S_y$ 배 만큼 크기 조절 변환
- $x' = S_x \cdot x$
- $y' = S_y \cdot y$
- $S_x, S_y$ 는 각각  $x, y$ 축 방향의 배율이고 1보다 크면 확대, 작으면 축소

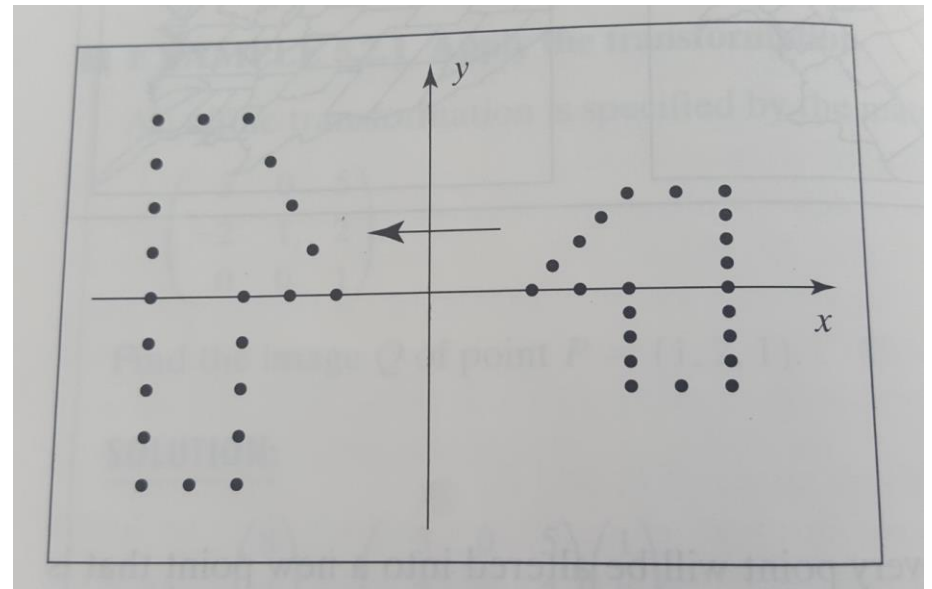
- 동차좌표로 표현시 : 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 만일 배율이 음이 된다면 어떻게 될까?

- 예:  $S_x=-1, S_y=2$ 인 경우

- $$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 점  $P = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, P' = \begin{bmatrix} -1 \\ 2 \end{bmatrix},$



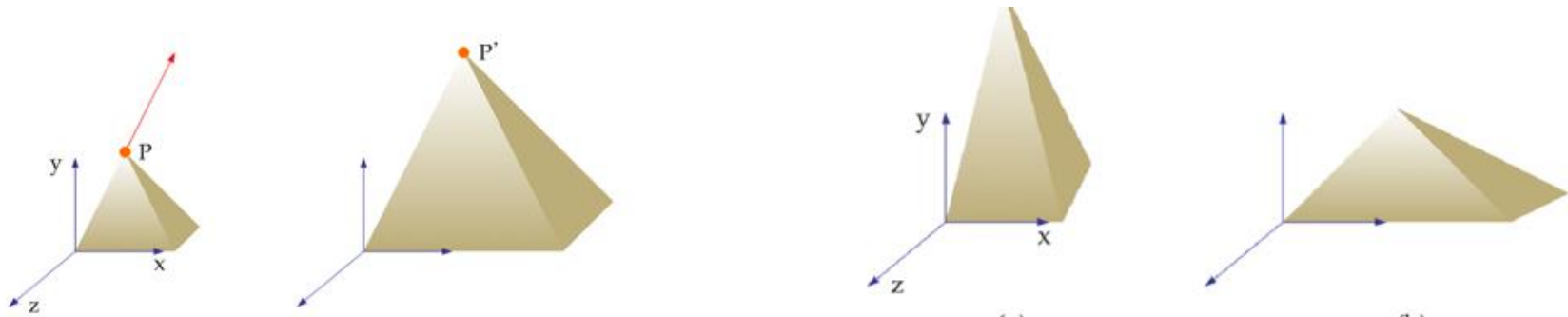
- 이를 **reflection (반사)**라 한다

### ■ 3D Scaling (크기 조절)

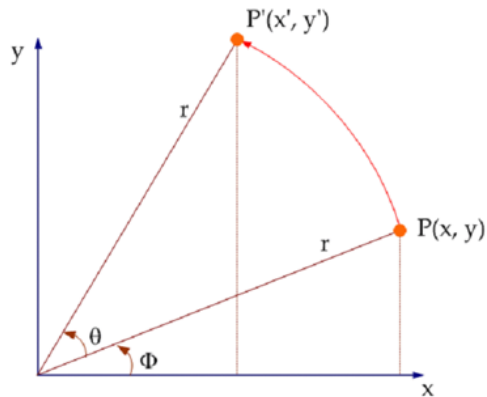
- 점  $P(x, y, z)$ 가 점  $P'(x', y', z')$ 로 x축  $S_x$ , y축  $S_y$ , z축  $S_z$  만큼 각각 크기 조절 변환
- $S_x, S_y, S_z$ 는 각각 x,y,z,축 방향의 배율이고 1보다 크면 확대, 작으면 축소
- $x'=S_x \cdot x, y'=S_y \cdot y, z'=S_z \cdot z$

■ 동차좌표로 표현시: 
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- 만일 모든 배율이 같은 경우 균등 크기 조절 (uniform scaling),
- 하나라도 다르다면 차등 크기 조절 (differential scaling)이라 한다
- 최초의 object가 아래의 가장 왼쪽과 같을 때
- 각각이 균등 크기 조절인지 차등 크기 조절인지 살펴보자



- 3. 2D 회전 (3D 회전은 후에)
- $P(x,y)$ 가 원점을 중심으로 **반시계 방향**으로  $\theta$  만큼  $P'(x',y')$ 로 회전
- $\Phi$ : 원점과 점  $P$ 를 연결한 선분이  $x$ 축과 이루는 각,  $r$ : 회전 반지름



$$x' = r \cos(\phi + \theta) = r \cos\phi \cos\theta - r \sin\phi \sin\theta = x \cos\theta - y \sin\theta$$

$$y' = r \sin(\phi + \theta) = r \cos\phi \sin\theta + r \sin\phi \cos\theta = x \sin\theta + y \cos\theta$$

$$P' = R \cdot P$$

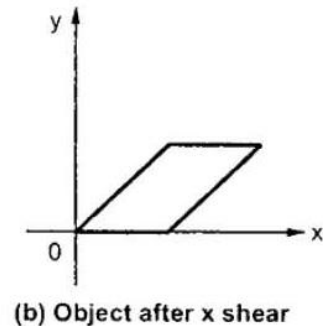
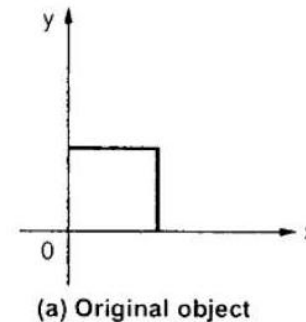
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- 
- 예: 어떤 점  $P(3, 5)$ 가 반시계 방향으로 60도 만큼 회전하였을 때의 위치를 구해보자



#### 4. 2D Shearing (전단): 물체를 한쪽으로 밀어낸 모습으로 물체 모양 변함

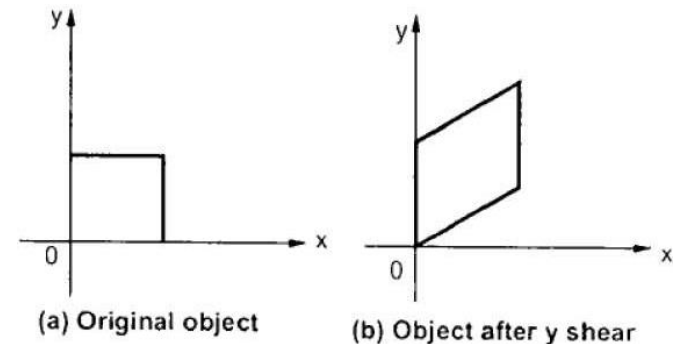
- **x축 방향의 shearing**
- x축에서는 y값이 클수록 변화가 크다
- y값은 변화가 없다
- 점  $P(x, y)$  가 점  $P'(x', y')$ 로 x축 shearing시
- $x' = x + hy$
- $y' = y$
- h: 전단 인수 (shearing factor)



x축 방향으로의 shearing: 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- 
- 예: x축 방향으로 shearing이 일어나고 x축 방향으로의 shearing factor=0.3일 때  $P(3,4)$ 는 어디로 이동하는가?

- **Y축 방향의 shearing**
- x값은 변화가 없다
- y값은 x값이 클수록 변화가 크다
- 점  $P(x, y)$  가 점  $P'(x', y')$ 로 y축 shearing시
- $x' = x$
- $y' = y + gx$
- $g$ : 전단 인수 (shearing factor)



- y축 방향으로의 shearing: 
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$





## ■ affine 변환의 특징

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

1. 평행성 유지: 두 개 (혹은 그 이상의) 평행한 직선은 Affine 변환 후에도 평행함을 유지
2. 직선은 affine 변환 후에도 직선
3. 같은 선분 위의 세 점 혹은 그 이상의 점은 Affine 변환 후에도 같은 선분 위에 있음

....

[https://en.wikipedia.org/wiki/Affine\\_transformation](https://en.wikipedia.org/wiki/Affine_transformation)

Affine Transform	Example
Translation	
Scale	
Shear	
Rotation	

- 3D affine 변환
- Translation, scaling, rotation, shearing 모두 affine 변환의 일종
- 변환 전후에 직선은 직선, 다각형은 다각형으로 유지된다

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

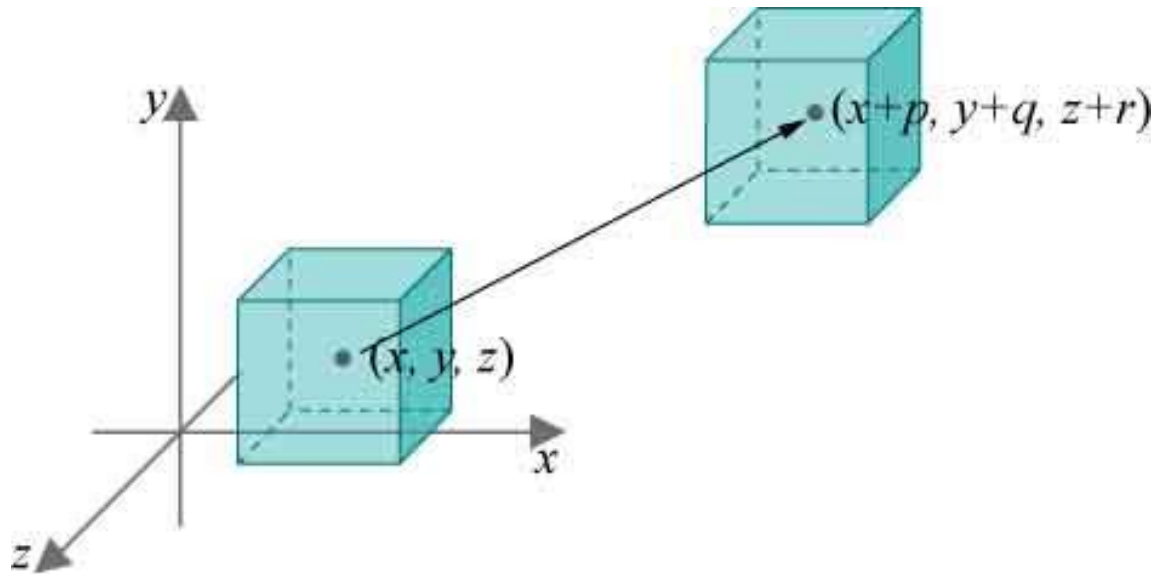
- 
- 복합 변환: Composite of transformations
  - 일반적으로 물체에 대한 변환이 여러 개가 연속적으로 가해질 수 있다
  - 앞에서 배운 변환이 연속적으로 가해지는 것을 **복합 변환**이라 한다
  - 복합 변환의 경우 변환의 순서가 매우 중요하다
  - 행렬 곱은 교환 법칙 성립 안함  $T_1T_2 \neq T_2T_1$

---

## ■ OpenGL에서의 object transformation

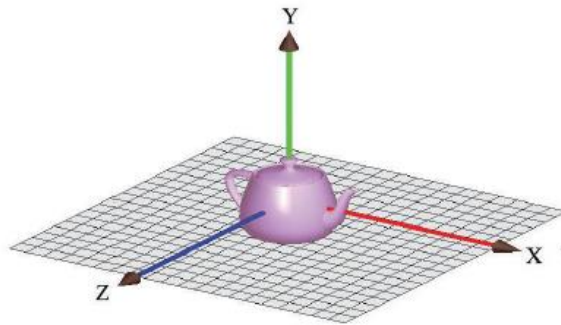
## ■ **glTranslatef(p, q, r)**

- Translate an object p units in the x direction, q units in the y direction, and r units in the z direction

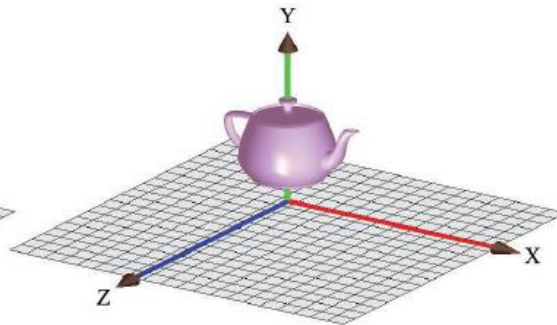




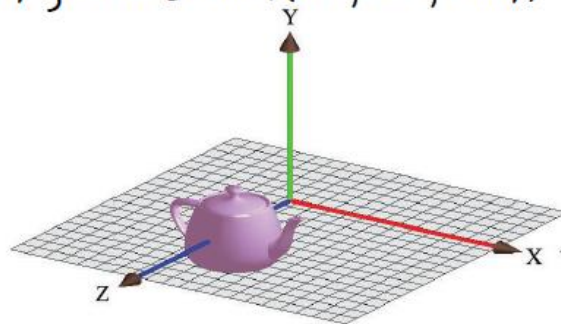
- Assume that a teapot is initially located at (0,0,0)



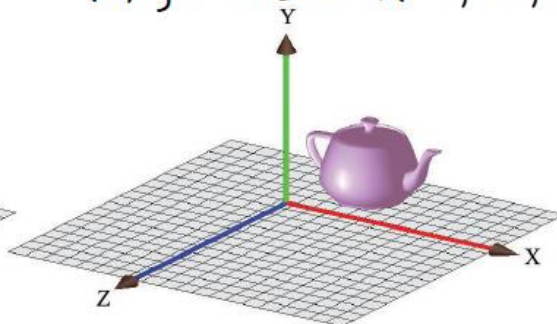
(A) `glTranslatef(0.0, 0.0, 0.0);`



(B) `glTranslatef(0.0, 1.0, 0.0);`

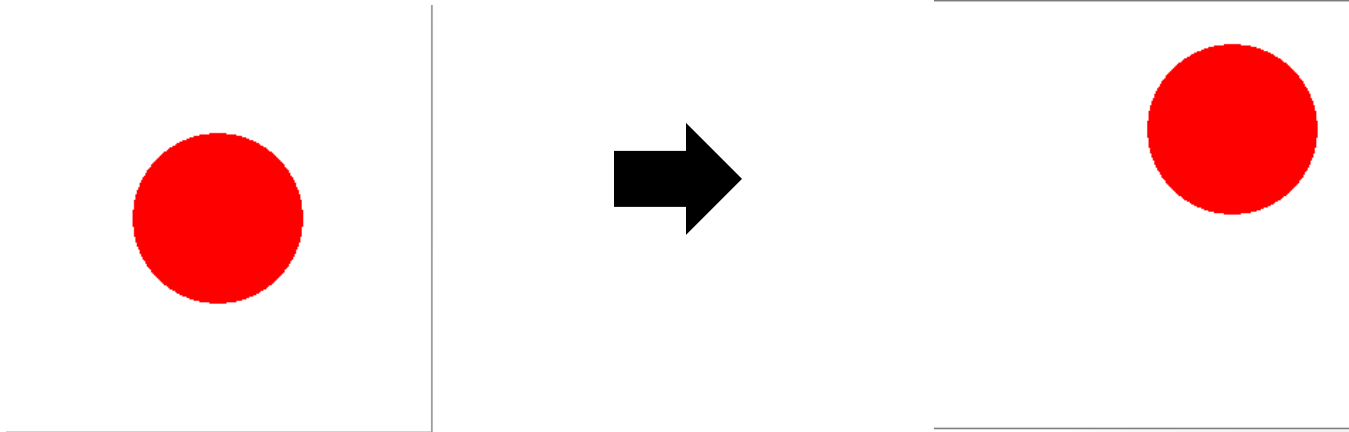


(C) `glTranslatef(0.0, 0.0, 1.0);`



(D) `glTranslatef(1.0, 1.0, 0.0);`

- 
- `glOrtho (0.0, 100.0, 0.0, 100.0, -1.0, 1.0)`
  - 원: 반지름 (20), 중심 (50.0, 50.0, 0.0)
  - X축으로 20, y축으로 20만큼 translate



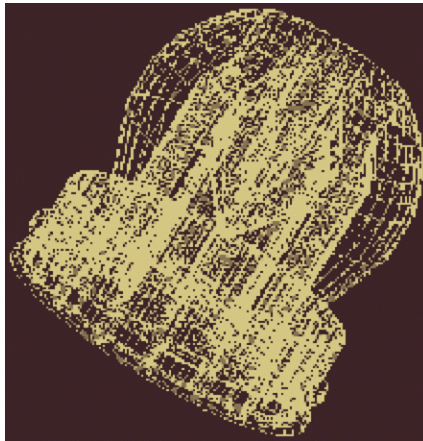
- 
- [https://www.dropbox.com/s/11aqiax7tsd8tjc/circle\\_translate.txt?dl=0](https://www.dropbox.com/s/11aqiax7tsd8tjc/circle_translate.txt?dl=0)
  - `// glTranslatef(20.0, 20.0, 0.0)`
  - 의 주석을 없애 보자

---

## ■ GLUT Object and Translation

- 
- GLUT object:
  - GLUT 라이브러리에서 쉽게 3D geometric objects를 생성해서 사용하도록 이미 만들어놓은 3D objects를 제공한다
  - 각각의 object들은 **wireframe, solid rendering (렌더링)** 모드 2가지를 제공 한다
  - GLUT object는 **기본적으로 (0,0,0)에 중심**을 두고 있다

- **Wireframe rendering (좌측)**
- 물체의 뼈대 만을 edge로 묘사. Drawing 속도가 빠른 장점이 있어서 복잡한 물체를 모델링시에 유리
- **Solid rendering (우측)**
- 물체에 조명을 가하여 색상이 드러난 물체를 그림. 실제 모습을 확인하기 좋음



## ■ GLUT object들의 예 (각각 solid, wireframe rendering)

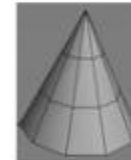
### Cube

- `glutSolidCube(size)`
- `glutWireCube(size)`



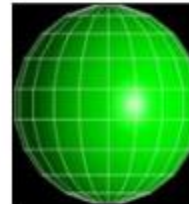
### Cone

- `glutSolidCone(base_radius, height, slices, stacks)`
- `glutWireCone(base_radius, height, slices, stacks)`



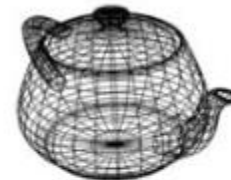
### Sphere

- `glutSolidSphere(radius, slices, stacks)`
- `glutWireSphere(radius, slices, stacks)`



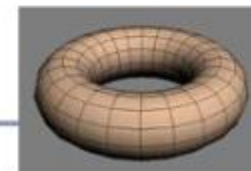
### Teapot

- `glutSolidTeapot(size)`
- `glutWireTeapot(size)`

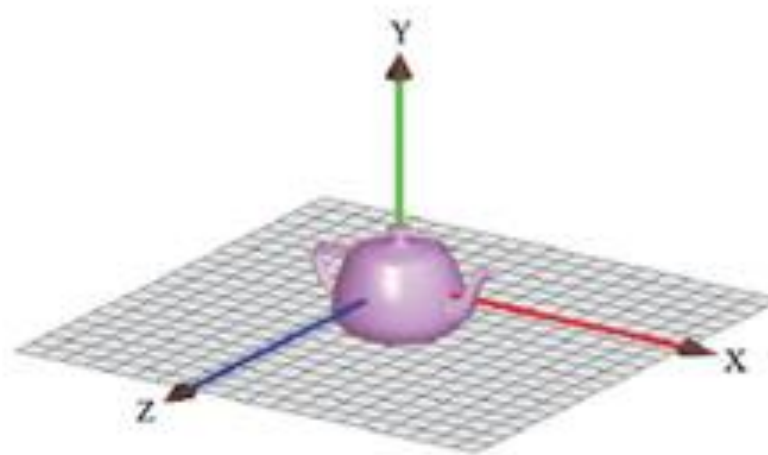


### Torus

- `glutSolidTorus(inner_radius, outer_radius, nsides, rings)`
- `glutWireTorus(inner_radius, outer_radius, nsides, rings)`

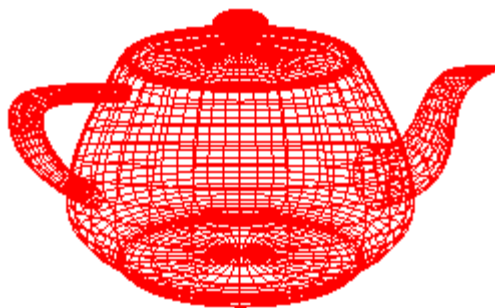


- glFrustum()과 GLUT object를 이용한 drawing
- glutWireTeapot(2.0)을 사용, Teapot의 중심 (0,0,0)
- [https://www.dropbox.com/s/0tqj634hrihg9vw/simple\\_teapot\\_frustum.txt?dl=0](https://www.dropbox.com/s/0tqj634hrihg9vw/simple_teapot_frustum.txt?dl=0)
- 왜 아무것도 보이지 않을까?
- 원근 투영 시 기본적인 Camera 위치는 ? (0, 0, 0)





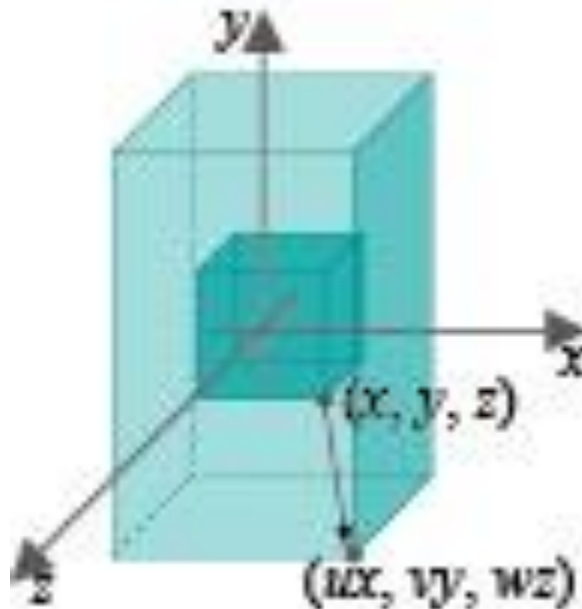
- 
- 이번엔 `glTranslatef` 주석을 제거하고 실행해 보자
  - Viewing frustum안에 있는가?

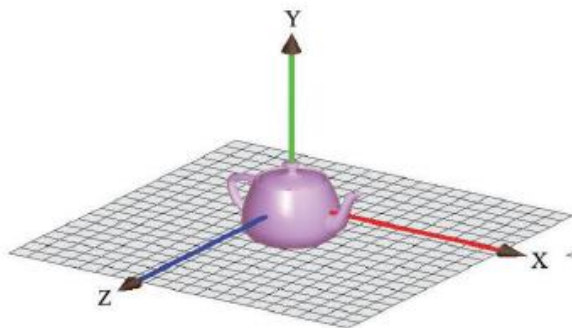


---

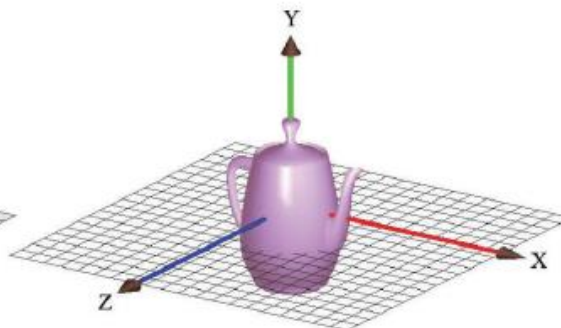
## ■ 2. Scaling

- OpenGL has **glScalef(u, v, w)**
- It maps each point  $(x, y, z)$  of an object to the point  $(ux, vy, wz)$ . This has the effect of stretching objects by a factor of  $u$  in  $x$ -direction,  $v$  in  $y$ -direction,  $w$  in  $z$ -direction

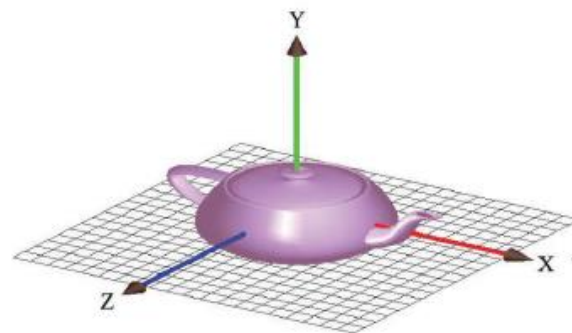




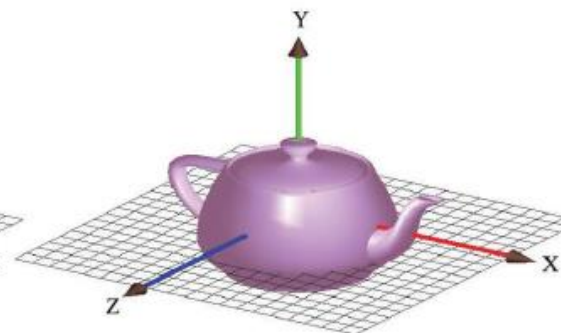
(A)  $glScalef(1.0, 1.0, 1.0);$



(B)  $glScalef(1.0, 2.5, 1.0);$



(C)  $glScalef(2.0, 1.0, 2.0);$



(D)  $glScalef(2.0, 2.0, 2.0);$

- 
- [https://www.dropbox.com/s/8fv0klIrhrrn0951/circle\\_scale.txt?dl=0](https://www.dropbox.com/s/8fv0klIrhrrn0951/circle_scale.txt?dl=0)
  - glScalef 부분의 주석을 제거하고 실행해보자

- 3D Teapot에 대해서 scaling
- 단, 이 경우 변환이 2번 연속적으로 발생하므로 복합 변환에 해당한다
- 복합 변환의 경우 변환 순서에 영향을 받지만 자세한 건 복합 변환 부분에서 배운다
- `void drawScene()`
- `{`
- `glClear(GL_COLOR_BUFFER_BIT);`
- `glTranslatef(0.0, 0.0, -10.0);`
- `glScalef(1.0, 2.5, 1.0);`
- `glutWireTeapot(5.0);`
- `glFlush();`

- 
- [https://www.dropbox.com/s/4rzrng6hkzaqqek/simple\\_teapot\\_scale.txt?dl=0](https://www.dropbox.com/s/4rzrng6hkzaqqek/simple_teapot_scale.txt?dl=0)

**glScalef 부분의 주석을 없애보자**

- 이번엔 reflection을 수행해 보자
- `glScalef(-1.0, 1.0, 1.0);`

