Chap 4. Geometric Objects and Transformations

3D CG 요소

- Objects : 목수, 디자이너, 건축가, …
  - 물체를 어떻게 만들 것인가 ?
    - 물체 표현법, 물체 위치화, 물체 변형
- Light : 조명 감독
  - 조명을 어떻게 비출 것인가 ?
    - 조명 위치, 조명 방향, …
- Camera : 카메라 감독
  - 주어진 환경에서 원하는 이미지를 만들기 위해서 어떤 위치에서 어떤 방향으로 어떤 렌즈를 사용할 것인가 ?
    - 카메라 위치, 카메라 방향, …
- 공통점
  - 3차원 공간, 기하학적 정보

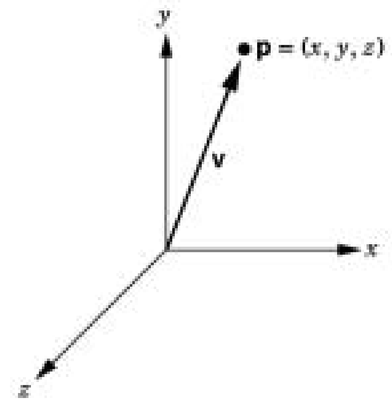
➔ 벡터, 행렬, 물체 표현법

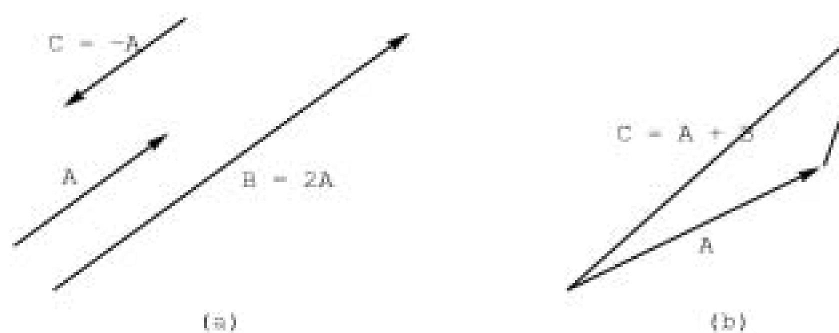
4.1 Scalars, Points, and Vectors

1.벡터의 정의
2. 벡터의 연산: 내적, 외적
3. 직선의 표현법
4. 평면의 표현법

Geometric View

- scalar : a real number 0.27
- point : location in space (0.27, 0.35, 0.19)
  - position 만 있음. no size
- vector : direction + magnitude
  - = directed line segment
- 연산 법칙
  - vector = point – point
  - point = point + vector
  - vector = vector + vector





Mathematical View : Space

- vector space : scalar, vector 가 정의됨
  - scalar-vector multiplication $\vec{v} = 2\vec{u}$
  - vector-vector addition $\vec{w} = \vec{u} + \vec{v}$
  - vector = sum of basis vectors $\vec{w} = a\vec{i} + b\vec{j} + c\vec{k}$

- affine space : point 개념이 들어감
  - vector-point addition $\mathbf{p} = \mathbf{q} + \vec{v}$
  - point-point subtraction $\vec{v} = \mathbf{p} - \mathbf{q}$

- Euclidean space : distance 개념이 들어감 $l = \sqrt{x^2 + y^2 + z^2}$

Computer Science View

- ADT : abstract data types
  - C++ / Java class 개념
  - 구현하기 쉽고, ADT 개념에 최적

- we need the operations:
  - scalar : α, β, γ
  - point : **p**, **q**, **r** (또는 P, Q, R)
  - vector : **u**, **v**, **w**

Operations

- magnitude of a vector

$$|\mathbf{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

$$|\alpha\,\mathbf{v}| = |\alpha|\,|\mathbf{v}|$$

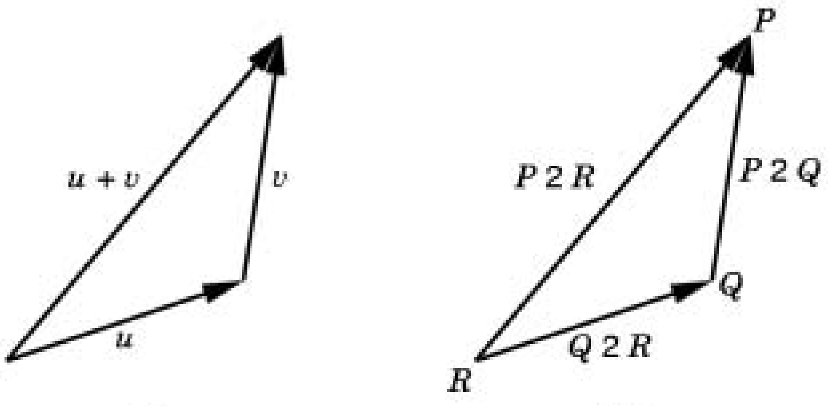
- vector-point relationship

$$\mathbf{v} = P - Q$$

$$P = \mathbf{v} + Q$$

- point-point subtraction

$$\mathbf{v} + \mathbf{u} = (P - Q) + (Q - R) = P - R$$



(a) (b)

Lines

- line : point, vector 로 정의
  - parametric form

    $$P(\alpha) = P_0 + \alpha \mathbf{d}$$

  - affine form

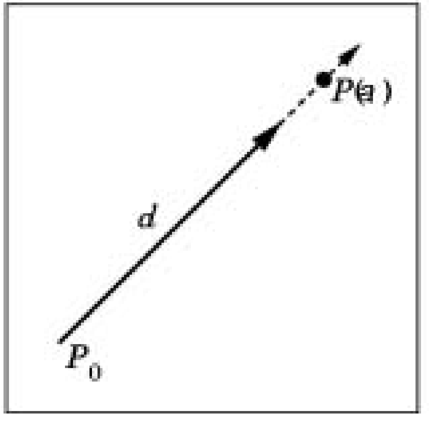
    $$P = Q + \alpha \mathbf{v}$$

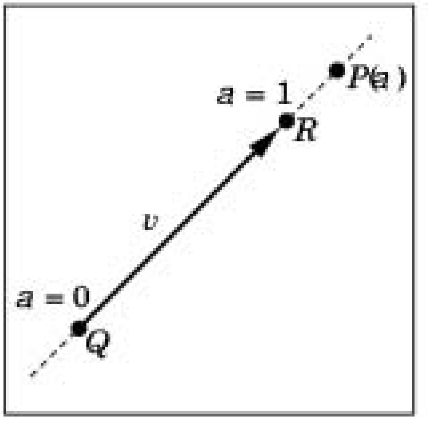
    $$\mathbf{v} = R - Q$$

    $$P = Q + \alpha(R - Q) = \alpha R + (1 - \alpha)Q$$

    $$P = \alpha_1 R + \alpha_2 Q$$

    $$\alpha_1 + \alpha_2 = 1$$





Convexity

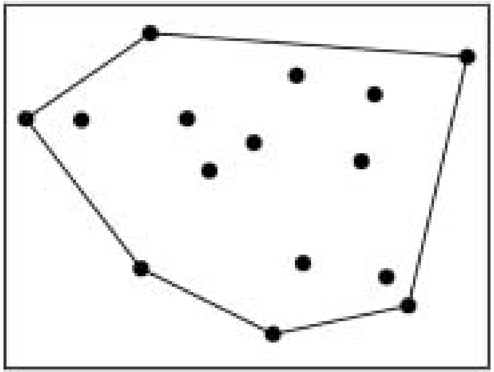
- convex object
  - object 내의 2 점을 연결한 선분이 항상 object 내에 포함되는 object
- 수학적 표현

$$n \text{ points}: P_1, P_2, P_3, \cdots, P_n$$

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \cdots + \alpha_n P_n$$

$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = 1$$

$$\alpha_i \geq 0, i = 1, 2, \cdots, n$$



- 기하학적 의미 : shrink wrapping

Product

- dot product : 내적

$$\mathbf{u}\cdot\mathbf{v} = |\mathbf{u}||\mathbf{v}|\cos\theta = u_x v_x + u_y v_y + u_z v_z$$

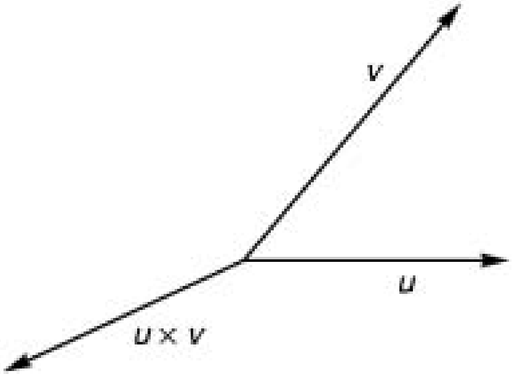
  – orthogonal : dot product 가 0 인 2개의 vector

$$\mathbf{u}\cdot\mathbf{v} = 0 \quad \Rightarrow \quad \mathbf{u} \text{ and } \mathbf{v} \text{ are orthogonal}$$

- cross product : 외적

$$\mathbf{u}\times\mathbf{v} = \begin{vmatrix} i & j & k \\ u_x & u_y & u_z \\ v_x & v_y & v_z \end{vmatrix}$$

$$|\mathbf{u}\times\mathbf{v}| = |\mathbf{u}||\mathbf{v}||\sin\theta|$$

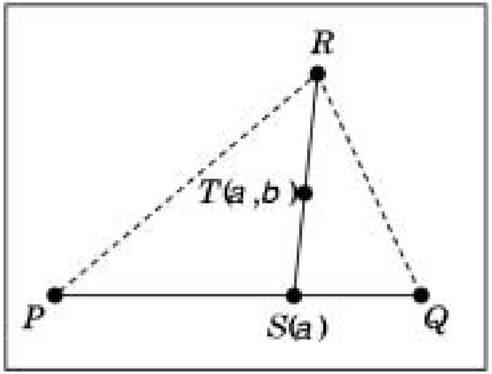


- right-handed coordinate system

Planes

- parametric form
  - line equation의 확장 (교재 참고)

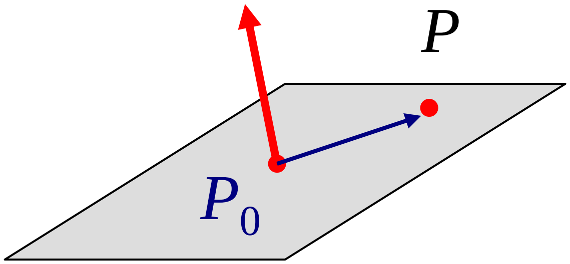
$$T(\alpha, \beta) = P_0 + \alpha\mathbf{u} + \beta\mathbf{v}$$



- plane equation

$$\mathbf{n} \cdot (P - P_0) = 0$$

$$\mathbf{n} = \mathbf{u} \times \mathbf{v} : \text{ normal vector}$$



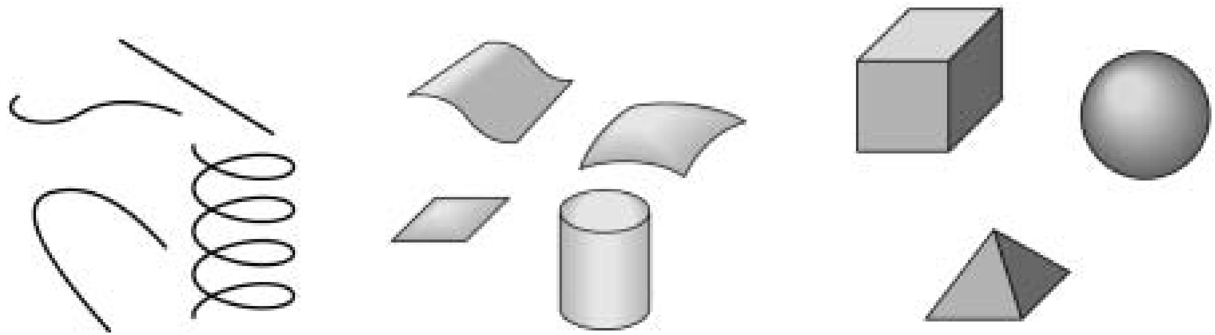
4.2 Three-dimensional Primitives

3차원 물체 표현법

1.곡선 2.곡면 3. 메쉬

Boundary Representation

- three-dimensional objects



curves surfaces volumetric objects

$$p(t) = (x(t), y(t), z(t))$$

$$s(u,v) = (x(u,v), y(u,v), z(u,v))$$

$$f(x, y, z) = 0$$

$$g(x, y, z) = 0$$

Boundary Representation

- What is the efficient way of representing 3D objects ?
  - 물체의 표면(surface)만 표현, 내부는 비었다고 가정
  - 물체의 표면은 꼭지점(vertex)을 연결해서 표현
  - 물체의 표면은 flat convex polygon 으로 구성

$\Rightarrow$ B-rep (boundary representation)

4.3 Coordinate Systems and Frames

물체 위치 표현법
(하나의 중심점 + 서로 수직인 3 벡터)

지역좌표계 : 물체 내부 상관 위치를 표현할 때
세계좌표계 : 물체를 원하는 위치에 놓을 때
카메라좌표계 : 카메라 위치에서 바라 볼 때

Coordinate System or Frame

- coordinate system : basis vector들로 정의
- frame : reference point (= origin) 가 더 필요

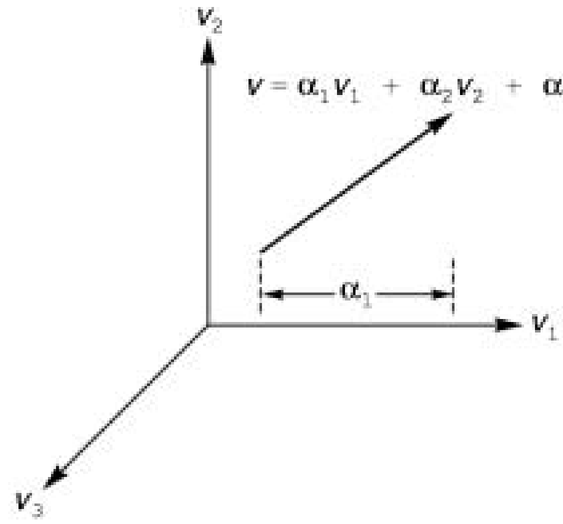
  - basis vectors : $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$
  - reference point : P_0

- vector :

$$\mathbf{w} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{bmatrix}$$

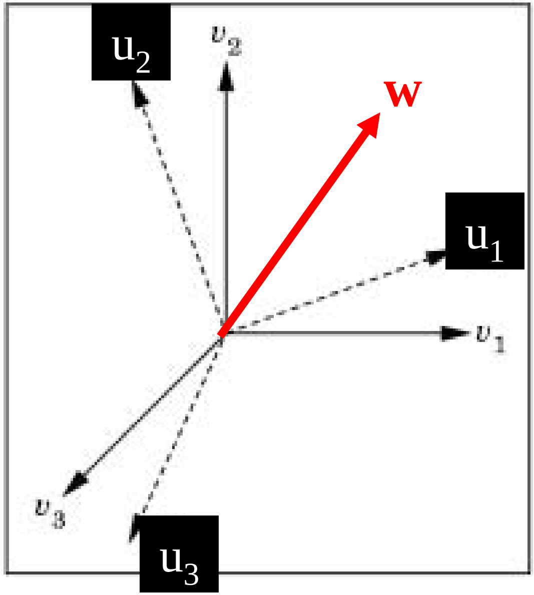
- point :

$$P = P_0 + \eta_1 \mathbf{v}_1 + \eta_2 \mathbf{v}_2 + \eta_3 \mathbf{v}_3$$



Coordinate Transform

- original basis vectors $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$
- new basis vectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$
- mapping



$$\mathbf{u}_1 = \gamma_{11}\mathbf{v}_1 + \gamma_{12}\mathbf{v}_2 + \gamma_{13}\mathbf{v}_3$$

$$\mathbf{u}_2 = \gamma_{21}\mathbf{v}_1 + \gamma_{22}\mathbf{v}_2 + \gamma_{23}\mathbf{v}_3$$

$$\mathbf{u}_3 = \gamma_{31}\mathbf{v}_1 + \gamma_{32}\mathbf{v}_2 + \gamma_{33}\mathbf{v}_3$$

$$\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\\ \mathbf{u}_3\end{bmatrix} = \begin{bmatrix}\gamma_{11} & \gamma_{12} & \gamma_{13}\\ \gamma_{21} & \gamma_{22} & \gamma_{23}\\ \gamma_{31} & \gamma_{32} & \gamma_{33}\end{bmatrix}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\end{bmatrix} = \mathbf{M}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix}\beta_1 & \beta_2 & \beta_3\end{bmatrix}\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\\ \mathbf{u}_3\end{bmatrix} = \begin{bmatrix}\beta_1 & \beta_2 & \beta_3\end{bmatrix}\mathbf{M}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\end{bmatrix} = \begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3\end{bmatrix}\begin{bmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\end{bmatrix}$$

Homogeneous Coordinates

- vector : 3 X 3 matrix 로 coordinate transform 가능

$$\mathbf{w} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3$$

- point

$$P = \eta_1 \mathbf{v}_1 + \eta_2 \mathbf{v}_2 + \eta_3 \mathbf{v}_3 + P_0$$

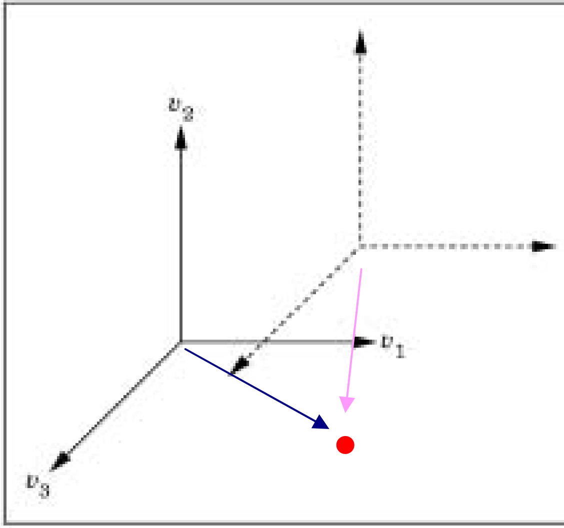
  - 어떻게 transform 할 것인가?
    - 4 X 4 matrix : homogeneous coordinate 도입

$$P = \eta_1 \mathbf{v}_1 + \eta_2 \mathbf{v}_2 + \eta_3 \mathbf{v}_3 + P_0 = \begin{bmatrix} \eta_1 & \eta_2 & \eta_3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ P_0 \end{bmatrix}$$

$$\mathbf{w} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ P_0 \end{bmatrix}$$

Coordinate Transform : Point

- original frame $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, P_0$
- new frame $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, Q_0$
- mapping



$$P = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + P_0 \quad \text{in original frame}$$

$$= \beta_1 \mathbf{u}_1 + \beta_2 \mathbf{u}_2 + \beta_3 \mathbf{u}_3 + Q_0 \quad \text{in new frame}$$

$$\mathbf{u}_1 = \gamma_{11}\mathbf{v}_1 + \gamma_{12}\mathbf{v}_2 + \gamma_{13}\mathbf{v}_3$$
$$\mathbf{u}_2 = \gamma_{21}\mathbf{v}_1 + \gamma_{22}\mathbf{v}_2 + \gamma_{23}\mathbf{v}_3$$
$$\mathbf{u}_3 = \gamma_{31}\mathbf{v}_1 + \gamma_{32}\mathbf{v}_2 + \gamma_{33}\mathbf{v}_3$$
$$Q_0 = \gamma_{41}\mathbf{v}_1 + \gamma_{42}\mathbf{v}_2 + \gamma_{43}\mathbf{v}_3 + P_0$$

$$\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}$$

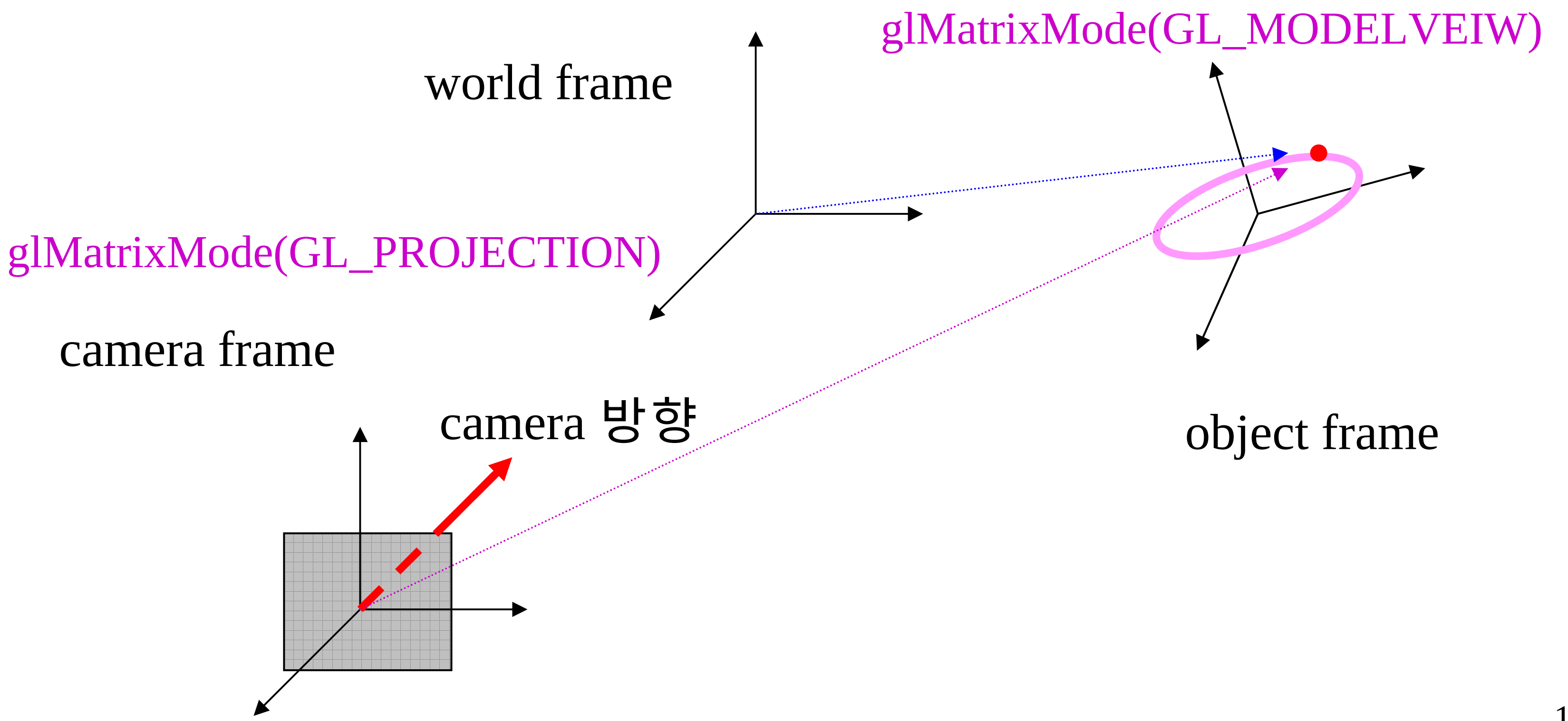
$$\mathbf{w} = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ Q_0 \end{bmatrix} = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & 1 \end{bmatrix} \mathbf{M} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ P_0 \end{bmatrix} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ P_0 \end{bmatrix}$$

Frames in OpenGL

- two transform matrices

object frame ⇒ world frame glMatrixMode(GL_MODELVEIW)

world frame ⇒ camera frame glMatrixMode(GL_PROJECTION)

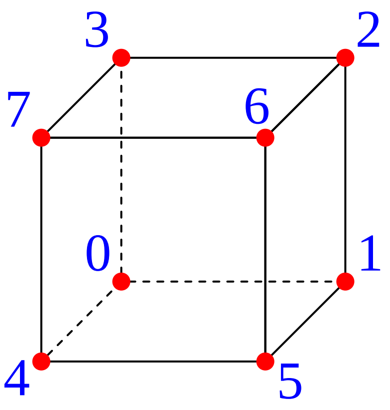


4.4 Modeling a Colored Cube

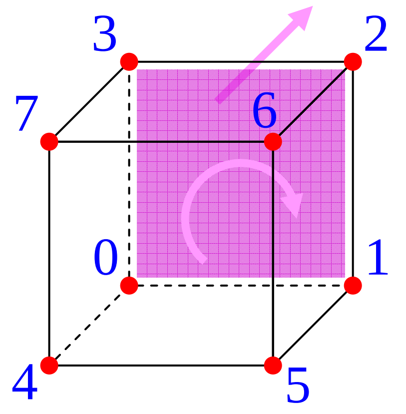
Modeling a Cube

- vertex 정의

```
GLfloat vertices[8][3] = {
  { –1.0, –1.0, –1.0 },  // 0
  { 1.0, –1.0, –1.0 },   // 1
  { 1.0, 1.0, –1.0 },    // 2
  { –1.0, 1.0, –1.0 },   // 3
  { –1.0, –1.0, 1.0 },   // 4
  { 1.0, –1.0, 1.0 },    // 5
  { 1.0, 1.0, 1.0 },     // 6
  { –1.0, 1.0, 1.0 }     // 7
};
```



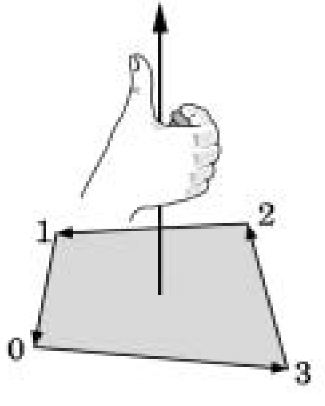
Modeling a Cube



- face 정의

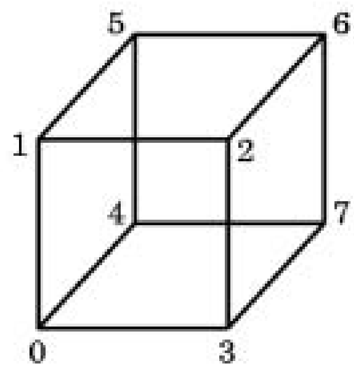
```
glBegin(GL_POLYGON);
  glVertex3fv(vertices[0]);
  glVertex3fv(vertices[3]);
  glVertex3fv(vertices[2]);
  glVertex3fv(vertices[1]);
glEnd( );
```

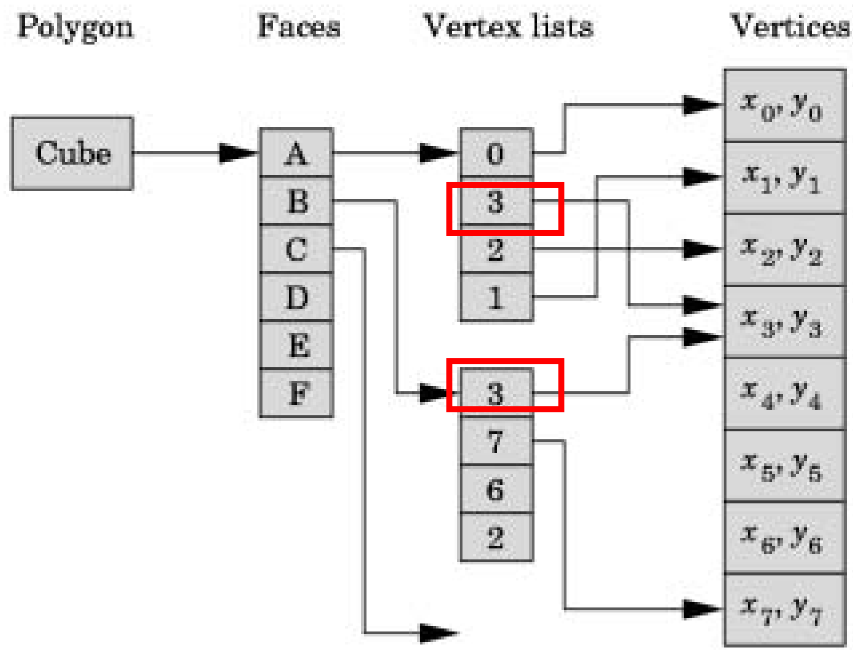
- outward facing
  - normal vector 는 항상 외부를 향한다.
  - right-hand rule



Data Structure

- vertex-list representation
  - 1 개의 vertex를 3개의 face가 공유
  - list 구조 : index 로 vertex 를 share

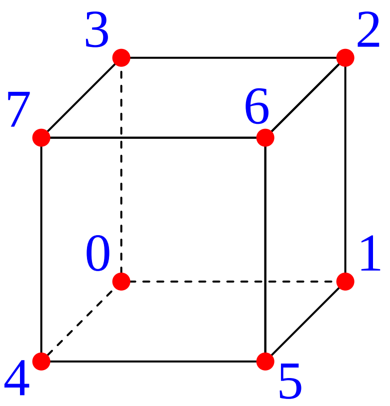


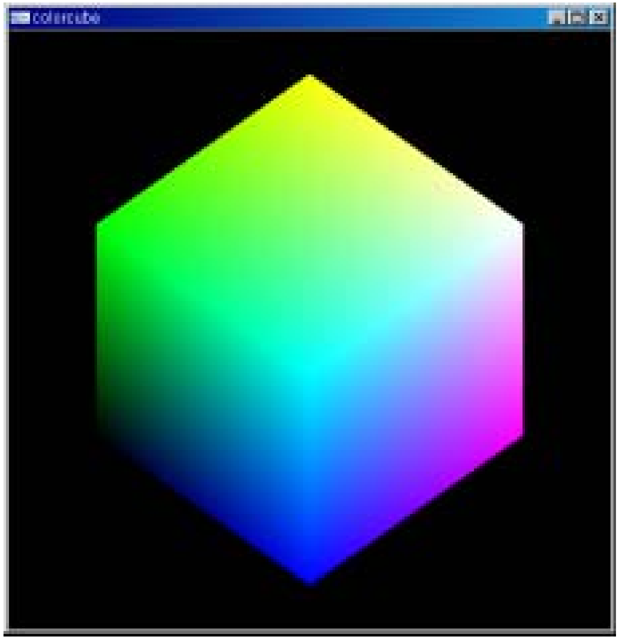


Color Cube

- vertex 마다 color 설정

```
GLfloat colors[8][3] = {
  { 0.0, 0.0, 0.0 },  // black
  { 1.0, 0.0, 0.0 },  // red
  { 1.0, 1.0, 0.0 },  // yellow
  { 0.0, 1.0, 0.0 },  // green
  { 0.0, 0.0, 1.0 },  // blue
  { 1.0, 0.0, 1.0 },  // magenta
  { 1.0, 1.0, 1.0 },  // white
  { 0.0, 1.0, 1.0 }   // cyan
};
```





Face 출력 함수

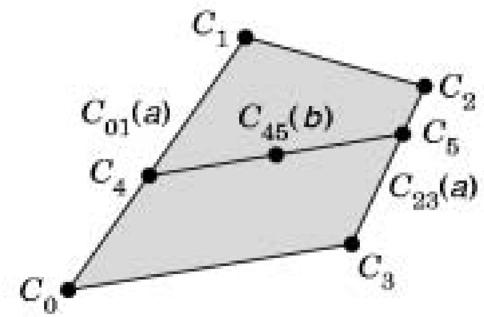
```
void  quad(int a, int b, int c , int d) {
  glBegin(GL_POLYGON);
    glColor3fv(colors[a]);
    glVertex3fv(vertices[a]);
    glColor3fv(colors[b]);
    glVertex3fv(vertices[b]);
    glColor3fv(colors[c]);
    glVertex3fv(vertices[c]);
    glColor3fv(colors[d]);
    glVertex3fv(vertices[d]);
  glEnd();
}
```

```
/* face 별로 출력 */

void colorcube(void) {
  quad(0,3,2,1);
  quad(2,3,7,6);
  quad(0,4,7,3);
  quad(1,2,6,5);
  quad(4,5,6,7);
  quad(0,1,5,4);
}
```

Bilinear Interpolation

- face 의 각 vertex 가 서로 다른 색깔일 수도 있다.
  - face 전체를 어떻게 칠할 것인가?
- bilinear interpolation
  - edge 상의 점 : linear interpolation



$$C_{01}(\alpha) = (1-\alpha)C_0 + \alpha C_1$$

$$C_{23}(\alpha) = (1-\alpha)C_2 + \alpha C_3$$

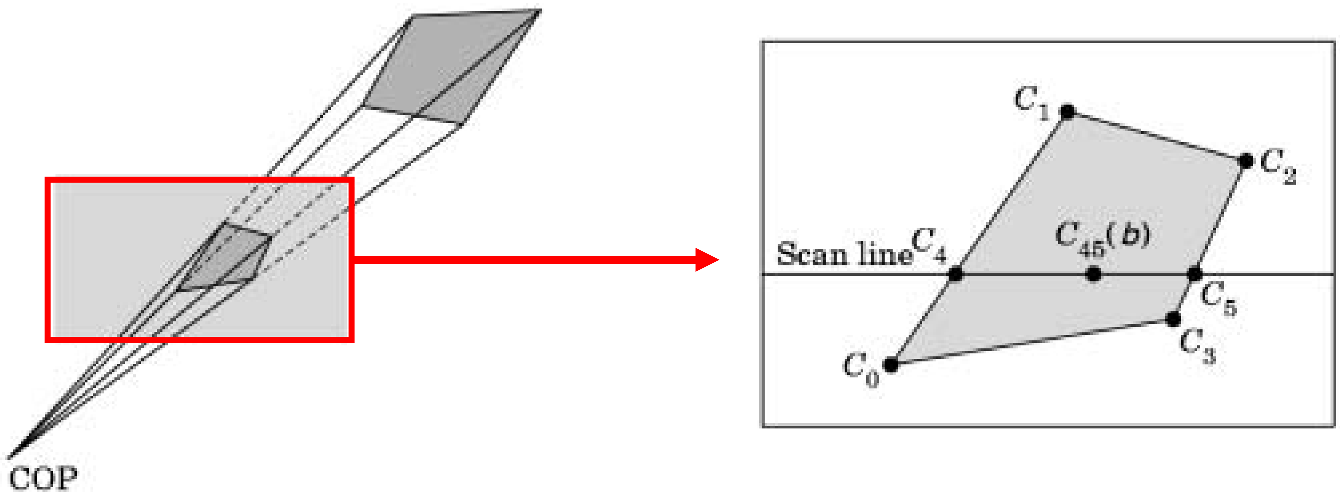
  - face 상의 점 : 다시 linear interpolation

$$C_{45}(\beta) = (1-\beta)C_4 + \beta C_5$$

  - 문제점 !   flatness 가 보장되어야 한다…

Scan-line Interpolation

- flatness 를 해결하기 위한 방법
  - polygon 을 일단 screen 에 project 한 후에,
  - bi-linear interpolation



Vertex Array

- object 를 draw 하는 과정에서의 bottleneck ?
  - 1개의 vertex 가 3번 사용된다.
  - vertex 정보 : 좌표 3개 × 4 byte / float = 12 bytes
  - color 정보 : R,G,B 3개 × 4 byte / float = 12 bytes
  - face 하나 출력에 총 4 × 24 bytes = 96 byte 전송

- 자료 전송을 최소화 하는 방법 ?
  - client-server model
  - vertex 정보, color 정보를 한 번만 server에 보낸다
  - index 로 이를 사용

Vertex Array

- enable the vertex array

  glEnableClinetState(GL_VERTEX_ARRAY);

  glEnableClientState(GL_COLOR_ARRAY);

- color, vertex array 전송

  glVertexPointer(3, GL_FLOAT, 0, vertices);

  glColorPointer(3, GL_FLOAT, 0, colors);

- void glVertexPointer(GLint size, GLenum type,
  GLsizei stride, const GLvoid* pointer);
  - size : 좌표의 수 (2, 3,4)
  - stride : 건너뛸 byte 수 (0 : tightly packed)
  - pointer : 자료의 시작 위치

Vertex Array

- index 정보 : 24 indices for 6 faces

```
GLubyte cubeIndices[24]={ 0,3,2,1, 2,3,7,6, 0,4,7,3,
                          1,2,6,5, 4,5,6,7, 0,1,5,4 };
```

- 실제 draw

```
glDrawElements(GL_QUADS, 24,
               GL_UNSIGNED_BYTE, cubeIndices);
```

- void glDrawElements(GLenum mode, GLsizei count,
  GLenum type, const GLvoid* indices);
  - mode : GL_POLYGON, GL_QUADS, …

4.5 Affine Transformations

Transformation

- transformation (= transform)
  - point / vector $\Rightarrow$ 다른 point / vector 로 mapping 하는 함수
- affine transformation
  - linear function 으로 제한한 경우
  - homogeneous coordinate 수식으로는

$$\mathbf{q} = A\mathbf{p} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{bmatrix}$$

Affine Transformation

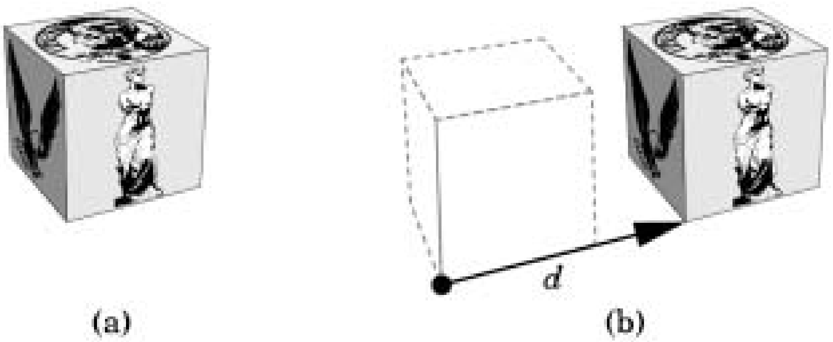
- 특징
  - linearity 보장 : line 은 transform 후에도 line
  - parallel 한 line들은 transform 후에도 parallel
  - angle preserving : 각도는 그대로 유지됨

4.6 Rotation, Translation, and Scaling

Translation

- translation : 물체를 평행 이동

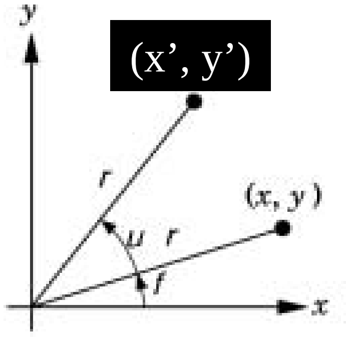
$$P' = P + d$$



이동 전 이동 후

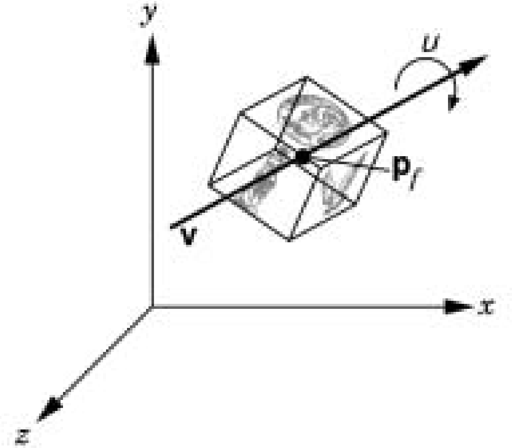
Rotation

- 2D rotation (원점 기준의 회전)



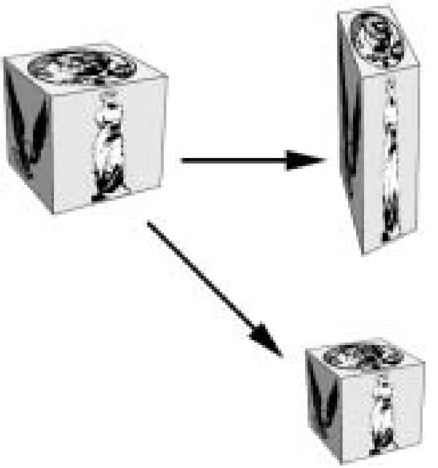
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- 3D rotation
  - a fixed point + a line : 회전축
  - rotation angle : 회전 각도

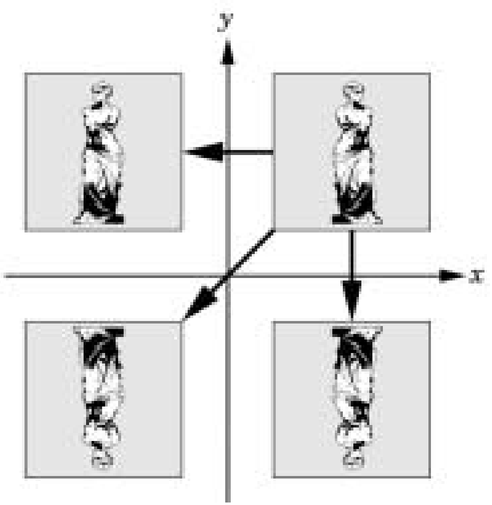


Scaling

- 물체의 크기를 조절
  - 물체의 크기를 α 배
  - 0 < α < 1 : 축소
  - α > 1 : 확대
  - α < 0 : reflection



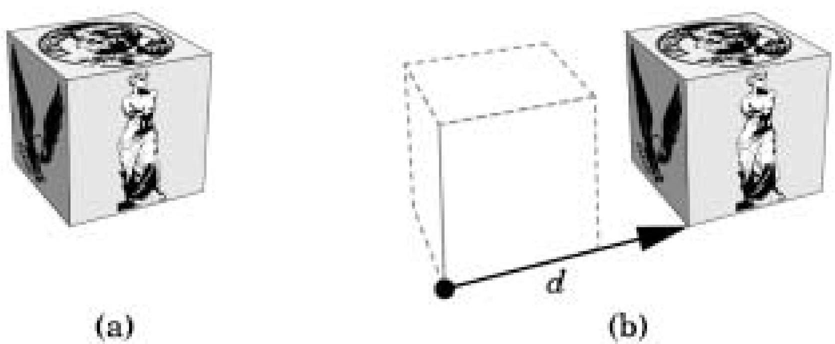
scaling



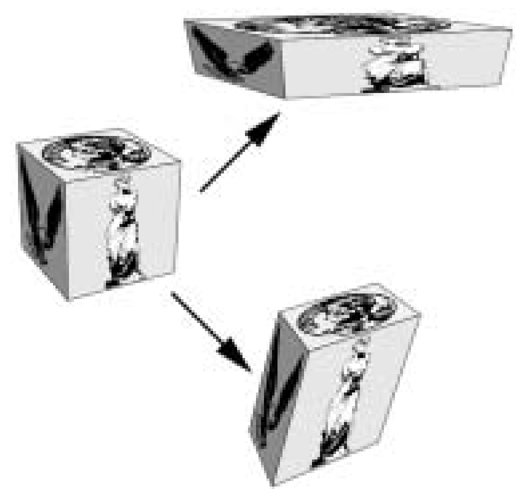
reflection

Rigid-body Transformation

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



rigid-body transformation



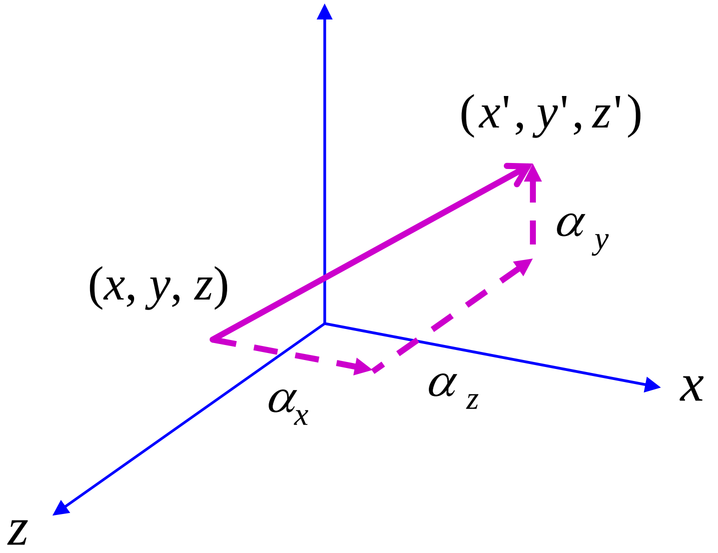
non-rigid-body transformation

4.7 Transformations in Homogeneous Coordinates

Translation

- $\mathbf{p}'$: 새로운 point, $\mathbf{p}$: 원래 point

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



- translation

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \mathbf{p} = \mathbf{T}^{-1}\mathbf{p}'$$

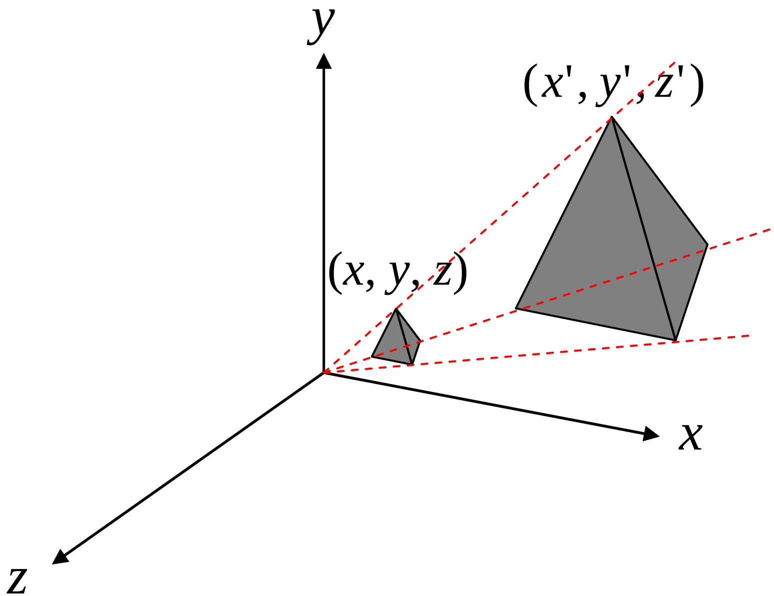
$$\mathbf{T}(\alpha_x, \alpha_y, \alpha_z) = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{T}^{-1}(\alpha_x, \alpha_y, \alpha_z) = \begin{bmatrix} 1 & 0 & 0 & -\alpha_x \\ 0 & 1 & 0 & -\alpha_y \\ 0 & 0 & 1 & -\alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$\mathbf{p}' = \mathbf{S}\,\mathbf{p}$$

$$\mathbf{S}(\beta_x, \beta_y, \beta_z) = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S}^{-1}(\beta_x, \beta_y, \beta_z) = \mathbf{S}(\frac{1}{\beta_x}, \frac{1}{\beta_y}, \frac{1}{\beta_z})$$

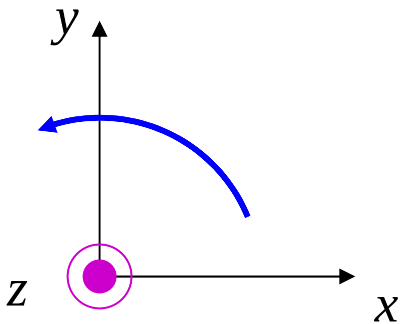


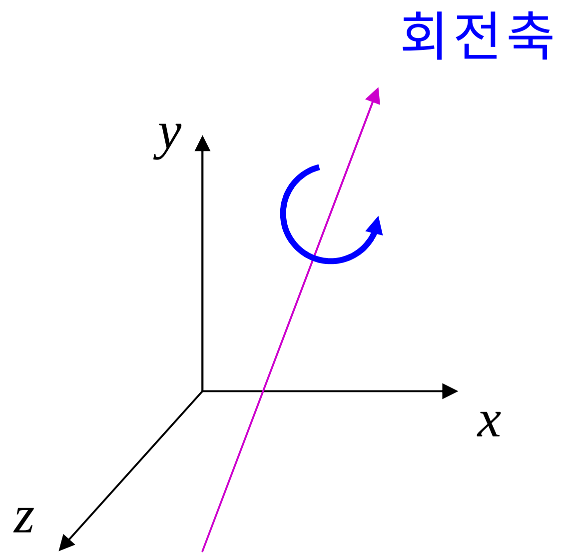
$$\alpha_x = \alpha_y = \alpha_z \quad \Rightarrow \quad \text{Uniform scaling}$$

$$\text{Otherwise, non-uniform scaling}$$

Rotation, 2D vs. 3D

- rotation : 회전축을 필요로 한다
  - 2D : *xy* 평면이므로, *z*축이 회전축
  - 3D : 회전축을 임의로 설정 가능
    - 각 좌표축에 대한 회전
    - 임의의 회전축에 대한 회전 : 뒤에…



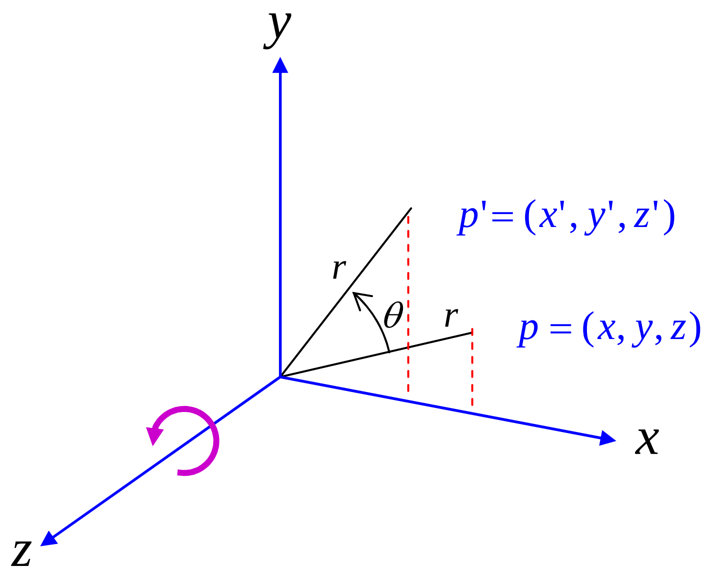


좌표축 중심의 rotation

- 비교적 간단
- z-axis rotation
  - 2D 경우와 동일

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

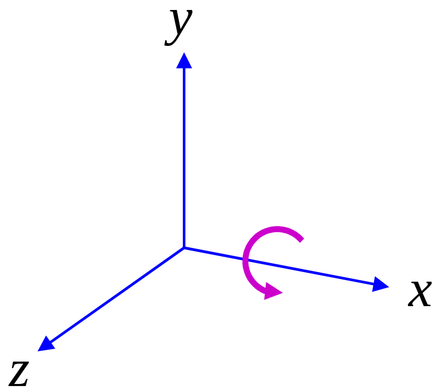
$$\mathbf{p}' = \mathbf{R}_z(\theta)\mathbf{p}$$



좌표축 중심의 rotation

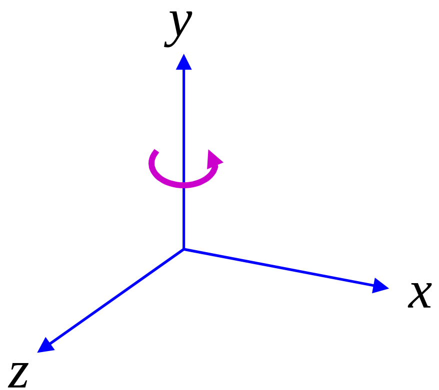
- *x*-axis rotation $\mathbf{p}' = \mathbf{R}_x(\theta)\mathbf{p}$

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



- *y*-axis rotation $\mathbf{p}' = \mathbf{R}_y(\theta)\mathbf{p}$

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



Rotation 의 중요 성질

- inverse 계산을 빨리 할 수 있다

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$$

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^{\mathrm{T}}(\theta)$$

$$\mathbf{R}_x(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R}_x^{-1}(\theta) = \mathbf{R}_x^{\mathrm{T}}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection

- xy plane : $(x, y, z) \rightarrow (x, y, -z)$

$$RF_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- yz plane : $(x, y, z) \rightarrow (-x, y, z)$
- zx plane : $(x, y, z) \rightarrow (x, -y, z)$

Shearing

- z-axis shearing

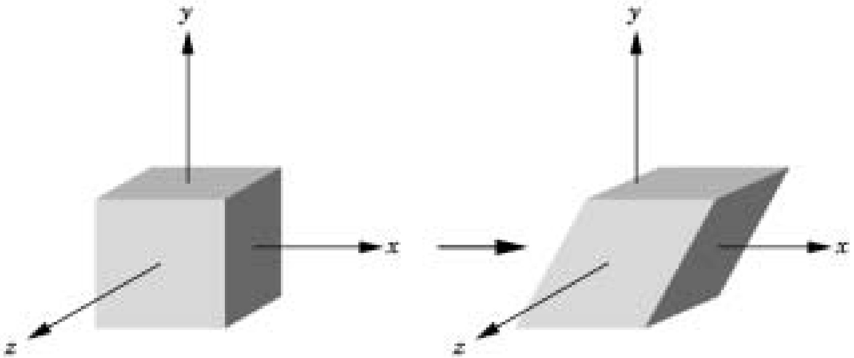
$$x' = x + y\cot\theta$$

$$y' = y$$

$$z' = z$$

$$\mathbf{H}_x(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_x^{-1}(\theta) = \mathbf{H}_x(-\theta)$$

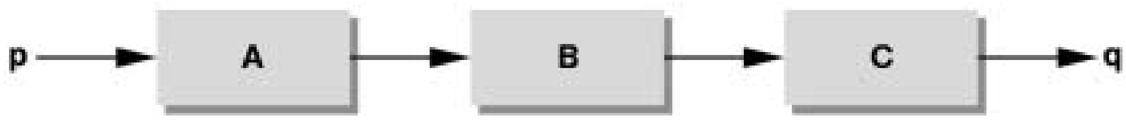


4.8 Concatenation of Transformations

Matrix Concatenation

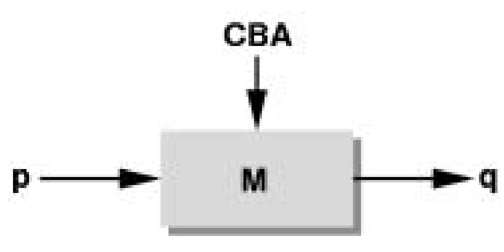
- matrix 여러 개를 곱하는 경우
  - 즉, transformation 여러 개가 적용되는 경우

$$\mathbf{q} = \mathbf{C}\mathbf{B}\mathbf{A}\mathbf{p}$$
$$\mathbf{q} = (\mathbf{C}(\mathbf{B}(\mathbf{A}\mathbf{p})))$$



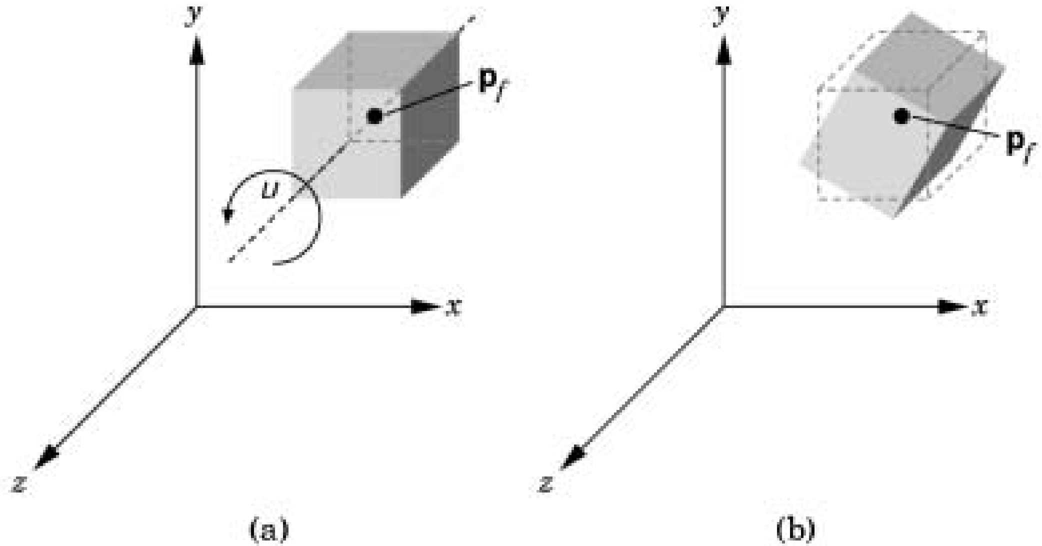
- 다른 방법 : point 여러 개를 transformation 하는 경우
  - graphics package 들이 사용 중.

$$\mathbf{M} = \mathbf{C}\mathbf{B}\mathbf{A}$$
$$\mathbf{q} = \mathbf{M}\mathbf{p}$$



Example : Rotation about a line

- fixed point $\mathbf{P_f}$ 를 지나면서
  z축에 평행한 line을 중심으로 회전

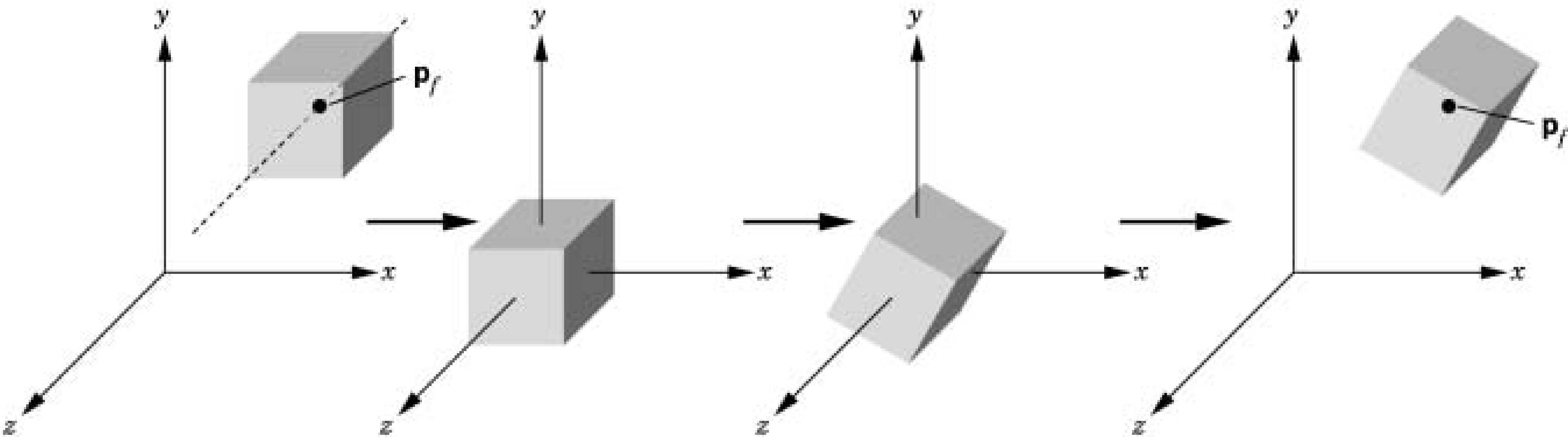


회전 이전　　회전 이후

Example : Rotation about a line

$$\mathbf{M} = \mathbf{T}(\mathbf{p}_f)\mathbf{R}_z(\theta)\mathbf{T}(-\mathbf{p}_f)$$

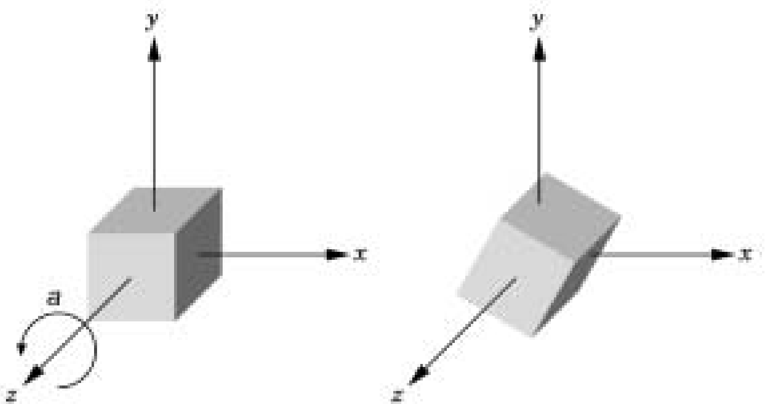
$$\mathbf{M} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & x_f - x_f\cos\theta + y_f\sin\theta \\ \sin\theta & \cos\theta & 0 & y_f - x_f\sin\theta + y_f\cos\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



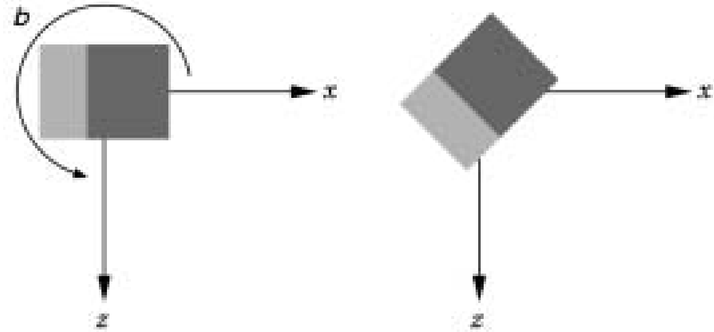
General Rotation

- z, y, x 축의 순서로 α, β, γ 각도씩 회전

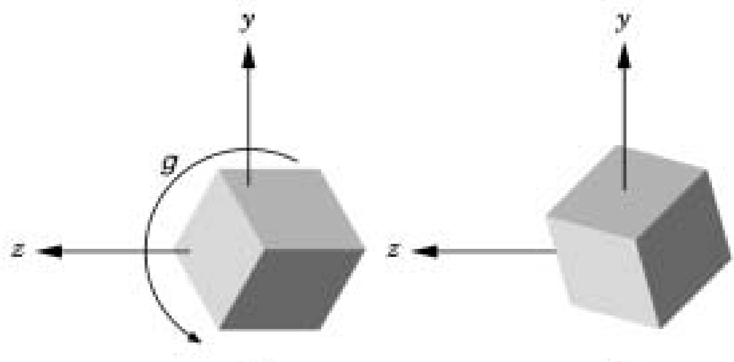
$$\mathbf{M} = \mathbf{R}_x(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_z(\alpha)$$



(a) (b)



(a) (b)

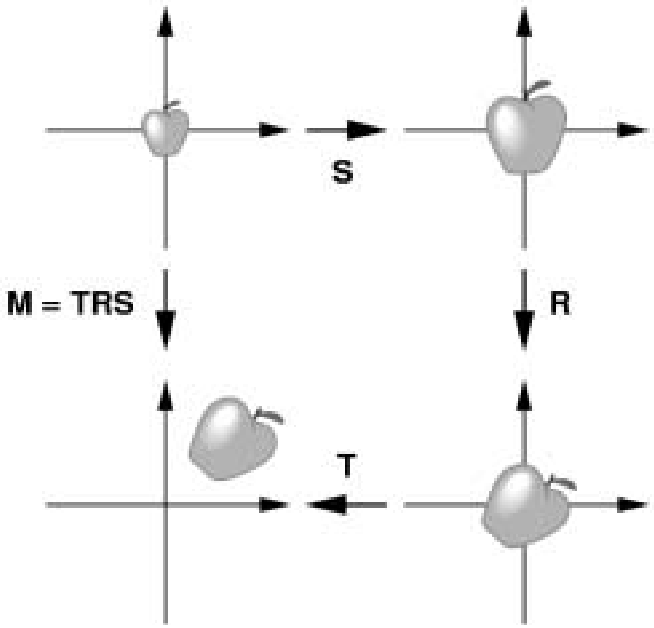


(a) (b)

Instance Transformation

- object 를 transformation 시키는 일반적인 방법
  - scale, rotate, translate 순서

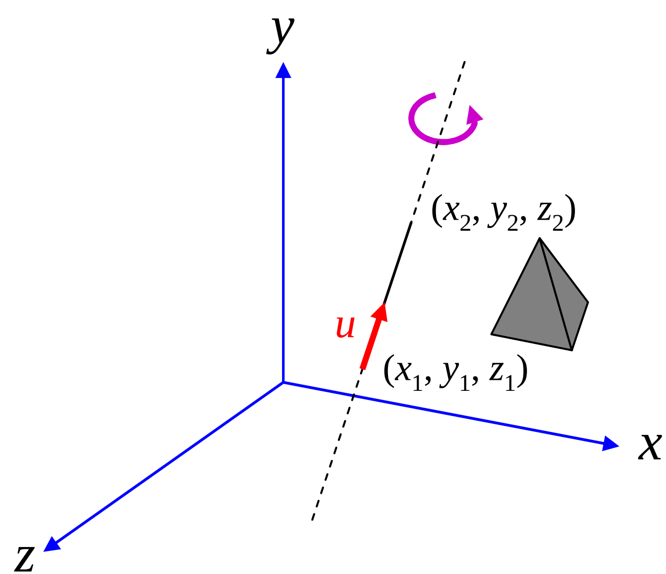
$$\mathbf{M} = \mathbf{T}(\mathbf{p}_f)\mathbf{R}_x(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_z(\alpha)\,\mathbf{S}(s_x, s_y, s_z)$$



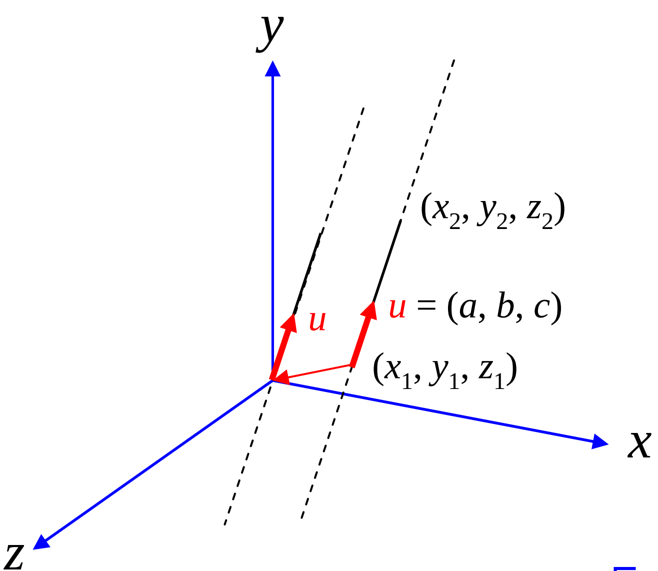
General 3D Rotation

- 임의의 회전축을 중심으로 rotation

1. Translate the object so that the rotation axis passes through the origin.
2. Rotate the object so that the rotation axis coincides with one of the coordinate axis.
3. Perform the specified rotation.
4. step 2의 역연산 R^{-1}
5. step 1의 역연산 T^{-1}



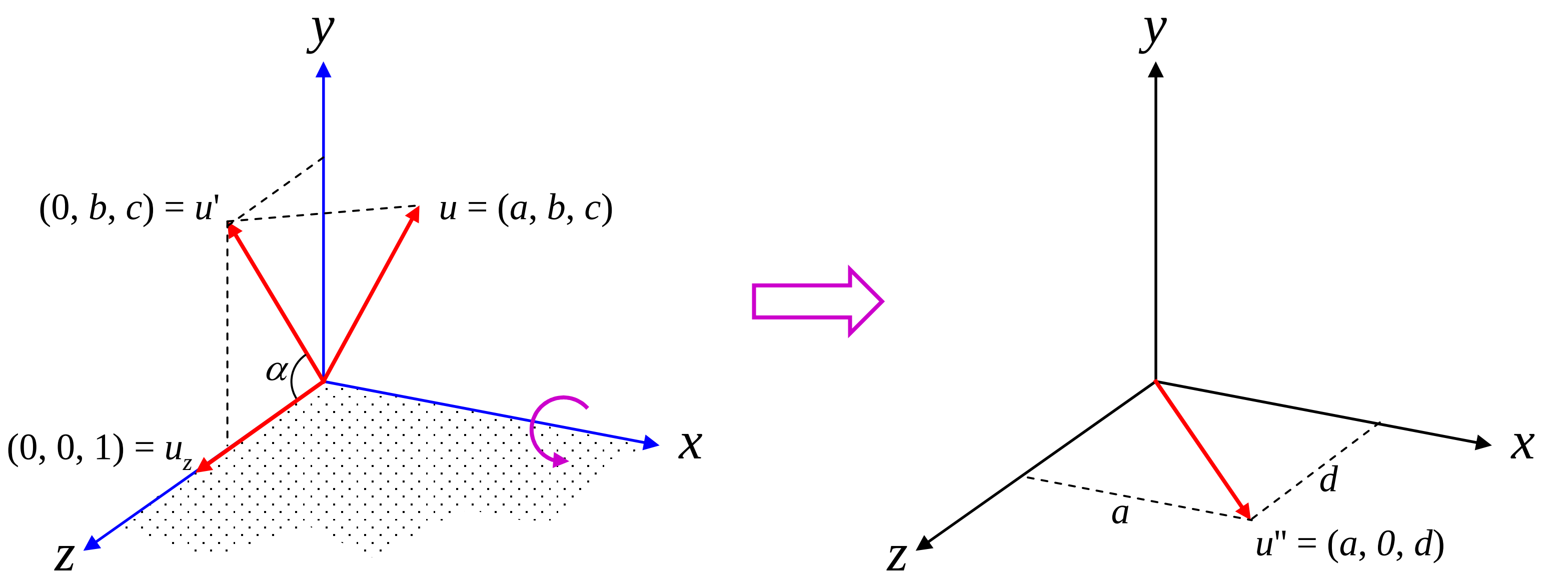
Step 1. Translation



회전축:
점 (x_1, y_1, z_1)을 지나고,
기울기가 $u = (a, b, c)$ 인 직선

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2. align *u* with *z*-axis (1/3)



$$u' \cdot u_z = |u'||u_z| \cos\alpha$$

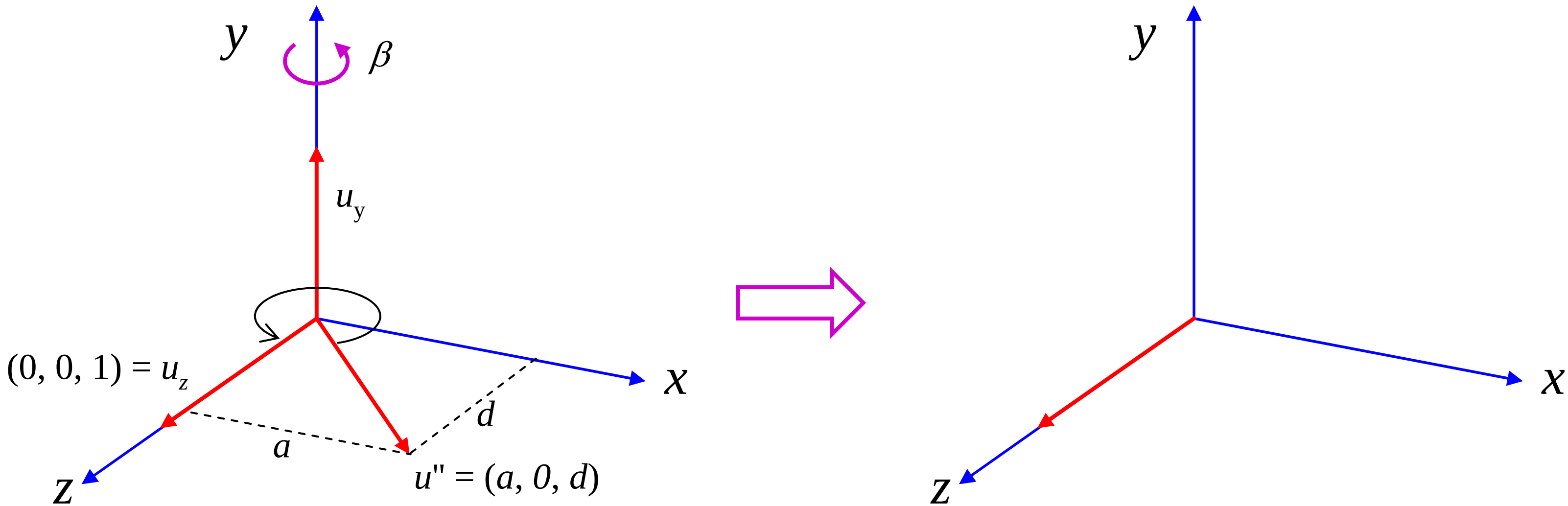
$$\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{c}{d}$$

$$|u'| = \sqrt{b^2+c^2}, \quad |u_z| \equiv 1, \quad d = \sqrt{b^2+c^2}$$

$$\sin\alpha = \frac{b}{d}$$

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2. align *u* with *z*-axis (2/3)



$$u''\times u_z = u_y\,|u''|\,|u_z|\sin\beta$$

$$= u_y \sin\beta$$

$$\cos\beta = \frac{u''\cdot u_z}{\underbrace{|u''|}_{\equiv 1}\,\underbrace{|u_z|}_{\equiv 1}} = d$$

$$\begin{vmatrix} u_x & u_y & u_z \\ a & 0 & d \\ 0 & 0 & 1 \end{vmatrix}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$u''\times u_z = u_y\cdot(-a)$$

$$\sin\beta = -a$$

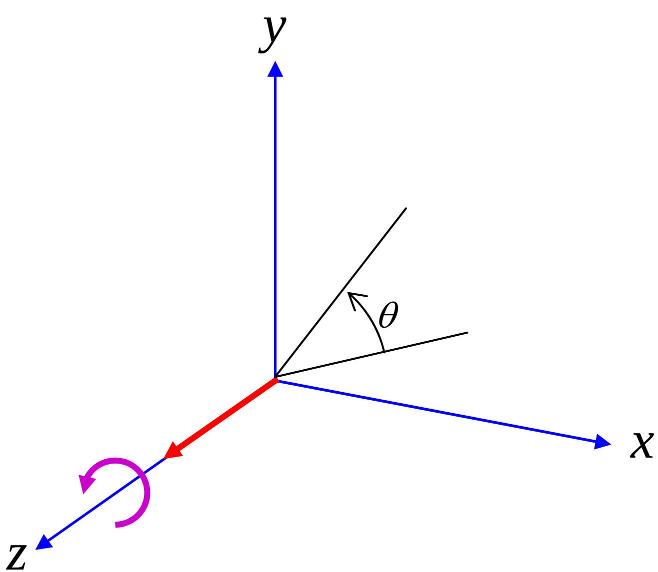
Step 2. align *u* with *z*-axis (3/3)

- 정리하면,
  - $\boldsymbol{u}$ 벡터를 z-axis 에 일치시키는 연산은

$$\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)=\begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - $\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)$ 의 역연산은 나중에 사용
    - $\mathbf{R}_x^{-1}(\alpha)\cdot\mathbf{R}_y^{-1}(\beta)$

Step 3. Rotate about *z*-axis



$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4, 5: 원상 복귀

- step 4: $\mathbf{R}_x^{-1}(\alpha)\cdot\mathbf{R}_y^{-1}(\beta)$
- step 5: $\mathbf{T}^{-1}$

- 정리하면,

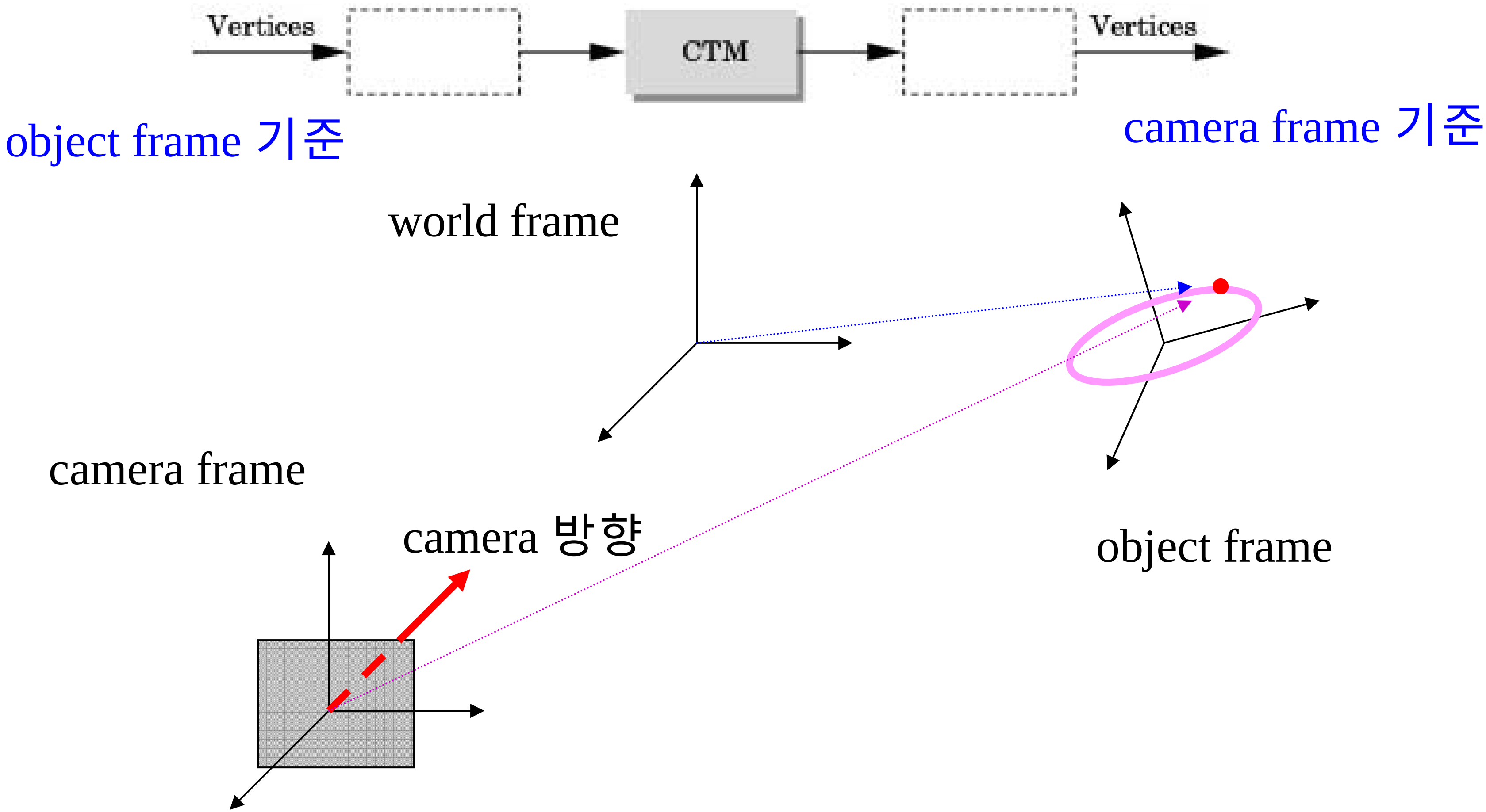
$$\mathbf{R}(\theta)=\mathbf{T}^{-1}\cdot\mathbf{R}_x^{-1}(\alpha)\cdot\mathbf{R}_y^{-1}(\beta)\cdot\mathbf{R}_z(\theta)\cdot\mathbf{R}_y(\beta)\cdot\mathbf{R}_x(\alpha)\cdot\mathbf{T}$$

$$\mathbf{P}'=\mathbf{R}(\theta)\cdot\mathbf{P}$$

4.9 OpenGL Transformation Matrices

CTM

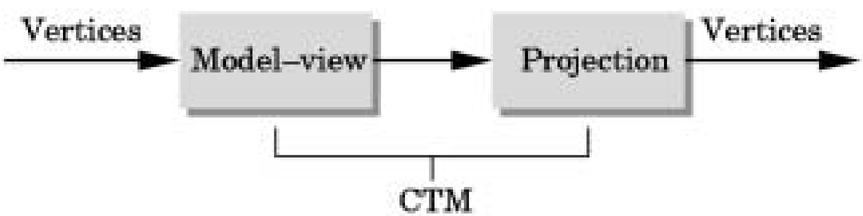
- current transformation matrix : OpenGL에서 관리



OpenGL CTM

- 2 matrix
  - model-view : world frame 까지 변환 (affine)
  - projection : camera frame 으로 변환 (non-affine)

$$\mathbf{vertex}_{\mathrm{camera}} = \mathbf{C}_{\mathrm{projection}}\,\mathbf{C}_{\mathrm{modelview}}\,\mathbf{vertex}_{\mathrm{object}}$$



CTM 관련 연산들

- matrix 선택 glMatrixMode(GL_MODELVIEW);

  $C = C_{projection}$ or $C_{modelview}$
- $C \leftarrow I$ glLoadIdentity( );
- $C \leftarrow CT$ glTranslatef(tx, ty, tz);
- $C \leftarrow CS$ glScalef(sx, sy, sz);
- $C \leftarrow CR$ glRotatef(degree, vx, vy, vz);
- $C \leftarrow M$ glLoadMatrixf(matrix);
- $C \leftarrow CM$ glMultMatrixf(matrix);

- 현재 matrix 보관 glPushMatrix( );
- 보관했던 matrix 복귀 glPopMatrix( );

Example : Spinning Cube

- register 3 callbacks in main(…)

```
glutDisplayFunc(display);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
```

- mouse callback : button 에 따라, 회전축 교체

```
static GLint axis = 2; // z axis
void mouse(int btn, int state, int x, int y) {
  if (state == GLUT_DOWN) {
    if (btn==GLUT_LEFT_BUTTON) axis = 0;     // x axis
    if (btn==GLUT_MIDDLE_BUTTON) axis = 1; // y axis
    if (btn==GLUT_RIGHT_BUTTON) axis = 2;   // z axis
  }
}
```

Example : Spinning Cube

- idle callback : 현재 회전축의 회전 각도 증가

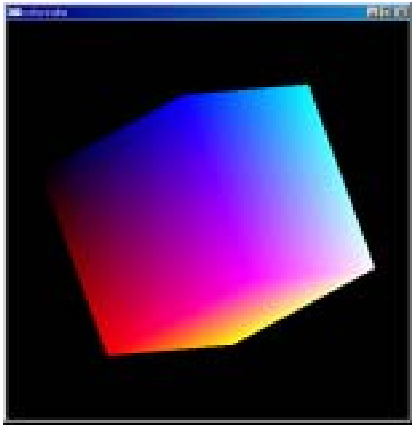
```
static GLfloat theta[3] = { 0.0, 0.0, 0.0 };

void spinCube(void) {
  theta[axis] += 2.0;
  if (theta[axis] > 360.0)
    theta[axis] -= 360.0;
  glutPostRedisplay();
}
```

Example : Spinning Cube

- display callback
  - rotate the cube, draw, and swap buffers

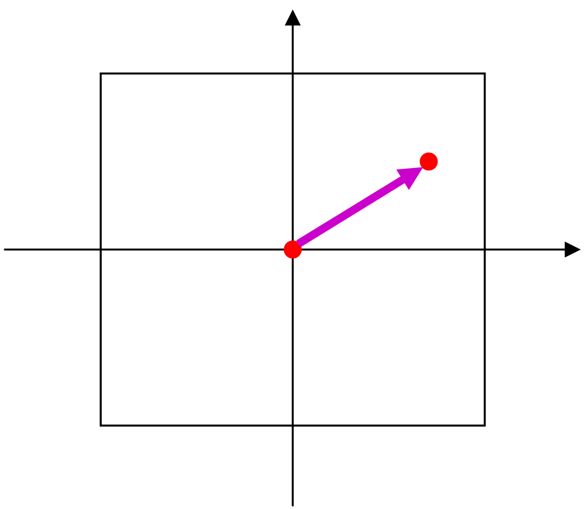
```
void display(void) {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef(theta[0], 1.0, 0.0, 0.0);
  glRotatef(theta[1], 0.0, 1.0, 0.0);
  glRotatef(theta[2], 0.0, 0.0, 1.0);
  colorcube();
  glFlush();
  glutSwapBuffers();
}
```



4.10 Interfaces to 3D Applications

Simple Screen-based Approach

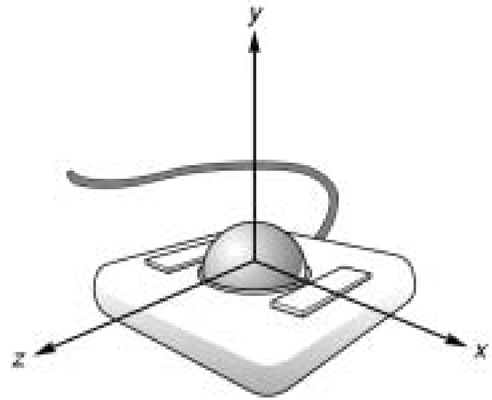
- 문제점 : too many control parameters
  - 회전각 : 각 축 마다 하나씩…
  - 위치 : translate vector 입력도 가능
  - but, mouse button 은 2 ~ 3 개 뿐…
- 해결책 a poor interface …
  - screen 전체를 input pad 로 사용

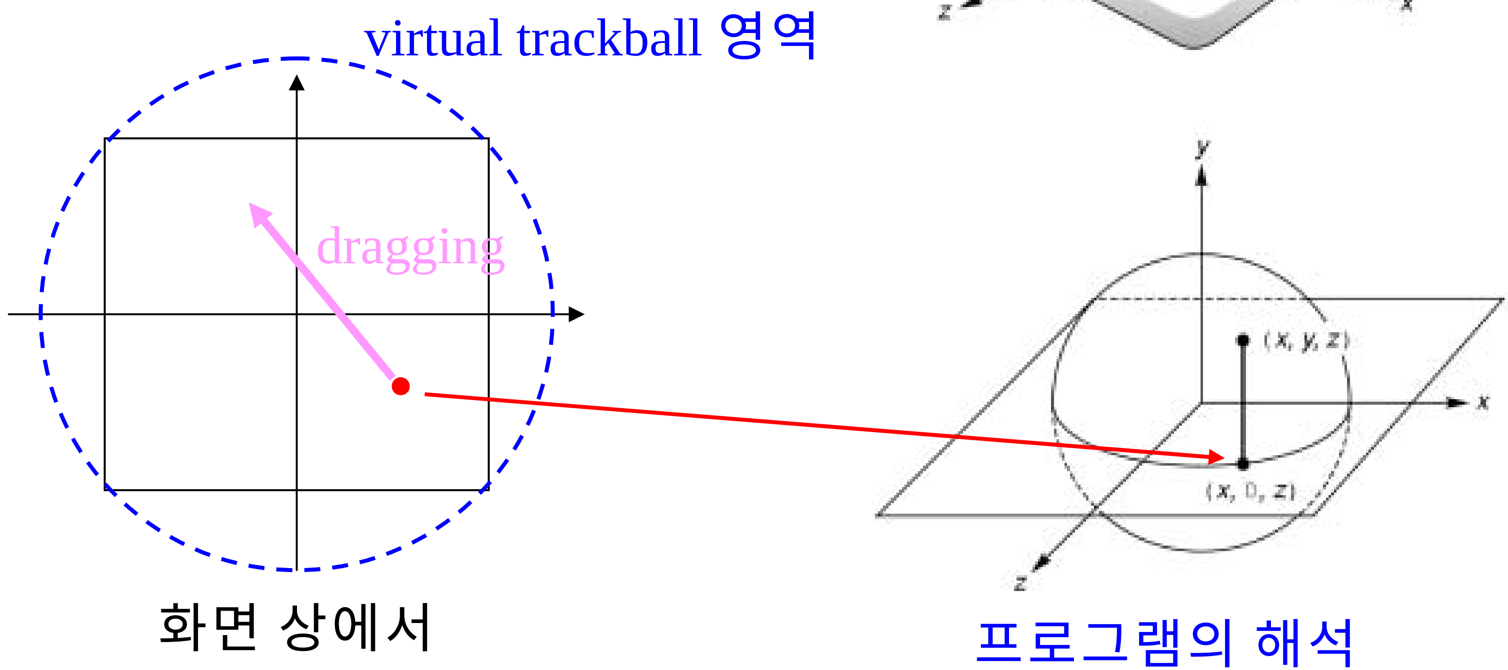


$\mathbf{v} = (\alpha, \beta)$
x 축 중심으로 β 만큼 회전
y 축 중심으로 α 만큼 회전

Virtual Trackball Approach

- mouse 가 화면 상에서 drag





Virtual Trackball Approach

- vector **n** 을 중심으로 각도 θ만큼 회전

$$\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2$$

$$|\sin\theta| = |\mathbf{n}|$$

  – quaternion approach 의 기본 개념