Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

- **Affine transformations**

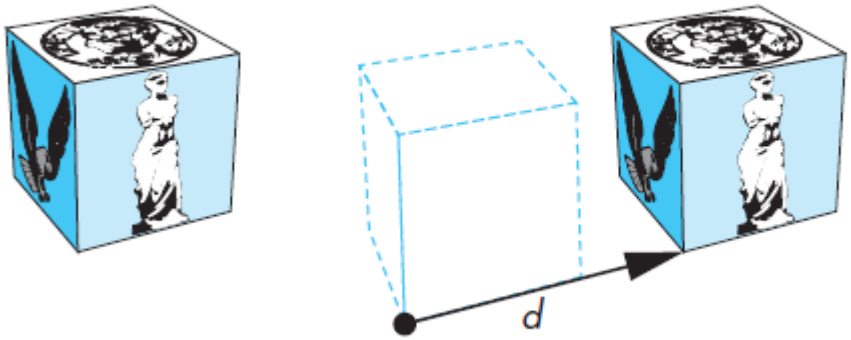
- **Affine transformations are the most common transformations used in CG**
- **예: Translation, Scaling, Rotation, Shear**
- **A succession of affine transformations can easily be combined into a single overall affine transformation, and affine transformations permit a compact matrix representation**
- **Affine transformations have a simple form**
- **Point P(x, y, z), Point P'(x', y', z') after affine transformation**

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

- Affine 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

- **Translation**

- **1. Translation is an operation that displaces points by a fixed distance in a given direction. To specify a translation, we need only specify a displacement vector d, because the transformed points are given by**
- **P'=P+d**
- **for all points P on the object**



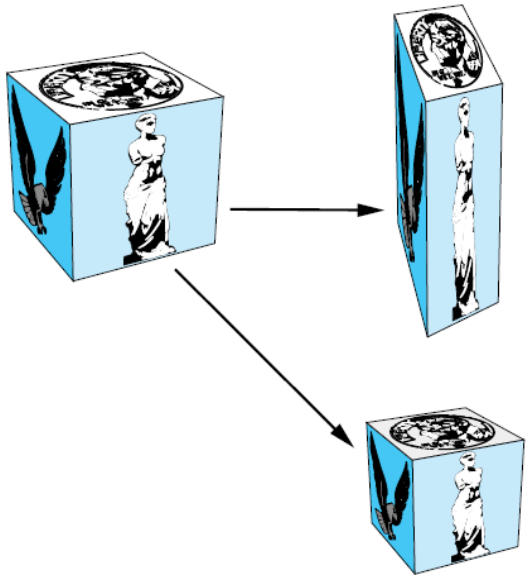
(a) (b)

- **Translation (천이)**
- **P(x, y, z)가 P'(x', y', z')로 x축으로 Tx, y축으로 Ty,z축으로Tz만큼 translate**
- **x'=x+Tx, y'=y+Ty, z'=z+Tz**
- 동차좌표로 표현시: **P'=TP**

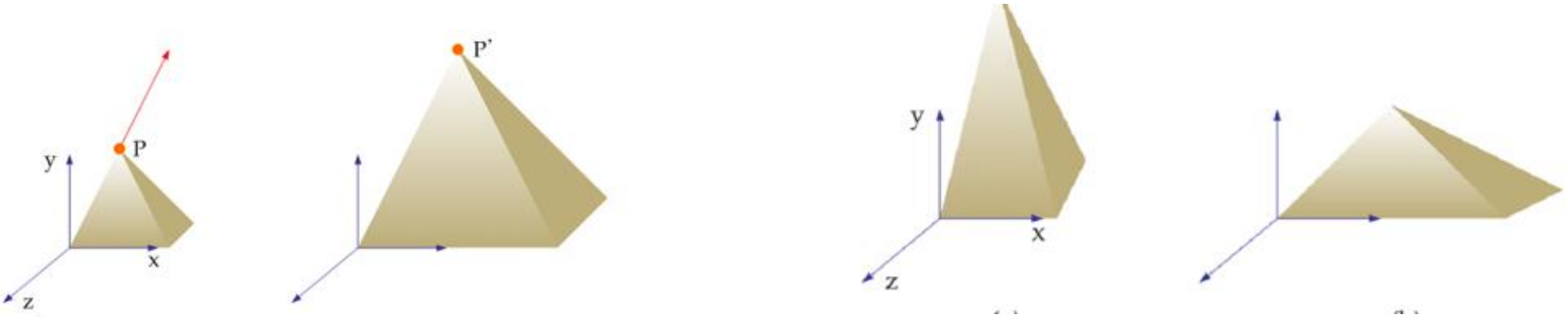
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \mathrm{Tx} \\ 0 & 1 & 0 & \mathrm{T}y \\ 0 & 0 & 1 & \mathrm{T}z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix},$$

- **T is called the translation matrix.**
- **Q: 동차 좌표를 쓰지 않고 P'=TP 형태로 표현 가능한가?**

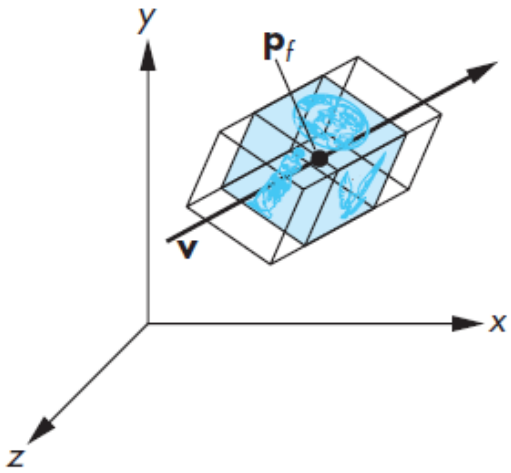
- **2. Scaling**
- **Scaling is an affine non-rigid-body transformation by which we can make an object bigger or smaller**
- **Below figure illustrates both uniform scaling in all directions and scaling in a single direction**



- 만일 모든 배율이 같은 경우 균등 크기 조절 (uniform scaling),
- 하나라도 다르면 차등 크기 조절 (differential scaling)이라 한다
- 최초의 object가 아래의 가장 왼쪽과 같을 때 각각이 균등 크기 조절인지 차등 크기 조절인지 살펴보자
- There is one point, the origin in this case, that is unchanged. We call this point the fixed point of the transformation



- **Scaling transformations have a fixed point.** **Hence, to specify a scaling, we can specify the fixed point, a direction in which we wish to scale, and a scale factor.**
- **We consider scaling with a fixed point at the origin**



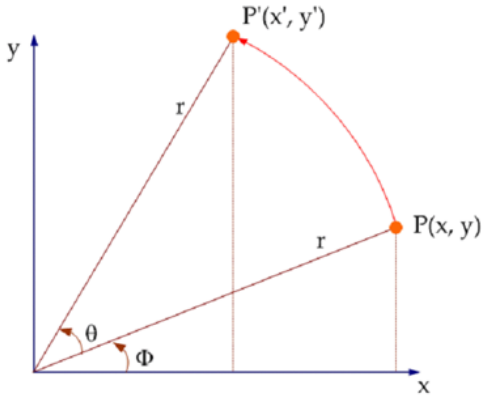
- **Scaling (크기 조절)**
- **점 P(x, y, z)가 점 P'(x', y', z')로 x축 Sx, y축 Sy, z축 Sz 만큼 각각 크기 조절 변환**
- **Sx, Sy, Sz는 각각 x,y,z,축 방향의 배율 (scale factor) 이고 1보다 크면 확대, 작으면 축소**
- **x'=Sx·x, y'=Sy·y , z'=Sz·z**
- 동차좌표로 표현시: **P'=SP**,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **S is called a scaling matrix**
- **Q: 앞의 transformation matrix와 같은 부분은? The final row !**

- **3. Rotation**
- **Rotation is more complicated to specify than translation because we must specify more parameters**
- **We start with the simple example of rotating a point about the origin in a 2D plane**
- **There is one point (the origin) that is unchanged by the rotation and we call this point the fixed point**

- **2D Rotation**
- **P(x,y)가 원점을 중심으로 반시계 방향으로 θ 만큼 P'(x',y')로 회전**
- **Φ: 원점과 점 P를 연결한 선분이 x축과 이루는 각, r: 회전 반지름**



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

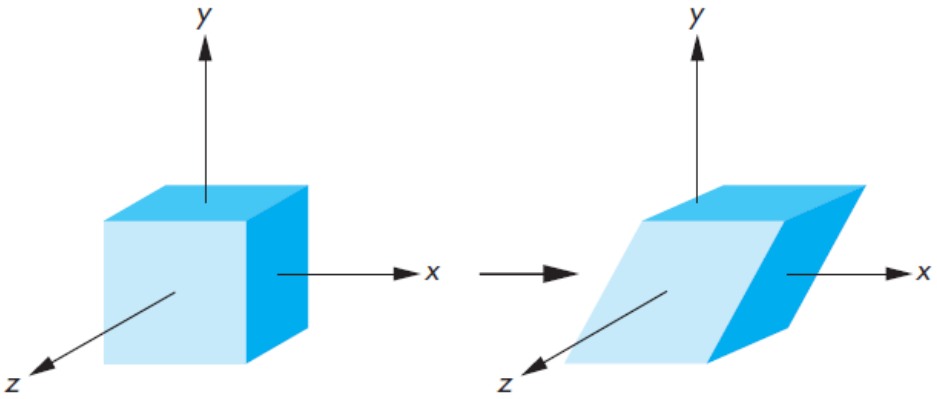
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta = x\sin\theta + y\cos\theta$$

$$P' = R \cdot P$$

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

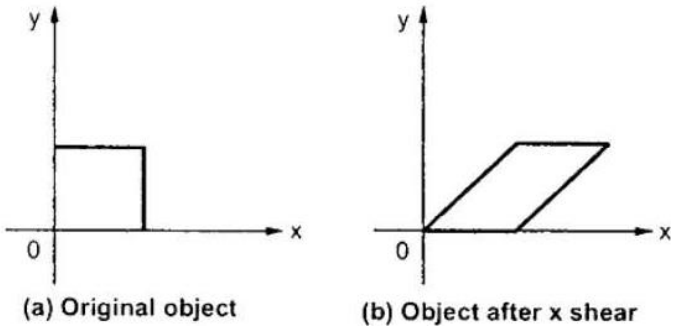
- 예: 어떤 점 P(3, 5)가 원점을 기준으로 반시계 방향으로 60도 만큼 회전하였을 때의 위치를 구해보자
- Find the transformed point, Q, caused by rotating P=(3,5) about the origin through an angle of 60도

- **4. Shear (shearing)**
- **Consider a cube centered at the origin. If we pull the top to the right and the bottom to the left, we shear the object in the x direction.**
- **Note that neither the y nor the z values are changed by the shear. Therefore, we call this operation "x shear"**



4. x축 방향의 shearing (x-shear)

- Each x-coordinate is translated by an amount that increase linearly with y
- The y-coordinate of each point is unaffected
- 점 P(x, y) 가 점 P'(x', y')로 x축 shearing시
- x'=x+hy
- y'=y
- h: 전단 인수 (shearing factor)



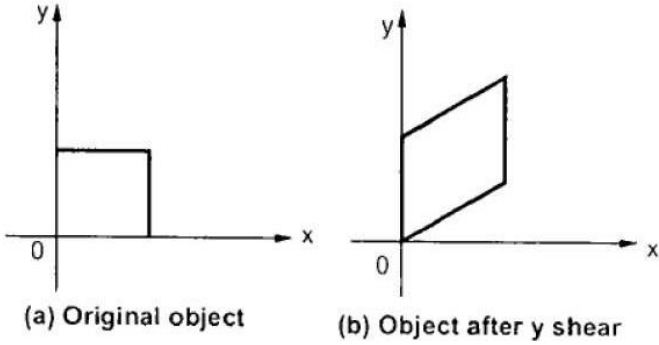
(a) Original object (b) Object after x shear

- 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?

- **2D Shearing; Y축 방향의 shearing (y-shear)**
- 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)



(a) Original object (b) Object after y shear

- 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 변환**
- **Translation, scaling, rotation, shearing 모두 affine 변환에 속함**
- **Homogeneous coordinates를 사용시에 모두 아래와 같은 형태의 행렬 곱 형태로 표현 가능**
- **Point P(x, y, z), Point P'(x', y', z') after transformation**
- **There are 12 elements that we can select. It is called 12 degrees of freedom (DOF)**

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

- **복합 변환: Composite of transformations**

- 복합 변환: Composite of transformations (concatenation of transformations)
- 일반적으로 물체에 대한 변환이 여러 개가 연속적으로 가해질 수 있다
- 앞에서 배운 변환이 연속적으로 가해지는 것을 복합 변환이라 한다
- 복합 변환의 경우 변환의 순서가 매우 중요하다
- 행렬 곱은 교환 법칙 성립 안함 $T_1 T_2 \neq T_2 T_1$

- 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?
- $\boldsymbol{P' = T1 \cdot P}$
- $\boldsymbol{P'' = T2 \cdot P'}$
- $\boldsymbol{P'' = T2 \cdot T1 \cdot P}$
- $\boldsymbol{P'' = T2(T1 \cdot P)}$

- 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 변환이다

- **Modelview matrix**

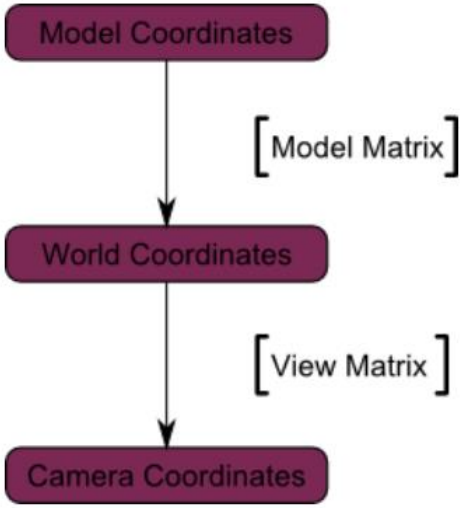
- OpenGL 좌표계 변환 순서

1. Local (model) coordinate (모델 좌표계)
2. World coordinate (세계 좌표계)
3. Eye coordinate (view, camera coordinate)
4. Clip coordinate (절단 좌표계)
5. Normalized device coordinate (NDC)
6. Screen coordinate (화면 좌표계)

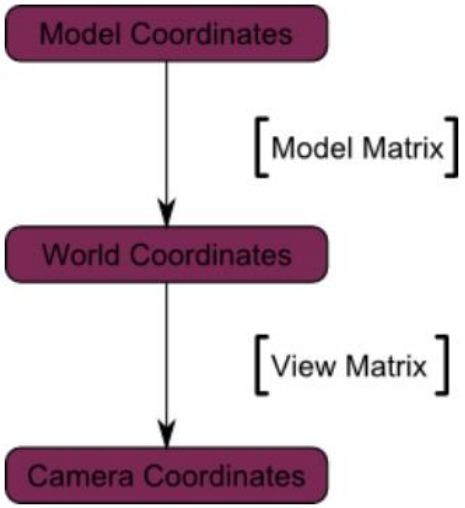
1. Local (model) coordinate (모델 좌표계): all vertices defined relatively to the center of the model

2. World coordinate (세계 좌표계): all vertices defined relatively to the center of the world

3. Eye coordinate (view, camera coordinate): all vertices relatively to the camera



- **There are 4x4 matrices that represent the transformation from model coordinates to worlds coordinates and from world coordinates to eye coordinates. These transformations are concatenated together into the model-view transformation, which is specified by the model-view matrix**



- **The model view matrix normally is an affine transformation matrix and has only 12 degrees of freedom.**
- **The model-view matrix is initialized to an identity matrix**

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

- **The projection matrix is also 4x4 matrix but is not affine**

- **Current transformation matrix (CTM)**

- OpenGL is a state machine
- There is a current transformation matrix (CTM) that holds the current state, 4X4 affine matrix
- Initially, we will set it to the 4X4 identity matrix
- CTM=I
- (Recall ‘glLoadIdentity’ function in OpenGL)

- **The CTM (C) is part of the pipeline; thus, if p is a vertex specified in the application, then the pipeline produces Cp**
- **If we use a CTM, we can regard it as part of the state of the system**

- 많은 그래픽스 시스템에서는 복합 변환을 수행하기 위하여 CTM 이라는 개념을 사용한다

- 예: // CTM = I (초기)

$$CTM = I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- 변환 1 (T1) // CTM=I ·T1=T1

  변환 2 (T2) // CTM=I ·T1·T2=T1·T2

  물체 (V) // CTM=T1·(T2·V)

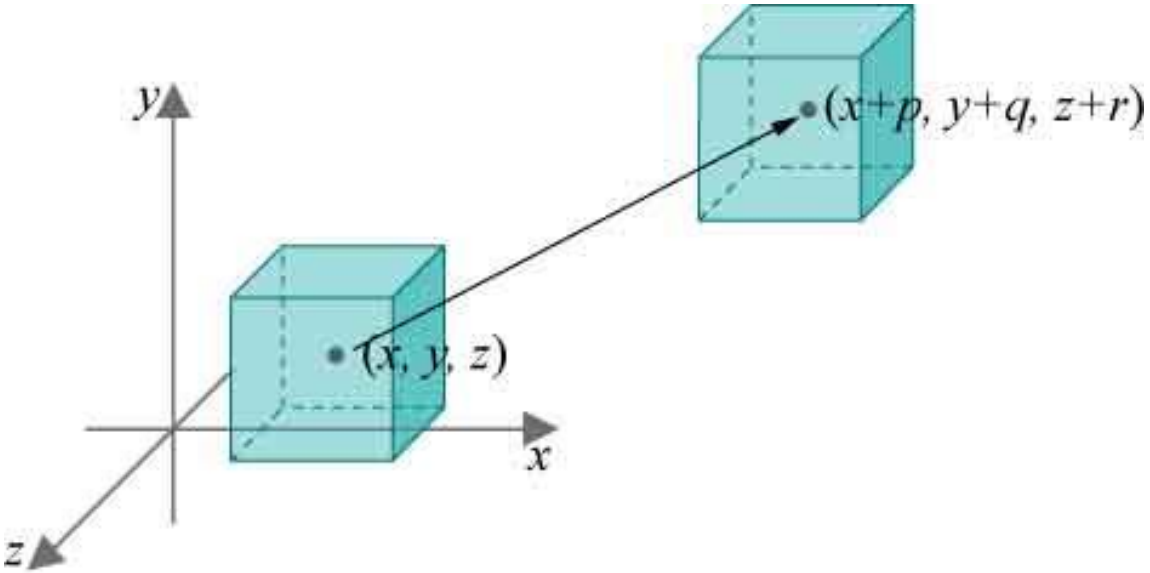
즉, OpenGL에서의 복합 변환은 역순으로 진행되는 것처럼 보인다

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

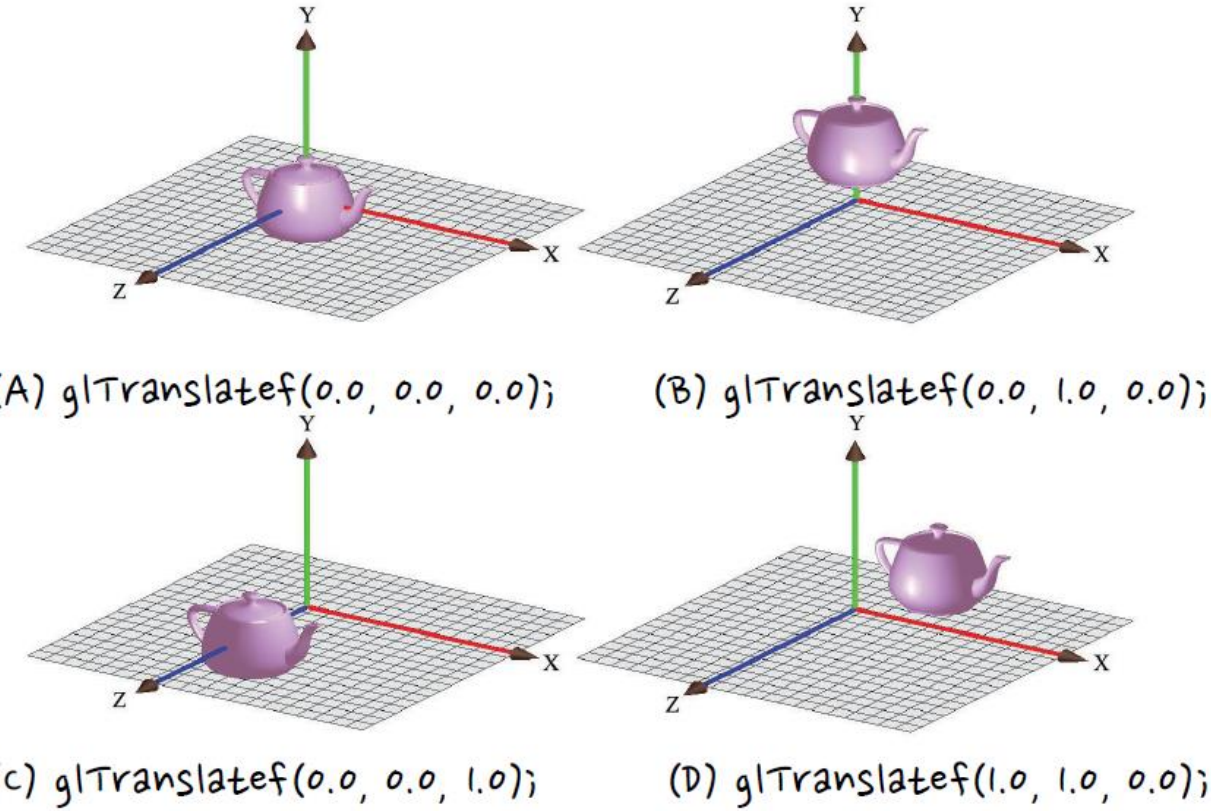
	// CTM=I
glScalef(sx, sy, sz);	// CTM=I· S
glRotatef(theta, vx, vy, vz);	// CTM=I· S · R
Object 1(v)	// I· S · (R · v)

- **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)**



- **glm library를 사용한 translation: 출처**
- **Tutorial 3 : 행렬(매트릭스) (opengl-tutorial.org)**

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

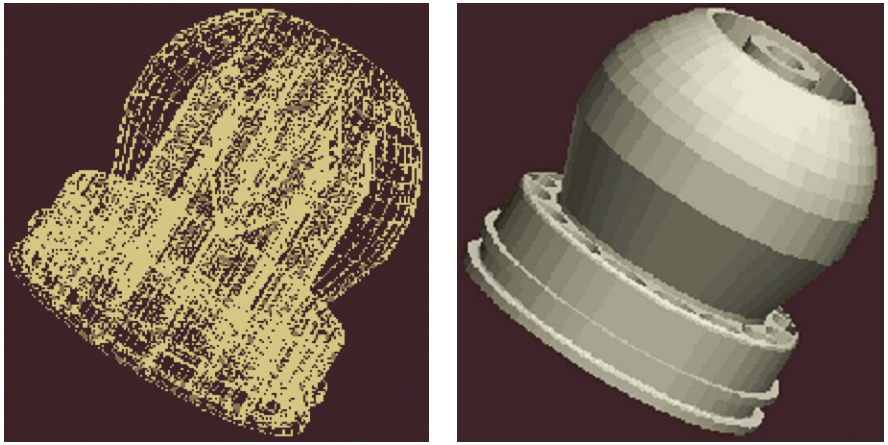
C++ 에서, GLM을 사용:

```
#include <glm/gtx/transform.hpp> // after <glm/glm.hpp>

glm::mat4 myMatrix = glm::translate(glm::mat4(), glm::vec3(10.0f, 0.0f, 0.0f));
glm::vec4 myVector(10.0f, 10.0f, 10.0f, 0.0f);
glm::vec4 transformedVector = myMatrix * myVector; // guess the result
```

- **GLUT Object and Translation**

- **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**

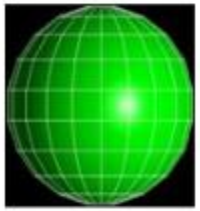


- **GLUT object들의 예 (각각 solid, wireframe rendering)**

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)

- 예: Chapter 4: Box.cpp
- “glFrustum”, 5x5x5 box
- Comment out the statement
- “glTranslatef(0.0, 0.0, -15.0);”
- What do you see? nothing

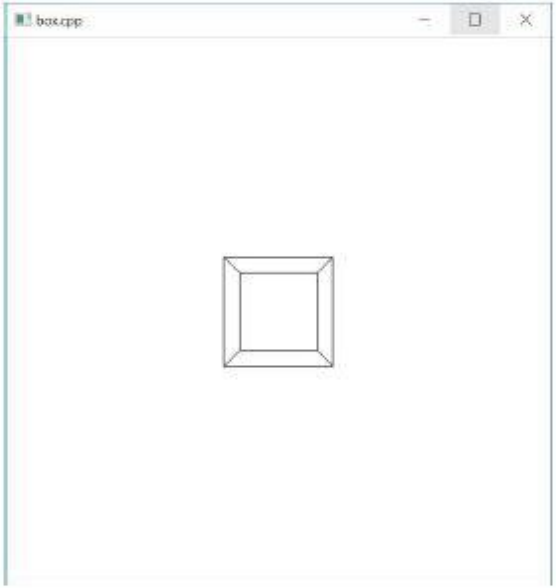


Figure 4.1: Screenshot of box.cpp.

- **“glTranslatef(0.0, 0.0, -15.0)” pushes the box 15units in the –z direction, to place it inside the viewing frustum**

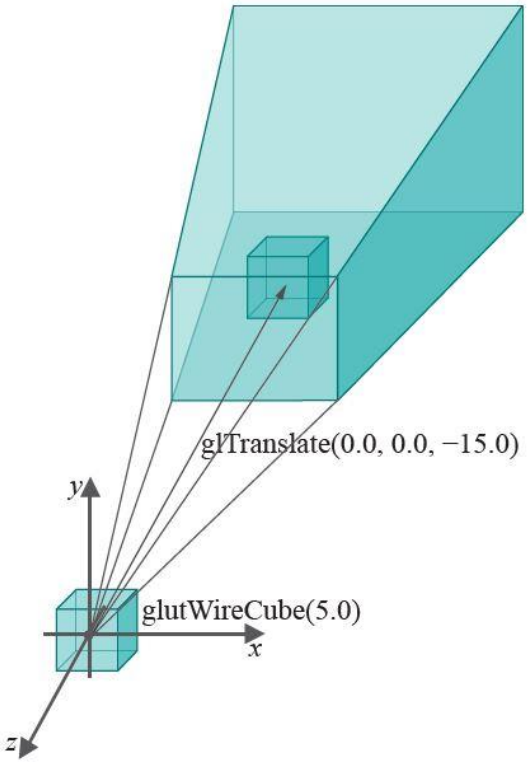


Figure 4.3: Translating into the viewing frustum.

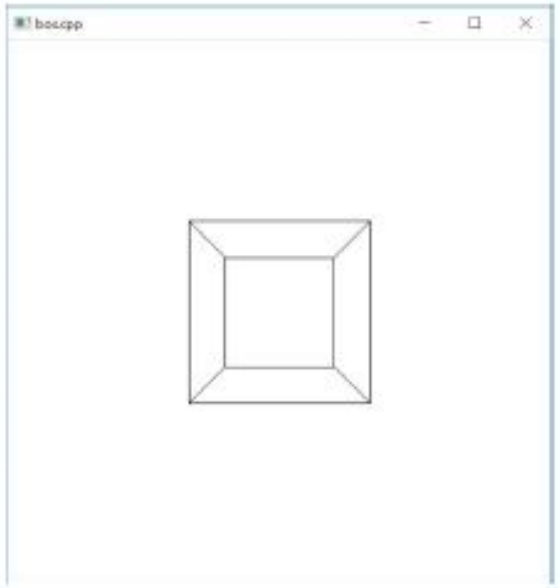
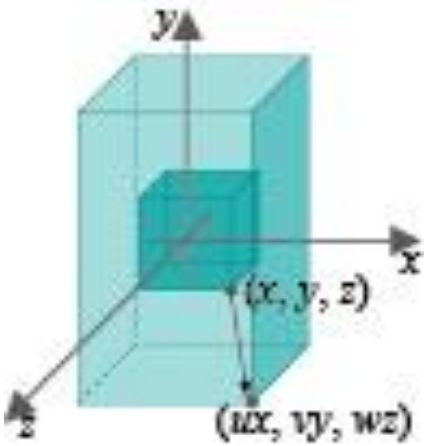
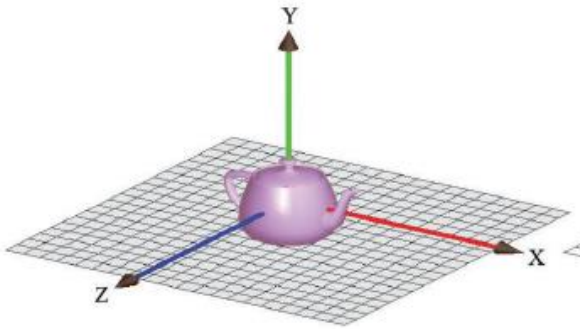


Figure 4.4: Screenshot of `box.cpp` with `glTranslatef(0.0, 0.0, -10.0)` instead (compare Figure 4.1).

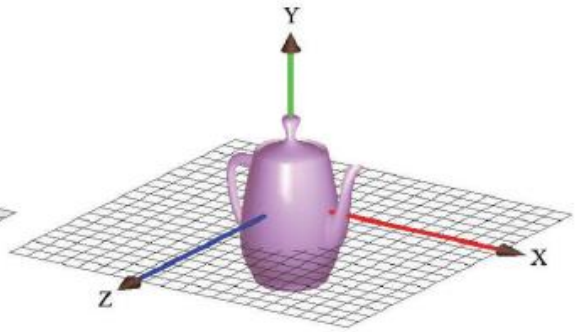
- **2. Scaling**

- 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
- Scaling with a fixed point at the origin

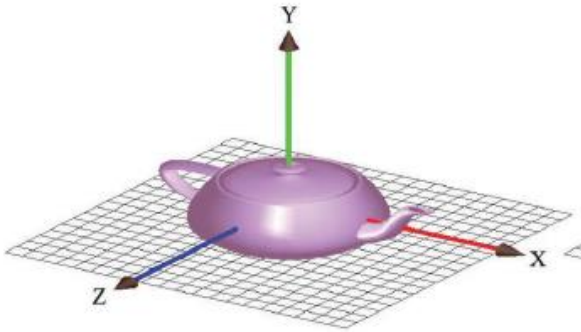




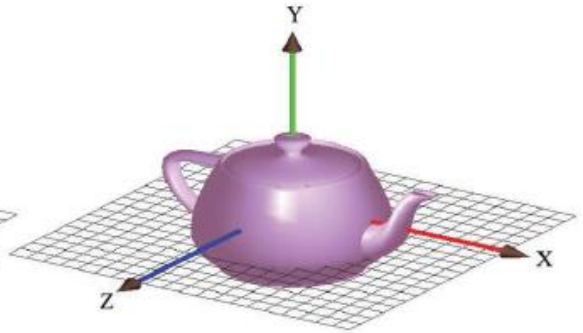
(A) glScalef(1.0, 1.0, 1.0);



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



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



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

- 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();
- }

예: add a scaling command

```
// Modeling transformations.
glTranslatef(0.0, 0.0, -15.0);
glScalef(2.0, 3.0, 1.0);
glutWireCube(5.0); // Box.
```

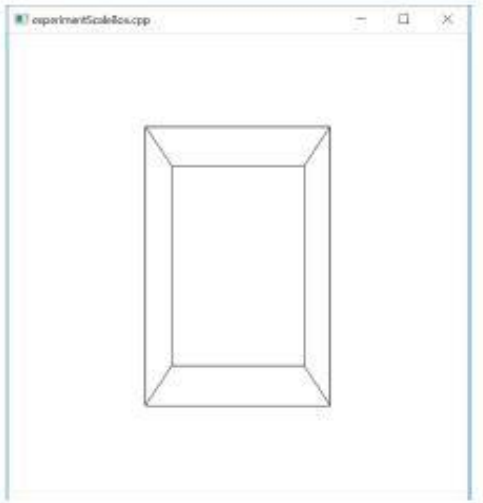
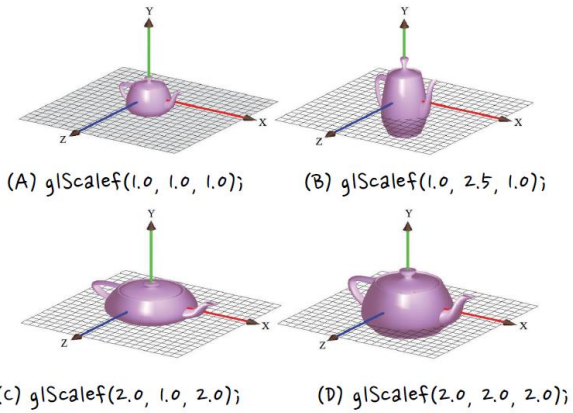


Figure 4.5: Screenshot of Experiment 4.3.

- **Change the modeling transformation and object definition part of box.cpp**

```
// Modeling transformations.
glTranslatef(0.0, 0.0, -15.0);
glScalef(1.0, 1.0, 1.0);
glutWireTeapot(5.0);
```

- **Change the scaling parameters**



- **The transformation (x, y, z)->(-x, y, z) is a mirror-like reflection about the yz plane.**
- **glScalef(-1.0, 1.0, 1.0)**

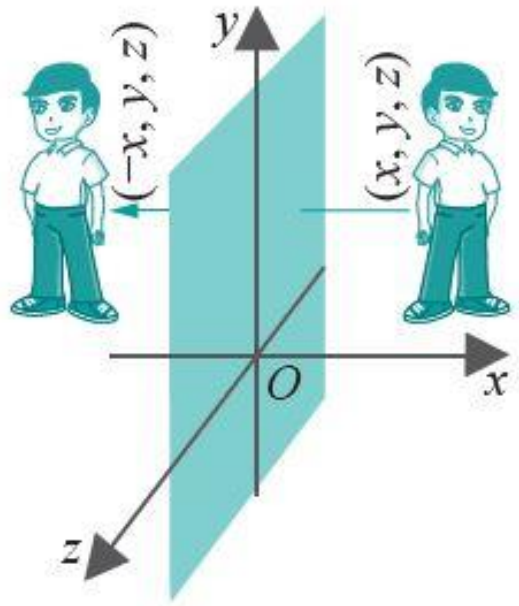


Figure 4.8: Reflection in the yz-plane.

- **glm library를 사용한 scaling: 출처**
- **Tutorial 3 : 행렬(매트릭스) (opengl-tutorial.org)**

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

C++ 에서 :

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm::mat4 myScalingMatrix = glm::scale(2.0f, 2.0f ,2.0f);
```