1. 3D 그래픽 기초(3D Graphics Fundamentals)

(7) 변환(Transformation)의 행렬 표현에 대한 수학적 이해

① 변환의 행렬 표현

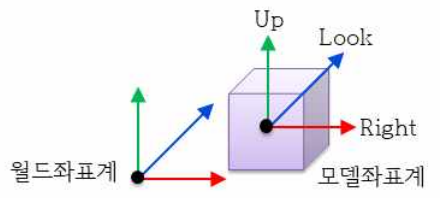
순차적으로 실행되는 변환 파이프라인의 4개의 단계(월드 변환, 카메라 변환, 투영 변환, 화면 변환)를 행렬을 사용하여 표현하자. 벡터는 점 또는 방향을 표현할 수 있고, 회전(Rotation) 변환, 크기(Scale) 변환을 포함하는 선형변환과 평행이동(Translation) 변환을 하나의 (4×4) 행렬을 사용하여 표현할 수 있음을 기억하라. 그리고 행렬을 사용하면 행렬의 곱셈의 유용한 성질, 역행렬 등을 통한 여러 가지 이점이 있게 된다.

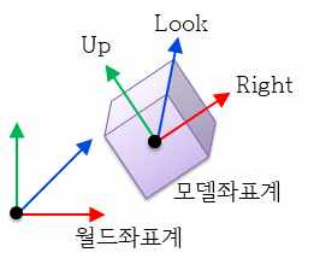
② 월드 좌표 변환(World Transformation)

지금까지 모델좌표계는 모델(메쉬)를 표현하기 위한 좌표계이고, 월드좌표계는 게임 세상을 표현하기 위한 전역좌표계라고 정의하였다. 이제 게임 객체를 표현하기 위한 객체좌표계(Object space, 지역좌표계: Local space)를 생각해보자. 우리는 당분간 객체좌표계가 모델좌표계와 같다고 간주할 것이다.

게임 객체가 회전을 하지 않으면 게임 객체의 x-축, y-축, z-축 방향은 월드좌표계의 x-축, y-축, z-축 방향과 같다. 어떤 직교좌표계에서 x-축은 $(1,0,0)$, y-축은 $(0,1,0)$, z-축은 $(0,0,1)$이다. 월드좌표계의 x-축, y-축, z-축 방향은 절대로 바뀌지 않는다. 게임 객체가 회전(자전)을 하면 게임 객체의 x-축, y-축, z-축 방향은 월드좌표계의 x-축, y-축, z-축 방향과 더 이상 같지 않게 된다. 그러나 회전을 한 후에도 객체좌표계에서 게임 객체의 x-축은 $(1,0,0)$, y-축은 $(0,1,0)$, z-축은 $(0,0,1)$이다. 이때 게임 객체의 x-축, y-축, z-축 방향은 월드좌표계로 표현하면 더 이상 x-축이 $(1,0,0)$, y-축이 $(0,1,0)$, z-축이 $(0,0,1)$이 아니다. 즉, 월드좌표계에서 게임 객체의 방향(x-축, y-축, z-축 방향)을 표현하려면, 객체좌표계로 표현된 게임 객체의 x-축, y-축, z-축이 월드좌표계로 어떻게 표현되는 가를 알아야 한다. 게임 객체가 이동을 하면 객체좌표계의 원점이 월드좌표계로 어떻게 바뀌는 가를 알아야 한다.

월드 변환은 모델좌표계의 한 점(벡터)을 월드좌표계로 변환을 하는 것이다. 이러한 변환은 변환 행렬(아핀 변환 행렬)로 표현할 수 있다.





3-차원 좌표계 A(모델좌표계)의 벡터 $\boldsymbol{a}_A=(x_1,y_1,z_1)$를 좌표계 B(월드좌표계)로 표현

한 벡터 $\boldsymbol{a}_B = (x_2, y_2, z_2)$는 다음과 같이 행렬 $\boldsymbol{M}$으로 표현할 수 있다. 이 행렬 $\boldsymbol{M}$의 1~3 행벡터는 좌표계 A의 기저 벡터 $\boldsymbol{u}_A$, $\boldsymbol{v}_A$, $\boldsymbol{w}_A$를 좌표계 B로 표현한 벡터 $\boldsymbol{u} = (u_x, u_y, u_z)$, $\boldsymbol{v} = (v_x, v_y, v_z)$, $\boldsymbol{w} = (w_x, w_y, w_z)$가 되고 4번째 행벡터는 좌표계 A의 원점 $\boldsymbol{O}_A$를 좌표계 B로 표현한 점 $\boldsymbol{O}_B = (O_x, O_y, O_z)$가 된다.

$$(x_2, y_2, z_2, k) = (x_1, y_1, z_1, k)\boldsymbol{M} = (x_1, y_1, z_1, k)\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ O_x & O_y & O_z & 1 \end{bmatrix}$$

3-차원 좌표계 A가 객체좌표계이고 좌표계 B가 월드좌표계일 때, 행렬 $\boldsymbol{M}$의 1~3 행벡터는 좌표계 A의 기저 벡터 $(1,0,0)$, $(0,1,0)$, $(0,0,1)$를 좌표계 B로 표현한 벡터 $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$가 된다. 벡터 $\boldsymbol{u}$,는 게임 객체의 $(1,0,0)$을 월드좌표계로 표현한 것이고 게임 객체가 사람이라면 오른쪽 방향에 해당한다. 우리는 이 벡터 $\boldsymbol{u}$를 Right 벡터(Right vector)라고 부를 것이다. 유사하게 벡터 $\boldsymbol{v}$는 Up 벡터(Up vector), $\boldsymbol{w}$는 Look 벡터(Look vector)라고 한다. 즉, 벡터 $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$는 게임 객체가 월드좌표계에서 어떤 방향을 가지는 가(회전을 한 결과)를 나타내게 된다. 또한 4번째 행벡터는 좌표계 A의 원점(게임 객체의 중심) $(0,0,0)$을 좌표계 B로 표현한 점 $\boldsymbol{O}_B$가 된다. 점 $\boldsymbol{O}_B$는 게임 객체의 위치(이동을 한 결과)를 나타낸다.

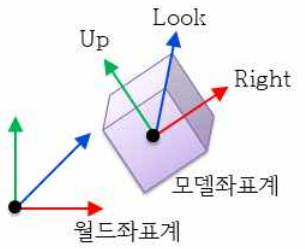
월드좌표계에서 게임 객체의 위치와 방향을 나타내기 위하여 하나의 (4×4) 행렬을 사용한다. 이 행렬을 게임 객체의 월드 변환 행렬(World transform matrix)이라고 하며 $\boldsymbol{W}$로 표기한다. 월드 변환 행렬은 월드좌표계에서 게임 객체의 위치와 방향을 포함한다. 방향은 게임 객체의 로컬 좌표축(Right, Up, Look 벡터)을 월드좌표계로 표현한 것을 의미한다.

게임 객체의 월드 변환 행렬이 $\boldsymbol{W}$이고 모델좌표계의 한 점을 $\boldsymbol{v} = (x_m, y_m, z_m)$라고 하면, 월드 좌표 변환은 $\boldsymbol{w} = \boldsymbol{v}\boldsymbol{W} = (x_w, y_w, z_w)$으로 표현할 수 있다.

행렬을 사용하여 월드 좌표 변환을 하기 위하여 게임 객체는 월드 변환 행렬을 가져야 하므로 게임 객체를 다음과 같이 표현할 수 있다.

```
class CObject
{
public:
  CMesh       *pMesh;
  Matrix      world;
  ...
};
```

Right Vector x	Right Vector y	Right Vector z	0
Up Vector x	Up Vector y	Up Vector z	0
Look Vector x	Look Vector y	Look Vector z	0
Position x	Position y	Position z	1



게임 객체의 월드 변환 행렬 W가 주어지면 W를 다음과 같은 4개의 3-차원 벡터로 나누어 표현할 수 있다. 4번째 열벡터는 항상 $(0,0,0,1)$이다. $(right, up, look)$는 월드좌표계에서 게임 객체의 방향을 나타내고, $position$ 벡터는 게임 객체의 위치를 나타낸다.

$$W = (right, up, look, position)$$

게임 객체의 월드 변환 행렬 W가 주어지면 게임 객체를 게임 객체의 전/후, 좌/우, 상/하 방향으로 이동하는 함수를 다음과 같이 표현할 수 있다. MoveForward() 함수에서 fDistance가 양수이면 $look$ 벡터 방향으로 전진하고 음수이면 후진한다.

```
void CObject::MoveForward(float fDistance)
{
  world._41 += world._31 * fDistance;
  world._42 += world._32 * fDistance;
  world._43 += world._33 * fDistance;
}

void CObject::MoveStrafe(float fDistance)
{
  world._41 += world._11 * fDistance;
  world._42 += world._12 * fDistance;
  world._43 += world._13 * fDistance;
}

void CObject::MoveUpDown(float fDistance)
{
  ????????
}
```

③ 카메라 좌표 변환(Camera Transformation)

월드좌표계에서 가상 카메라의 위치와 방향을 표현하기 위하여 4개의 벡터 또는 (4×4) 행렬이 필요하다.

$$C = (right, up, look, position)$$

```
class CCamera
{
  Vector      right;
  Vector      up;
  Vector      look;
  Vector      position;
```

```
    Matrix      view;
    ...
};
```

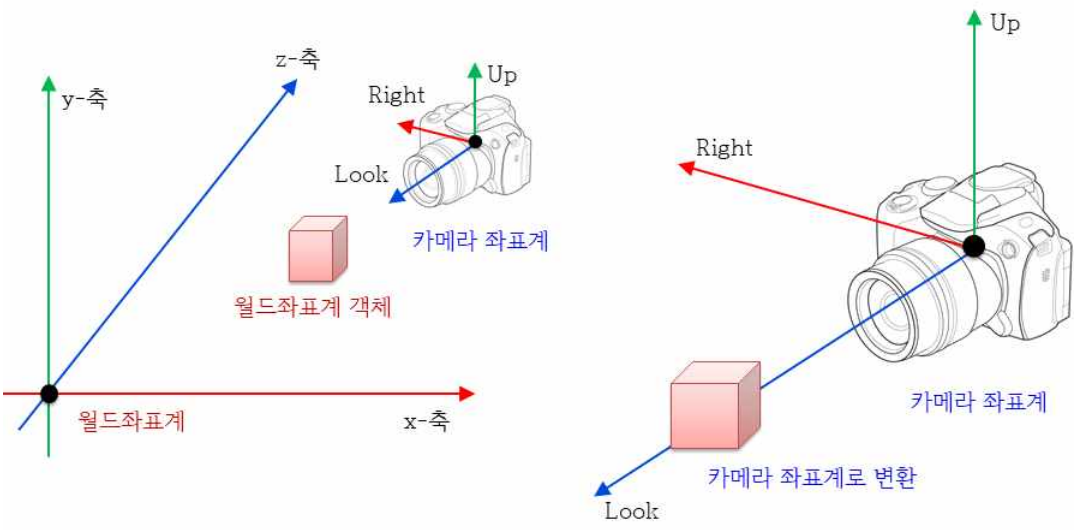
월드좌표계에서 가상 카메라의 위치와 방향을 표현하는 (4×4) 행렬(카메라 월드 변환 행렬)은 다음과 같다. 직교좌표계에서 x-축, y-축, z-축은 서로 직교하며, 회전을 한 후에도 서로 직교한다. 그러므로 ***right***, ***up***, ***look*** 벡터는 서로 수직이다.

right.x	right.y	right.z	0
up.x	up.y	up.z	0
look.x	look.y	look.z	0
position.x	position.y	position.z	1

카메라 변환은 다음과 같이 정리할 수 있다.

❶ 카메라를 월드좌표계의 원점으로 옮기는 평행이동 변환을 월드좌표계로 표현된 점들에 적용한다. 카메라를 월드좌표계의 원점으로 옮기는 평행이동 변환은 카메라를 카메라의 위치로 이동한 평행이동 변환의 반대 방향으로 이동하는 것이다(월드좌표계의 점에서 카메라의 위치를 빼면 된다).

❷ 카메라 좌표계의 축이 월드좌표계의 축과 일치하도록 카메라를 회전하는 변환을 월드좌표계로 표현된 점들에 적용한다. 카메라 좌표계의 축이 월드좌표계의 축과 일치하도록 회전하는 변환은 카메라를 회전한 방향과 반대 방향으로 회전하는 것이다(회전 방향이 반대인 회전은 회전 각도의 부호를 반대로 회전하는 것이다).



카메라 변환은 월드좌표계로 표현된 점을 카메라 좌표계로 변환하는 것이고, 이 변환은 카메라를 월드좌표계의 원점으로 이동하고 카메라의 축 방향을 월드좌표계의 축 방향과 일치하게 하는 변환이다. 이 변환은 월드좌표계에서 카메라의 방향이 $(\boldsymbol{right}, \boldsymbol{up}, \boldsymbol{look})$이고, 카메라의 위치가 ***position*** 벡터가 되도록 하는 월드 변환의 역 변환이다. 그러므로 카메라 변환을 나타내는 행렬은 다음과 같다.

$$V = C^{-1} = (\boldsymbol{right}, \boldsymbol{up}, \boldsymbol{look}, \boldsymbol{position})^{-1}$$

$\boldsymbol{A}$가 (3×3) 선형 변환 행렬이고 $\boldsymbol{y}$가 3-차원 벡터일 때, 아핀 변환 $\boldsymbol{C}$를 다음과 같이 블럭 행렬(Block matrix)으로 표현할 수 있다.

$$C = \begin{bmatrix} A & 0 \\ y & 1 \end{bmatrix}$$

변환 행렬 $\boldsymbol{C}$의 역행렬 $\boldsymbol{C}^{-1}$은 다음과 같다. $\boldsymbol{A}$가 직교행렬이므로 $\boldsymbol{A}^{-1}$는 $\boldsymbol{A}^{T}$이다.

$$C^{-1} = \begin{bmatrix} A & 0 \\ y & 1 \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -yA^{-1} & 1 \end{bmatrix} = \begin{bmatrix} A^{T} & 0 \\ -yA^{T} & 1 \end{bmatrix}$$

카메라 변환 행렬 $\boldsymbol{V}$는 다음과 같다.

right.x	up.x	look.x	0
right.y	up.y	look.y	0
right.z	up.z	look.z	0
-(position • right)	-(position • up)	-(position • look)	1

카메라 변환 행렬이 $\boldsymbol{V}$이고 월드 좌표계의 한 점을 $\boldsymbol{w} = (x_w, y_w, z_w)$라고 하면, 카메라 좌표 변환은 $\boldsymbol{c} = \boldsymbol{w}\boldsymbol{V} = (x_c, y_c, z_c)$으로 표현할 수 있다.

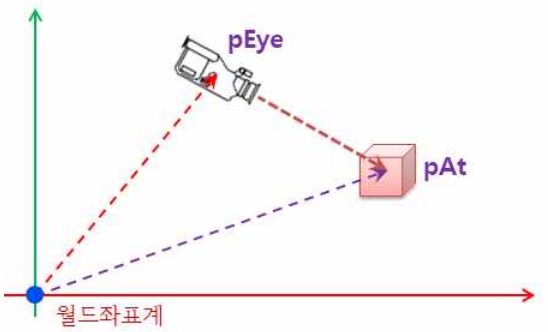
다음은 카메라 변환 행렬을 구하는 함수의 코드이다.

```
void CCamera::GenerateViewTransform()
{
  look = Vector3Normalize(look);
  right = Vector3Normalize(Vector3CrossProduct(up, look));
  up = Vector3Normalize(Vector3CrossProduct(look, right));

  view._11 = right.x; view._12 = up.y; view._13 = look.z;
  view._21 = right.y; view._22 = up.y; view._23 = look.z;
  view._31 = right.z; view._32 = up.y; view._33 = look.z;
  view._14 = view._24 = view._34 = 0.0f;
  view._41 = -Vector3DotProduct(position, right);
  view._42 = -Vector3DotProduct(position, up);
  view._43 = -Vector3DotProduct(position, look);
  view._44 = 1.0f;
}
```

카메라 변환을 행렬로 직접 표현하지 않고 4개의 벡터로 표현하는 이유는 3D 게임에서 카메라는 이동이 빈번하게 일어나고, 카메라 변환 행렬에서 카메라의 위치를 직접 알기가 어렵기 때문이다. 카메라가 이동하면 벡터의 덧셈으로 $\boldsymbol{position}$ 벡터를 변경하고, 필요할 때 카메라 변환 행렬을 4개의 벡터로부터 구해서 사용한다.

월드좌표계에서 카메라의 위치 벡터 $\boldsymbol{pEye}$와 카메라가 바라볼 지점 $\boldsymbol{pAt}$ 벡터가 주어지면 다음과 같이 카메라의 방향 $(\boldsymbol{right}, \boldsymbol{up}, \boldsymbol{look})$을 구할 수 있다(일반적으로 LookAt이라고 한다).




```
look = Vector3Normalize(Vector3Subtract(pAt, pEye));
right = Vector3Normalize(Vector3CrossProduct(pUp, look));
up = Vector3CrossProduct(look, right);
```

다음은 카메라 변환 행렬을 "LookAt"을 사용하여 구하는 함수의 코드이다.

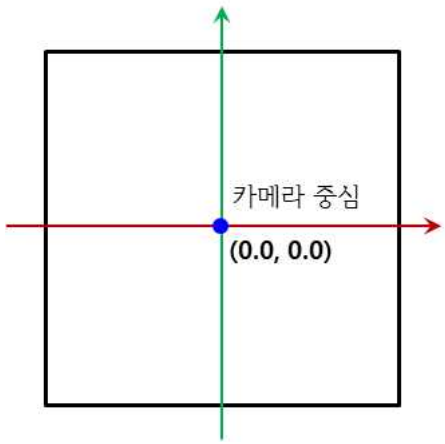
```
void CCamera::GenerateViewTransform(Vector lookAt, Vector newUp)
{
  look = Vector3Normalize(Vector3Subtract(lookAt, position));
  right = Vector3Normalize(Vector3CrossProduct(newUp, look));
  up = Vector3CrossProduct(look, right);
  CCamera::GenerateViewTransform();
}
```

④ 원근 투영 변환(Perspective Projection Transformation)

3차원 카메라 좌표계의 점 (x_c, y_c, z_c)를 2차원 좌표계의 점으로 변환(투영)하는 방법은 점 (x_c, y_c, z_c)의 z-좌표(z_c) 값으로 x_c, y_c, z_c를 모두 나누는 것이다(원근 투영 나누기).

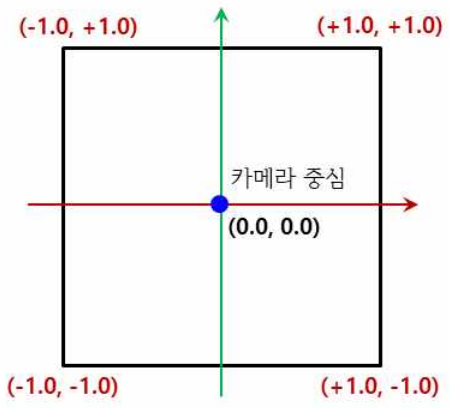
$$(x_c, y_c, z_c) \rightarrow \left(\frac{x_c}{z_c}, \frac{y_c}{z_c}, 1\right)$$

카메라에 보일 수 있는 모든 점 (x_c, y_c, z_c)을 원근 투영을 하면 다음과 같은 투영 사각형이 된다.



카메라의 시야각(FOV)이 90°이면 투영 사각형은 다음 그림과 같이 정사각형(정규화된 투영 사각형, NDC(Normalized Device Coordinates) 또는 클립 공간(Clip space))이 된다.

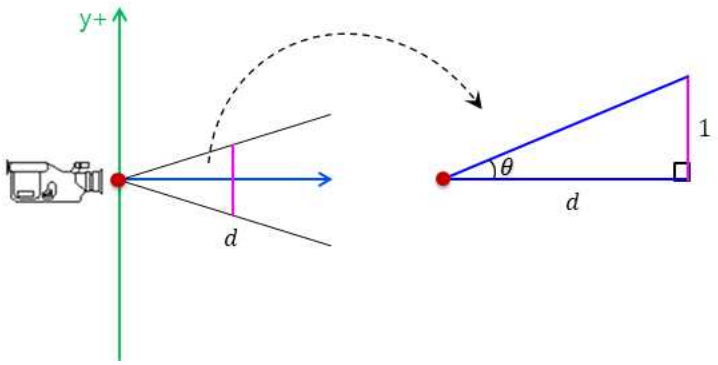
$$-1 \le \frac{x_c}{z_c} \le 1 \qquad -1 \le \frac{y_c}{z_c} \le 1$$



카메라의 시야각(FOV)이 90°가 아닌 카메라를 시야각이 90°인 카메라처럼 동작하게 하려면, 원근 투영 나누기를 하기 전에 카메라 좌표계의 점 (x_c, y_c, z_c)의 z-좌표를 투영 평면과 카메라 사이의 거리 d로 나누면 된다.

$$tan(\theta) = \frac{1}{d} \qquad d = \frac{1}{tan(\theta)}$$

$$(x_c, y_c, z_c) \rightarrow \left(x_c, y_c, \frac{z_c}{d}\right) = \left(x_c, y_c, \frac{z_c}{\frac{1}{tan(\theta)}}\right) = (x_c, y_c, \tan(\theta) z_c)$$



$(x_c, y_c, \tan(\theta) z_c)$를 원근 투영 나누기를 하면 다음과 같다.

$$(x_c, y_c, tan(\theta) z_c) \rightarrow \left(\frac{x_c}{tan(\theta) z_c}, \frac{y_c}{tan(\theta) z_c}, 1\right) = \left(\frac{1}{tan(\theta)} \frac{x_c}{z_c}, \frac{1}{tan(\theta)} \frac{y_c}{z_c}, 1\right)$$

이것은 (x_c, y_c, z_c)의 x-좌표와 y-좌표를 $tan(\theta)$로 나누고 원근 투영 나누기를 하는 것과 같다.

$$(x_c, y_c, z_c) \rightarrow \left(\frac{1}{tan(\theta)} x_c, \frac{1}{tan(\theta)} y_c, z_c\right)$$

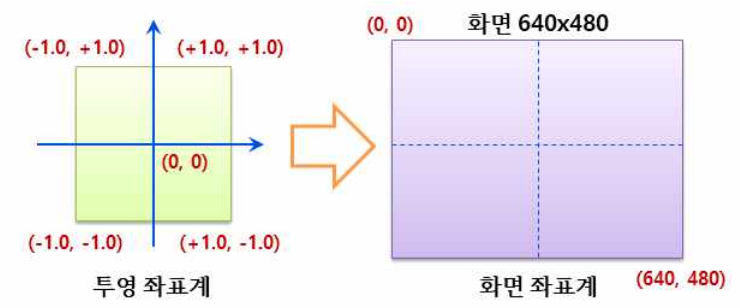
(x_c, y_c, z_c)의 x-좌표와 y-좌표를 $tan(\theta)$로 나누는 것(크기를 바꾸는)도 변환이므로 다음

과 같은 행렬로 표현할 수 있다.

$$\left(\frac{1}{tan(\theta)}x_c,\ \frac{1}{tan(\theta)}y_c,\ z_c,\ 1\right) = (x_c,\ y_c,\ z_c,\ 1)\begin{bmatrix} \frac{1}{tan(\theta)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

카메라의 시야각이 항상 90°인 것처럼 카메라 좌표계의 (x_c, y_c, z_c)를 변환할 수 있으므로 투영 사각형은 정사각형(정규화된 투영 사각형)이 된다.

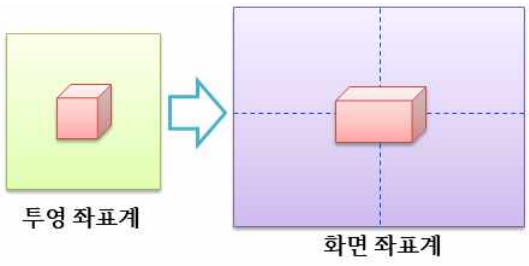
투영 좌표 변환 다음의 과정은 화면 좌표 변환이다. 화면 좌표 변환은 이 투영 사각형의 2차원 점들을 화면의 픽셀로 매핑하는 것이다. 일반적으로 화면(모니터 스크린) 또는 뷰포트는 다음 그림과 같이 가로가 세로보다 길다. 투영 좌표계의 정사각형을 가로가 640 픽셀이고 세로가 480 픽셀인 화면(뷰포트)으로 매핑하면 가로가 더 긴 직사각형이 될 것이다. 또는 투영 좌표계의 정육면체를 화면 좌표 변환을 하면 가로가 더 긴 직육면체가 될 것이다.



뷰포트의 가로 길이와 세로 길이의 비율을 종횡비(Aspect ratio)라고 한다. (640×480) 뷰포트에서 종횡비 r는 (640 / 480) $\approx$ 1.33333가 된다. 종횡비는 세로의 길이가 1일 때 가로의 길이의 비율을 의미한다.

$$Aspect\,Ratio = r = \frac{viewport.width}{viewport.height}$$

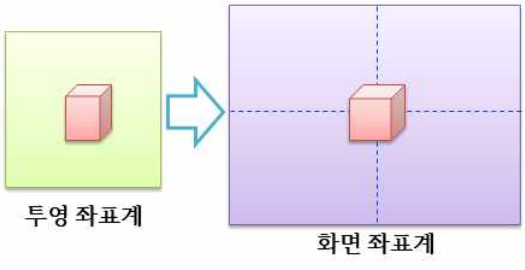
$$(x_c, y_c, z_c) \rightarrow (rx_c, y_c, z_c)$$



투영 좌표계의 정육면체는 화면 좌표계에서도 정육면체가 되어야 한다. 뷰포트의 가로(640)가 세로(480)보다 더 길다고 하면(종횡비 $r \approx 1.33333$), 화면 좌표 변환을 할 때 가로가 세로보다 ($r \approx 1.33333$)배 더 길어질 것이다. 그러므로 화면 좌표 변환을 하기 전에 투

영 좌표계의 점 (x_p, y_p)의 x-좌표를 화면 좌표 변환에서 늘어나는 비율만큼 줄이면 된다.

$$(x_p, y_p) \rightarrow \left(\frac{x_p}{r}, y_p\right) \rightarrow \left(r\left(\frac{x_p}{r}\right), y_p\right) = (x_p, y_p)$$



투영 좌표계의 점 (x_p, y_p)의 x_p를 종횡비 r로 나누는 것은 원근 투영 나누기를 하기 전에 카메라 좌표계의 점 (x_c, y_c, z_c)의 x-좌표를 종횡비 r로 나누는 것과 같다.

$$(x_c, y_c, z_c) \rightarrow \left(\frac{x_c}{r}, y_c, z_c\right) \rightarrow \left(\frac{x_c}{rz_c}, \frac{y_c}{z_c}, 1\right) \rightarrow \left(r(\frac{x_c}{rz_c}), \frac{y_c}{z_c}, 1\right) \rightarrow \left(\frac{x_c}{z_c}, \frac{y_c}{z_c}\right)$$

카메라 좌표계의 점 (x_c, y_c, z_c)의 x-좌표를 종횡비 r로 나누는 것도 변환이므로 다음과 같은 행렬로 표현할 수 있다.

$$\left(\frac{x_c}{r}, y_c, z_c, 1\right) = (x_c, y_c, z_c, 1)\begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이제 원근 투영 변환을 행렬로 표현하는 것에 대하여 생각해 보자. 카메라 좌표계의 점 (x_c, y_c, z_c)에 대하여 원근 투영 나누기를 하여 2차원 투영 좌표계의 점 (x_p, y_p)을 얻는 것은 위하여 행렬의 곱을 할 필요는 없다. 카메라 좌표계의 점 (x_c, y_c, z_c)에 대하여 단순히 나누기 연산을 하면 된다.

$$(x_c, y_c, z_c) \rightarrow \left(\frac{x_c}{z_c}, \frac{y_c}{z_c}, 1\right)$$

FOV가 90°가 아닌 경우와 뷰포트의 종횡비의 1이 아닌 경우를 위하여, 카메라 좌표계의 점 (x_c, y_c, z_c)에 대하여 x-좌표와 y-좌표를 다른 값으로 바꾸어야 한다. 이것은 카메라가 볼 수 있는 공간(영역)인 사각뿔을 변환시키는 의미를 가진다. 이러한 변환을 위하여 사용하는 행렬 $\boldsymbol{P}$를 원근 투영 변환 행렬(Pretspective projection matrix)이라고 한다. FOV가 90°가 아닌 경우와 뷰포트의 종횡비의 1이 아닌 경우를 위하여 카메라 좌표계의 점 (x_c, y_c, z_c)을 다음과 같은 변환을 하고 원근 투영 나누기를 한다.

$$(x_c, y_c, z_c, 1) \rightarrow \left(\frac{1}{tan(\theta)r}x_c, \frac{1}{tan(\theta)}y_c, z_c, 1\right)$$

$$\left(\frac{1}{tan(\theta)r}x_c,\ \frac{1}{tan(\theta)}y_c,\ z_c,\ 1\right) = (x_c, y_c, z_c, 1)\begin{bmatrix} \frac{1}{tan(\theta)} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left(\frac{1}{tan(\theta)r}x_c,\ \frac{1}{tan(\theta)}y_c,\ z_c,\ 1\right) = (x_c, y_c, z_c, 1)\begin{bmatrix} \frac{1}{tan(\theta)r} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{P} = \begin{bmatrix} \frac{1}{tan(\theta)r} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

카메라 좌표계의 점 (x_c, y_c, z_c)에 대하여 원근 투영 변환 행렬 $\boldsymbol{P}$를 곱하면 $(x_p, y_p, z_p, 1)$가 된다. 다음에 원근 투영 나누기 연산을 하면 x-좌표와 y-좌표는 정규화되지만 z-좌표는 1이 된다.

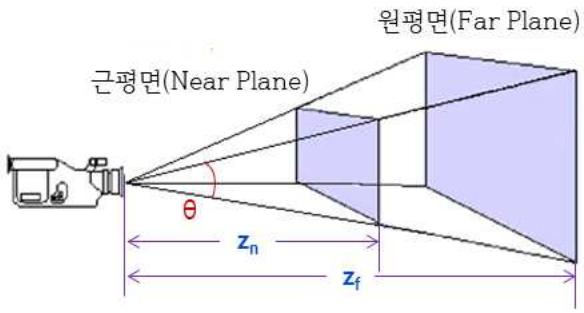
$$(x_c, y_c, z_c, 1)\boldsymbol{P} = (x_p, y_p, z_p, 1)$$

$$(x_p, y_p, z_p, 1) \rightarrow (x_p, y_p, z_p) \rightarrow \left(\frac{x_p}{z_p}, \frac{y_p}{z_p}, 1\right) \rightarrow \left(\frac{x_p}{z_p}, \frac{y_p}{z_p}\right)$$

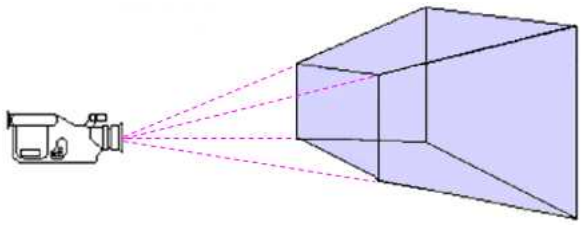
$$-1 \le \frac{x_p}{z_p} \le 1 \qquad -1 \le \frac{y_p}{z_p} \le 1$$

- 절두체(Frustum)

  카메라의 시야각이 θ일 때, 카메라가 볼 수 있는 영역은 다음 그림과 같이 사각뿔 모양이다. 카메라가 z-축 방향으로 볼 수 있는 가장 먼 거리를 z_f라고 하면 카메라 좌표계의 z-축과 수직이고 원점에서 거리(카메라 좌표계의 z-좌표)가 z_f인 평면을 원평면(Far plane)이라고 한다. 카메라가 z-축 방향으로 볼 수 있는 가장 가까운 거리(카메라 좌표계의 z-좌표)를 z_n이라고 하면 카메라 좌표계의 z-축과 수직이고 원점에서 거리가 z_n인 평면을 근평면(Near plane)이라고 한다.



근평면보다 더 가까운 거리의 점 또는 원평면보다 더 먼 거리의 점은 카메라에 보이지 않는다고 가정한다. 그러면 카메라가 볼 수 있는 영역은 사각뿔의 뾰족한 부분을 잘라낸 모양이다. 이러한 영역을 카메라 절두체라고 한다. 절두체는 6개의 사각형을 가지는 입체(육면체)이다.



원근 투영 나누기 연산을 하면 z-좌표가 0~1로 정규화될 수 있도록 원근 투영 변환 행렬을 수정하도록 하자. 원근 투영 나누기 연산을 하면 근평면까지의 거리가 0, 원평면까지의 거리가 1이 되고, 절두체 내의 점들은 z-좌표가 0~1로 정규화되도록 하자.

카메라 좌표계의 점 (x_c, y_c, z_c)을 원근 투영 변환 행렬 $\boldsymbol{P}$로 변환하면 (x_p, y_p, z_p, z)가 되도록 행렬 $\boldsymbol{P}$를 수정하자. 행렬 $\boldsymbol{P}$의 4번째 열벡터를 $(0, 0, 1, 0)$으로 수정하면 카메라 좌표계의 z-좌표가 변환된 벡터의 w-좌표가 된다.

$$(x_c, y_c, z_c, 1)\boldsymbol{P} = \left(\frac{1}{tan(\theta)r}x_c,\ \frac{1}{tan(\theta)}y_c, z_c, z_c\right)$$

$$\boldsymbol{P} = \begin{bmatrix} \frac{1}{tan(\theta)r} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

행렬 $\boldsymbol{P}$의 3행 3열($\boldsymbol{P}_{33}$)과 4행 3열($\boldsymbol{P}_{43}$)을 다음과 같이 바꾸어 보자.

$$\boldsymbol{P}_{33} = \frac{z_f}{(z_f - z_n)}$$

$$\boldsymbol{P}_{43} = \frac{-z_n z_f}{(z_f - z_n)}$$

$$\boldsymbol{P} = \begin{bmatrix} \frac{1}{tan(\theta)r} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & \frac{z_f}{(z_f - z_n)} & 1 \\ 0 & 0 & \frac{-z_n z_f}{(z_f - z_n)} & 0 \end{bmatrix}$$

이제 카메라 좌표계의 점 (x_c, y_c, z_c)을 원근 투영 변환 행렬 $\boldsymbol{P}$로 변환하면 (x_p, y_p, z_p, z_c)가 된다.

$$(x_c, y_c, z_c, 1)\boldsymbol{P} = (x_p, y_p, z_p, z_c) = \left(\frac{1}{tan(\theta)r}x_c, \frac{1}{tan(\theta)}y_c, \frac{(z_c - z_n)z_f}{(z_f - z_n)}, z_c\right)$$

4차원 벡터 (x_p, y_p, z_p, z_c)를 3차원 동차좌표계로 바꾸기 위하여 w-좌표(z_c)로 원근 투영 나누기를 하자.

$$(x_p, y_p, z_p, z_c) \rightarrow \left(\frac{x_p}{z_c}, \frac{y_p}{z_c}, \frac{z_p}{z_c}, 1\right) = \left(\frac{1}{tan(\theta)r}\frac{x_c}{z_c}, \frac{1}{tan(\theta)}\frac{y_c}{z_c}, \frac{(z_c - z_n)z_f}{(z_f - z_n)z_c}, 1\right)$$

$$(x_p, y_p, z_p, z_c) \rightarrow \left(\frac{1}{tan(\theta)r}\frac{x_c}{z_c}, \frac{1}{tan(\theta)}\frac{y_c}{z_c}, \frac{(z_c - z_n)z_f}{(z_f - z_n)z_c}\right)$$

$$f(z_c) = \frac{z_p}{z_c} = \frac{(z_c - z_n)z_f}{(z_f - z_n)z_c}$$

$f(z_c)$를 깊이 값(Depth)라고 한다. $(z_c = z_f)$일 때 $f(z_c) = 1$이고, $(z_c = z_n)$일 때 $f(z_c) = 0$이다. $(z_c < z_n)$이면 $f(z_c) < 0$이고, $(z_c > z_f)$이면 $f(z_c) > 1$이다.

$$(z_c = z_n) \rightarrow f(z_n) = \frac{(z_n - z_n)z_f}{(z_f - z_n)z_n} = 0$$

$$(z_c = z_f) \rightarrow f(z_f) = \frac{(z_f - z_n)z_f}{(z_f - z_n)z_f} = 1$$

카메라 좌표계의 점 (x_c, y_c, z_c)의 z-좌표가 $(z_n \le z_c \le z_f)$를 만족하면 다음이 성립한다.

$$0 \le f(z_c) = \frac{(z_c - z_n)z_f}{(z_f - z_n)z_c} \le 1$$

이제 카메라 좌표계의 점 $(x_c, y_c, z_c, 1)$에 원근 투영 변환 행렬 $\boldsymbol{P}$를 곱하고 원근 투영 나누기 연산을 하는 것을 원근 투영 변환이라고 하자. 카메라 절두체 내부의 점 (x_c, y_c, z_c)를 원근 투영 변환한 점 (x_p, y_p, z_p)은 다음을 모두 만족한다.

$$(x_p, y_p, z_p) = \left(\frac{1}{tan(\theta)r}\frac{x_c}{z_c}, \frac{1}{tan(\theta)}\frac{y_c}{z_c}, \frac{(z_c - z_n)z_f}{(z_f - z_n)z_c}\right)$$

$$-1 \le x_p \le 1$$

$$-1 \le y_p \le 1$$

$$0 \le z_p \le 1$$

원근 투영 변환을 한 후에 $(|x_p| > 1)$, $(|y_p| > 1)$, $(z_p < 0)$, 또는 $(z_p > 1)$이면 그 점은 카메라의 절두체 밖의 점(카메라에 보이지 않는)이므로 렌더링을 하지 않아도 된다. 이렇게 판단하여 렌더링을 하지 않는 것을 클리핑(Clipping)이라고 한다.

카메라는 원근 투영 변환을 위하여 다음과 같이 원근 투영 변환 행렬을 가져야 한다.

```
class CCamera
{
  ...
  Matrix      project;
```

```
  ...
};

void CCamera::GenerateProjectionMatrix(float zn, float zf, float
width, float height, float fov)
{
  project._22 = 1.0f / tan(float((fov/2.0f)*3.141592654f/180.0f));
  project._11 = project._22 / (width / height);
  project._33 = zf / (zf - zn);
  project._34 = 1.0f;
  project._43 = -(zn * zf) / (zf - zn);
  project._12 = project.13 = ... = 0.0f;
}
```

- 깊이 값(Depth Value) 그래프

  $f(z_c)$는 z_c의 함수이므로 함수 $f(z_c)$의 그래프를 그려보자.

$$f(z_c)=\frac{(z_c-z_n)z_f}{(z_f-z_n)z_c}$$

$$f(z_c)=\frac{(z_c-z_n)z_f}{(z_f-z_n)z_c}=\frac{z_cz_f-z_nz_f}{(z_f-z_n)z_c}=\frac{z_cz_f}{(z_f-z_n)z_c}-\frac{z_nz_f}{(z_f-z_n)z}=\frac{z_f}{(z_f-z_n)}-\frac{z_nz_f}{(z_f-z_n)z_c}$$

$$f(z_c)=\frac{(z_c-z_n)z_f}{(z_f-z_n)z_c}=\frac{z_f}{(z_f-z_n)}\left(1-\frac{z_n}{z_c}\right)=k\left(1-\frac{z_n}{z_c}\right)$$

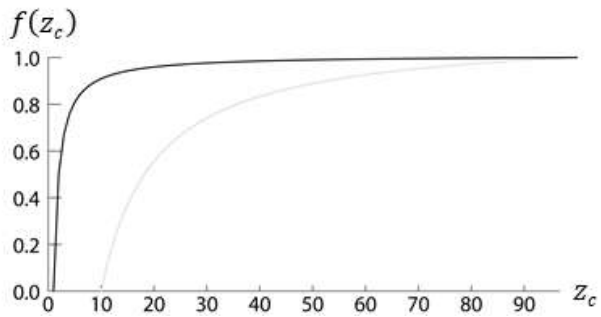
$$f(z_c)=k\left(1-\frac{z_n}{z_c}\right)\qquad k=\frac{z_f}{(z_f-z_n)}$$

$(z_n=1)$이고 $(z_f=5000)$일 때, $z_p=f(z_c)$의 값을 구하여 그래프를 그려보자.

$$k=\frac{z_f}{(z_f-z_n)}=\frac{5000}{(5000-1)}=\frac{5000}{4999}\approx 1.0$$

$$f(z_c)=k\left(1-\frac{z_n}{z_c}\right)\approx\left(1-\frac{1}{z_c}\right)$$

$f(1)=0, f(2)\approx 0.5, f(5)\approx 0.8, f(10)\approx 0.9, f(100)\approx 0.99, f(2000)\approx 0.99999975$



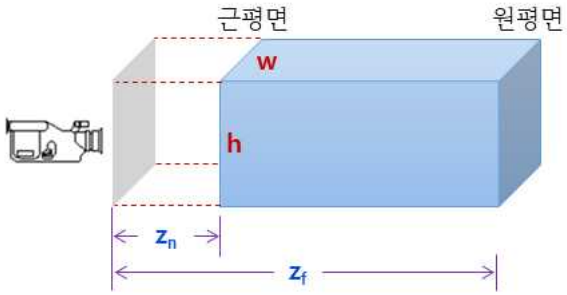
이 그래프는 깊이 값이 급격하게 1에 가까워짐을 보이고 있다. z_n이 z_f 보다 아주 작을 때, 카메라 좌표계의 점 (x_c, y_c, z_c)의 z-좌표(z_c)가 조금만 커도 깊이 값은 1.0에

가깝게된다. 즉, 깊이 값이 z_n~z_f에 균일하게 분포하지 않는다. 이것은 컴퓨터의 부동소수점 실수 표현(float, IEEE754)의 방식 때문에 문제가 될 수 있다. 어떤 문제가 있을 까요?

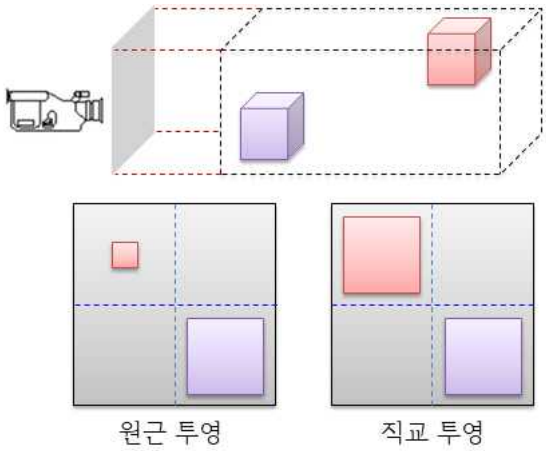
⑤ 직교 투영 변환(Orthogonal Projection Transformation)

카메라 좌표계의 점들이 투영 평면에 수직하게 투영되면 직교 투영이라 한다. 직교 투영을 하는 카메라에 보이는 점들의 공간은 다음 그림과 같이 직육면체가 된다. 투영 평면에 수직으로 투영이 되기 때문에(z-좌표에 상관없기 때문에) 투영 나누기를 할 필요가 없다. 즉, 투영 연산을 할 때 카메라 좌표계의 z-좌표가 필요하지 않다. 카메라 좌표계의 점 $(x_c, y_c, z_c, 1)$을 직교 투영 변환 행렬 $\boldsymbol{P}$로 변환하면 $(x_p, y_p, z_p, 1)$가 되며, w-좌표가 1이 되므로 동차좌표계로 바꾸기 위하여 w-좌표로 나눌 필요가 없다.

$$(x_p, y_p, z_p, 1) \rightarrow (x_p, y_p, z_p)$$



그리고 투영 평면에 수직으로 투영이 되기 때문에 카메라에서 멀고 가까운 것에 상관없이(z-좌표에 상관없이) 카메라 좌표계의 x-좌표와 y-좌표가 같으면 투영 평면의 같은 점으로 투영된다.



직교 투영 변환 행렬 $\boldsymbol{P}$는 다음과 같다. w는 투영 사각형의 가로 길이이고, h는 투영 사각형의 세로 길이이다. z_n은 근평면까지의 거리이고 z_f은 원평면까지의 거리이다.

$$\boldsymbol{P} = \begin{bmatrix} \frac{2}{w} & 0 & 0 & 0 \\ 0 & \frac{2}{h} & 0 & 0 \\ 0 & 0 & \frac{1}{(z_f - z_n)} & 0 \\ 0 & 0 & \frac{-z_n}{(z_f - z_n)} & 1 \end{bmatrix}$$

$$(x_c, y_c, z_c, 1)\boldsymbol{P} = (x_p, y_p, z_p, 1) = \left(\frac{2x_c}{w}, \frac{2y_c}{h}, \frac{(z_c - z_n)}{(z_f - z_n)}, 1\right)$$

카메라에 보이는 카메라 좌표계의 점 $(x_c, y_c, z_c, 1)$를 직교 투영 변환을 하면 다음과 같이 정규화된다.

$$-1 \le \frac{2x_c}{w} \le 1$$

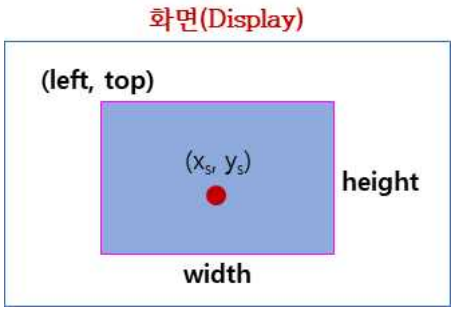
$$-1 \le \frac{2y_c}{h} \le 1$$

$$0 \le \frac{(z_c - z_n)}{(z_f - z_n)} \le 1$$

⑥ 화면 좌표 변환(Screen Space Mapping)

화면 좌표 변환은 투영 사각형을 화면(스크린)으로 매핑하는 변환이다. 카메라의 투영 사각형이 매핑될 화면 사각형 영역을 뷰포트(Viewport)라고 하며 다음과 같이 표현할 수 있다.

```
struct Viewport
{
   float       left;
   float       top;
   float       width;
   float       height;
};
```



뷰포트가 주어지면 투영 사각형의 점 (x_p, y_p)를 다음과 같이 화면 좌표계 (x_s, y_s)로 변환할 수 있다.

$$x_s = viewport_{left} + \frac{viewport_{width}}{2}(x_p + 1)$$

$$y_s = viewport_{top} + \frac{viewport_{height}}{2}(-y_p + 1)$$

l, t, w, h가 뷰포트의 left, top, width, height일 때, 투영 사각형의 점 (x_p, y_p)를 다음과 같은 행렬 $\boldsymbol{S}$를 사용하여 화면 좌표계 (x_s, y_s)로 변환할 수 있다.

$$(x_s, y_s, 1, 1) = (x_p, y_p, 1, 1)\boldsymbol{S}$$

$$\boldsymbol{S} = \begin{bmatrix} \frac{w}{2} & 0 & 0 & 0 \\ 0 & \frac{-h}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (l+\frac{w}{2}) & (t+\frac{h}{2}) & 0 & 1 \end{bmatrix}$$

- 픽킹(Picking)

  게임에서 플레이어가 게임 세상의 게임 객체를 마우스로 선택하는 것을 마우스 픽킹이라고 한다. 마우스로 어떤 게임 객체를 선택했는 가를 알려면 마우스 커서의 위치(화면 좌표계)를 월드 좌표계로 변환을 해야 한다. 모델 좌표계의 점을 화면 좌표계로 변환하는 과정을 역으로 변환하면 화면 좌표계의 점으로부터 모델 좌표계의 점을 구할 수 있다.

화면 좌표계의 점 (x_s, y_s)에서 카메라 좌표계의 점 (x_c, y_c, z_c)를 구하는 과정을 먼저 알아 보자. 카메라 좌표계의 점 (x_c, y_c, z_c)에서 월드 좌표계의 점 (x_w, y_w, z_w)를 구하려면 카메라 변환의 역 변환(카메라 변환 행렬 $\boldsymbol{V}$의 역행렬로 곱셈 연산)을 하면 된다.

$$(x_w, y_w, z_w) = (x_c, y_c, z_c)(\boldsymbol{V}^{-1})$$

화면 좌표계의 점 (x_s, y_s)에서 카메라 좌표계의 점 (x_c, y_c, z_c)를 구하는 과정은 다음 그림과 같은 과정을 거치면 된다. 윈도우 응용 프로그램에서 클라이언트 영역의 한 점을 마우스 버튼을 클릭하면 마우스 커서의 위치 (x_s, y_s)를 알 수 있다(GetCursorPos() 또는 WM_LBUTTONDOWN 메시지).

화면 좌표 (x_s, y_s)에서 투영 좌표계의 점 (x_p, y_p)를 구하자. 플레이어가 게임 화면에서 화면 좌표 (x_s, y_s)의 객체를 선택할 수 있는 것은 투영 좌표계의 점 $(x_p, y_p) = (x_p, y_p, 1)$을 뷰포트 매핑(화면 좌표 변환)하여 화면에 그렸기 때문이다. 화면 좌표 (x_s, y_s)가 주어지면, 화면 좌표에 대응되는 투영 좌표계의 점 (x_p, y_p)은 뷰포트

매핑(화면 좌표 변환)의 역변환을 하여 다음과 같이 구할 수 있다.

$$x_p = \frac{2(x_s - viewport_{left})}{viewport_{width}} - 1$$

$$y_p = \frac{-2(y_s - viewport_{top})}{viewport_{height}} + 1$$

투영 좌표계의 점 $(x_p, y_p) = (x_p, y_p, 1)$에 대응되는 카메라 좌표계의 점을 구하자. 위에서 구한 점 (x_p, y_p)은 카메라 좌표계의 점 $(x_c, y_c, z_c, 1)$에 다음의 원근 투영 변환 행렬 $\boldsymbol{P}$를 곱한 것이다.

$$(x_p, y_p, z_p, z_c) = (x_c, y_c, z_c, 1)\boldsymbol{P}$$

$$\boldsymbol{P} = \begin{bmatrix} \frac{1}{tan(\theta)r} & 0 & 0 & 0 \\ 0 & \frac{1}{tan(\theta)} & 0 & 0 \\ 0 & 0 & \frac{z_f}{(z_f - z_n)} & 1 \\ 0 & 0 & \frac{-z_n z_f}{(z_f - z_n)} & 0 \end{bmatrix}$$

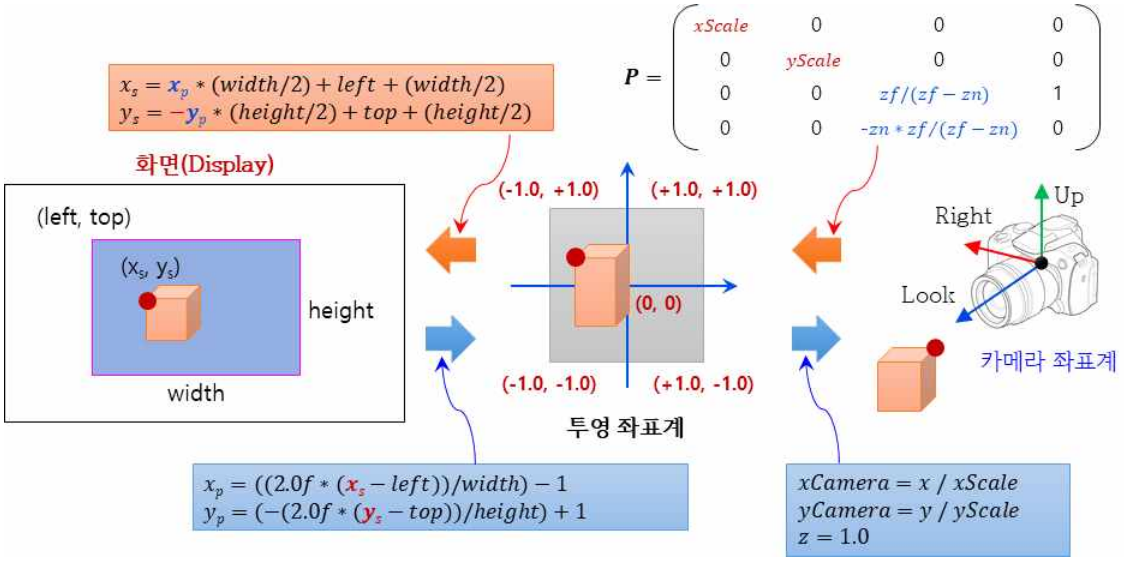
원근 투영 변환 행렬 $\boldsymbol{P}$의 1행 1열의 원소를 $\boldsymbol{P}_{11}$, 2행 2열의 원소를 $\boldsymbol{P}_{22}$라고 하면, x_p와 y_p는 다음과 같다.

$$x_p = x_c\boldsymbol{P}_{11} \qquad y_p = y_c\boldsymbol{P}_{22}$$

x_c와 y_c는 다음과 같다.

$$x_c = \frac{x_p}{\boldsymbol{P}_{11}} \quad y_c = \frac{y_p}{\boldsymbol{P}_{22}}$$

화면 좌표계의 점은 z-좌표가 없다는 것에 유의하라(2차원 좌표계). 화면 좌표계의 점 (x_s, y_s)에 대응되는 카메라 좌표계의 점 (x_c, y_c)는 동차 좌표계 $(x_c, y_c, 1)$과 같다. $(x_c, y_c, 1)$은 카메라 좌표계에서 투영 평면(투영 사각형)에 있는 점이다.

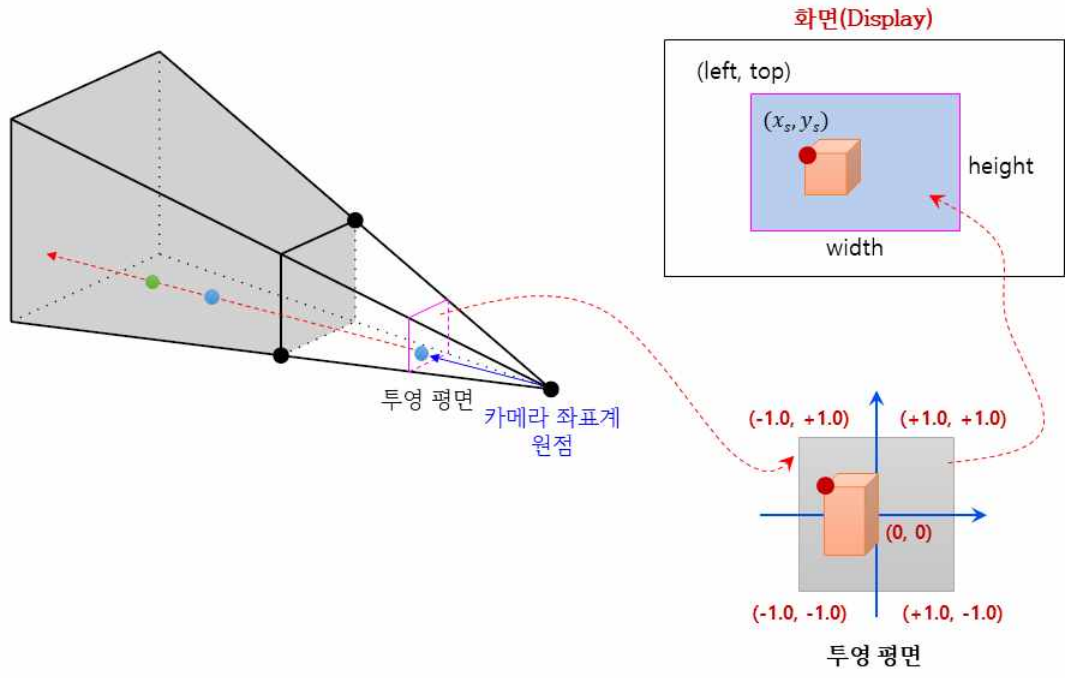


$$(x_c, y_c, 1) \rightarrow (x_c, y_c)$$

$$(x_c, y_c) \rightarrow (x_c, y_c, 1)$$

다음 그림과 같이 투영 평면의 한 점 $(x_c, y_c, 1)$에 투영되는 카메라 좌표계의 점은 무수히 많을 수 있다. 그러므로 화면의 한 점 또는 투영 평면의 한 점에서 카메라 좌표계의 점의 z-좌표를 직접 구할 수는 없다.

$$(2x_c, 2y_c, 2) \rightarrow (x_c, y_c, 1)$$
$$(3x_c, 3y_c, 3) \rightarrow (x_c, y_c, 1)$$
$$\cdots$$
$$(kx_c, ky_c, k) \rightarrow (x_c, y_c, 1)$$



$(x_c, y_c, 1)$로 투영될 수 있는 많은 점들 중에서 화면에 그려지는(마우스로 선택할 수 있는) 점은 카메라에 가장 가까운 점이다. 카메라에 가장 가까운 점은 광선 교차를 사용하면 구할 수 있다.

뷰포트(화면)의 점 (x_s, y_s)에 대응되는 투영 평면의 점(카메라 좌표계의 점) $(x_c, y_c, 1)$을 구했다. 이제, 카메라 좌표계의 원점 $(0, 0, 0)$에서 점 $(x_c, y_c, 1)$으로 향하는 광선(픽킹 광선)을 생각하자. 게임 객체(표면)는 삼각형들의 집합이라고 생각할 수 있다. 광선의 시작점이 $(0, 0, 0)$이고 방향이 $(x_c, y_c, 1)$인 광선이 게임 객체를 구성하는 삼각형 중 하나와 교차한다면 광선이 게임 객체와 교차한 것이다. 이 게임 객체를 렌더링하면 게임 객체의 표면을 구성하는 한 점이 뷰포트(화면)의 점 (x_s, y_s)에 그려지게 된다. 이렇게 광선과 교차하는 모든 게임 객체들 중에서 카메라에 가장 가까운 게임 객체를 찾으면 플레이어는 이 게임 객체를 화면에서 픽킹한 것이다.

광선과 삼각형의 교차와 카메라에 가장 가까운 게임 객체를 찾는 것은 앞에서 다룬 벡터의 연산으로 해결할 수 있을 것이라 기대한다.

⑦ 변환과 역변환에 대한 정리와 고찰

모델 좌표계로 주어지는 한 점을 화면 좌표계로 변환하기 위한 과정(파이프라인)은 다음 그림에서 오른쪽 방향으로 진행되며, 각 단계에는 변환을 나타내는 행렬이 필요하다. 각 단계의 역변환은 역행렬로 표현할 수 있다.

모델 좌표계의 점 (x_m, y_m, z_m)을 월드 좌표계의 점 (x_w, y_w, z_w)으로 변환하려면 게임 객체의 월드 변환 행렬이 필요하다. 월드 좌표계의 점 (x_w, y_w, z_w)을 카메라 좌표계의 점 (x_c, y_c, z_c)으로 변환하려면 카메라 객체의 카메라 변환 행렬이 필요하다. 카메라 좌표계의 점 (x_c, y_c, z_c)을 투영 좌표계의 점 (x_p, y_p, z_p)으로 변환하려면 카메라 객체의 투영 변환 행렬이 필요하다. 투영 좌표계의 점 (x_p, y_p, z_p)을 화면 좌표계의 점 (x_s, y_s)으로 변환하려면 원근 투영 나누기 연산을 하고 카메라 객체의 뷰포트를 사용하면 된다.

각 단계의 역변환을 하려면 단계의 변환 행렬의 역행렬을 사용하면 될 것이다. 주의할 것은 게임 세상을 구성하는 게임 객체는 여러 개이지만 카메라 객체는 하나이다. 변환의 관점에서 카메라 객체는 카메라 변환 행렬, 투영 변환 행렬, 뷰포트로 구성된다.

3D 게임 프로그램에서 어떤 문제를 카메라 좌표계에서 해결하기 위한 Algorithm() 함수를 가지고 있다고 가정하자. 이 함수 Algorithm()는 카메라 좌표계의 점에 대하여 어떤 계산을 수행하여 참/거짓을 반환한다. 예를 들어, 알고리즘 Algorithm()는 한 점이 카메라 절두체에 완전히 포함되는 가를 판단할 수 있다고 가정하자. 게임 세상의 모든 게임 객체들에 대하여 카메라 절두체에 완전히 포함되는 게임 객체들의 리스트를 구해 보자. gnGameObjects가 씬의 게임 객체의 개수, gpGameObjects[]가 게임 객체들의 배열, gpCamera가 카메라 객체에 대한 포인터이다. 카메라 객체는 절두체(CFrustum 클래스)를 멤버 변수(frustum)로 포함하고 있고, CFrustum.Contain() 함수는 한 점이 절두체 내부의 점인가를 판단한다고 가정하자.

- 경우 1

TransformModelToCamera() 함수가 모델 좌표계의 점을 카메라 좌표계로 변환하는 함수 일 때, 대략적인 알고리즘은 다음과 같을 것이다.

```
for (int i = 0; i < gnGameObjects; i++)
{
   if (IsGameObjectInFrustum(&gpGameObject[i])
   {
      ...;
   }
```

```
}

bool IsGameObjectIncludedInFrustum(CObject *pObject)
{
   CMesh *pMesh = pObject->pMesh;
   for (int j = 0; j < pMesh->nFaces; j++)
   {
      CPolygon* pPolygon = pMesh->pFaces[j];
      for (int k = 0; k < pPolygon->nVertices; k++)
      {
         CVertex model = pPolygon->pVertices[k];
         CVeretex camera = TransformModelToCamera(model, pObject);
         if (!Algorithm(camera)) return(false);
      }
   }
   return(true);
}

CVertex TransformModelToCamera(CVertex model, CObject* pObject)
{
  CVertex world = WorldTransform(model, pObject);
  CVertex camera = CameraTransform(world);
  return(camera);
}

bool Algorithm(CVertex camera)
{
  return(gpCamera->frustum.Contain(camera));
}
```

게임 세상의 게임 객체들의 개수 gnGameObjects = 100개, 게임 객체의 메쉬의 다각형 개수가 평균적으로 100000개이고 다각형은 삼각형이라 가정하면(정점의 개수는 300000개), 함수 TransformModelToCamera()는 $(100 \times 300000) = 30000000$번 호출될 것이다. Algorithm() 함수도 $(100 \times 300000) = 30000000$번 호출될 것이다. 위의 알고리즘은 원하는 대로 잘 수행될 것이다. 그러나 이 알고리즘이 최선의 방법인 가를 생각해볼 필요가 있다.

카메라 객체는 1개이고 게임 객체들의 개수와 메쉬의 정점들은 여러 개이다. 이제, 함수 Algorithm()이 카메라 좌표계가 아닌 모델 좌표계에서 문제를 해결한다고 가정하자. 그러면 카메라 좌표계의 절두체(육면체)를 모델 좌표계로 변환해야 한다. 카메라 좌표계의 절두체(육면체)를 모델 좌표계로 변환하려면 카메라 변환 행렬의 역행렬과 게임 객체의 월드 변환 행렬의 역행렬이 필요하다.

- 경우 2

  TransformFrustumToModel() 함수는 이러한 역행렬을 사용하여 절두체를 모델 좌표계로 변환하는 함수일 때, 대략적인 알고리즘은 다음과 같을 것이다.

```
for (int i = 0; i < gnGameObjects; i++)
{
```

```
    if (IsGameObjectInFrustum(&gpGameObject[i], &frustum)
    {
       ...;
    }
  }

  bool IsGameObjectInFrustum(CObject* pObject)
  {
    CFrustum frustum = TransformFrustumToModel(pObject);
    CMesh *pMesh = pObject->pMesh;
    for (int j = 0; j < pMesh->nFaces; j++)
    {
       CPolygon* pPolygon = pMesh->pFaces[j];
       for (int k = 0; k < pPolygon->nVertices; k++)
       {
          CVertex model = pPolygon->pVertices[k];
          if (!Algorithm(model, &frustum) return(false);
       }
    }
    return(true);
  }

  CFrustum TransformFrustumToModel(CObject *pObject)
  {
    return(gpCamera->frustum.TransformToModel(pObject));
  }

  bool Algorithm(CVertex model, CFrustum* pFrustum)
  {
    return(pFrustum->Contain(model));
  }
```

카메라 좌표계의 절두체를 모델 좌표계로 변환하는 TransformFrustumToModel() 함수는 100번 호출되고, Algorithm() 함수도 $(100 \times 300000) = 30000000$번 호출될 것이다. TransformFrustumToModel() 함수와 TransformModelToCamera() 함수가 같은 연산양을 가진다고 가정하면 경우 1의 연산양이 경우 2의 연산양보다 300000 배 더 많게 된다.