3D 변환 행렬과 벡터 연산 예제 (Direct3D 기준)

이 문서는 벡터 `v = (1, 1, 1, 1)`을 사용하여 Direct3D(D3D) 기준의 변환 행렬 계산 예제와 벡터의 내적, 외적에 대해 설명합니다.

D3D의 규칙:

- **좌표계**: 왼손 좌표계 (Left-Handed)
- **행렬**: 행 우선 행렬 (Row-Major)
- **곱셈 순서**: `v' = v * M` (벡터 * 행렬)

---

1. 변환 행렬(SRT) 예제

사용할 원본 벡터는 동차 좌표계로 표현된 `v = [1, 1, 1, 1]` 입니다.

1.1. 스케일 (Scale) 변환 예제

조건: X, Y, Z축으로 각각 2, 3, 4배 확대합니다. (`Sx=2, Sy=3, Sz=4`)

스케일 행렬 (S):

```
[ 2  0  0  0 ]
[ 0  3  0  0 ]
[ 0  0  4  0 ]
[ 0  0  0  1 ]
```

계산 (`v' = v * S`):
`v' = [1, 1, 1, 1] * S = [2, 3, 4, 1]`

해석: 원점(0,0,0)을 기준으로 각 축의 방향으로 2, 3, 4배만큼 늘어난 위치 `(2, 3, 4)`로 이동했습니다.

1.2. 회전 (Rotation) 변환 예제

조건: 각 축을 기준으로 90도(π/2 라디안) 회전합니다. (`cos(90°)=0`, `sin(90°)=1`)

X축 기준 90도 회전

X축 회전 행렬 (Rx):

```
[ 1    0        0       0 ]
[ 0  cosθ    -sinθ      0 ]   ->   [ 1  0  0  0 ]
[ 0  sinθ     cosθ      0 ]   ->   [ 0  0 -1  0 ]
[ 0    0        0       1 ]        [ 0  1  0  0 ]
                                    [ 0  0  0  1 ]
```

계산 (`v' = v * Rx`):
`v' = [1, 1, 1, 1] * Rx = [1, 1, -1, 1]`

해석: X축을 중심으로 90도 회전하여, `(1, 1, 1)` 점이 `(1, 1, -1)` 위치로 이동했습니다.

Y축 기준 90도 회전

Y축 회전 행렬 (Ry):

```
[ cos(θ)  0   sin(θ)  0 ]
[    0     1     0     0 ]
```

```
[-sin(θ)  0   cos(θ)  0 ]
[    0     0     0     1 ]
```

예제: Y축을 기준으로 90도 회전
`θ = 90도`, `cos(90) = 0`, `sin(90) = 1`

```
Ry = [ 0  0  1  0 ]
     [ 0  1  0  0 ]
     [-1  0  0  0 ]
     [ 0  0  0  1 ]
```

계산 (`v' = v * Ry`):
`v' = [1, 1, 1, 1] * Ry`

- `x' = (1*0) + (1*0) + (1*-1) + (1*0) = -1`
- `y' = (1*0) + (1*1) + (1*0) + (1*0) = 1`
- `z' = (1*1) + (1*0) + (1*0) + (1*0) = 1`
- `w' = (1*0) + (1*0) + (1*0) + (1*1) = 1`

결과 벡터: `v' = (-1, 1, 1, 1)`

해석:

> D3D의 왼손 좌표계에서, Y축의 양수 방향에서 원점을 내려다볼 때 양수(+) 회전은 **반시계 방향**입니다. 이 규칙에 따라 점 `(x, z)`를 90도 회전시키면 `(-z, x)`로 이동합니다.
> 따라서 벡터 `(1, 1, 1)`의 XZ 평면 위 점 `(1, 1)`은 `(-1, 1)`로 이동하게 됩니다. 최종 결과는 `(-1, 1, 1)`이 됩니다.

Z축 기준 90도 회전

Z축 회전 행렬 (Rz):

```
[ cosθ -sinθ  0  0 ]   ->   [ 0 -1  0  0 ]
[ sinθ  cosθ  0  0 ]   ->   [ 1  0  0  0 ]
[  0     0    1  0 ]   ->   [ 0  0  1  0 ]
[  0     0    0  1 ]   ->   [ 0  0  0  1 ]
```

계산 (`v' = v * Rz`):
`v' = [1, 1, 1, 1] * Rz = [1, -1, 1, 1]`

해석: Z축을 중심으로 90도 회전하여, `(1, 1, 1)` 점이 `(1, -1, 1)` 위치로 이동했습니다.

1.3. **이동** (Translation) **변환 예제**

조건: X축으로 5, Y축으로 6, Z축으로 7만큼 이동합니다. (`Tx=5, Ty=6, Tz=7`)

이동 행렬 (T):

```
[ 1  0  0  0 ]
[ 0  1  0  0 ]
[ 0  0  1  0 ]
[ 5  6  7  1 ]
```

계산 (`v' = v * T`):
`v' = [1, 1, 1, 1] * T = [6, 7, 8, 1]`

해석: `(1, 1, 1)` 위치에서 각 축으로 5, 6, 7만큼 더해져 `(6, 7, 8)` 위치로 평행 이동했습니다.

---

2. 벡터 연산: 내적과 외적

2.1. 내적 (Dot Product)

설명:
내적은 두 벡터의 방향 관계를 스칼라(숫자) 값으로 나타내는 연산입니다. 주로 두 벡터 사이의 각도를 구하거나, 한 벡터를 다른 벡터에 투영(projection)시키는 길이를 계산하는 데 사용됩니다.

- **결과 > 0**: 두 벡터 사이의 각도가 90도보다 작다 (예각).
- **결과 = 0**: 두 벡터가 서로 직교(수직)한다.
- **결과 < 0**: 두 벡터 사이의 각도가 90도보다 크다 (둔각).

공식:
`A · B = Ax * Bx + Ay * By + Az * Bz`
`A · B = |A| * |B| * cos(θ)`

예제:
`A = (1, 2, 3)`, `B = (4, -5, 6)`
`A · B = (1 * 4) + (2 * -5) + (3 * 6) = 4 - 10 + 18 = 12`
(결과가 12이므로 두 벡터는 예각을 이룹니다.)

2.2. 외적 (Cross Product)

설명:
외적은 3차원 공간에서 두 벡터에 **동시에 수직인 새로운 벡터**를 계산하는 연산입니다. 결과로 나오는 벡터의 방향은 오른손 법칙(OpenGL) 또는 왼손 법칙(Direct3D)을 따릅니다. 주로 폴리곤의 법선 벡터(Normal Vector)를 구하는 데 사용됩니다.

공식:
`A x B = (Ay*Bz - Az*By, Az*Bx - Ax*Bz, Ax*By - Ay*Bx)`

예제:
`A = (1, 2, 3)`, `B = (4, 5, 6)`

- `Cx = (2 * 6) - (3 * 5) = 12 - 15 = -3`
- `Cy = (3 * 4) - (1 * 6) = 12 - 6 = 6`
- `Cz = (1 * 5) - (2 * 4) = 5 - 8 = -3`

결과 벡터: `C = A x B = (-3, 6, -3)`
(벡터 C는 벡터 A와도 수직이고, 벡터 B와도 수직입니다.)

---

3. 월드 행렬 (World Matrix) - SRT 결합

실제 3D 환경에서는 물체(정점)에 스케일, 회전, 이동 변환을 순차적으로 적용해야 합니다. 매번 세 번의 행렬 곱셈을 하는 것은 비효율적이므로, 이 세 행렬을 미리 곱해서 **하나의 최종 변환 행렬**, 즉 **월드 행렬**을 만들어 사용합니다.

3.1. 행렬 곱셈 순서의 중요성

행렬 곱셈은 교환 법칙이 성립하지 않으므로 곱하는 순서가 매우 중요합니다. 일반적으로 **스케일** → **회전** → **이동** 순서로 적용합니다.

`WorldMatrix = ScaleMatrix * RotationMatrix * TranslationMatrix`

이 순서는 다음과 같은 논리적 흐름을 따릅니다.

1. 물체의 원점(local origin)을 기준으로 크기를 조절합니다 (Scale).
2. 크기가 조절된 물체를 그 원점을 기준으로 회전시킵니다 (Rotation).
3. 회전된 물체를 월드 공간의 최종 위치로 옮깁니다 (Translation).

3.2. 월드 행렬 계산 예제

조건:

- **스케일(S)**: 모든 축으로 2배 확대 (`Sx=2, Sy=2, Sz=2`)
- **회전(R)**: Y축을 기준으로 90도 회전
- **이동(T)**: `(5, 6, 7)` 만큼 이동

1. 개별 행렬 준비

- **S 행렬**

```
[ 2  0  0  0 ]
[ 0  2  0  0 ]
[ 0  0  2  0 ]
[ 0  0  0  1 ]
```

- **R 행렬 (Y축 90도)**

```
[ 0  0 -1  0 ]
[ 0  1  0  0 ]
[ 1  0  0  0 ]
[ 0  0  0  1 ]
```

- **T 행렬**

```
[ 1  0  0  0 ]
[ 0  1  0  0 ]
[ 0  0  1  0 ]
[ 5  6  7  1 ]
```

2. 월드 행렬 계산 (`World = S * R * T`)

- **임시 행렬 `SR = S * R` 계산**

```
SR = [ 2  0  0  0 ] * [ 0  0 -1  0 ] = [ 0  0 -2  0 ]
     [ 0  2  0  0 ]   [ 0  1  0  0 ]   [ 0  2  0  0 ]
     [ 0  0  2  0 ]   [ 1  0  0  0 ]   [ 2  0  0  0 ]
     [ 0  0  0  1 ]   [ 0  0  0  1 ]   [ 0  0  0  1 ]
```

- **최종 `World = SR * T` 계산**

```
World = [ 0  0 -2  0 ] * [ 1  0  0  0 ] = [ 0  0 -2  0 ]
        [ 0  2  0  0 ]   [ 0  1  0  0 ]   [ 0  2  0  0 ]
        [ 2  0  0  0 ]   [ 0  0  1  0 ]   [ 2  0  0  0 ]
        [ 0  0  0  1 ]   [ 5  6  7  1 ]   [ 5  6  7  1 ]
```

3. 최종 변환 (`v' = v * World`)

이제 원본 벡터 `v = [1, 1, 1, 1]`에 최종 월드 행렬을 한 번만 곱해줍니다.

```
v' = [1, 1, 1, 1] * [ 0  0 -2  0 ]
                    [ 0  2  0  0 ]
                    [ 2  0  0  0 ]
                    [ 5  6  7  1 ]
```

- `x' = (1*0) + (1*0) + (1*2) + (1*5) = 7`
- `y' = (1*0) + (1*2) + (1*0) + (1*6) = 8`
- `z' = (1*-2) + (1*0) + (1*0) + (1*7) = 5`

- `w' = (1*0) + (1*0) + (1*0) + (1*1) = 1`

결과 벡터: `v' = [7, 8, 5, 1]`

해석: `(1,1,1)` 점이 2배 커지고, Y축으로 90도 회전한 후, `(5,6,7)`만큼 평행 이동하여 최종적으로 `(7,8,5)` 위치로 이동했습니다.