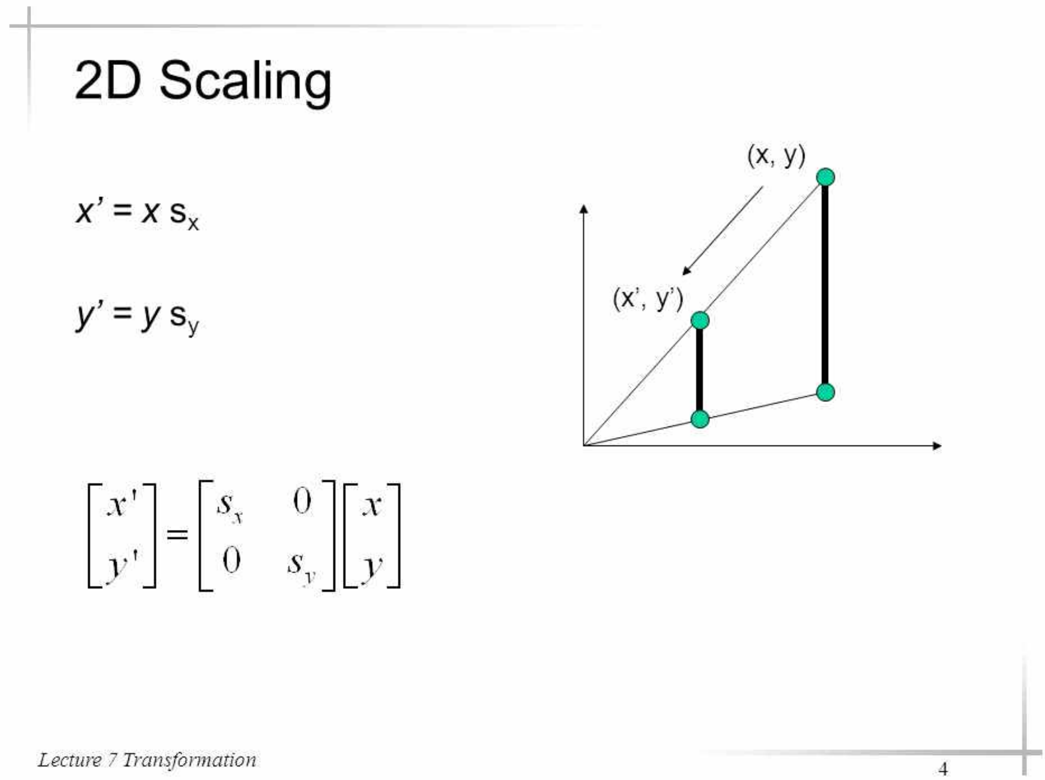
필기노트 - 컴퓨터 그래픽스 4장 좌표계와 변환

<table>
<tr><th></th><th>내 용</th></tr>
<tr><td>Scaling
축소확대</td><td>

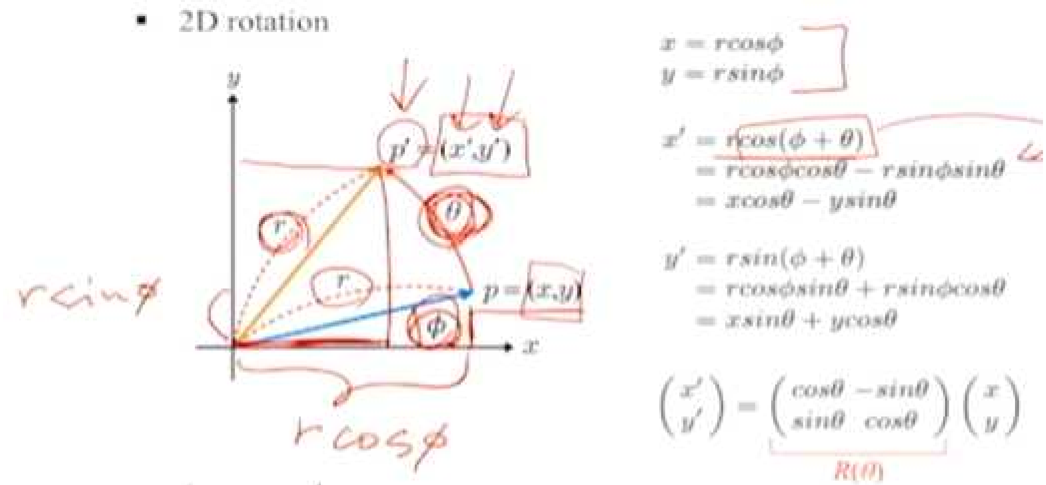
● 2차원 Scaling(축소확대)에서는 scaling factors(축소확대 인자)을 사용

- scaling factors -> Sx, Sy 일 때 $\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix}$

- 물체의 크기나 비율을 변경하고 싶을 때 각 좌표에 축소확대 인자를 곱해서 변화를 주는 듯



</td></tr>
<tr><td>Rotation
회전 변환</td><td>



● 한 벡터가 θ만큼 회전하는 경우에 그 벡터가 x = rcosθ, y = rsinθ라면 위와 같은 식으로 나타낼 수 있다.

R(θ)에 이동하고자 하는 각도를 집어넣고 x,y에 곱해주면 원하는 좌표를 만들 수 있음

● 시계 반대방향이 정방향으로 시계방향으로 회전을 원하는 경우 -θ로 간주함

- 시계방향으로 90도 회전하는 것은 반시계방향으로 270도 회전하는 것과 같기 때문

</td></tr>
</table>

<table>
<tr>
<td>Translation
&
동차좌표</td>
<td>

● 주어진 점 x, y를 d_x, d_y만큼 이동시키는 것

$$\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \end{pmatrix} = \begin{pmatrix} x+d_x \\ y+d_y \end{pmatrix}$$

● homogeneous coordinates(동차좌표)를 이용하면 행렬의 곱셈으로 표현 가능

Question 1. Translation in homogeneous coordinates.
Create a **function** called `translate(dx,dy)`. Be sure your function suppresses all output, as we will want to use it later inside a large `for` loop. This function should return the matrix:

$$T = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

When a 2D vector (x, y) is represented in (3D) homogeneous coordinates as $\vec{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, this matrix does exactly what you would expect, translates the x and y values by dx and dy respectively.

$$T\,\vec{\mathbf{x}} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x+dx \\ y+dy \\ 1 \end{bmatrix}$$

행렬의 곱셈으로 표현하기 위해 3차원 좌표로 바꾸려면 세 번째 요소로 1을 추가 해줘야함. 1이 아닌 수로 하기 위해선 ex) 2, 나머지 요소들도 그만큼 곱해야함 ex) [2x, 2y, 2]

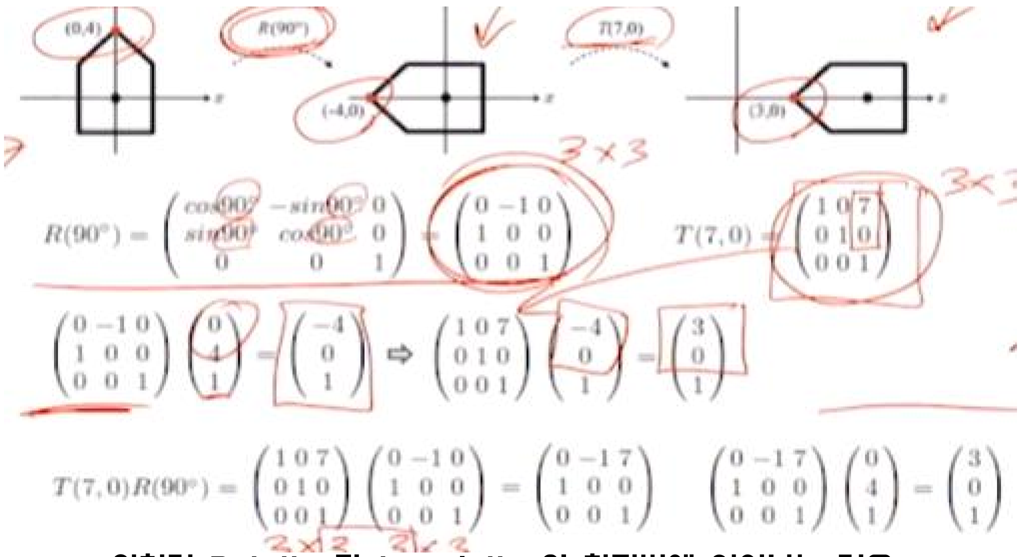
세 번째 요소가 1이 아닌 경우 ex) [3, 6, 3]
첫 번째 두 번째 요소를 세 번째 요소로 나누어주고 ex) [1, 2, 1]
세 번째 요소를 제외시킴 ex) [1, 2]

● 위의 scaling과 rotation은 2차원 좌표인데 이건 3차원 좌표이므로
행렬의 곱셈으로 함께 나타내기 위해 scaling과 rotation도 세 번째 요소로 1을 추가하면 됨.

</td>
</tr>
<tr>
<td>Transform Composition
변환 구성</td>
<td>

● 축소확대, 회전변환, Translation 모두 3차원 행렬로 표현할 수 있다는 것을 알았음

● 변환은 2개 이상 일어날 수 있으므로

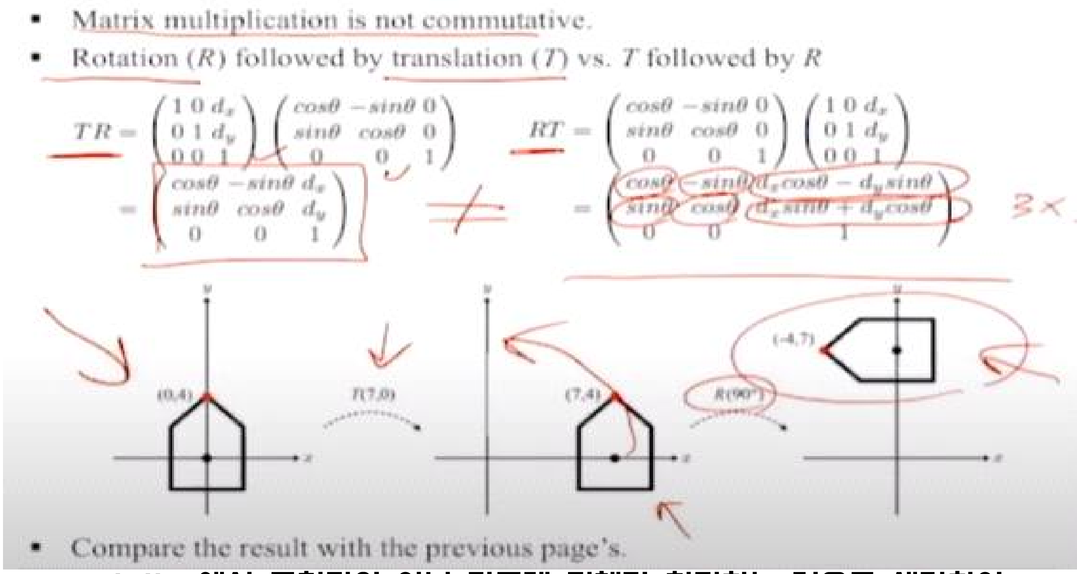


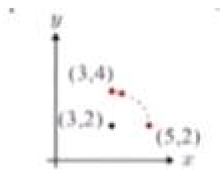
위처럼 Rotation과 translation이 한꺼번에 일어나는 경우
1. 두 행렬을 먼저 곱해 하나의 행렬로 만들어주고
2. 변환할 좌표에 행렬을 곱해주어 좌표를 변환한다

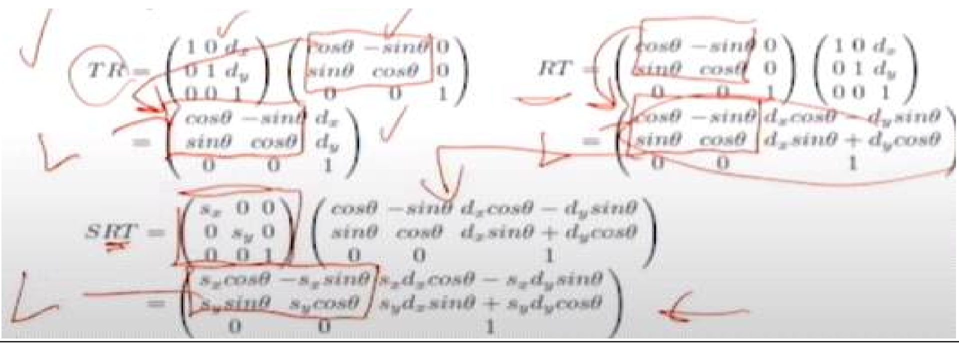
\# 중요한 것은 변환의 순서는 그대로 지켜야 함

↓ 계속

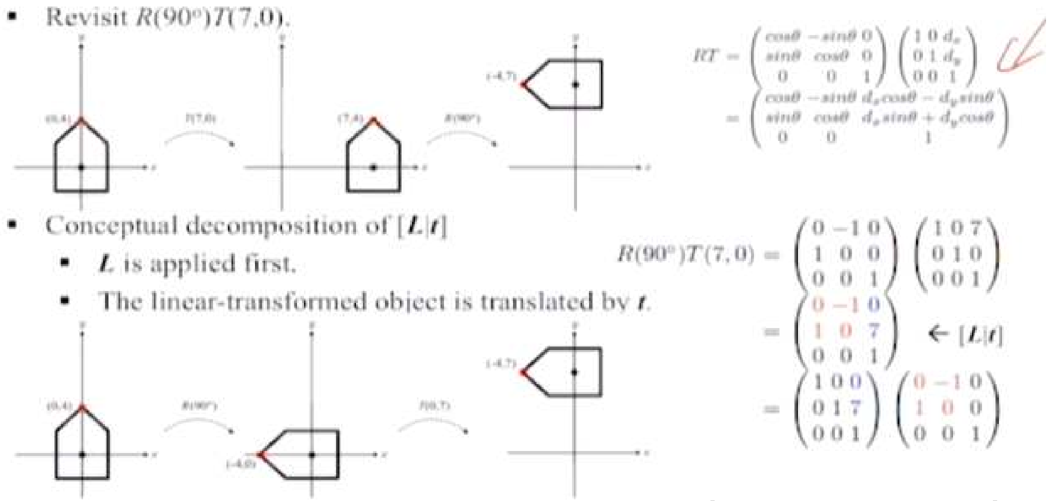
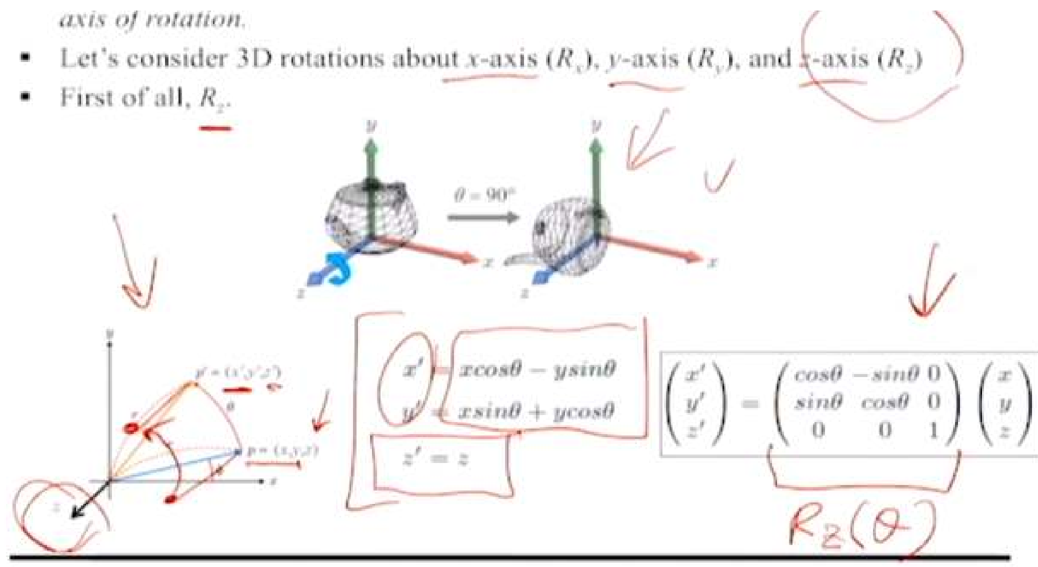
</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td>


rotation에선 도형만이 아닌 좌표계 전체가 회전하는 것으로 생각하여
translation된 만큼 rotation에 적용되기 때문에 둘의 순서가 바뀌어선 안된다.
</td>
</tr>
<tr>
<td>Rotation 회전변환
(임의의 점 중심)</td>
<td>
● 원점 아닌 임의의 점을 기준으로 한 회전변환


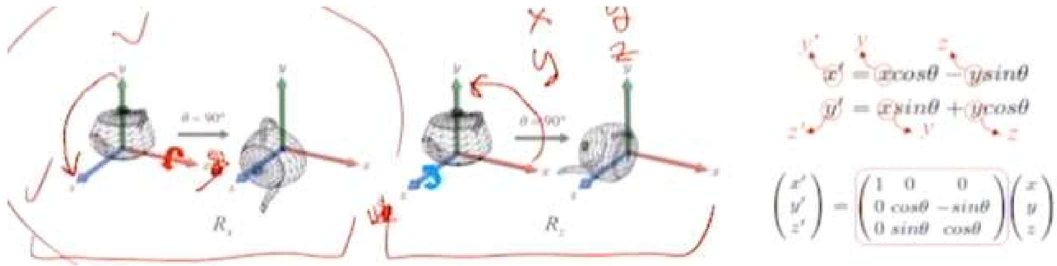
* (3,2)를 기준으로 한 회전변환을 한다고 생각하면
1. 전체를 T(-3,-2) <- (-3,-2)만큼 translation해주어서 (3,2)를 (0,0)으로 만든다
2. 그리고 R(90˚) 해준다
3. 다시 T(3,2)를 해주어 기준점을 원상 복귀 해준다
</td>
</tr>
<tr>
<td>Affin Transform</td>
<td>
● Affine transform
- Linear transform
· Scaling
· Rotation
- Translation
● 3X3 행렬에서
EX) 0 0 0
0 0 0
0 0 0
맨 아래 행을 제외한 좌상단 2X2 행렬을 L, 우측 상단 열을 t라고 한다.
그러면 [L|t] 는 2X3 행렬인데
L은 오직 선형변환(Linear transform)의 결합체,
t는 translation의 결합체 이다.


</td>
</tr>
</table>

	항상 L을 먼저 적용시키고 t를 적용해야 한다 -> translation을 적용한 후 rotation과 같은 선형 변환을 적용하는 경우 t가 L의 영향을 받아 t의 입력값과 다르게 나타난다.  그림에서도 위를 보면 t의 입력값은 (7,0) 이지만 L을 거치며 t값이 (0,7)로 변한다. 위처럼 t -> L 순서가 아닌 아래의 L -> t 순서로 진행해야 한다.
Rigid Motion (Rigid-body motion) (강체)	Rotation과 translation만 생각해보자 -> 물체의 모양은 변하지 않음 -> 강체라고 함 rotation과 translation만 있는 경우에 [L | t] 대신 [R | t]라고 함 이것도 R먼저 계산하고 t를 사용함
3D Scailing 3차원 축소확대	2차원에서 (Sx, Sy)를 각각 (x, y)에 곱해 크기를 변경했던 것처럼 3차원에서도 (Sx, Sy, Sz)를 (x, y, z)에 곱해 크기를 변경해준다. (Sx, Sy, Sz) -> 이런 것들을 scaling factor라고 부르고 이 scaling factor가 모두 같은 경우 uniform scaling 아니면 non-uniform scaling이라 부름
3D Rotation 3차원 회전변환	● z축을 중심으로 한 회전변환 Rz(θ)  z축을 중심으로 한 회전은 원점을 중심으로 한 회전과 같기 때문에 식의 변화는 없음 x' = xcosθ - ysinθ, y' = xsinθ + ycosθ, z' = z 로 그대로 생각

<table>
<tr>
<td></td>
<td>

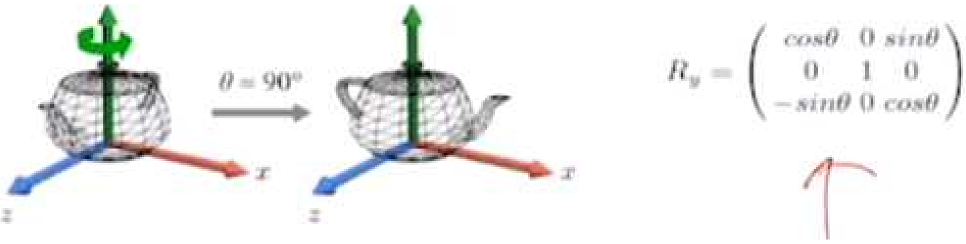
● x축을 중심으로 한 회전변환 Rx(θ)



생각해보면 z축 중심일때는 우리가 z축 화살표 끝에서 x축과 y축을 바라본 것과 같다
x축이 중심인 경우에는 x축 화살표 끝에서 y축과 z축을 바라보면
y축이 x축, z축이 y축처럼 보인다.
따라서 x' = xcosθ - ysinθ, y' = xsinθ + ycosθ식에서 x를 y로 바꾸고 y를 z로 바꾼다.
x' =x, y' = ycosθ - zsinθ, z' = ysinθ + zcosθ 식으로 나타낼 수 있다.

● y축을 중심으로 한 회전변환 Ry(θ)

y축 화살표 끝에서 바라본다고 생각하면 z축이 x축으로, x축을 y축으로 생각할 수 있다.
x' = zsinθ + xcosθ, y' = y, z' = zcosθ - xsinθ



</td>
</tr>
<tr>
<td>3차원 Translation</td>
<td>

● 2차원에서 했던 것처럼 3x3 행렬을 4x4로 바꾸어서 수행

- Matrix-point multiplication

4x4

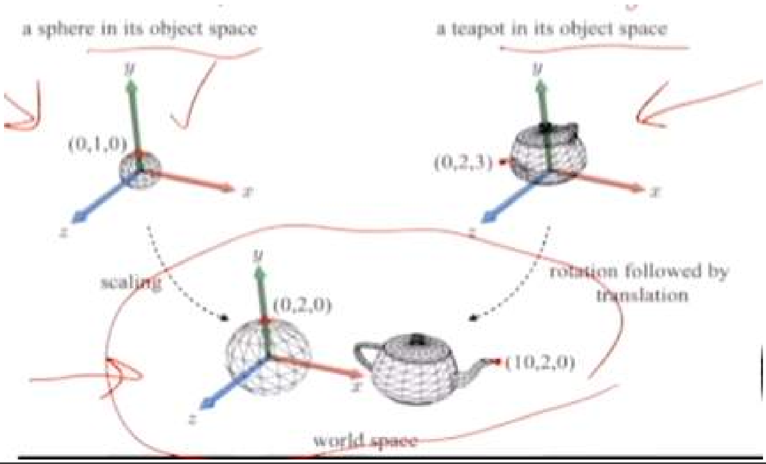
$$\begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{pmatrix}$$

4x4 I

</td>
</tr>
<tr>
<td>World transform
월드 변환</td>
<td>

● 각각 만들어진 폴리곤 메쉬가 있는 공간(좌표계)을 Object space라고 한다.
이렇게 각각의 object space에서 만들어진 폴리곤 메쉬를
하나의 world space 안에 넣으려고 할 때 World transform이 필요하다

● 월드 변환에서 scailing, rotation, translation과 같은 변환을 사용



</td>
</tr>
</table>

<table>
<tr>
<td>Object Space</td>
<td>

● 폴리곤 메쉬를 사용하여 object를 만들 때 object space도 같이 만들어진다.
여기서 우리는 object와 object space는 항상 함께하는 것으로 생각하자

● world space는 e1, e2, e3를 축으로 이루어져 있고
object space는 u, v, n을 축으로 이루어져 있음
(x, y, z)와 같은 것으로 생각

- Similarly, R transforms e_2 and e_3 into v and n, respectively:

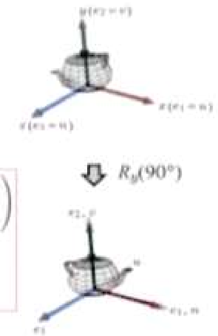
$$Re_2 = R\begin{pmatrix}0\\1\\0\end{pmatrix} = \begin{pmatrix}v_x\\v_y\\v_z\end{pmatrix} \quad Re_3 = R\begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}n_x\\n_y\\n_z\end{pmatrix}$$

- The above three are combined:

R=

$$R\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} = \begin{pmatrix}\boxed{\begin{matrix}u_x\\u_y\\u_z\end{matrix}}&\boxed{\begin{matrix}v_x\\v_y\\v_z\end{matrix}}&\boxed{\begin{matrix}n_x\\n_y\\n_z\end{matrix}}\end{pmatrix}$$
$$\qquad\qquad\qquad u \quad\; v \quad\; n$$

$$R_y(90°) = \begin{pmatrix}\cos 90° & 0 & \sin 90°\\ 0 & 1 & 0\\ -\sin 90° & 0 & \cos 90°\end{pmatrix} = \begin{pmatrix}0&0&1\\0&1&0\\-1&0&0\end{pmatrix}$$



- R's columns are u, v, and n. Given the 'rotated' object-space basis, $\{u, v, n\}$, the rotation matrix is immediately determined, and vice versa.

rotation에서 R은 변환된 object의 object space의 (u, v, n)축이 가지는 좌표를
사용하여 빠르게 구할 수있다.

$$R\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} = \begin{pmatrix}\boxed{\begin{matrix}u_x\\u_y\\u_z\end{matrix}}&\boxed{\begin{matrix}v_x\\v_y\\v_z\end{matrix}}&\boxed{\begin{matrix}n_x\\n_y\\n_z\end{matrix}}\end{pmatrix}$$
$$\qquad\qquad\qquad u \quad\; v \quad\; n$$

</td>
</tr>
<tr>
<td>Invers of
Translation and Scaling
변환들의 역함수</td>
<td>

● Translation의 경우 (dx, dy, dz) 만큼 이동했으면
반대로 (-dx, -dy, -dz) 만큼 이동한다.

* scaling의 경우 (Sx, Sy, Sz) 만큼 변환한 경우
역으로 (1/Sx, 1/Sy, 1/Sz) 만큼 변환한다.

● rotation의 경우 {u,v,n}이 각각 orthonormal(직교)하므로
R의 전치행렬이 역행렬이다.
-직교한 벡터의 내적은 0, 같은 벡터끼리의 내적은 1

- Note that $\{u, v, n\}$ is an orthonormal basis, i.e., $u \cdot u = v \cdot v = n \cdot n = 1$ and $u \cdot v = v \cdot n = n \cdot u = 0$.
- Let's multiply R's transpose (R^T) with R:

$$\begin{aligned} R^TR &= \begin{pmatrix}u_x & u_y & u_z\\ v_x & v_y & v_z\\ n_x & n_y & n_z\end{pmatrix}\begin{pmatrix}u_x & v_x & n_x\\ u_y & v_y & n_y\\ u_z & v_z & n_z\end{pmatrix} \\ &= \begin{pmatrix}u\cdot u & u\cdot v & u\cdot n\\ v\cdot u & v\cdot v & v\cdot n\\ n\cdot u & n\cdot v & n\cdot n\end{pmatrix} \\ &= \begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix} \\ &= I\end{aligned}$$

</td>
</tr>
</table>