3D Game Programming Basic with Direct3D

2.3 변환(Transform)

2.3.1 기하 변환(Geometric Transform)

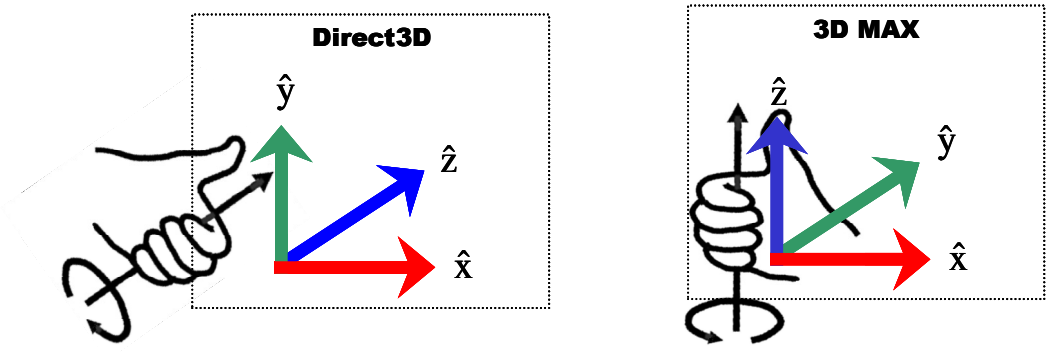
앞서 기초 수학을 시작할 때에 3D의 그래픽 파이프라인은 정점의 위치 벡터를 주어진 행렬을 통해서 변환이 된다고 했습니다. 정점의 위치를 바꾸는 변환은 크기 변환(Scaling transform), 회전 변환(Rotation Transform), 이동 변환(Translation Transform) 3 종류가 있으며 이러한 변환을 통틀어 기하 변환(Geometry Transform) 이라 합니다.

기하 변환의 수학적인 형태는 행렬과 열 벡터 또는 행 벡터와 행렬의 곱입니다.

EX) 기하 변환의 예

$$[x \quad y \quad z]\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}, \qquad \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

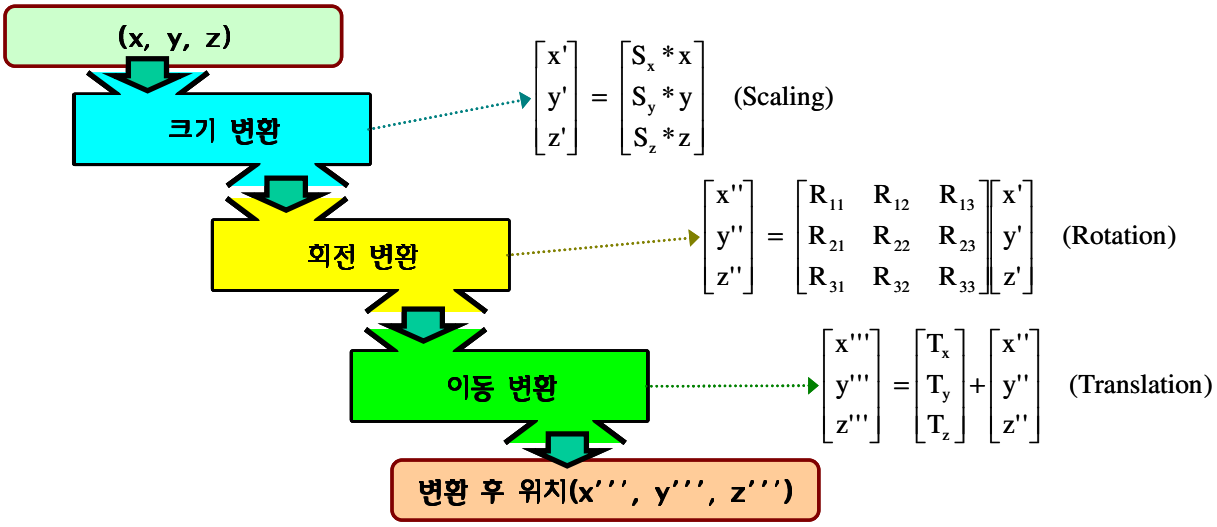
D3D는 왼손 좌표계를 사용하기 때문에 위의 예의 왼쪽처럼 행 벡터 * 행렬 곱을 변환을 수행하며, OpenGL, 3D MAX등 오른손 좌표계를 사용하는 시스템에서는 행렬 * 열 벡터로 변환을 수행합니다. 다음 그림은 왼손 좌표계와 오른손 좌표계의 차이를 보여주는 그림입니다.



<왼손 좌표계와 오른손 좌표계>

수학의 모든 이론은 오른손 좌표계를 중심으로 설명되어 있습니다. 그런데 D3D는 행렬 연산을 컴퓨터 시스템에서 빠르게 처리하기 위해 왼손 좌표계를 사용합니다. 따라서 변환에서 행 벡터 * 행렬 연산을 진행합니다.

만약 3차원 벡터가 크기 변환, 회전 변환, 이동을 한다고 가정합시다. 3차원 벡터는 (x, y, z) 3개의 성분으로 구성되어 있어서 이것을 행렬과 곱셈을 하려면 3x3 행렬이 필요 합니다. 따라서 다음 그림처럼 순서대로 벡터가 변환 될 때 회전에 대한 3x3 행렬, 각 벡터 성분에 대한 곱셈, 덧셈 과정이 필요합니다.



이런 과정으로 정점을 변환해도 됩니다. 그런데 위 3개의 변환을 단 하나의 행렬로 처리할 수 있음을 사람들을 알게 되었습니다.
크기, 회전, 이동을 하나의 행렬로 처리하기 위해서 3차원 벡터는 4차원 동차 좌표(Homogeneous Coordinate)로 바꾸어야 됩니다. 예를 들어 다음과 같은 4차원 동차 좌표(x, y, z, w) 들은 전부 동차 좌표 입니다.

EX) 4차원 동차 좌표 (1,2,3,4) ↔ (2,4, 6, 8) ↔ (0.5, 1, 1.5, 2)

4 차원에서 두 개의 좌표가 동차 좌표가 되려면 $\vec{a} = k * \vec{b}$ 관계가 성립 되면 됩니다. 이 관계를 이용해서 3차원 좌표 (x, y, z)는 4차원 동차 좌표에서 무수히 많은 좌표를 가질 수 있으나 특별히 마지막 w 값을 1로 하는 동차 좌표를 사용합니다.

3차원 좌표 (x, y, z) ➔ 4차원 동차 좌표 (x, y, z, 1)

4 차원 좌표는 변환을 행렬을 이용할 경우 4x4 행렬이 필요하고, 이 행렬을 통해서 변환 후에 w 값으로 x, y, z, w를 나누어 항상 w= 1이 되도록 합니다.

이것을 수식으로 나타내면 다음과 같습니다.
먼저 3차원 좌표 (x, y, z)를 4차원 동차 좌표 (x, y, z, 1)로 바꾼 후에 기하 변환 행렬 적용 합

니다.

$$[x' \quad y' \quad z' \quad w'] = [x \quad y \quad z \quad 1] \begin{bmatrix} R_{11} & R_{12} & R_{13} & M_{14} \\ R_{21} & R_{22} & R_{23} & M_{24} \\ R_{31} & R_{32} & R_{33} & M_{34} \\ T_x & T_y & T_z & M_{44} \end{bmatrix}$$

위의 벡터 행렬 연산은 다음과 같이 될 것입니다.

$$x' = x*R_{11} + y*R_{21} + z*R_{31} + T_x$$
$$y' = x*R_{12} + y*R_{22} + z*R_{32} + T_y$$
$$z' = x*R_{13} + y*R_{23} + z*R_{33} + T_z$$
$$w' = x*R_{14} + y*R_{24} + z*R_{34} + R_{44}$$
$$(\text{if } R_{14} \leftarrow 0 \text{ and } R_{24} \leftarrow 0 \text{ and } R_{34} \leftarrow 0 \text{ and } R_{44} \leftarrow 1 \text{ then } w' = 1)$$

이렇게 구한 x', y', z', w' 값을 w'=1인 좌표로 다시 바꿉니다.

$$x'/=w', x'/=w', x'/=w', x'/=w'$$
$$[x' \quad y' \quad z' \quad w'] = [x'/w' \quad y'/w' \quad z'/w' \quad 1]$$

이것을 3차원 좌표 (x', y', z')으로 하면 크기, 회전, 이동에 대해서 하나의 행렬로 대치가 가능합니다.
다음으로 이동, 크기, 회전 변환에 대해서 4차원 동차 좌표에 해당하는 변환 행렬을 만들어 봅시다.

2.3.2 이동 변환(Translation Transform)

이동 변환은 정점의 평행 이동을 의미하며 정점의 평행 이동은 정점의 위치를 상대적으로 이동하는 것입니다. 이것을 벡터 수식으로 나타내면 다음과 같습니다.

$$V' = V(V_x, V_y, V_z) + T(T_x, T_y, T_z)$$
$$= (V_x + T_x, V_y + T_y, V_z + T_z)$$

이것을 벡터 T의 성분 T_x, T_y, T_z 로 4x4 이동 행렬을 다음과 같이 구성하면 연산의 결과가 앞의 수식과 같음을 알 수 있습니다.

$$\begin{bmatrix} V_x & V_y & V_z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix} = \begin{bmatrix} V_x + T_x & V_y + T_y & V_z + T_z & 1 \end{bmatrix}$$

벡터 T에 대한 이동 변환 행렬

$$T(t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

이 이동 행렬의 역 행렬은 다음과 같이 벡터 T의 방향을 반대로 해서 만든 이동 행렬과 동일합니다.

$$T^{-1}(t) = T(-t) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 1 \end{bmatrix}$$

DXSDK는 이동 벡터를 이용해서 이동 행렬을 구하는 함수가 있습니다. 위의 벡터 T의 성분으로 DXSDK 함수를 사용해서 이동 행렬을 다음과 같이 만듭니다.

```
D3DXMATRIX      OutMatrix;
D3DXMatrixTranslation(&OutMatrix, Tx,Ty,Tz);
```

2.3.3 크기 변환(Scaling Transform)

크기 변환은 정점의 위치에 scalar 배를 적용한 것입니다. 수식으로 나타내면 다음과 같습니다.

$$V' = S(S_x, S_y, S_z) \otimes V(V_x, V_y, V_z)$$
$$= (S_x * V_x, S_y * V_y, S_z * V_z)$$

이동 변환과 마찬 가지로 이것을 벡터 S의 성분 S_x, S_y, S_z 로 4x4 크기 변환 행렬을 다음과 같이 구성할 수 있으며 연산의 결과가 위의 수식과 같음을 알 수 있습니다.

$$\begin{bmatrix} V_x & V_y & V_z & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} V_x * S_x & V_y * S_y & V_z * S_z & 1 \end{bmatrix}$$

벡터 S에 대한 크기 변환 행렬

$$S(s) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이 크기 변환 행렬의 역 행렬은 다음과 같이 벡터 S의 성분을 역수로 하면 간단하게 만들 수 있습니다.

$$S^{-1}(s) = S(\frac{1}{s_x} \quad \frac{1}{s_y} \quad \frac{1}{s_z}) = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DXSDK는 이동 벡터를 이용해서 이동 행렬을 구하는 함수가 있습니다. 위의 벡터 T의 성분으로 DXSDK 함수로 이동 행렬을 다음과 같이 만듭니다.

```
D3DXMATRIX      OutMatrix;
D3DXMatrixScaling(&OutMatrix, Sx,Sy,Sz);
```

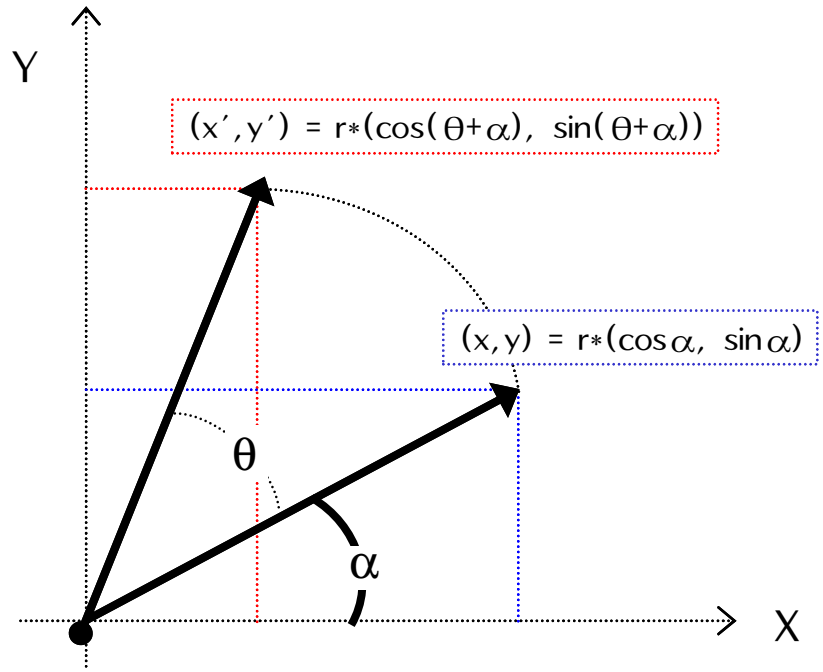
2.4 회전(Rotation)

아마도 회전 부분이 3D 수학에서 가장 어려운 축에 속하는 부분입니다. 회전은 오일러 앵글(Euler Angle)을 이용한 방법과 임의 축에 대한 회전 두 가지가 있습니다. 임의의 축에 대한 회전은 사원수(Quaternion)으로 풀 수 있고, 서로의 공식을 유도 할 수 있습니다. 사원수에서 행렬을 얻거나 행렬에서 사원수를 얻는 방법은 난이도가 있으니 처음 접하는 분들은 그 결과만 알고 있기 바랍니

다.

2.4.1 2차원에서 정점의 회전

오일러 앵글을 사용하려면 2차원에서 정점의 회전 변환을 알아야 합니다. 다음 그림과 같이 점(x, y)가 θ만큼 회전을 해서 (x', y')로 옮겨지는 것을 수식으로 풀어 봅시다.



<2차원 정점의 회전>

회전 이동한 x'과 y'은 다음과 같이 구할 수 있습니다.

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

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

이것을 행렬로 표현하면 다음과 같습니다.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} r*\cos(\theta+\alpha) \\ r*\sin(\theta+\alpha) \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

우리는 D3D를 가지고 게임을 만들 것이므로 벡터 * 행렬 곱으로 바꿉니다. 바꾸는 방법은 전치(Transpose) 하면 됩니다.

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

이 결과를 3 차원으로 확장 시켜서 다음과 같은 좌표 축에 대한 행렬을 만들 수 있습니다.

X 축에 대한 회전 행렬: $\mathbf{M}_{RotX}(\theta) = \begin{bmatrix}1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1\end{bmatrix}$

Y 축에 대한 회전 행렬: $\mathbf{M}_{RotY}(\theta) = \begin{bmatrix}\cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1\end{bmatrix}$

Z 축에 대한 회전 행렬: $\mathbf{M}_{RotZ}(\theta) = \begin{bmatrix}\cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1\end{bmatrix}$

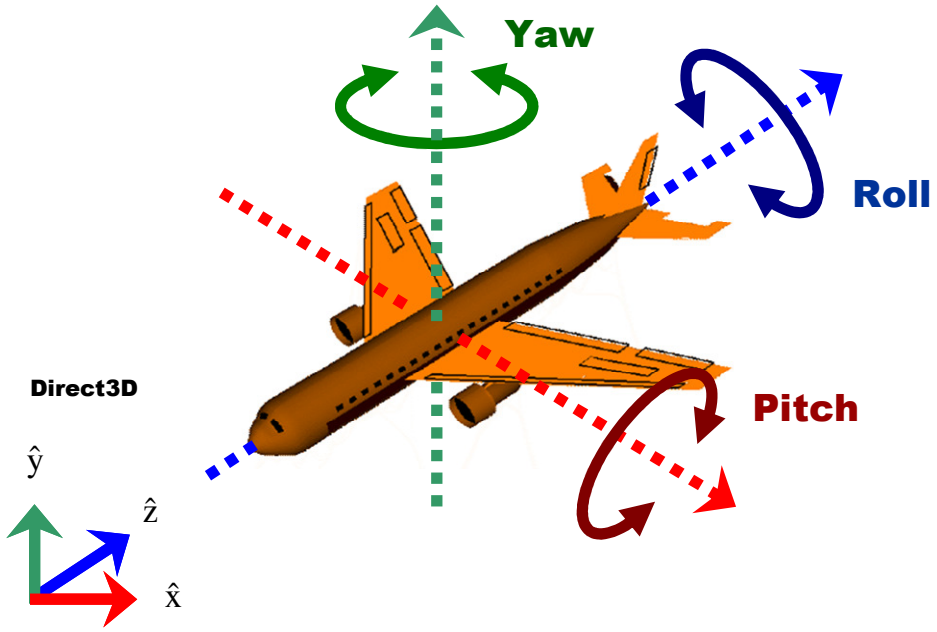
이들 행렬은 DXSDK에서 D3DXMatrixRotationX(), D3DXMatrixRotationY(), D3DXMatrixRotationZ() 함수로 구할 수 있습니다.

위의 회전 행렬들의 역 행렬을 구하는 방법은 두 가지가 있습니다. 가장 간단한 방법은 각도에 반대 각도(-θ)를 적용하면 됩니다.
또 다른 방법은 직교 행렬의 특성을 이용하는 것입니다. 모든 회전 행렬은 수학에서 직교 행렬(Orthogonal Matrix)이라는 행렬의 한 종류 입니다. 그런데 이 직교 행렬의 특징 중에 하나가 역 행렬과 전치 행렬이 같다는 것입니다. (이 부분 설명은 안 하겠습니다.) 따라서 위에서 구한 행렬은 전부 회전 행렬이므로 이 행렬들의 역 행렬은 전치 행렬로 구하면 됩니다. 반대 각도(-θ)를 적용한 것과 전치한 행렬을 비교해 보면 둘의 결과는 같음을 알 수 있고, 계산은 반대 각도를 넣어서 역 행렬을 구하는 것보다 전치 시키는 것이 더 빠르다는 것은 말할 필요도 없습니다.

2.4.2 오일러 앵글(Euler Angle)

오일러 앵글은 위에서 구한 축에 대한 행렬을 연속으로 곱해서 최종 회전 행렬을 만들 때 그 각도를 말합니다.

D3D는 X, Z 축을 평면으로 Y축이 하늘 방향을 지시하는 형태가 되므로 먼저 Y축에 대한 회전을 진행 합니다. 이것을 Yaw라 합니다. 다음으로 X축에 대한 회전을 합니다. 이것을 Pitch라 합니다. 마지막으로 Z축에 대한 회전을 합니다. 이것을 Roll이라 합니다.



<D3D에서의 Yaw, Pitch, Roll>

오일러 앵글로 구한 회전 행렬:

$$\vec{R}(Yaw, Pitch, Roll) = M_{RotY}(Yaw) * M_{RotX}(Pitch) * M_{RotZ}(Roll)$$

DXSDK에서는 다음과 같이 오일러 앵글에 대한 회전 행렬을 구하는 함수가 존재합니다.

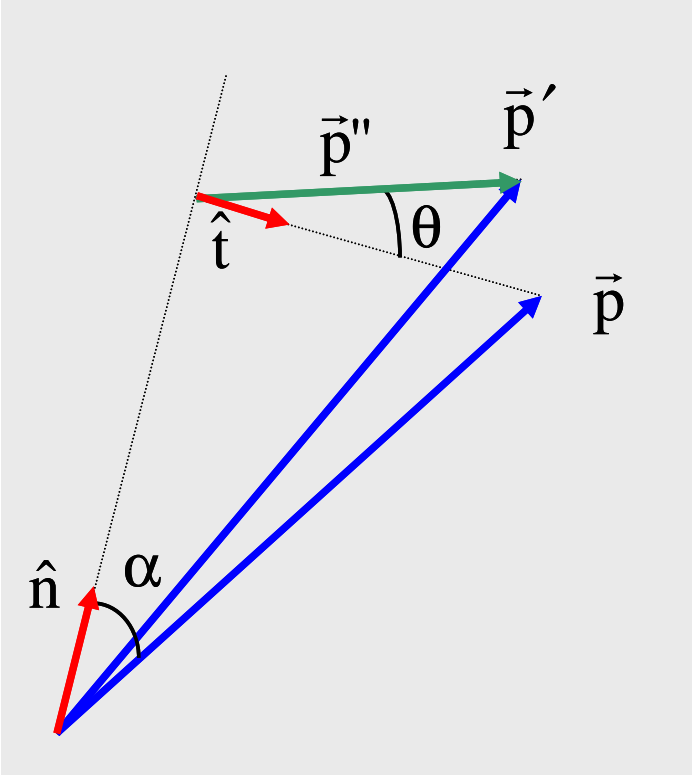
```
D3DXMatrixRotationYawPitchRoll( D3DXMATRIX *pOut, FLOAT Yaw, FLOAT Pitch, FLOAT Roll );
```

행렬의 연산 시간에서 두 행렬의 곱은 교환 법칙이 성립되지 않는다고 했습니다.(M*N≠N*M)
회전 행렬도 행렬이므로 행렬의 곱하기 순서에 따라 그 결과가 달라집니다. 예를 들면 위에서 Yaw->Pitch->Roll순으로 행렬을 곱했는데 이 순서가 다르면 그 결과고 달라집니다.
오일러 앵글이 적용하기가 쉽지만 모든 오브젝트가 이 방법대로 움직이지 않는다는데 문제가 발생하고 이것을 짐벌락(Gimbal Lock)이라 합니다.
짐벌락 문제는 회전에서 그 곱의 순서를 알고 있지 않다면 전혀 다른 행렬을 만들어 다른 위치로

이동 할 수 있다는 것이며 실제로 인공 위성이나 탐사 로켓 등에서와 같이 하늘에 떠 있는 물체들과 같은 제어 장치에서 많이 연구 되고 있습니다.
게임에서도 종종 회전하는 물체에 자주 나타나는데 이 짐벌락 문제를 해결하기 위해서는 회전 각도를 누적시키는 것이 아니라 임의의 축에 대해서 회전 행렬을 만들고 이 회전 행렬을 계속 곱해야만 해결이 됩니다.

2.4.3 임의의 축에 대한 회전

다음 그림과 같이 임의 점 $\vec{p}$가 축 $\hat{n}$에 대해서 θ만큼 회전한 점 $\vec{p}'$을 구해 봅시다.



<임의의 축에 대한 정점의 회전>

회전의 결과인 $\vec{p}'$는 다음과 같이 $\vec{p}$, $\hat{n}$, $\vec{p}''$으로 나타낼 수 있습니다.

$$\vec{p}' = (\vec{p} \cdot \hat{n})\hat{n} + \vec{p}''$$

또한 $\vec{p}''$은 위의 그림을 이용해 $\vec{p}$와 $\hat{n}$의 조합으로 만들 수 있습니다.

$$\vec{p}'' = \|\vec{p}''\| \cos\theta \hat{t} + \|\vec{p}''\| \sin\theta \frac{(\hat{n} \times \vec{p})}{\|\hat{n} \times \vec{p}\|}$$

$$\|\vec{p}''\| = \|\vec{p}\| \sin\alpha$$
$$\|\vec{p}\| \sin\alpha = \|\hat{n} \times \vec{p}\|$$

이므로

$$\vec{p}'' = \|\vec{p}''\| \cos\theta \hat{t} + \sin\theta * (\hat{n} \times \vec{p})$$
$$\|\vec{p}''\| \hat{t} = \vec{p} - (\vec{p} \cdot \hat{n})\hat{n}$$
$$\vec{p}'' = \cos\theta(\vec{p} - (\vec{p} \cdot \hat{n})\hat{n}) + \sin\theta(\hat{n} \times \vec{p})$$

으로 $\vec{p}''$을 다음과 같이 구합니다.

$$\vec{p}'' = \|\vec{p}''\| \cos\theta \hat{t} + \sin\theta * (\hat{n} \times \vec{p})$$
$$\|\vec{p}''\| \hat{t} = \vec{p} - (\vec{p} \cdot \hat{n})\hat{n}$$
$$\vec{p}'' = \cos\theta(\vec{p} - (\vec{p} \cdot \hat{n})\hat{n}) + \sin\theta(\hat{n} \times \vec{p})$$

이제 나머지 계산을 정리하면 됩니다.

$$\vec{p}' = (\vec{p} \cdot \hat{n})\hat{n} + \cos\theta(\vec{p} - (\vec{p} \cdot \hat{n})\hat{n}) + \sin\theta(\hat{n} \times \vec{p})$$

$$\therefore \vec{p}' = \cos\theta\vec{p} + \sin\theta(\hat{n} \times \vec{p}) + (1 - \cos\theta)(\hat{n} \cdot \vec{p})\hat{n}$$

수식이 길지만 임의의 축에 대한 회전은 다음 공식으로 구할 수 있게 됩니다.

$$\vec{p}' = \cos\theta\vec{p} + \sin\theta(\hat{n} \times \vec{p}) + (1 - \cos\theta)(\hat{n} \cdot \vec{p})\hat{n}$$

<임의의 축에 대한 회전>

다음 단계로 위의 공식에서 회전 행렬을 만들어야 합니다. 공식을 보면 일이 많아질 수 있음을 느낌으로 알 수 있습니다. 위의 공식에서 직접 회전 행렬을 구할 수 있지만 다음에 소개될 사원수를 이용해 공식을 다시 유도할 수 있고 이 사원수로 회전 행렬을 얻는 것이 좋습니다.

DXSDK에서는 임의 축에 대한 회전 행렬을 구해주는 다음과 같은 함수가 있습니다.

```
D3DXMATRIX* WINAPI D3DXMatrixRotationAxis
    ( D3DXMATRIX *pOut, CONST D3DXVECTOR3 *pV, FLOAT Angle );
```

2.5 사원수(Rotation)

2.5.1 사원수

아일랜드 태생의 천재 수학자 윌리엄 로원 해밀턴(William Rowan Hamilton)이 발명한 사원수(Quaternion)은 복소수(Complex Number) 표현을 확장하기 위해 만들었습니다. 최근까지 벡터 해석보다 사용이 난해하여 비주류로 밀려났다가 최근에 컴퓨터 시뮬레이션에서 다시 각광 받기 시작했고 3D 프로그램에서는 회전에서 자주 사용되고 있습니다.

사원수를 배우기 전에 명심할 것이 있습니다. 그것은 사원수를 사용하지 않고 벡터만 가지고도 임의 축에 대한 회전을 구할 수 있습니다. 그런데 사원수를 사용하는 것은 벡터로 구한 결과보다 간결하고, C++의 연산자 다중정의(Overloading)을 적용하면 프로그램을 구현하기가 쉽기 때문입니다.

사원수의 시작은 복소수부터 시작합니다. 복소수는 실수(Real Number)와 허수(Imaginary Number)의 결합임을 중고등학교 수학시간에 이미 배웠습니다.

EX) 복소수 예. 2+3i, a + bi,

사원수는 복소수를 연장해서 하나의 실수와 3개의 허수로 구성된 형태입니다.
다음은 사원수 $\hat{q}$를 여러 방법들로 표현한 것입니다.

$$\hat{q} = w + x\hat{i} + y\hat{j} + z\hat{k} = \cos\theta + \sin\theta\hat{v} = [x, y, z, w] = (s, \hat{v})$$
$$s = w$$
$$\hat{v} = [x, y, z]$$

여기서 $\hat{v}$는 회전 축 단위 벡터(방향 벡터)입니다. $\hat{i},\ \hat{j},\ \hat{k}$는 좌표 축 단위벡터이고 $x\hat{i} + y\hat{j} + z\hat{k}$

는 $\sin\theta * \hat{v}$와 같습니다. 벡터의 좌표 축과 다르게 $\hat{i},\ \hat{j},\ \hat{k}$ 벡터는 곱셈 연산자 "*"가 있습니다.

이 연산자의 성질은 다음과 같이 정의합니다.

$$\hat{i}*\hat{i} = -1 \qquad \hat{i}*\hat{j} = -\hat{j}*\hat{i} = \hat{k}$$
$$\hat{j}*\hat{j} = -1 \qquad \hat{j}*\hat{k} = -\hat{k}*\hat{j} = \hat{i}$$
$$\hat{k}*\hat{k} = -1 \qquad \hat{k}*\hat{i} = -\hat{i}*\hat{k} = \hat{j}$$

자신끼리 곱셈을 하면 스칼라 -1이 되고 다른 벡터와 곱셈을 하면 Cyclic Order로 또 다른 벡터를 만들어 냅니다.

복소수와 마찬가지로 사원수도 복소 공액(Conjugate)가 있습니다. 이 복소 공액으로 사원수의 크

기를 구할 수 있습니다.

$$\hat{q} = w + x\hat{i} + y\hat{j} + z\hat{k},$$
$$\hat{q}' = \text{Conjuage}(\hat{q}) = w - x\hat{i} - y\hat{j} - z\hat{k}$$

다음은 사원수와 이의 복소 공액으로 사원수의 크기를 구하는 방법입니다.

$$\|\hat{q}\| = \sqrt{\hat{q} * \hat{q}'} = \sqrt{w^2 + x^2 + y^2 + z^2}$$

사원수의 성질을 이용해서 임의의 사원수 $\hat{q}_1 = (s_1, \hat{v}_1)$, $\hat{q}_2 = (s_2, \hat{v}_2)$ 다음과 같을 때 곱셈을 정리해 봅시다.

$$\hat{q}_1 = (s_1, \hat{v}_1),\ \hat{q}_2 = (s_2, \hat{v}_2)$$

$$\hat{q}_1 * \hat{q}_2 = (s_1, \hat{v}_1) * (s_2, \hat{v}_2)$$
$$= s_1 s_2 + s_1 * \hat{v}_2 + s_2 * \hat{v}_1 + \hat{v}_1 * \hat{v}_2$$

다음으로 $\hat{v}_1$와 $\hat{v}_2$를 벡터 성분으로 나누어서 $\hat{v}_1 * \hat{v}_2$를 계산합니다.

$$= s_1 s_2 + s_1 * \hat{v}_2 + s_2 * \hat{v}_1 + (\hat{v}_{1x}\hat{i} + \hat{v}_{1y}\hat{j} + \hat{v}_{1z}\hat{k}) * (\hat{v}_{2x}\hat{i} + \hat{v}_{2y}\hat{j} + \hat{v}_{2z}\hat{k})$$
$$= s_1 s_2 + s_1 * \hat{v}_2 + s_2 * \hat{v}_1 - (\hat{v}_{1x} * \hat{v}_{2x} + \hat{v}_{1y} * \hat{v}_{2y} + \hat{v}_{1z} * \hat{v}_{2y})$$
$$+ (\hat{v}_{1y} * \hat{v}_{2z} - \hat{v}_{1z} * \hat{v}_{2y})\hat{i} + (\hat{v}_{1z} * \hat{v}_{2x} - \hat{v}_{1x} * \hat{v}_{2z})\hat{j} + (\hat{v}_{1x} * \hat{v}_{2y} - \hat{v}_{1y} * \hat{v}_{2x})\hat{k}$$

실수끼리, 벡터끼리 묶고 내적, 외적으로 식을 간소화 시켜 두 사원수의 곱셈을 정리합니다.

$$= s_1 s_2 - \hat{v}_1 \cdot \hat{v}_2 + s_1 * \hat{v}_2 + s_2 * \hat{v}_1 + \hat{v}_1 \times \hat{v}_2$$

$$\therefore \hat{q}_1 * \hat{q}_2 = (s_1 s_2 - \hat{v}_1 \cdot \hat{v}_2, s_1 \hat{v}_2 + s_2 \hat{v}_1 + \hat{v}_1 \times \hat{v}_2)$$

이 식의 마지막 부분에 벡터의 외적이 포함되고 있습니다. 이로서 벡터의 외적은 순서가 바뀌면 방향이 반대가 됩니다. 따라서 위의 식만으로도 우리는 사원수 사이의 곱셈에 대한 교환 법칙은 성립하지 않음을 직관적으로 알 수 있습니다.

사원수 사이의 곱은 결합법칙만 가능하고 교환법칙은 성립이 안됨을 기억하기 바랍니다. 다음은 교환 법칙과 더불어 사원수의 성질을 정리한 것입니다.

사원수의 결합 법칙: $\hat{q}_1 * \hat{q}_2 * \hat{q}_3 = (\hat{q}_1 * \hat{q}_2) * \hat{q}_3 = \hat{q}_1 * (\hat{q}_2 * \hat{q}_3)$

사원수의 교환 법칙(성립 안됨): $\hat{q}_1 * \hat{q}_2 \neq \hat{q}_2 * \hat{q}_1$

단위 사원수의 크기: $\|\hat{q}\| = 1$

단위 사원수의 역수: $\hat{q}' = \hat{q}^{-1}$ ← 단위 사원수 복소 공액과 단위 사원수의 역수는 같다.

2.5.2 사원수와 회전

각도가 $\theta/2$ 인 단위 사원수 $\hat{q}(\theta/2, \hat{v})$ 와, 실수가 0인 임의의 사원수 $\vec{p}(0, \hat{v})$ 에 대해서 곱셈 $\vec{p}' = q * \vec{p} * q^{-1}$ 을 구해 봅시다.

$$\vec{p}' = q * \vec{p} * q^{-1}$$

$$= \left(\cos\frac{\theta}{2},\ u\sin\frac{\theta}{2}\right) * (0, \vec{v}) * \left(\cos\frac{\theta}{2},\ -\hat{u}\sin\frac{\theta}{2}\right)$$

$$= \left(-\sin\frac{\theta}{2}(\hat{u}\cdot\vec{v}), \cos\frac{\theta}{2}\vec{v} + \sin\frac{\theta}{2}(\hat{u}\times\vec{v})\right) * \left(\cos\frac{\theta}{2}, -\hat{u}\sin\frac{\theta}{2}\right)$$

$$= \left(-\sin\frac{\theta}{2}(\hat{u}\cdot\vec{v}), \cos\frac{\theta}{2}\vec{v} + \sin\frac{\theta}{2}(\hat{u}\times\vec{v})\right) * \left(\cos\frac{\theta}{2}, -\hat{u}\sin\frac{\theta}{2}\right)$$

$$= \begin{pmatrix} -\sin\frac{\theta}{2}\cos\frac{\theta}{2}(\hat{u}\cdot\vec{v}) + \sin\frac{\theta}{2}\cos\frac{\theta}{2}(\vec{v}\cdot\hat{u}) - \sin^2\frac{\theta}{2}(\hat{u}\times\vec{v})\cdot\hat{u}, \\ \sin^2\frac{\theta}{2}(\hat{u}\cdot\vec{v})\hat{u} + \cos^2\frac{\theta}{2}\vec{v} + \sin\frac{\theta}{2}\cos\frac{\theta}{2}(\hat{u}\times\vec{v}) \\ -\sin\frac{\theta}{2}\cos\frac{\theta}{2}(\vec{v}\times\hat{u}) - \sin^2\frac{\theta}{2}(\hat{u}\times\vec{v})\times\hat{u} \end{pmatrix}$$

$$= \sin^2\frac{\theta}{2}(\hat{u}\cdot\vec{v})\hat{u} + \cos^2\frac{\theta}{2}\vec{v} + \sin\theta(\hat{u}\times\vec{v}) + \sin^2\frac{\theta}{2}\hat{u}\times(\hat{u}\times\vec{v})$$

$$= \sin^2\frac{\theta}{2}(\hat{u}\cdot\vec{v})\hat{u} + \cos^2\frac{\theta}{2}\vec{v} + \sin\theta(\hat{u}\times\vec{v}) + \sin^2\frac{\theta}{2}(\hat{u}\cdot\vec{v})\hat{u} - \sin^2\frac{\theta}{2}\vec{v}$$

$$= (\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2})\vec{v} + \sin\theta(\hat{u}\times\vec{v}) + 2\sin^2\frac{\theta}{2}(\hat{u}\cdot\vec{v})\hat{u}$$

$$\therefore \vec{p}' = \cos\theta\vec{v} + \sin\theta(\hat{u}\times\vec{v}) + (1-\cos\theta)(\hat{u}\cdot\vec{v})\hat{u}$$

이 공식은 2.4.3의 임의 축에 대한 회전에서 얻은 것과 동일합니다. 이것은 임의의 축에 대한 회전을 간단하게 사원수의 곱셈 $q * \vec{p} * q^{-1}$ 으로 표현이 된다는 것입니다. 물론 단위 사원수의 회전각

은 절반으로 해야 합니다.
만약 여러분이 사원수 클래스를 만들고 여기에 앞서 정리한 사원수 사이의 곱셈을 연산자 다중 정의로 구현했다면 회전을 간단하게 한 줄의 코드: $q*\vec{p}*q^{-1}$ 으로 구할 수 있음을 의미합니다.

이 것을 다음과 같이 정리할 수 있습니다.

<사원수를 이용한 회전>

1. 임의 축(Axis)과 각도를 사원수로 바꾼다. 이 때 각도는 ½배 한다.
   q = (sin(Θ/2)*AxisX, sin(Θ/2)*AxisY, sin(Θ/2)*AxisZ, cos(Θ/2))
   q-1 = q(-q.x, -q.y, -q.z, q.w)

2. 회전을 적용할 정점을 사원수로 바꾼다
   P=(Px, Py, Pz , 0)

3. 사원수 곱 q*P*q-1 연산을 수행한다(D3D 왼손 좌표라 q-1*P*q 순으로 곱한다.)

4. 최종 연산의 결과도 사원수 이므로 이중에서 허수 부를 선택한다

다음은 <사원수를 이용한 회전>을 이용해서 이전 벡터 해석과 결과를 비교하기 위한 코드 입니다.

```
#pragma comment(lib, "d3dx9.lib")
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include <d3dx9.h>

void main()
{
	for(int i=0; i<100; ++i)
	{
		D3DXMATRIX		mtT;

		// 임의의 축을 만든다.
		D3DXVECTOR3		vAxis(1.f + rand()%100
			, 1.f + rand()%100
			, 1.f + rand()%100);
```

```
// 임의의 각도를 만든다.
FLOAT fAngle = D3DXToRadian( (1.f+rand()%100) * 3.f);

// 임의의 위치를 만든다.
D3DXVECTOR3            Ps(1.f + rand()%100
                        , 1.f + rand()%100
                        , 1.f + rand()%100);

// 변환 후 벡터
D3DXVECTOR3            Po;

// 임의의 축 벡터를 정규화 한다.
D3DXVec3Normalize(&vAxis, &vAxis);

////////////////////////////////////////////////////////////////
// 벡터 해석을 통한 결과
D3DXMatrixRotationAxis(&mtT, &vAxis, fAngle );
D3DXVec3TransformCoord(&Po, &Ps, &mtT);
printf("%f %f %f₩n", Po.x, Po.y, Po.z);
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
// 다음 코드들은 <사원수를 이용한 회전> 방법이다.
// 사원수에 적용하기 위해 각도를 절반으로 하고 Sin, Cos를 구한다.
float   fCos = cosf(fAngle/2);
float   fSin = sinf(fAngle/2);

// 단위 사원수를 구한다.
D3DXQUATERNION  q(  fSin* vAxis.x,  fSin* vAxis.y,  fSin* vAxis.z, fCos);
// 단위 사원수의 역수를 구한다.
D3DXQUATERNION  qi(-q.x, -q.y, -q.z, q.w);
// 사원수에 변환할 벡터를 복사한다.w=0이다.
D3DXQUATERNION  P(Ps.x, Ps.y, Ps.z, 0);
// 사원수 곱셈을 진행한다. D3DSDK는 왼손 좌표계를 따르므로
// 순서를 뒤집는다.q * P * qi를 뒤집는다.
P = qi * P * q;
printf("%f %f %f₩n", P.x, P.y, P.z);
```

```
            ////////////////////////////////////////////////////////////
            // 사원수에서 얻은 행렬로 변환해서 출력해 본다.
            D3DXMatrixRotationQuaternion(&mtT, &q);
            D3DXVec3TransformCoord(&Po, &Ps, &mtT);
            printf("%f %f %fWnWn", Po.x, Po.y, Po.z);
            ////////////////////////////////////////////////////////////
        }
}
```

임의의 축이 여러 개 일 때 회전을 구하는 것은 다음과 같습니다.

만약 q_1, q_2 순서대로 회전이 적용되는 경우를 생각해 봅시다. 먼저 $\vec{P} = q_1 * \vec{P} * q_1^{-1}$ 를 계산한 후 $\vec{P} = q_2 * \vec{P} * q_2^{-1}$ 를 계산할 수 있습니다. 그런데 이것은 다음과 같이 식으로 전개가 가능합니다.

$$
\begin{aligned}
& q_2 * (q_1 * \vec{P} * q_1^{-1}) * q_2^{-1} \\
&= (q_2 * q_1) * \vec{P} * (q_1^{-1} * q_2^{-1}) \\
&= (q_2 * q_1) * \vec{P} * (q_2 * q_1)^{-1}
\end{aligned}
$$

이것은 $q = q_2 * q_1$ 을 먼저 계산한 후 이의 공액을 구하고 나서 $q * \vec{P} * q^{-1}$ 을 계산하는 것과 같이 됩니다. 즉, 사원수의 쌍들을 미리 곱한 후에 계산하게 되어 속도에 이득이 생깁니다.

이것을 정리하면 다음과 같습니다.

1. 사원수의 곱셈을 회전 적용 역순으로 곱한다. $q = q_n * \cdots * q_2 * q_1$

2. 1에서 구한 사원수의 역수를 복소 공액으로 구한다. $q^{-1} = q^*$

3. 회전에 적용한다. $q * \vec{P} * q^{-1}$

2.5.3 사원수에서 회전 행렬 얻기

사원수[x, y, z, w]에서 회전을 얻는 과정입니다. 벡터와 행렬이 익숙하지 않은 분들은 그냥 결과 정도만 확인하기 바랍니다.

$$\vec{p}_{rotated} = q * \vec{p} * q^{-1}$$
$$= \cos\theta\vec{v} + \sin\theta(\hat{u}\times\vec{v}) + (1-\cos\theta)\hat{u}(\hat{u}\cdot\vec{v})$$

$$= \left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right)\vec{I}\vec{v}$$

$$+ 2\cos\frac{\theta}{2}\begin{bmatrix} 0 & -\sin\frac{\theta}{2}\hat{u}_z & \sin\frac{\theta}{2}\hat{u}_y \\ \sin\frac{\theta}{2}\hat{u}_z & 0 & -\sin\frac{\theta}{2}\hat{u}_x \\ -\sin\frac{\theta}{2}\hat{u}_y & \sin\frac{\theta}{2}\hat{u}_x & 0 \end{bmatrix}\vec{v}$$

$$+ 2\left(\sin\frac{\theta}{2}\hat{u}\right)\left(\sin\frac{\theta}{2}\hat{u}\right)\cdot\vec{v}$$

$$= \begin{bmatrix} w^2-x^2-y^2-z^2 & 0 & 0 \\ 0 & w^2-x^2-y^2-z^2 & 0 \\ 0 & 0 & w^2-x^2-y^2-z^2 \end{bmatrix}\vec{v}$$

$$+ \begin{bmatrix} 0 & -2wz & 2wy \\ 2wz & 0 & -2wx \\ -2wy & 2wx & 0 \end{bmatrix}\vec{v} + \begin{bmatrix} 2x^2 & 2xy & 2xz \\ 2wz & 2y^2 & 2yz \\ 2xz & 2yz & 2z^2 \end{bmatrix}\vec{v}$$

$$= \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}v$$

$$\therefore R_q = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

다행히도 DXSDK에서 D3DXMatrixRotationQuaternion() 함수를 지원해서 사원수에서 행렬을 얻어낼 수 있습니다. 이 함수의 사용법은 앞서 보여주었던 <사원수를 이용한 회전>의 마지막 단계에 있으니 참고 하기 바랍니다

2.6 보간(Interpolation)

2.6.1 선형 보간

보간(補間)은 이미 알고 있는 값을 가지고 측정하지 않거나 못한 값을 추정하는 방법으로 한자를 그대로 직역하면 사이의 틈을 메우는 것이라 할 수 있겠습니다. 보간은 다항식 함수의 계수를 근사(Approximation)해서 구합니다.

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots a_n x^n = \sum a_n x^n$$

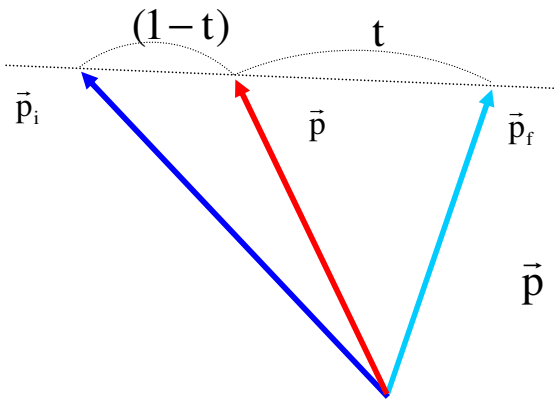
이 방정식 중에서 특별히 $f(x) = a_0 + a_1 x$ 로 만들어진 보간을 선형 보간이라 하며 선형 보간은 가장 많이 사용하는 보간입니다.

선형 보간의 좀 더 자세한 표현은 다음과 같습니다.

보간 값: $p = (1-t) * p_i + t * p_f$, or $p = p_i + t * (p_f - p_i)$

p_i: 시작, p_f :끝, t: Weight(비중)

선형 보간을 벡터에 적용하면 다음 그림처럼 보간 값은 두 벡터가 만드는 직선 위의 위치가 됩니다.

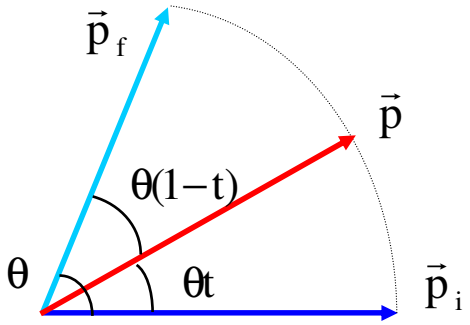


$$\vec{p} = (1-t) * \vec{p}_i + t * \vec{p}_f$$

<벡터에 대한 선형 보간>

2.6.2 회전에 대한 선형 보간

앞의 선형 보간을 회전에도 적용해 봅시다. 회전에서의 선형 보간은 각도에 대한 선형 보간과 같은 의미이며 다음 그림처럼 표현할 수 있습니다.



<각도에 대한 선형 보간>

점 $\vec{p}$ 를 구하기 위해서 다음과 같은 수식을 만들 수 있습니다.

$$\theta = \theta(1-t) + \theta t$$

$$\vec{p} = a \cdot \vec{p}_i + b \cdot \vec{p}_f$$

여기서 a, b는 미지의 수입니다.

위의 두 식을 가지고 $\vec{p}_i$, $\vec{p}_f$ 를 $\vec{p}$ 에 번갈아 외적을 적용하면 다음과 같이 Sine함수를 얻게 됩니다.

$$\vec{p}_i \times \vec{p} = a \cdot \vec{p}_i \times \vec{p}_i + b \cdot \vec{p}_i \times \vec{p}_f$$

$$\sin\theta t \cdot \hat{n} = a \cdot \vec{0} + b \sin\theta \cdot \hat{n}$$

양변에 벡터 $\hat{n}$ 으로 내적을 적용합니다.

$$\sin\theta t \cdot (\hat{n} \bullet \hat{n}) = a \cdot \vec{0} + b \sin\theta \cdot (\hat{n} \bullet \hat{n})$$

$\hat{n}$ 은 단위벡터이므로 $(\hat{n} \bullet \hat{n})$=1이 되며 위의 식은 다음과 같이 정리되어 미지수 b를 구하게 됩니다.

$$\sin\theta t = b \sin\theta$$

$$\therefore b = \frac{\sin\theta t}{\sin\theta}$$

b를 구한 것처럼 a도 같은 방식으로 구할 수 있습니다.

$$\vec{p} \times \vec{p}_f = a \cdot \vec{p}_i \times \vec{p}_f + b \cdot \vec{p}_f \times \vec{p}_f$$

$$\sin\theta(1-t) = a \sin\theta$$

$$\therefore a = \frac{\sin\theta(1-t)}{\sin\theta}$$

미지수 a, b를 구했으므로 점 $\vec{p}$ 의 보간 위치를 구할 수 있습니다.

$$\therefore \vec{p} = \frac{1}{\sin\theta}\left[\sin\theta(1-t)\ \vec{p}_i + \sin(\theta t)\ \vec{p}_f\right]$$

Θ ~ 0 이면 $\sin\theta \approx \theta, \sin\theta t \approx \theta t, \sin\theta(1-t) \approx \theta(1-t)$ 이 되어 위의 점 $\vec{p}$ 는 선형보간에서 구한 결과와 일치합니다.

$$\therefore \vec{p} = (1-t) \cdot \vec{p}_i + t \cdot \vec{p}_f$$

즉, 각도가 0에 가까우면 위치에 대한 선형 보간을 적용하는 것이 속도에 이득이 됩니다.

2.7 변환 정리

크기 변환, 회전, 이동 변환을 동시에 적용할 경우에 가장 일반적인 방법은 다음과 같은 순서대로 변환을 진행합니다.

1. 회전 중심점으로 상대적으로 이동하는 행렬을 구한다. **T'**
2. 크기 변환 행렬을 구한다. **S**
3. 회전 변환 행렬을 구한다 **R**
4. 다시 원래의 위치로 이동하는 행렬을 구한다. **T**
5. 최종 행렬은 1~4까지 구한 행렬의 곱으로 만든다. **M = T'SRT**

게임에서 사용되는 3차원 물체들은 원점을 중심으로 작업을 합니다. 따라서 위의 과정 중에서 1번을 생략해서 최종 행렬을 다음과 같이 만듭니다.

변환 최종 행렬: **M = SRT**

변환 최종 행렬에 정점을 곱하면 변환된 정점의 위치가 구해집니다.
또한 이 행렬은 다음과 모습을 합니다.

$$\vec{\overleftrightarrow{M}} = \begin{bmatrix} S_x*R_{11} & S_x*R_{12} & S_x*R_{13} & 0 \\ S_y*R_{21} & S_y*R_{22} & S_y*R_{23} & 0 \\ S_z*R_{31} & S_z*R_{32} & S_z*R_{33} & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

<최종 변환 행렬>

이 행렬을 주의 깊게 보면 행렬의 원소 14, 14, 34, 는 항상 0이고 마지막 44는 1입니다. 또한 41, 42, 43 은 이동의 정보가 있고, 이것은 곧 이 오브젝트의 위치를 의미합니다.
이런 것들을 알고 있으면 좀 더 빠르게 변환 행렬을 설정하거나 정보를 얻는데 이점이 있습니다.

DXSDK는 정점 변환에 대해서 다음과 같은 두 종류의 함수를 지원합니다.
D3DXVec3TransformCoord(): 이 함수는 w=1로 놓고 변환하는 함수입니다.
이 함수는 크기+회전+이동 변환이 진행 됩니다.
D3DXVec3TransformNormal(): 이 함수는 w=0로 놓고 정점을 변환 하며 회전과 크기만 변환됩니다.

함수 이름에 있는 Normal로부터 이 함수의 목적은 법선 벡터를 변환하는데 있음을 짐작할 수 있습니다.

D3DXVec3TransformCoord() 함수 사용은 <사원수를 이용한 회전> 예제 코드에 나와 있으니 참고하기 바랍니다.

위의 두 함수와 자신이 만든 변환 함수의 비교를 할 때 위의 함수는 "V' = V * Matrix" 인 벡터 * 행렬 연산을 하는 함수임을 기억하기 바랍니다.

지금까지 소개한 변환은 아핀(Affine) 변환의 한 종류 입니다. 아핀 변환의 일반적인 표현은 다음과 같습니다.

$$\vec{T}' = \sum \vec{M}_{ij}\vec{T}_j + \vec{V}_i$$

3D 프로그램과 그래픽 파이프라인의 모든 변환은 전부 아핀 변환입니다. 이 아핀 변환의 특징 중에 하나는 이전의 특성을 그대로 유지합니다. 예를 들어 평행선은 변환 후 평행선에 대응하고, 유한한 점은 이전의 유한 점에 대응합니다.
이동, 회전, 크기, 대칭, 밀림(Shear) 변환은 아핀 변환의 한 종류입니다. 아핀 변환은 동차 좌표를 적용해서 여러 복합 변환을 하나의 행렬로 표현할 수 있습니다.

과제1) 벡터와 행렬 사이의 연산을 Operator Overloading을 이용해서 구현하시오.

과제2) 크기, 이동, X축, Y축, Z축에 대한 회전 변환 행렬을 구현하시오. 구현의 결과가 DXSDK 함수와 비슷한지 비교해보시오.

2.8 벡터와 동차 좌표(Homogeneous Coordinate)

이장은 보충 설명이라 넘어가도 됩니다.

2.8.1 동차 좌표

1차원 좌표에서 (2), (4)가 있다고 가정 합시다. 이 둘은 당연히 같지 않습니다. 그런데 이것을 2차원으로 확장하는 동차 좌표계를 사용하면 (2) ->(2, 1), (4) ->(4,1)로 변경 될 수 있습니다. 새롭게 추가된 좌표의 값은 일반적으로 값은 1로 유지합니다. 이렇게 좌표계를 확장하더라도 원래의 좌표 값을 그대로 유지 시켜주는 것이 동차 좌표입니다.
그런데 동차 좌표계를 사용할 경우 주의해야 합니다. 예를 들어 2차원 동차좌표계에서 (2,1) ≡ (4, 2) 가 되고, 또한 (2,1)과 같은 좌표 값은 동차 좌표계에서는 무수히 많은 좌표 값을 가질 수 있습니다. 2차원의 경우 이러한 내용을 일반화 시키면 (a, b) ≡ (a/b, 1)로 표현할 수 있습니다. 예를 들면 (2,4) ≡ (3, 6) ≡ (1.5, 3) ≡ (0.5, 1)이 가능합니다.

3차원은 2차원의 확장입니다. 그러므로 2차원 좌표(x, y)를 3차원 동차 좌표계의 좌표로 바꾸면 (x, y, 1)이 되고 이것을 일반화 하면 다음과 같이 됩니다.

(C *x, C* y, C)(단, C ≠ 0)

3차원 벡터를 4차원 동차 좌표계로 확장 할 수 있습니다. 위의 과정을 그대로 적용하면 3차원 좌표에 대한 4차원 동차 좌표계 좌표는 다음과 같이 변환 됩니다.

(x, y, z) ->(x, y, z, 1) ≡ (w * x, w * y, w * z, w)

4차원 동차좌표계에서는 C 대신 w를 사용하고, 입력한 벡터의 연산의 마지막에 이 값으로 x 성분, y 성분, z 성분을 나누어 원래 차원의 크기로 바꾸어 주는데 사용됩니다. 이 때 w 값이 이용되는 과정을 요약하면 다음과 같습니다.

V(x, y, z) ->동차 좌표계로 변환 (x, y, z, 1)
 -> 행렬 변환 (x', y', z', w')
 -> 원 좌표계의 크기로 환원 (x'/w', y'/w', z'/w', w'/w')
 -> 최종 (x'/w', y'/w', z'/w')

컴퓨터 그래픽스에서는 이 w를 특별히 Reciprocal of Homogeneous W 라 부르며 줄여서 RHW 라 합니다.

문제 1-1)(2,3) 과 같은 3차원 동차 좌표계의 좌표 3개만 구하시오.

문제 1-2)(3,4,2) 와 같은 4차원 동차 좌표계의 좌표 3개만 구하시오.

풀이 1-1) (4,6), (-2, -3), (200, 3) 등
풀이 1-2) (-3, -4, -2), (-1.5, -2, -1) (6, 8, 4) 등

2.8.2 동차 좌표와 행렬

1) 벡터 $\vec{v} = \vec{v}$에 대한 식을 $\vec{v} = \vec{v}\,\mathbf{M}$ 을 만드는 행렬 M을 동차 좌표계에서 구하도록 합시다.
행렬을 공부한 사람은 위의 수식에 대한 행렬 M은 3X3 항등 행렬 임을 알고 있을 것입니다. 그러나 우리가 구하고자 하는 것은 동차좌표계에서의 행렬이므로 먼저 주어진 벡터를 동차좌표계의 좌표로 바꾸어 주어야 합니다.

벡터 $\vec{v} = (x, y, z)$라 합시다. 이것을 4차원 동차 좌표계로 바꾸면 다음과 같이 됩니다.

(x, y, z) -> $(x, y, z, 1)$
≡ $(w*x, w*y, w*z, w)$

이것을 행렬로 나타내면

$$(x, y, z, 1) = (x, y, z, 1)\begin{pmatrix} w & 0 & 0 & 0 \\ 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & w \end{pmatrix}$$

이 되고, 동차 좌표계에서 행렬 M = $\begin{pmatrix} w & 0 & 0 & 0 \\ 0 & w & 0 & 0 \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & w \end{pmatrix}$ 이 됩니다.

일반 좌표계에서 항등 행렬은 거칠게 표현 한다면 동차 좌표계에서는 대각선의 값이 같고, 나머지가 0인 행렬의 특별한 값이 되는 특별한 행렬일 뿐입니다.

2) $\vec{v} = -\vec{v}$ 에 대한 식을 $\vec{v} = \vec{v}\,\mathbf{M}$ 을 만드는 행렬 M을 동차 좌표계에서 구해 봅시다.

벡터 $\vec{v} = (x, y, z, 1)$라 하면 벡터 $-\vec{v} = (-x, -y, -z)$가 됩니다. 주어진 문제의 식을 4차원 동차

좌표계로 바꾸면 다음과 같이 됩니다.

$-\vec{v} = (-x, -y, -z, 1)$

이것을 $\vec{v} = \vec{v}\,M$ 식으로 변경하면 다음과 같이 됩니다.

$$(-x, -y, -z, 1) = (x, y, z, 1)\begin{pmatrix} -w & 0 & 0 & 0 \\ 0 & -w & 0 & 0 \\ 0 & 0 & -w & 0 \\ 0 & 0 & 0 & w \end{pmatrix}$$

따라서 행렬 M=$\begin{pmatrix} -w & 0 & 0 & 0 \\ 0 & -w & 0 & 0 \\ 0 & 0 & -w & 0 \\ 0 & 0 & 0 & w \end{pmatrix}$ 가 됩니다.

3) $\vec{v} = \vec{c}$ 에 대한 식을 $\vec{v} = \vec{v}\,M_c$ 을 만드는 벡터 c의 성분으로 구성된 행렬 M을 동차 좌표계에서 구해 봅시다.

벡터 $\vec{v} = (v_x, v_y, v_z, 1)$ 라 하고 $\vec{c} = (c_x, c_y, c_z, 1)$ 라 하면

$$(v_x, v_y, v_z, 1) = (c_x, c_y, c_z, 1)$$

$$= (v_x, v_y, v_z, 1)\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ w*c_x & w*c_y & w*c_z & w \end{pmatrix}$$

따라서 행렬 M_c=$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ w*c_x & w*c_y & w*c_z & w \end{pmatrix}$ 가 됩니다.

4) $\vec{v} = -\vec{c}$ 에 대한 식을 $\vec{v} = \vec{v}\,M_c$ 을 만드는 벡터 c의 성분으로 구성된 행렬 M을 동차 좌표계에

서 구해 보면 3)의 경우와 같으므로

벡터 $\vec{v} = (v_x, v_y, v_z, 1)$ 라 하고 $\vec{c} = (c_x, c_y, c_z, 1)$ 라 하면

$$(v_x, v_y, v_z, 1) = (-c_x, -c_y, -c_z, 1)$$

$$= (v_x, v_y, v_z, 1)\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -w*c_x & -w*c_y & -w*c_z & w \end{pmatrix}$$

따라서 행렬 $M_c = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -w*c_x & -w*c_y & -w*c_z & w \end{pmatrix}$ 가 됩니다.

주의할 것은 w 값을 아직 1로 만들지 않았다는 것입니다. 이 것은 동차좌표계의 행렬이기 때문이다. 이것을 실제로 적용하려면 위의 행렬을 w값으로 나누어야 할 것입니다.

문제1-3) $\vec{A}, \vec{B}, \vec{C}$ 가 3차원 벡터일 때 $\vec{C} = \vec{A} + \vec{B}$ 를 $\vec{C} = \vec{A}\, M_B$ 으로 나타내고자 할 때 벡터 $\vec{B}$ 의 성분으로 구성된 행렬 M_B 을 4차원 동차좌표계에서 구하시오.

문제1-4) $\vec{A}, \vec{B}, \vec{C}$ 가 3차원 벡터일 때 $\vec{C} = \vec{A} - \frac{1}{k}\vec{B}$ 를 $\vec{C} = \vec{A}\, M_B$ 으로 나타내고자 할 때 벡터 $\vec{B}$ 의 성분으로 구성된 행렬 M_B 을 4차원 동차좌표계에서 구하시오.

풀이 1-3) 동차 좌표계로 풀면

$$\vec{A} = \vec{A}\ ,\ (A_x, A_y, A_z, 1) = (A_x, A_y, A_z, 1)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\vec{A}=\vec{B},\ (A_x, A_y, A_z, 1)=(A_x, A_y, A_z, 1)\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\B_x&B_y&B_z&1\end{pmatrix}$$

$$\vec{C}=\vec{A}+\vec{B}$$

$$=\vec{A}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}+\vec{A}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\B_x&B_y&B_z&1\end{pmatrix}$$

$$=\vec{A}\left(\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}+\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\B_x&B_y&B_z&1\end{pmatrix}\right)\quad=\vec{A}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\B_x&B_y&B_z&1\end{pmatrix}$$

$$\therefore M_B=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\B_x&B_y&B_z&1\end{pmatrix}\ \text{또는}\ \therefore M_B=\begin{pmatrix}w&0&0&0\\0&w&0&0\\0&0&w&0\\w*B_x&w*B_y&w*B_z&w\end{pmatrix}$$

위의 풀이 과정을 보면 동차좌표의 W 값은 연산에서 같은 값이어야 하고, 또한 행렬의 덧셈에는 관여하지 않음을 주의해야 합니다.

풀이 1-4) 동차 좌표계로 풀면

$$\vec{A}=\vec{A},\ \vec{A}=\vec{A}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

$$-\frac{1}{k}\vec{B}\quad=\vec{A}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\-\frac{1}{k}B_x&-\frac{1}{k}B_y&-\frac{1}{k}B_z&1\end{pmatrix}\quad=\vec{A}\begin{pmatrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\-B_x&-B_y&-B_z&k\end{pmatrix}$$

연산을 위해서 $\vec{A} = \vec{A}\begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & k \end{pmatrix}$ 로 바꿉니다.

$$\vec{C} = \vec{A} - \frac{1}{k}\vec{B}$$

$$= \vec{A}\begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & k \end{pmatrix} + \vec{A}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -B_x & -B_y & -B_z & k \end{pmatrix} = \vec{A}\begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ -B_x & -B_y & -B_z & k \end{pmatrix}$$

$$\therefore M_B = \begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ -B_x & -B_y & -B_z & k \end{pmatrix}$$

2.8.3 벡터의 내적과 행렬의 특별한 표현

3차원벡터 $\vec{A}, \vec{B}, \vec{C}$ 에서 $(\vec{A} \cdot \vec{B})\vec{C}$ 를 행렬로 표현해 봅시다.

$\vec{A} = (A_x, A_y, A_z)$, $\vec{B} = (B_x, B_y, B_z)$, $\vec{C} = (C_x, C_y, C_z)$ 라 한다면

$$(\vec{A} \cdot \vec{B})\vec{C}$$

$$= (\quad (A_xB_x + A_yB_y + A_zB_z)C_x ,$$
$$(A_xB_x + A_yB_y + A_zB_z)C_y ,$$
$$(A_xB_x + A_yB_y + A_zB_z)C_z)$$

$$= (\quad A_xB_xC_x + A_yB_yC_x + A_zB_zC_x ,$$
$$A_xB_xC_y + A_yB_yC_y + A_zB_zC_y ,$$
$$(A_xB_xC_z + A_yB_yC_z + A_zB_zC_z))$$

$$= (A_x, A_y, A_z) \begin{pmatrix} B_xC_x & B_xC_y & B_xC_z \\ B_yC_x & B_yC_y & B_yC_z \\ B_zC_x & B_zC_y & B_zC_z \end{pmatrix}$$

$(\vec{A}\cdot\vec{B})\vec{C} = \vec{A}\cdot\vec{B}\,\vec{C}$ 라 하고 이때 $\vec{B}\,\vec{C} = \begin{pmatrix} B_xC_x & B_xC_y & B_xC_z \\ B_yC_x & B_yC_y & B_yC_z \\ B_zC_x & B_zC_y & B_zC_z \end{pmatrix}$ 인 행렬로 만들 수 있습니다.

물리에서는 $\vec{B}\,\vec{C}$와 같이 종종 두 벡터의 조합으로 이루어진 행렬을 사용하므로 의미를 기억하도록 합시다.

$(\vec{A}\cdot\vec{B})\vec{C} = (\vec{B}\cdot\vec{A})\vec{C} = \vec{B}\cdot\vec{A}\vec{C}$ 이 가능하므로 $\vec{A}\,\vec{C} = \begin{pmatrix} A_xC_x & A_xC_y & A_xC_z \\ A_yC_x & A_yC_y & A_yC_z \\ A_zC_x & A_zC_y & A_zC_z \end{pmatrix}$ 행렬을 얻을 수 있습니다.

문제 1-5) $\vec{P}, \vec{A}, \vec{B}, \vec{C}$가 3차원 벡터일 때 $\vec{P} = \vec{A} + \frac{D}{k}\vec{B} + -\frac{(\vec{A}\cdot\vec{C})}{k}\vec{B}$ 를 $\vec{C} = \vec{A}\,M_{B,C}$ 식을 만족하는 벡터 $\vec{B}, \vec{C}$ 의 성분으로 구성된 행렬 M_{BC}을 4차원 동차좌표계에서 구하시오.(단, D, k는 스칼라)

문제에서 k값으로 나누고 있으므로 이것을 w 값으로 사용합니다.

$$\vec{A} = \vec{A}\begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & k \end{pmatrix}$$

$$\frac{D}{k}\vec{B} = \vec{A}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ D*B_x & D*B_y & D*B_z & k \end{pmatrix}$$

$$-\frac{(\vec{A}\cdot\vec{C})}{k}\vec{B} \quad = \vec{A}\cdot\left(-\frac{1}{k}(\vec{C}\vec{B})\right) \quad = \vec{A}\begin{pmatrix} -C_xB_x & -C_xB_y & -C_xB_z & 0 \\ -C_yB_x & -C_yB_y & -C_yB_z & 0 \\ -C_zB_x & -C_zB_y & -C_zB_z & 0 \\ 0 & 0 & 0 & k \end{pmatrix}$$

$$\vec{P} = \vec{A}\begin{pmatrix} k & 0 & 0 & 0 \\ 0 & k & 0 & 0 \\ 0 & 0 & k & 0 \\ 0 & 0 & 0 & k \end{pmatrix} + \vec{A}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ D*B_x & D*B_y & D*B_z & k \end{pmatrix} + \vec{A}\begin{pmatrix} -C_xB_x & -C_xB_y & -C_xB_z & 0 \\ -C_yB_x & -C_yB_y & -C_yB_z & 0 \\ -C_zB_x & -C_zB_y & -C_zB_z & 0 \\ 0 & 0 & 0 & k \end{pmatrix}$$

$$\vec{P} = \vec{A}\begin{pmatrix} k-C_xB_x & -C_xB_y & -C_xB_z & 0 \\ -C_yB_x & k-C_yB_y & -C_yB_z & 0 \\ -C_zB_x & -C_zB_y & k-C_zB_z & 0 \\ D*B_x & D*B_y & D*B_z & k \end{pmatrix}$$

$$\therefore M_{BC} = \begin{pmatrix} k-C_xB_x & -C_xB_y & -C_xB_z & 0 \\ -C_yB_x & k-C_yB_y & -C_yB_z & 0 \\ -C_zB_x & -C_zB_y & k-C_zB_z & 0 \\ D*B_x & D*B_y & D*B_z & k \end{pmatrix}$$

또는 $$M_{BC} = \begin{pmatrix} -k+C_xB_x & C_xB_y & C_xB_z & 0 \\ C_yB_x & -k+C_yB_y & C_yB_z & 0 \\ C_zB_x & C_zB_y & -k+C_zB_z & 0 \\ -D*B_x & -D*B_y & -D*B_z & -k \end{pmatrix}$$

행렬 M_{BC} 는 평행광의 그림자를 만드는 행렬로 그림자 행렬(Shadow Matrix) 라 합니다.