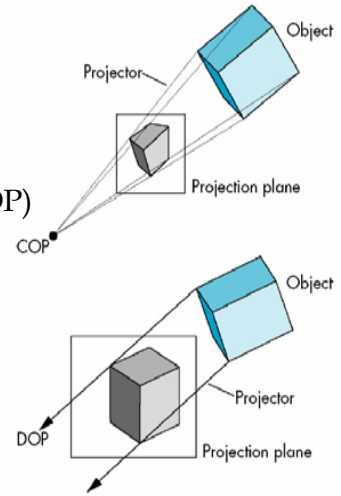
Viewing

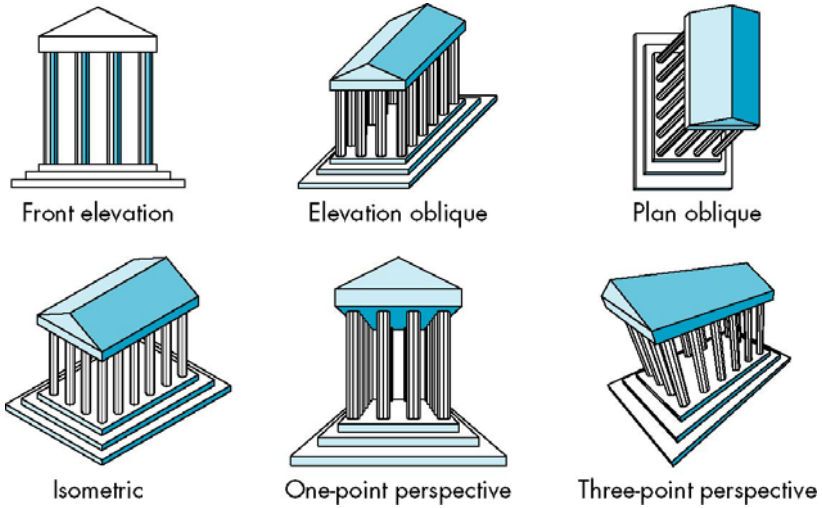
321190
2007년 봄학기
5/1/2007
박경신

Viewing

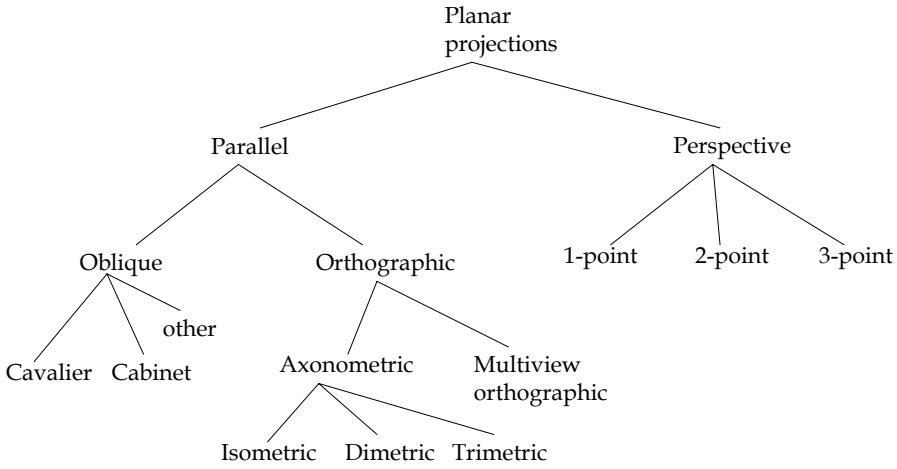
- 관측의 기본요소
  - 객체 (Objects)
  - 관측자 (Viewer)
  - 투영선 (Projector)
  - 투영면 (Projection plane)
- 투영중심 (Center of Projection: COP)
  - COP가 유한한 경우 – 투시관측 (Perspective views)
  - COP가 무한한 경우 – 평행관측 (Parallel views)



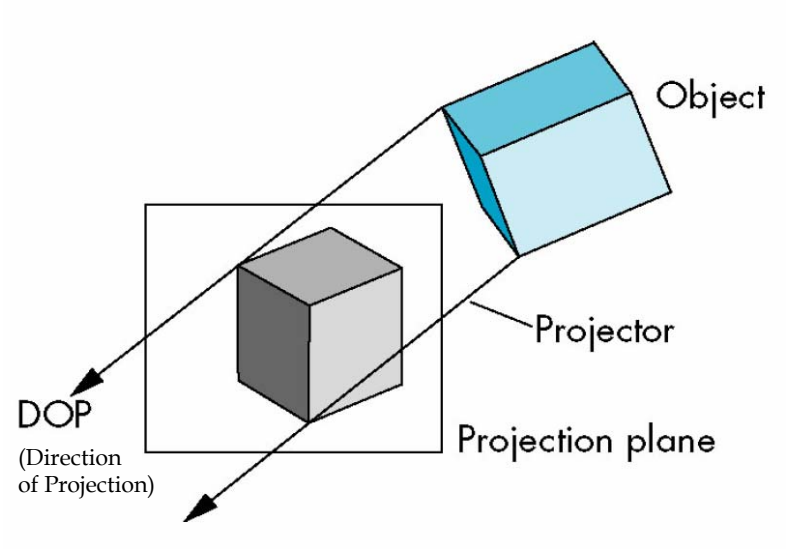
Classical Viewing



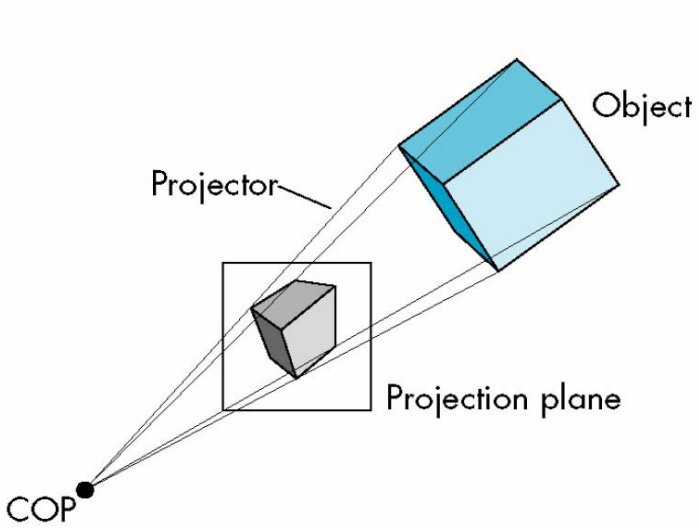
Classical Viewing



Parallel Viewing

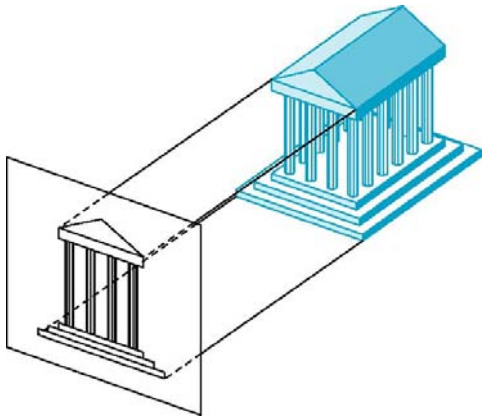


Perspective Viewing



Orthographic Projection

- 직교 투영 (Orthographic Projection)에서는 투영선 (Projector) 은 투영면 (Projection plane)에 수직이다.



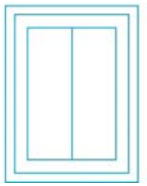
Multiview Orthographic Projection

- 다중 관측 직교 투영 (Multiview Orthographic Projection)에서는 여러 개의 투영면을 만드는데, 각각은 객체의 주면 중 하나와 평행하다.
- 일반적으로 세 개의 관측 (전면, 상면, 우측면 등)을 표시한다.

Isometric (not multiview orthographic view)

front

top

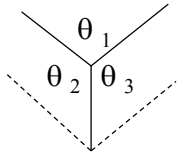


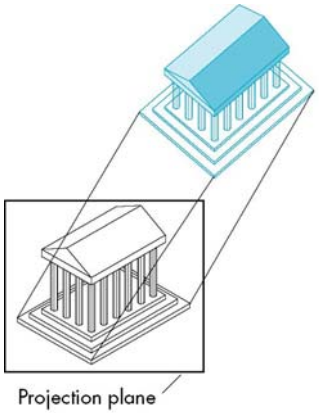
side

- 거리와 각이 모두 보존된다
- 또한, 거리와 모양의 왜곡이 없다
- 그러나, 다중관측직교투영으로부터 객체가 어떻게 생겼는지 그리기 어렵다.
- 그래서, Isometric view과 함께 제공함

Axonometric Projections

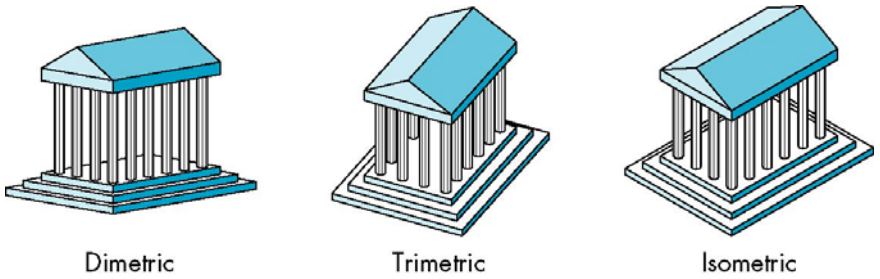
- 축측 관측 (Axonometric view)에서 투영선은 투영면에 수직이지만, 투영면은 객체에 대해 어떠한 방향에도 존재할 수 있다.





- 등축관측(isometric) - 만일 투영면이 사각형 객체의 모서리에서 만나는 세 개의 주면에 대해서 대칭으로 놓여짐
- 이축관측(dimetric) – 투영면이 두 개의 주면이 대칭되도록 놓여짐
- 삼축관측(trimetric)

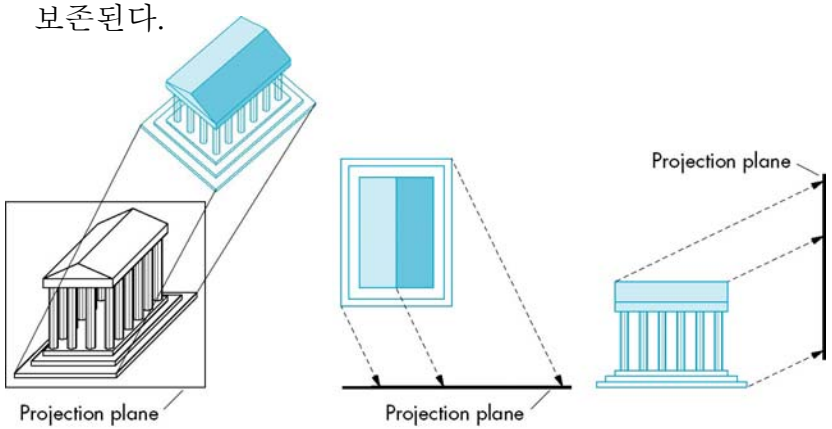
Types of Axonometric Projections



- 선들의 평행은 이미지 안에서 보존되지만 각은 보존되지 않는다.
- 등축 관측: 이미지공간에서 선분의 길이는 객체 공간에서 측정된 길이보다 짧다. 이러한 거리의 단축 (foreshortening)은 세 개의 주면에서 똑같이 발생한다. 따라서 거리 비교는 가능하다.
- 이축 관측: 두 개의 다른 단축비를 가진다.
- 삼축 관측: 세 개의 다른 단축비를 가진다.

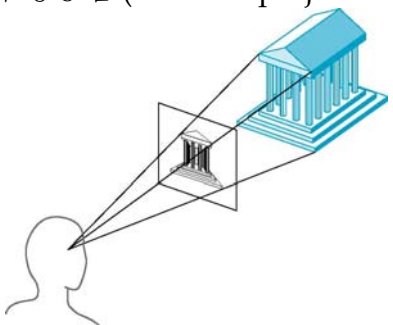
Oblique Projection

- 경사투영 (Oblique Projection)에서 투영선은 투영면과 임의의 각을 가질 수 있다. 투영면에 평행한 면내의 각은 보존된다.



Perspective Projection

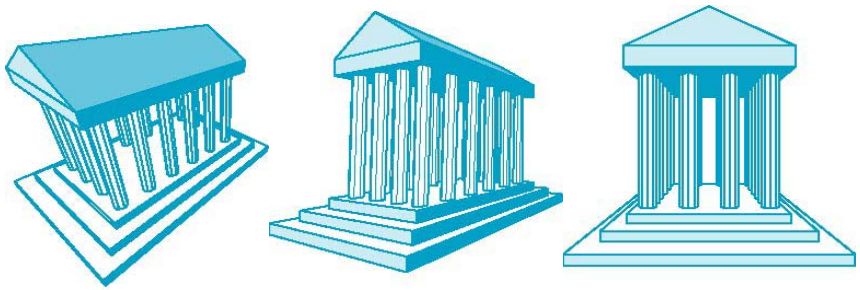
- 투시관측 (Perspective Projection)은 객체가 관측자로부터 멀리 떨어질 수록 크기가 축소된다.
- 투영선은 투영중심 (center of projection)으로 모인다.



- 투시 관측은 크기가 축소 (diminution)되는 특징을 가진다.
- 이러한 크기변화는 자연스러운 모습의 관측(realistic view)을 얻게한다.
- 그러나, 선분의 길이가 얼마나 짧아지는 가는 그 선분이 관측자로부터 얼마나 떨어져 있는가에 의존하기 때문에 길이 측정을 할 수 없다.

1-,2-,3-Point Perspective

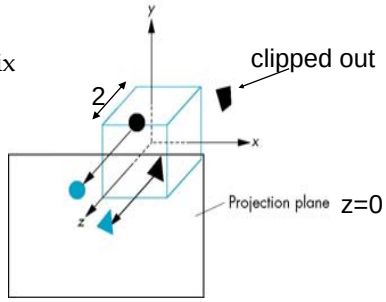
- 일점, 이점, 삼점 투시 (one, two, three point perspectives) 관측의 차이는 객체의 세가지 주 방향 가운데 얼마나 많은 방향이 투영면에 평행한 가에 있다.
- 삼점 투시의 경우 세 개의 주 방향에 평행한 모든 직선들은 세 개의 소실점 (vanishing point)에서 만난다.



3점 투시 2점 투시 1점 투시

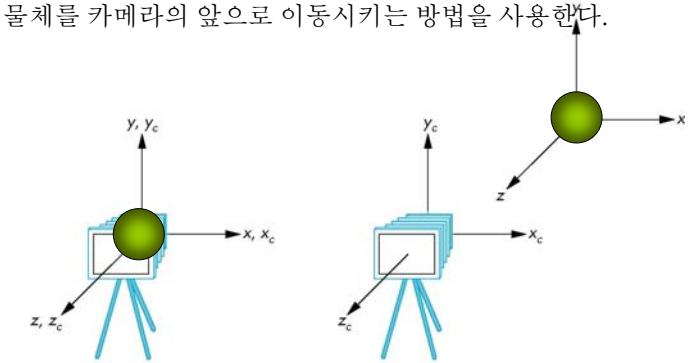
Computer Viewing

- 컴퓨터에서의 관측은 다음과 같이 구성된다.
  - 카메라의 위치와 방향을 잡아준다.
    - Model-view transformation matrix
  - 투영변환을 적용한다.
    - Projection transformation matrix
  - 클리핑 (Clipping) 한다.
    - View volume
- OpenGL에서 초기 카메라는
  - 객체 프레임의 원점에 놓이고,
  - Z축의 음수방향을 향한다.
  - 직교관측으로 설정되어 있고,
  - 원점을 중심으로 한 각 변의 길이가 2인 정육면체로 된 관측 공간을 가진다.
  - 기본 투영면은 z=0인 면이고, 투영방향은 z축과 나란하다.



Positioning the Camera Frame

- OpenGL에서 카메라의 위치지정
  - 카메라를 원점으로부터 먼 곳으로 이동시키는 방법, 또는
    glTranslatef(0.0, 0.0, -d)
  - 물체를 카메라의 앞으로 이동시키는 방법을 사용한다.



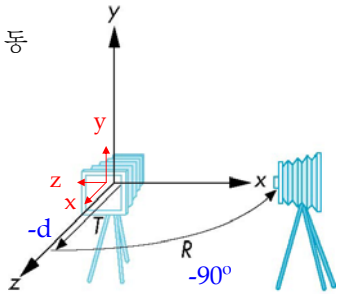
World frame = Camera frame

Moving the camera frame after translation by –d, d > 0

Positioning the Camera

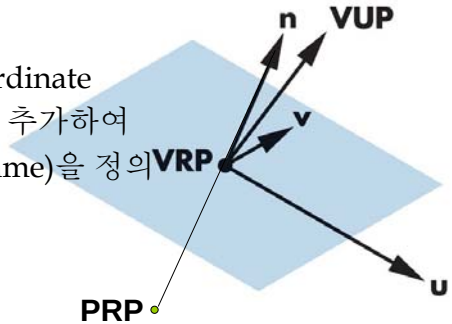
- 연속된 회전 (rotation)과 이동 (translation)으로 카메라 위치를 지정할 수 있다.
- 예제: x축 방면에서 바라보는 관측
  - R = 카메라를 y축을 중심으로 회전
  - T = 카메라를 원점에서 먼 곳으로 이동
  - Model-view matrix C = TR

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -d);
glRotatef(-90.0, 0.0, 1.0, 0.0);
```



Camera Frame

- 관측 참조점 (View reference point, VRP)
- 관측면 법선 (View plane normal, VPN), n = PRP - VRP
- 관측 상향벡터 (View-up vector, VUP)
- u = VUP x n
- v = n x u
- u, v, n를 정규화
- 관측 좌표계 (viewing coordinate system, u'-v'-n')와 VRP를 추가하여 카메라 프레임 (camera frame)을 정의



Camera Frame

- View-orientation matrix, M

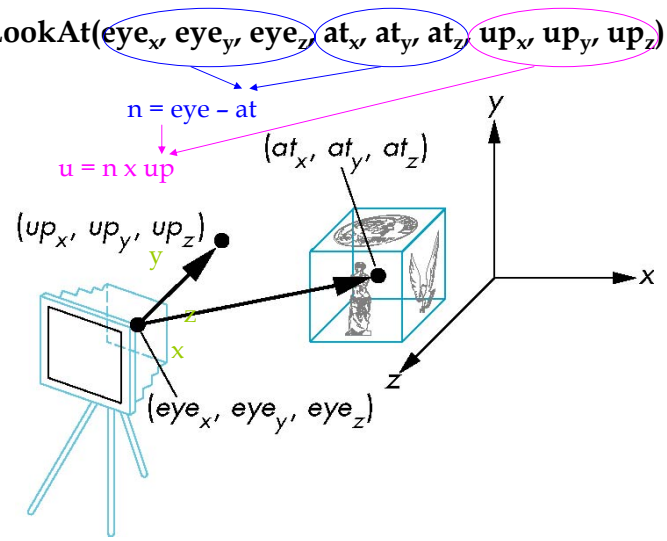
$$\mathbf{M} = \begin{bmatrix} u'_x & v'_x & n'_x & 0 \\ u'_y & v'_y & n'_y & 0 \\ u'_z & v'_z & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation matrix, $M^{-1} = M^T = R$
- World frame에서 카메라의 위치 지정: V = RT

$$\begin{bmatrix} u'_x & u'_y & u'_z & 0 \\ v'_x & v'_y & v'_z & 0 \\ n'_x & n'_y & n'_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} u'_x & u'_y & u'_z & -e \bullet u' \\ v'_x & v'_y & v'_z & -e \bullet v' \\ n'_x & n'_y & n'_z & -e \bullet n' \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

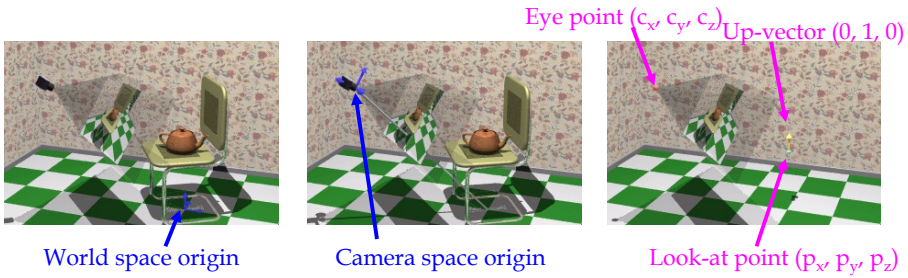
gluLookAt

- **glLookAt(eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z)**



gluLookAt

- *Eye Point* : 카메라의 원점 (월드 좌표계)
- *Look-At* : 카메라가 쳐다보고 있는 위치 (카메라 이미지의 중심이 되는 위치)
- *Up-Vector* : 월드 좌표계에서 카메라가 보는 up 벡터 (카메라 이미지에서 어디로 향하는지에 대한 방향벡터)

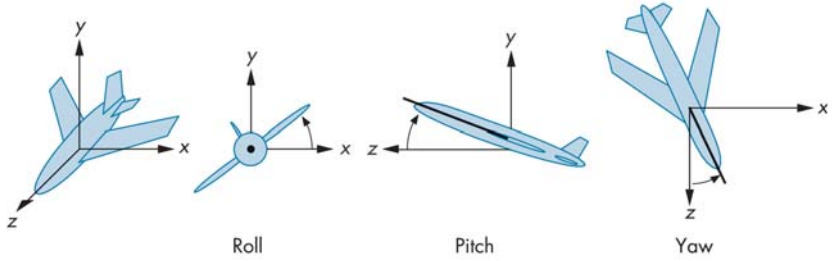


gluLookAt

```
void gluLookAt(GLdouble ex, GLdouble ey, GLdouble ez, GLdouble ax, GLdouble ay, GLdouble az,
      GLdouble ux, GLdouble uy, GLdouble uz) {
      GLdouble M[16]; GLdouble x[3], y[3], z[3]; GLdouble mag;

      z[0] = ex – ax; z[1] = ey – ay; z[2] = ez – az;                  // n
      mag = sqrt(z[0]*z[0] + z[1]*z[1] + z[2]*z[2]);
      if (mag) { z[0] /= mag; z[1] /= mag; z[2] /= mag; }

      y[0] = ux; y[1] = uy; z[2] = uz;                                 // u
      x[0] = y[1]*z[2] – y[2]*z[1]; x[1] = -y[0]*z[2] + y[2]*z[0]; x[2] = y[0]*z[1] - y[1]*z[0];
      mag = sqrt(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);
      if (mag) { x[0] /= mag; x[1] /= mag; x[2] /= mag; }

      y[0] = z[1]*x[2] – z[2]*x[1]; y[1] = -z[0]*x[2] + z[2]*x[0]; y[2] = z[0]*x[1] - z[1]*x[0]; // v
      mag = sqrt(y[0]*y[0] + y[1]*y[1] + y[2]*y[2]);
      if (mag) { y[0] /= mag; y[1] /= mag; y[2] /= mag; }

      M[0] = x[0]; M[1] = x[1]; M[2] = x[2]; M[3] = 0.0;               // R
      M[4] = y[0]; M[5] = y[1]; M[6] = y[2]; M[7] = 0.0;
      M[8] = z[0]; M[9] = z[1]; M[10] = z[2]; M[11] = 0.0;
      M[12] = 0.0; M[13] = 0.0; M[14] = 0.0; M[15] = 1.0;
      glMultMatrix(M);

      glTranslated(-ex, -ey, -ez);                                     // RT
}
```

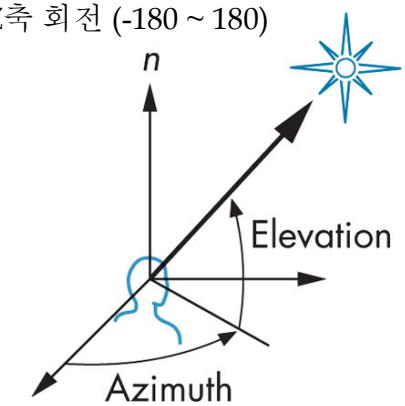
Yaw, Pitch, Roll

- 편요 (Yaw) – Y축 회전
- 종전 (Pitch) – X축 회전
- 횡전 (Roll) – Z축 회전



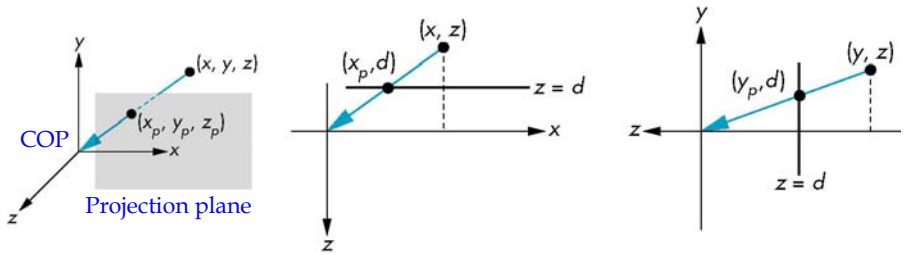
Elevation and Azimuth

- 방위각 (Azimuth) – X축 회전 (-180 ~ 180)
- 앙각 (Elevation) – Y축 회전 (-90 ~ 90)
- 꼬임각 (Twist angle) – Z축 회전 (-180 ~ 180)



Perspective Projection

- 원근 투영 (Perspective projection)
  - 투영중심 (Center of projection)은 원점 (Origin)
  - 투영면 (Projection plane) $z_p = d$



$$x_p = \frac{x}{z/d} \qquad y_p = \frac{y}{z/d} \qquad z_p = d$$

Perspective Projection

Perspective projection

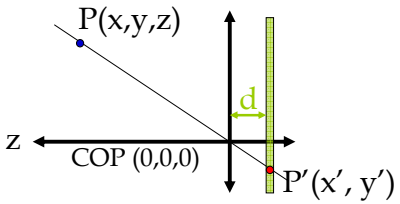
$x_p = \dfrac{x}{z/d}$ $\quad y_p = \dfrac{y}{z/d}$ $\quad z_p = d$

$\mathbf{q} = \mathbf{Mp}$ $\quad \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$

Perspective Projection

- 투영면 (Projection plane, PP)이 투영중심 (Center of projection, COP) 의 뒤에 있는 경우



$-\dfrac{x'}{d} = \dfrac{x}{z}$ $\quad x' = -\dfrac{x}{z/d}$

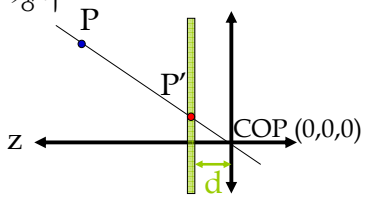
$-\dfrac{y'}{d} = \dfrac{y}{z}$ $\quad y' = -\dfrac{y}{z/d}$

$z' = -d$ $\quad z' = -d$

$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix}$

Perspective Projection

- 투영면 (Projection plane, PP)이 투영중심 (Center of projection, COP) 의 앞에 있는 경우



$\dfrac{x'}{d} = \dfrac{x}{z}$ $\quad x' = \dfrac{x}{z/d}$

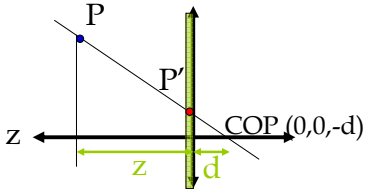
$\dfrac{y'}{d} = \dfrac{y}{z}$ $\quad y' = \dfrac{y}{z/d}$

$z' = d$ $\quad z' = d$

$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$

Perspective Projection

- 투영면 (Projection plane, PP)이 z = 0에 있고, 투영중심 (Center of projection, COP)이 z = -d에 있는 경우



$\dfrac{x'}{d} = \dfrac{x}{z+d}$ $\quad x' = \dfrac{x}{(z+d)/d}$

$\dfrac{y'}{d} = \dfrac{y}{z+d}$ $\quad y' = \dfrac{y}{(z+d)/d}$

$z' = 0$ $\quad z' = 0$

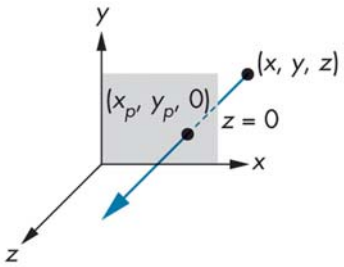
$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$

Orthographic Projection

- 직교 투영 (Orthographic projection)
  - 투영선이 관측 평면에 수직인 평행투영의 특수한 경우이다.
  - 렌즈와 카메라의 뒷면이 평행하고 초점거리가 무한대이다.

Orthographic projection

$x_p = x$
$y_p = y$
$z_p = 0$
$w_p = 1$

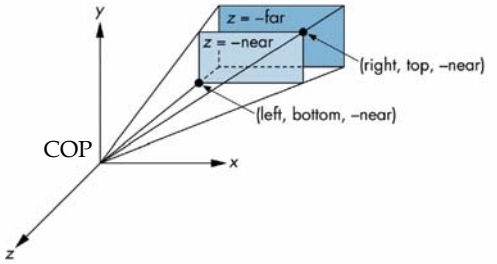


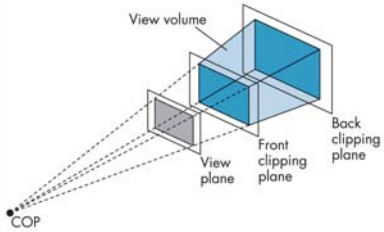
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OpenGL Perspective Projection

- OpenGL에서 투시투영 (perspective projection)은 바늘구멍 카메라 (pin-hole camera)가 원점(origin)에 위치하고 있으며 -Z축을 바라보고 있다.
- glFrustum(left, right, bottom, top, near, far)
  - 앞면과 뒷면의 거리는 양수이어야 하며, COP에서 평면까지의 거리로 측정된다.
  - 관측공간은 절두체 (frustum, i.e. truncated pyramid)이다.

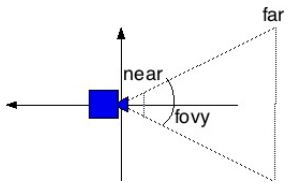


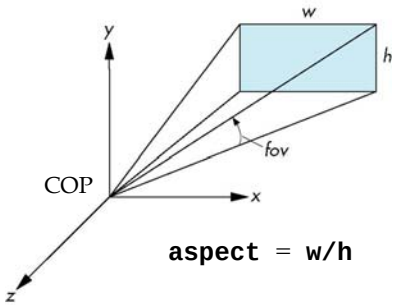


OpenGL Perspective Projection

- gluPerspective(fovy, aspect, near, far)
  - fovy – Y-축 방향에서의 시야 (field of view) 각도
  - aspect – 투영면의 (너비를 높이로 나눈) 종횡비 (aspect ratio)
  - near – 앞쪽 클리핑면
  - far – 뒤쪽 클리핑면

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, 1.333, 0.1, 100);
glMatrixMode(GL_MODELVIEW);
```

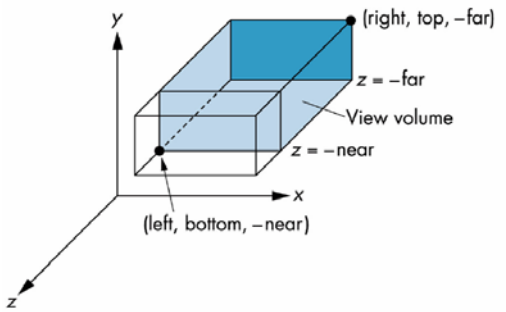




aspect = w/h

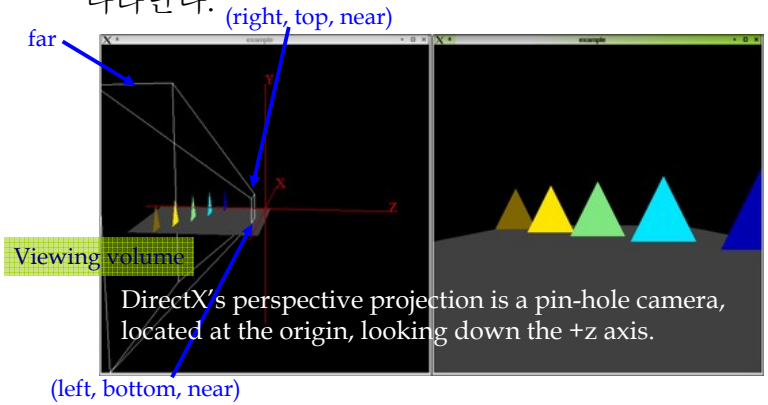
OpenGL Orthographic Projection

- glOrtho(left, right, bottom, top, near, far)
  - 이 함수의 매개변수는 glFrustum의 매개변수와 동일하다.
  - 관측공간은 직육면체이다.



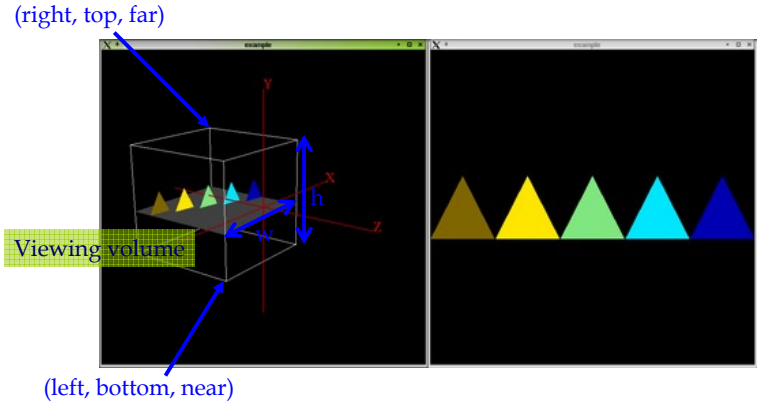
Perspective Projection

- 원근투영 (Perspective projection)은 절두체 (*frustum, i.e., truncated pyramid)* 관측공간을 화면에 투영한다.
- 가까운 객체는 크게 나타나고, 멀리 있는 객체는 작게 나타난다.



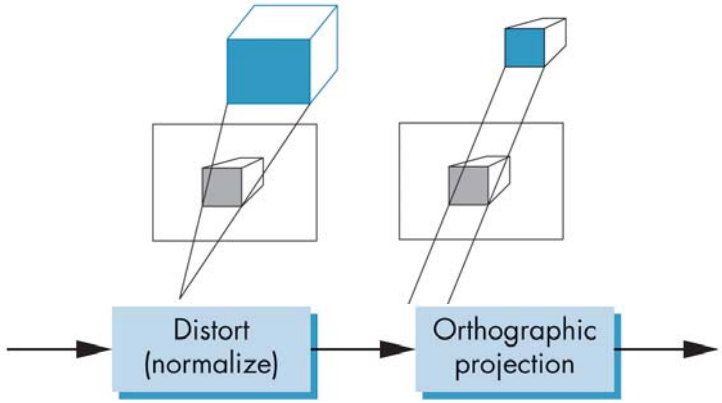
Orthographic Projection

- 직교 투영 (Orthographic projection)은 직육면체 (rectilinear box)의 관측공간을 화면에 투영한다.
- 객체의 크기가 거리에 따라 변하지 않는다.



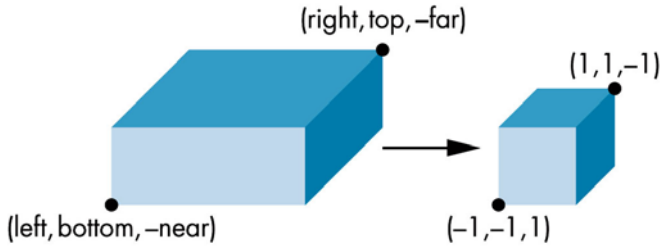
Projection Normalization

- 투영 정규화 (projection normalization)는 왜곡된 객체의 직교 투영이 원래 객체의 원하는 투영이 되도록 먼저 객체들을 왜곡시킴으로써, 모든 투영을 직교 투영으로 변환시키는 작업이다.



Orthogonal Projection Matrix

- 관측 공간을 정규관측공간 (Canonical view volume)으로 매핑한다.



직육면체 -> 정육면체
직교투영 -> 직교투영

Orthogonal Projection Matrix

- 지정된 관측공간의 중심을 정규관측공간의 중심으로 이동

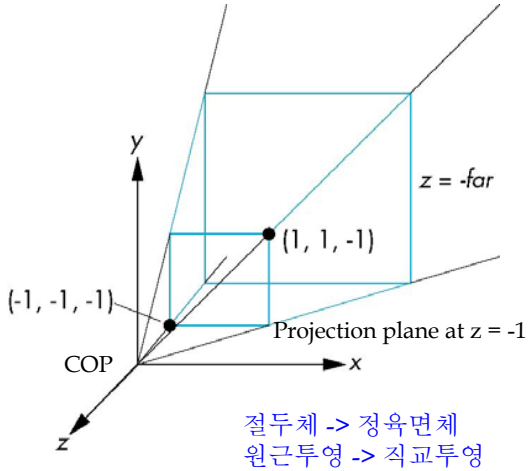
$$T\left(\frac{-(left+right)}{2}, \frac{-(top+bottom)}{2}, \frac{(far+near)}{2}\right)$$

- 지정된 관측공간의 변을 길이가 2가 되도록 크기 변환

$$S\left(\frac{2}{(right-left)}, \frac{2}{(bottom-top)}, \frac{2}{(far-near)}\right)$$

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

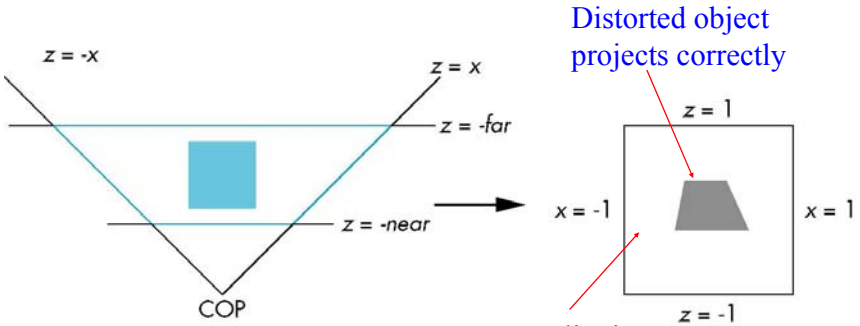
Perspective Projection Matrix



절두체 -> 정육면체
원근투영 -> 직교투영
한 점으로 모이던 투영선들이 평행해짐 - 원근감생성

Perspective Projection Matrix

- 원근 정규화 (Perspective normalization)



$x = \pm z \rightarrow \pm 1$
$y = \pm z \rightarrow \pm 1$
$z = near/far \rightarrow \pm 1$

Perspective Projection Matrix

- 원근 정규화 (Perspective normalization)는 원근투영을 직교투영으로 바꾸는 작업이다.
  - 투영면(PP)이 z = -1, 투영중심(COP)이 원점인 원근투영 행렬, M

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

  - 관측공간의 측면이 투영면을 45도로 교차하도록 함으로써 시야를 90도로 고정

$x = \pm z$
$y = \pm z$

Perspective Projection Matrix

- N 행렬:

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- p′=Np:

$$x' = x, y' = y, z' = \alpha z + \beta, w' = -z$$

- 원근 나눗셈 (Perspective division)한 후, p′->p′′:

$$\Rightarrow x'' = -\frac{x}{z}, y'' = -\frac{y}{z}, z'' = -\left(\alpha + \frac{\beta}{z}\right)$$

Perspective Projection Matrix

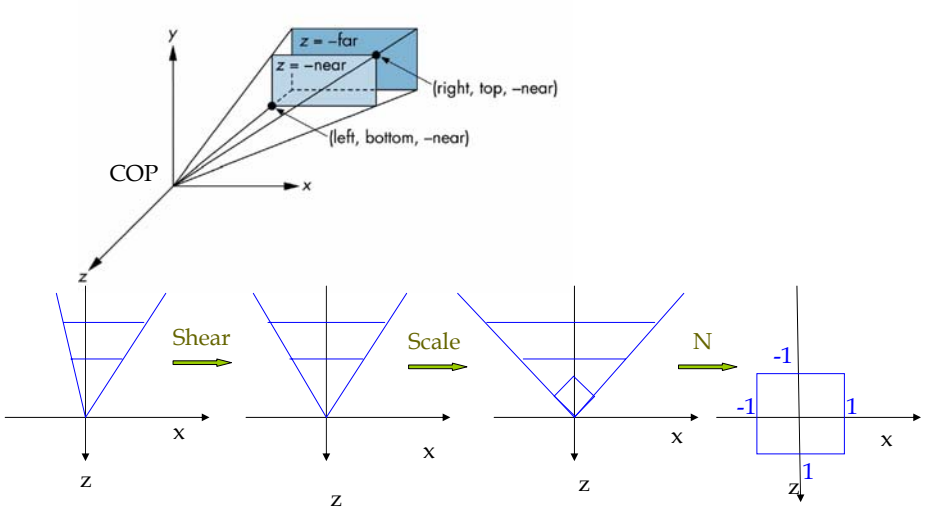
- x = ± z 이면, x′′ = ± 1
- y = ± z 이면, y′′ = ± 1
- Far plane z = z_{max} 이면, $z'' = -\left(\alpha + \frac{\beta}{z_{max}}\right)$
- Near plane z = z_{min} 이면, $z'' = -\left(\alpha + \frac{\beta}{z_{min}}\right)$
- z_{min} -> -1 그리고 z_{max} -> 1 매핑하도록 α와 β를 선정

$$\alpha = \frac{z_{max} + z_{min}}{z_{max} - z_{min}}$$

$$\beta = \frac{2 z_{max} z_{min}}{z_{max} - z_{min}}$$

OpenGL Perspective Projection

- glFrustum(left, right, bottom, top, near, far)



OpenGL Perspective Projection

- Shear $H(\cot\theta, \cot\phi) = H\left(\frac{right + left}{2z_{min}}, \frac{top + bottom}{2z_{min}}\right)$
- Then, $x = \pm\frac{right - left}{2z_{min}}, y = \pm\frac{top - bottom}{2z_{min}}, z = z_{max}, z = z_{min}$
- Scale $x = \pm z, y = \pm z$
- $\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$

$$\alpha = \frac{z_{max} + z_{min}}{z_{max} - z_{min}}$$

$$\beta = \frac{2 z_{max} z_{min}}{z_{max} - z_{min}}$$

OpenGL Perspective Projection

$$\mathbf{P} = \mathbf{NSH} = \begin{bmatrix} \frac{2z_{min}}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2z_{min}}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{z_{max}+z_{min}}{z_{max}-z_{min}} & -\frac{2z_{max}z_{min}}{z_{max}-z_{min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

z_{max} = far
z_{min} = near

Camera Movement

- The OpenGL camera is always positioned at the origin, looking down the –Z axis.
- The camera itself never moves.
- Looking at the final image produced, moving a camera is equivalent to moving the world in the opposite direction.
- E.g. moving everything to the right will produce the same image as moving the camera to the left.

Camera Movement

- To give the effect of moving the perspective viewpoint in OpenGL, use a transformation at the beginning of the frame, which affects all objects.
- For example, to move the camera 10 units from the origin in the +Z direction, translate the world by -10 in Z

```
void display()
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(45, 1, 0.1, 100);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(0, 0, -10);
   drawObjects();
}
```

Camera Movement

- For general camera movement
  - Keep track of camera position & orientation
  - Apply inverse transformation to the world
- Example

```
float cameraX, cameraY, cameraZ, cameraHeading;
void display()
{
   glLoadIdentity();
   glRotatef(-cameraHeading, 0, 1, 0);
   glTranslatef(-cameraX, -cameraY, -cameraZ);

   drawObjects();
}
```