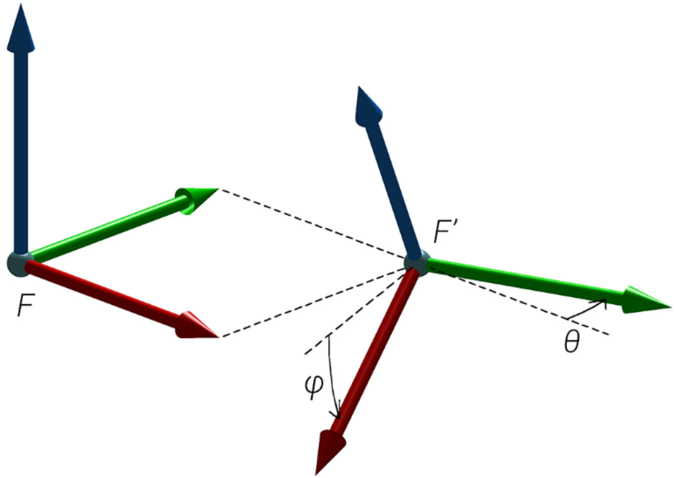
실감사업단 소개

- 10명 정도 씩 5번에 걸쳐
- 사진 촬영 7425
- 동문 앞 맘스터치
  - 다음주에 쿠폰 배급 (1만원)

게임 개발 로드맵 (2025 버전)

- https://roadmap.sh/game-developer



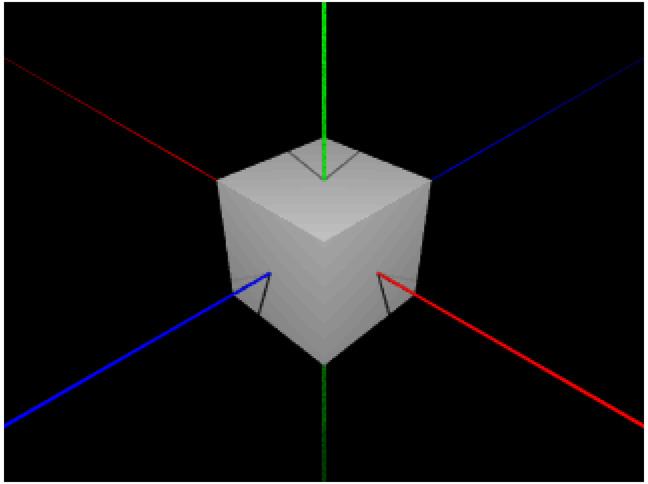
3차원 오리엔테이션(Orientation) 표현 방법

A theory behind the orientation representation

※ 이 슬라이드는 학생들의 이해를 도모하기 위해 매주 업데이트될 가능성이 높음

아직 까지 다루지 않은 부분

- 모든 파티클은 방향이 없다고 가정
  - 왜? 구(Sphere) 형태 이므로..
- 회전을 고려 하지 않았음
  - 회전해 봐야 여전히 구 이므로..
- 하지만, 파티클이 box 형태라면...?
  - 3차원 오리엔테이션에 대해 알아야 함...

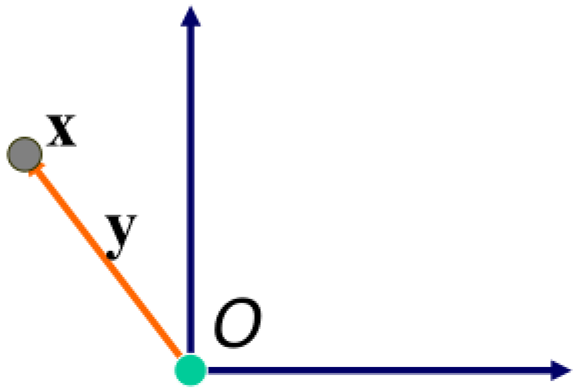


Orientation(상태) & Rotation (작업)

- Orientation은 하나의 기준이 되는 reference frame (origin) 이 필요
- rotation은 오브젝트의 orientation을 하나의 상태에서 다른 상태로 변경
- 하나의 orientation은 기준 reference frame 으로 부터의 rotation을 통해 표현 할 수 있음

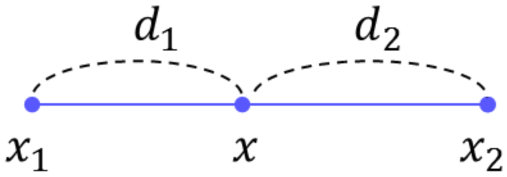
비유(Analogy)

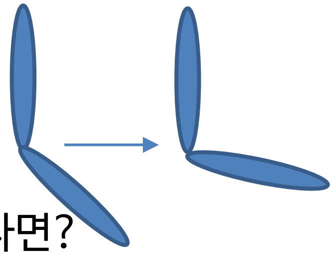
- 같은 의미: position(<->orientation) and translation(<->rotation)
- Reference 는 원점(origin)
- position을 원점으로 부터의 위치이동으로 표현 가능



Issues

- 생각할 문제 점, 3차원 로테이션을 표현하기 위해..
  - # elements : 얼마나 많은 값이 필요하나? (matrix : 16(4x4), Euler angle: 3, quaternion 4…)
  - 결합(Concatenation) : 두 개 이상의 rotation을 결합 할 수 있나?
    - 로테이션 1 -> 로테이션2 == 로테이션1 * 로테이션2
  - 보간(Interpolation) : 주어진 두 개의 orientation이 있을 때 , 둘 사이를 보간 가능한가?
    - position에서의 보간은 간단하나 orientation은 무척 어렵다
      - 카메라 제어: 공간상에 배치된 카메라 위치 와 방향을 보간해서 smooth하게 움직일 필요가 많은
      - 캐릭터 애니메이션 : 조인트의 orientation을 보간해서 부드러운 관절 움직임을 나타내야 함





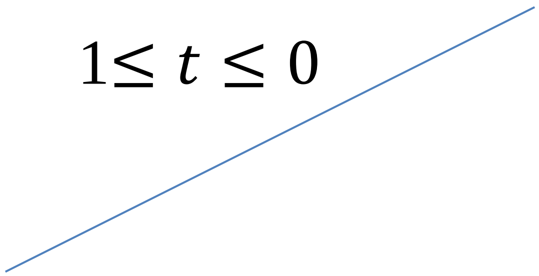
선형보간의 예 : x1, x2가 주어 졌을 때 t 가 0, 1사이로 주어 진다면?

LERP (Linear Interpolation)

$$(x_1 - x_0)t + x_0$$

x_1

$$1 \leq t \leq 0$$



x_0

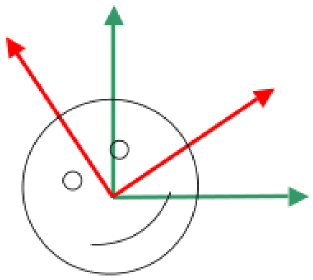
$$t = 0.5 \rightarrow x_0 + (x_1 - x_0)/2$$

$$t = 0 \rightarrow x_0$$

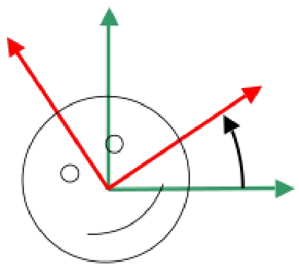
$$t = 1 \rightarrow x_1$$

Intro

- Orientation 은 reference frame에 상대적
  - 녹색 축 : Reference frame



- Rotation은 object의 orientation을 변경
  - Rotation 은 하나의 operation(검은 화살표)
  - orientation을 reference frame으로 부터의 rotation으로 표현 가능



이상적인 표현 방법 Ideal Rotation Format

- 최소의 값을 통해 표현 가능해야 함
- rotation 끼리의 결합이 쉬어야 함
  - 하나의 로테이션 -> 또 하나의 로테이션
- 기본 수학 개념이 간단하고 구현이 쉬어야 함
  - concatenation (결합)
  - Interpolation (보간)

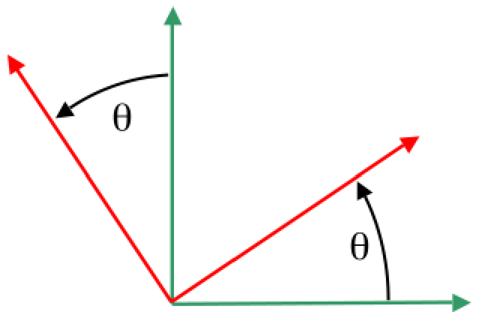
- 어떠한 표현방법들이 있나?

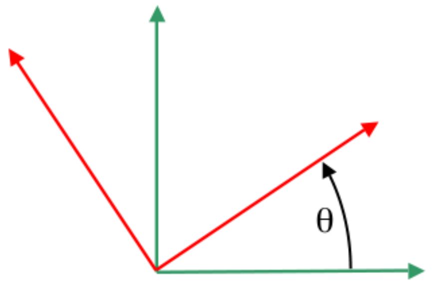
Orientation Representations

- Angle (2D)
- Euler Angles (3D)
- Axis-Angle (3D)
- Matrix (2D)
- Matrix(3D)
- Complex Number (2D)
- Quaternion (3D) : 가장 대표적인 방법

2D angle

- 가장 간단한 표현 방법은 하나의 angle 값으로 2차원 orientation을 표현하는 것임

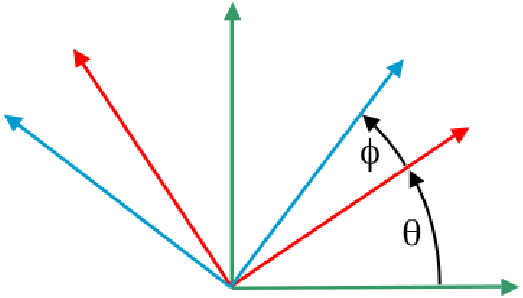




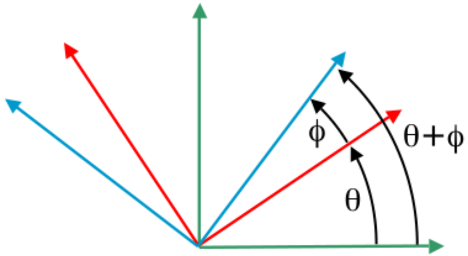
좌표계가 회전 한다고 생각하는 것이 올바름

2D Angle: Concatenation

- 결합은 아주 쉬움. θ 다음 Φ 이라면

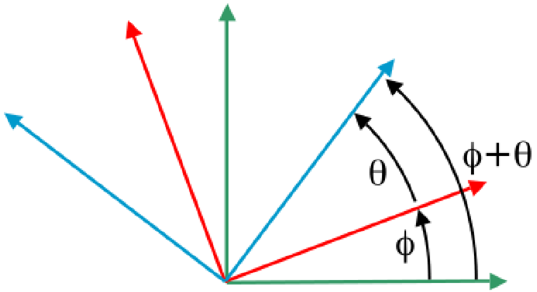


- 이 둘을 단순히 더하면 됨 (θ + Φ)

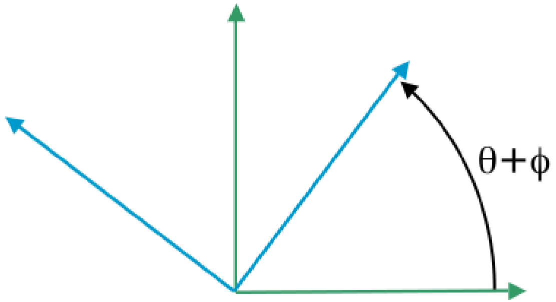


2D Angle: Concatenation

- 교환 법칙이 성립 함
  - $(\theta + \Phi) = (\Phi + \theta)$

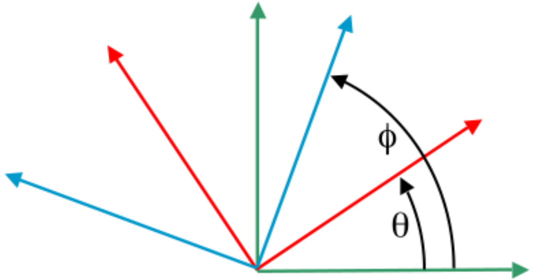


Note: 2D rotation is commutative

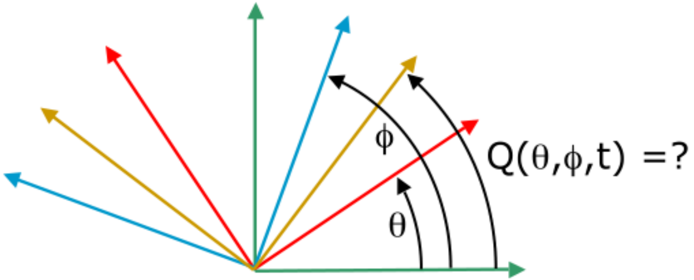


2D Angle: Interpolation

- 보간도 쉬움, 하지만 주의해야 할 점 이 있음
  만일, θ 만큼 회전한 후에 Φ 만큼 더
  회전했다고 가정하자

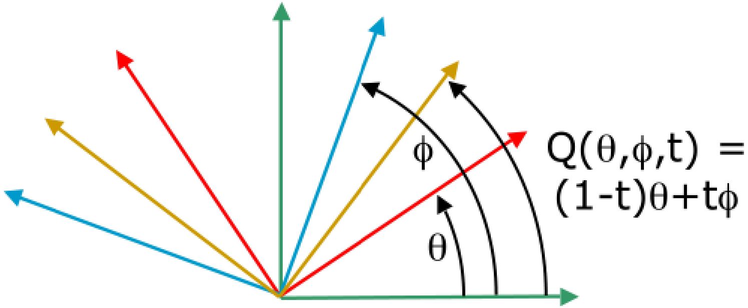


- 0에서 1로 변하는 값 t를 이용해 보간한다고
  가정한다면 (t->0 : θ, t->1 : Φ, t->0.5 : ?)



2D Angle: 보간(Interpolation)

- 보간 또한 간단히 , θ 와 ϕ 사이를 선형 보간 하면 됨.
  - 0 => θ
  - 1 => Φ
  - 0.5 => ?
- 하지만, 만일 θ = 30˚ & ϕ = 390˚ 라면?
  - 같은 각도를 의미 (390˚=30˚)
  - 하지만 선형 보간을 한다면 (1-t)θ+tϕ 결과 값은 from 30˚ to 390˚ 까지 변화 함



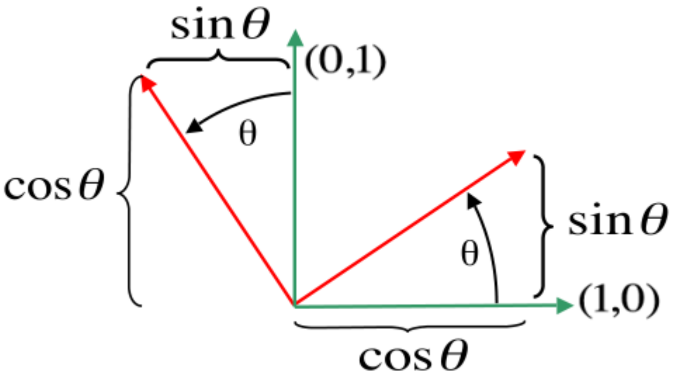
- 즉, non-linear한 각도의 특성 상 보간이 어려움

2D Angle: Interpolation

- 문제점 :
  - Angles 값은 not well-formed
  - 같은 각도를 나타내는 무한개의 orientation: 30º = 390º = -330º
  - 해결방법으로는 제약을 둘 수 있음 [0,360) or [0, 2π)

2D Angle: Rotation

- 원래 좌표 계는 x축으로 (1,0) y축으로 (0,1)를 가지며.그들의 길이는 1 임, 반지름이 1은 원을 생각해보면, 원 상의 한점의 좌표값 을 cos, sin을 통해 구할 수 있음
- rotation of vector (x,y) : R(x,y, θ) = (xcos θ - ysin θ ,xsin θ + ycos θ)



$$(x, y) \rightarrow (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

행렬

행렬 표현

- 축 의 변경
  - (1,0) -> (cosθ, sinθ)
  - (0,1) -> (-sinθ, cosθ)

- 2D 회전 행렬 표현

$$R = \begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

3D 회전

- 2D와 유사
  - 2x2 -> 4x4 (동차 좌표계 이용 할 경우)
    - 동차 좌표계
      - 2차원 ->3차원
      - 3차원->4차원
- 회전 행렬의 특징 (반드시 지켜지는 점)

$$R = \begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

basis 벡터라고 함 : unit 길이, 서로 끼리 내적은 0, $R^{-1} = R^T$

회전 행렬 간 결합

$$R_\emptyset = \begin{bmatrix} \cos\emptyset & -sin\emptyset \\ sin\emptyset & cos\emptyset \end{bmatrix}$$

$$R_\theta = \begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

$$\Rightarrow \quad R_\emptyset R_\theta$$

2D 회전 행렬

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta & x\sin\theta + y\cos\theta \end{bmatrix}$$

- 결합

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} = \begin{bmatrix} \cos(\theta+\phi) & \sin(\theta+\phi) \\ -\sin(\theta+\phi) & \cos(\theta+\phi) \end{bmatrix}$$

2D Matrix: 보간(Interpolation)

- 선형 보간(Lerp)을 한다면…

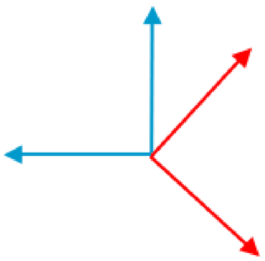
$$0.5\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}+0.5\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}=\begin{pmatrix}0.5 & -0.5\\ 0.5 & 0.5\end{pmatrix}$$

- 결과는 rotation 행렬이 안됨
  - 왜? basis vector가 unit length를 갖지 못함

선형 보간의 문제점

보간

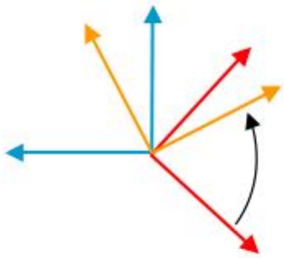
- Example



두 개의 프레임이 위와 같이 정의되어 있다고 가정하자. 빨간축은 -45도로 회전한것이고, 파란 축은 90도로 회전한것이다.

보간

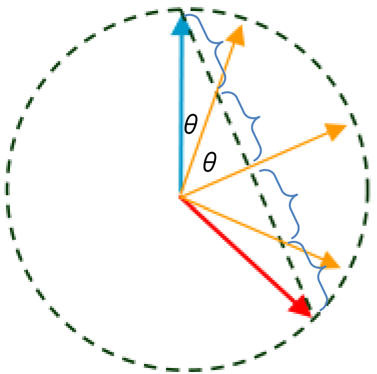
- Example



이 두 회전 사이를 보간해 보자.
일단 x, y 축 중에 x축을 대상으로 해 보면…

2D Matrix: 보간(Interpolation)

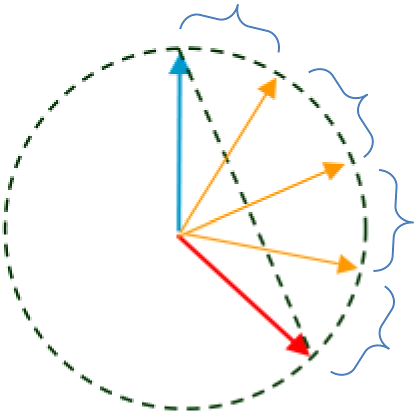
- 만일 x축만 보면…



만일 두 개의 벡터를 보간한다고 가정하자(red and blue), 만일 이 두 끝점일 잇는 선분을 4개의 같은 길이로 나눈다고 가정한다면 나눈 길이는 같지만, 각도를 다르다. 우리가 보간으로 원하는 것은 같은 각도로 구분하는 것임

2D Matrix: Interpolation

- arc를 Subdivide 해야 함



- 이를 Spherical linear interpolation, or slerp이라고 함
  - arc of rotation 가 같게 subdivide됨.

2D Matrix: 보간(Interpolation)

- 어떻게 slerp를 수행하나?
- 기본 아이디어: 두 position에 대한 operations을 orientation에 대해 수행하도록 변경함

position	orientation
$x+y$	$\Rightarrow xy$
$x-y$	$\Rightarrow xy^{-1}$
ax	$\Rightarrow x^{a}$

회전에 대해 이런 식으로 바뀌어야 함

오리엔테이션을 위해 새 개의 새로운 operation(연산자)를 정의

아주 중요!!!!!

2D Matrix: 보간(Interpolation)

- 선형 보간 LERP
  - $(x_1 - x_0)t + x_0$
- Gives SLERP formula
  - $(M_1 M_0^{-1})^t M_0$

x0, x1대신 M0, M1대입

$$x + y \Rightarrow xy$$
$$x - y \Rightarrow xy^{-1}$$
$$ax \Rightarrow x^a$$

순서가 중요함 ! – 원래는 x+y = y+x이나 오리엔테이션일 경우에는 다름. x+y = xy 임 (! = yx)
또한 x−y = xy^{-1} (! = $y^{-1}x$)

- 이 수식에서 M^t (계승) 은 어떻게 계산하나? $(M_1 M_0^{-1})^t M_0$
  - 2D rotation simpler:
    - $(M_\theta)^t = M_{t\theta}$

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \rightarrow \begin{bmatrix} \cos(t\theta) & -\sin(t\theta) \\ \sin(t\theta) & \cos(t\theta) \end{bmatrix}$$

2D Matrix: 보간 Interpolation

- SLERP process

- 1. 곱셈 계산
  - $M = M_1 M_0^{-1}$
- 2. 각도 계산
  - $\theta = \mathbf{atan2}(M_{01}, M_{00}) \Rightarrow M_{01} = \sin, M_{00} = \cos$
- 3. 마지막 계산 (t는 주어짐)
  - $M_{t\theta} M_0$

$$(M_1 M_0^{-1})^t M_0$$

$$M=\begin{array}{cc} M_{00} & M_{01} \\ \end{array}\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

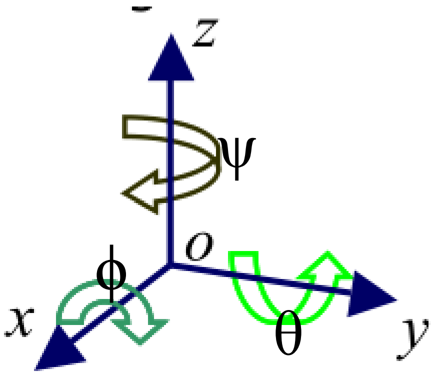
3D 행렬

- 2D 행렬과 유사 함 (한 차원의 확대)
  - 행렬의 각 row는 변환된 축에 해당 됨
  - 벡터를 회전하기 위해서는 3D회전행렬*벡터 을 수행 하면 됨
  - 결합을 위해서는 두 행렬을 곱함(곱해지는 순서가 중요)
- 하지만 LERP는 문제를 여전히 가짐
- Slerp 또한 2차원보다 더 복잡해짐

Euler Angle

Euler Angle

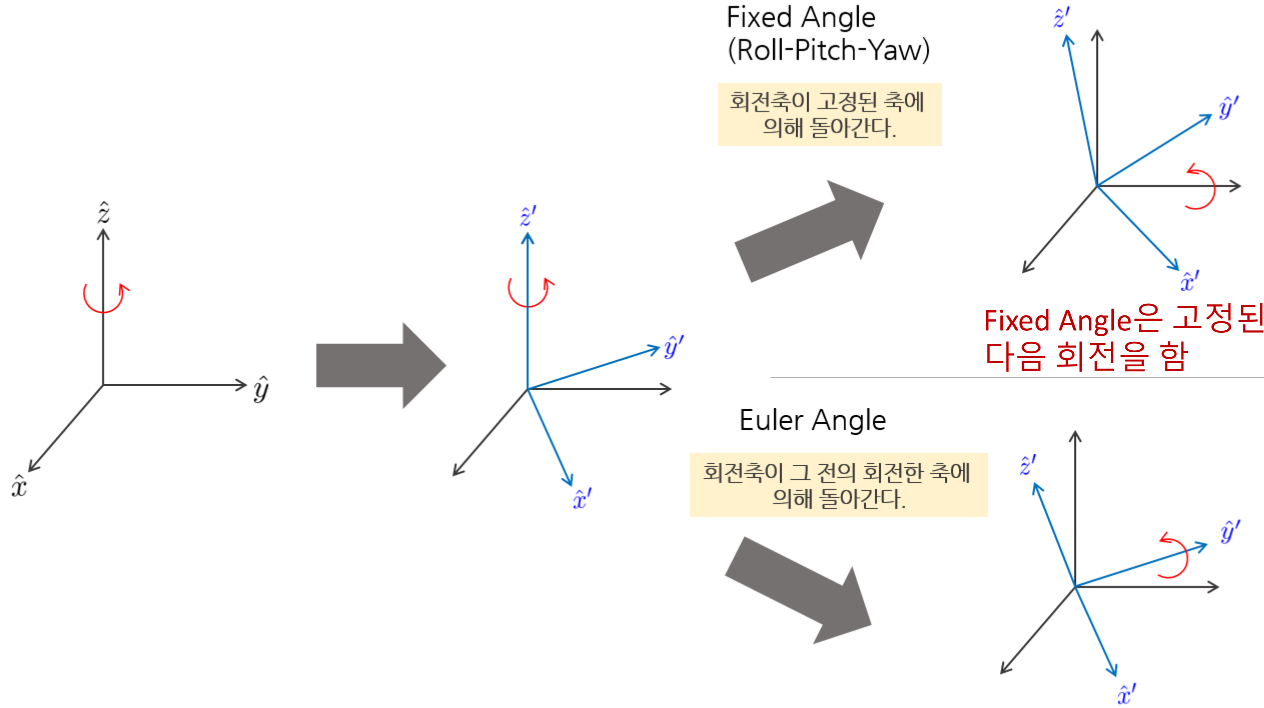
- 행렬 표현에 사용된 3축의 각도를 (ϕ,θ,ψ)로 표현
- 3차원 좌표 계에서 3개의 축에 대한 ordered rotations 을 통해 orientation을 나타 냄
  - 예) x -> y ->z
    - x축으로 rotation결과 위에 y축으로 rotation하고 이 결과 위에 z축으로 rotation함



- Order가 중요하며 교환법칙이 성립하지 않음

- Euler angles 은 2D angle 과 비슷하나 3개의 축에 대해 회전한다는 점에서 다르다.

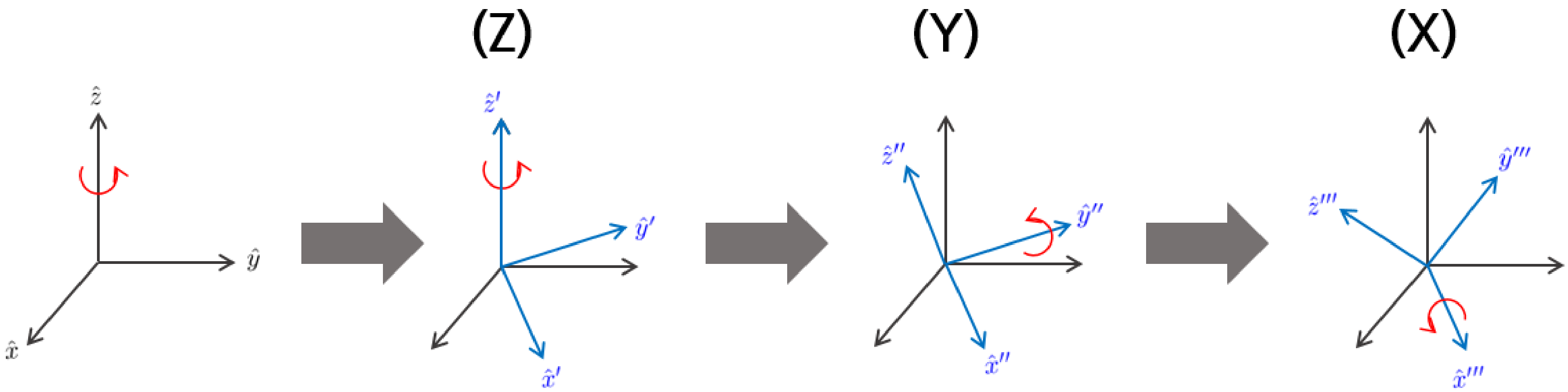
Euler angle vs Fixed angle



Fixed Angle은 고정된 축을 기준으로 다음 회전을 함

오일러 각도는 회전한 축을 기준으로 다음 회전을 함

Euler Angle의 예



ZYX의 경우

Euler Angle과 Fixed Angle의 변환 법

- Rotation 순서를 바꾸면 같아 짐
  - 예) x-y-z Euler Angle == z-y-x Fixed Angle
  - 왜? 수학적으로 증명이 되나, 복잡하므로 생략..

- 논문이나 분야에 따라 Euler Angle과 Fixed Angle을 혼용하는 경우도 있음
  - 예) 로보틱스
  - 어차피 서로간의 변환이 가능하므로..

Euler Angle 문제 점

- 결합이 어려움
  - 최선의 방법:
    - 3차원 로테이션 행렬로 변환 (Euler에서 행렬로 변환식이 있음)
    - 행렬끼리의 곱함
    - 결과 행렬로 부터 Euler angle로 다시 변환
- “gimbal lock” 문제
  - 일련의 rotation을 수행 한 뒤에 때에 따라, 두개의 축이 align됨. 이 경우에는 Degree of Freedom하나를 잃어버림

https://www.youtube.com/watch?v=zc8b2Jo7mno

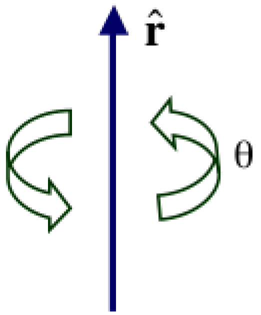
Gimbal Lock

- 다른 값이 같은 orientation을 의미할 수 있음
- Example with z-y-x fixed angles:
  - (90, 90, 90) = (0, 90, 0)
- Why? y축으로 Rotation of 90˚ 는 x 축과 z축을 align해 버림

Axis angle

Axis Angle

- 하나의 회전 벡터를 명시하며, 항상 ccw(counter-clock-wise) 방향으로 회전한다고 가정 함
  - 임의의 축으로 회전함.
  - 임의의 축에서 얼마나 회전하는 지를 나타내는 값(스칼라)이 필요



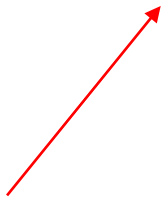
- 보간은 가능하나, 결합은 어려움
  - 결합을 위해서는 행렬로 변환이 필요

Axis angle

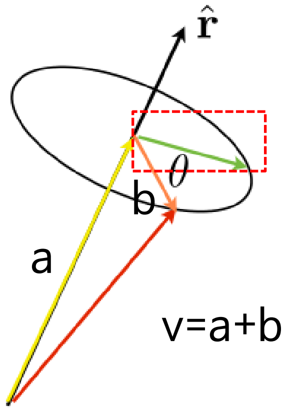
- 회전 법칙
  - 어떤 포인트 P를 r 회전 축으로 θ만큼 회전 시키려면..
  - $R(\mathbf{p}, \hat{\mathbf{r}}, \theta) = cos\theta \cdot \mathbf{p} + (1 - cos\theta)(\mathbf{p} \cdot \hat{\mathbf{r}})\hat{\mathbf{r}} + \sin(\theta)\hat{\mathbf{r}} \times \mathbf{p})$
    - 복잡함... (Rodrigues Rotation Theorem)

- Axis angle은 완전한 포맷이라기 보다 중간 단계로 많이 사용됨.
  - convert to axis-angle, manipulate angle or axis, convert back

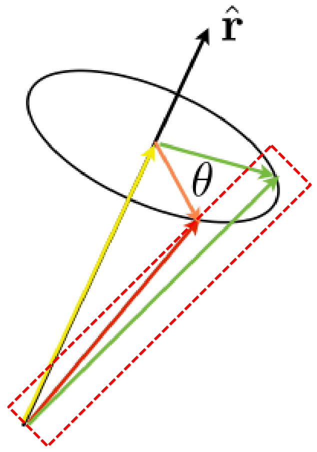
Axis angle를 이용한 벡터 회전



회전하고자 하는 벡터가
있다고 가정하면



이 벡터(v)를 회전축(r)에 평행한 벡터(a)와
나머지 벡터(b)로 decompose가능.
나머지 벡터 b는 항상 회전축에 직각이며,
평면을 생성, 이 평면상에 θ만큼 2차원 회전 하면 됨



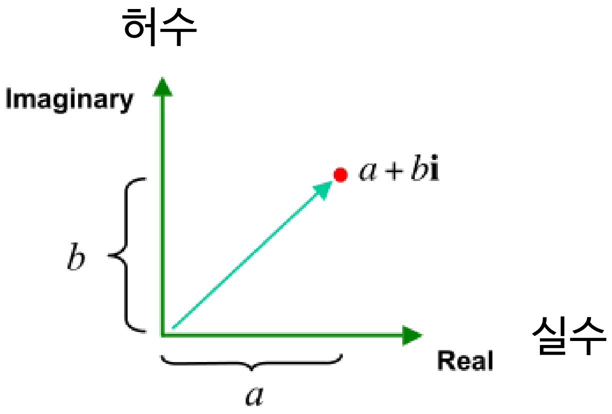
최종 로테이션된 벡터

복소수

Complex Number

2D: 복소수(Complex Number)

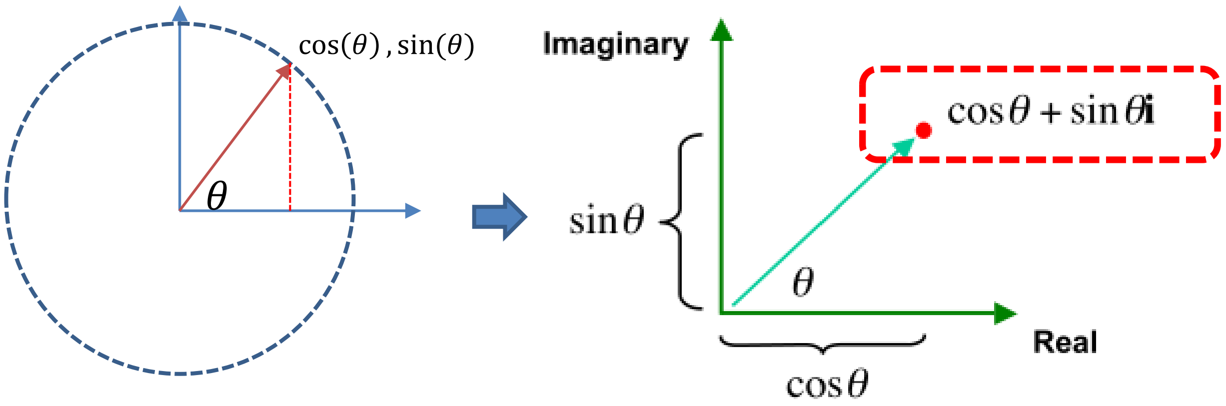
- 복소수
  - $a + b\mathbf{i}\ \ where\ \mathrm{i} = \sqrt{-1}$



  - 두 복소수의 결합 $= (a + b\mathbf{i})(c + d\mathbf{i}) = (ac - bd) + (bc + ad)\mathbf{i}$
    - 결합 계산에 대한 다른 의미=> 내적(dot product) $(a, \boldsymbol{b}) \cdot (\boldsymbol{c}, \boldsymbol{d}) = (\boldsymbol{ac} - \boldsymbol{bd}, \boldsymbol{bc} + \mathbf{ad})$

2D: 복소수(Complex Numbers)

- 만일 복소수 길이를 1로 제한한다면 -> $cos\theta - isin\theta$



복소수 성질

- $z = x + yi = r[\cos\theta + i sin\theta] = re^{i\theta} \Rightarrow e^{i\theta}$ $(r = x^2 + y^2 = \cos\theta^2 + \sin\theta^2 = 1)$
- $z_1 z_2 = (x_1 + y_1 \mathrm{i})(x_2 + y_2 \mathrm{i}) = r_1 e^{i\theta_1} r_2 e^{i\theta_2} = r_1 r_2 e^{i(\theta_1+\theta_2)} = e^{i(\theta_1+\theta_2)}$
  - 이것의 의미는? 결합은 단순히 θ 끼리 더하면 끝
- 복소수 conjugate $z^* = re^{-i\theta} = x - y\mathbf{i}$
- 복소수 Inverse $z^{-1} = \frac{1}{r} e^{-i\theta} = e^{-i\theta}$
- $(z)(z^*) = r^2,\ (z)(z^{-1}) = 1$
- If r=1, then $z^* = z^{-1}$

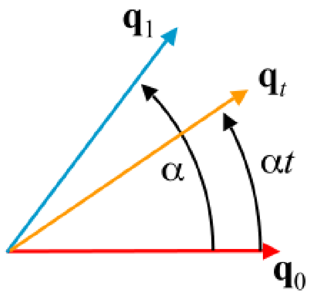
2D: 복소수(Complex Numbers)

- 만일 일반 복소수와 길이가 1인 복소수를 곱한다면
  - $(x + y\mathbf{i})(cos\theta + sin\theta\mathbf{i}) = (xcos\theta - ysin\theta) + (xsin\theta + ycos\theta)\mathbf{i}$

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

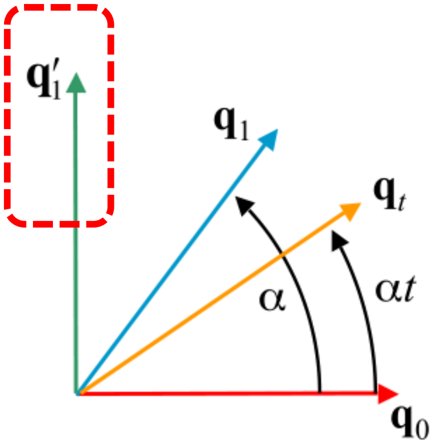
회전행렬을 의미!

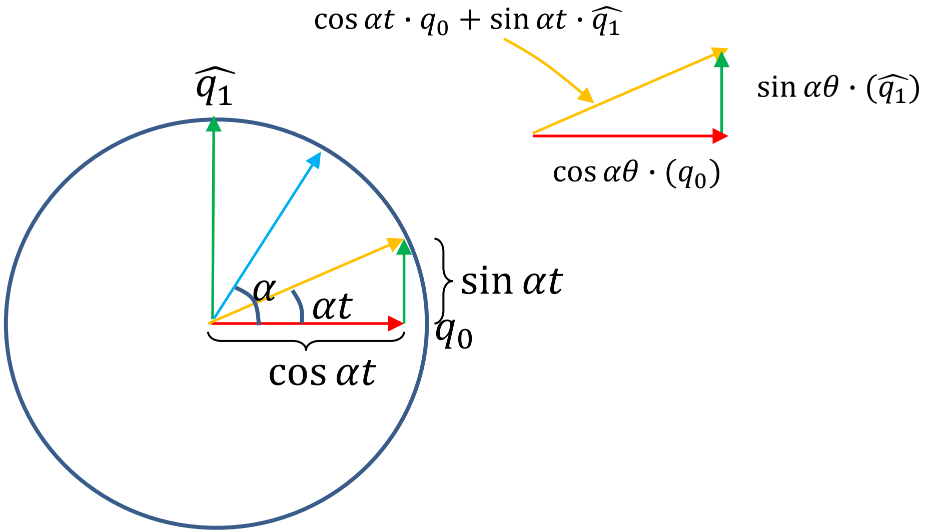
- 결합
  - $(cos\theta + sin\theta\mathbf{i})(cos\emptyset + sin\emptyset\mathbf{i}) = (cos(\theta + \emptyset)) + (sin(\theta + \emptyset))\mathbf{i}$
- 보간
  - 두 개의 복소수 q0 와 q1 가 있다고 가정하고, 이를 t를 이용해 보간 한다고 하자. 두 복소수 사이의 각도를 α라고 한다면, 우리가 원하는 것은 α t에 해당하는 복소수를 찾는 것이다.
    - α는 어떻게 구하나?
      - 벡터 사이의 각도 구하듯이 내적을 구한 후 acos()



2D: 복소수 Complex Numbers 보간

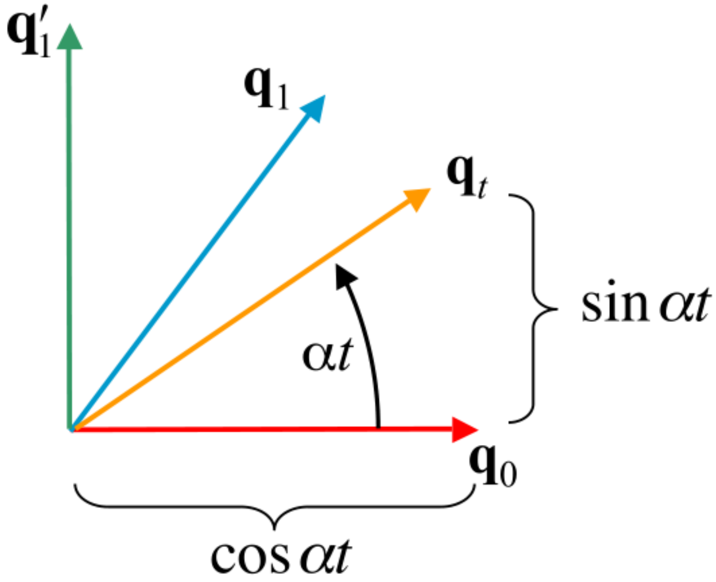
- 먼저 q_0에 직각 벡터에 해당하는 q_1'를 구한다. (basis 프레임(기본 축) 에 해당)




$\cos\alpha t \cdot q_0 + \sin\alpha t \cdot \widehat{q_1}$
$\sin\alpha\theta \cdot (\widehat{q_1})$
$\cos\alpha\theta \cdot (q_0)$
$\widehat{q_1}$
α
αt
$\sin\alpha t$
q_0
$\cos\alpha t$

2D: 복소수 Complex Numbers 보간

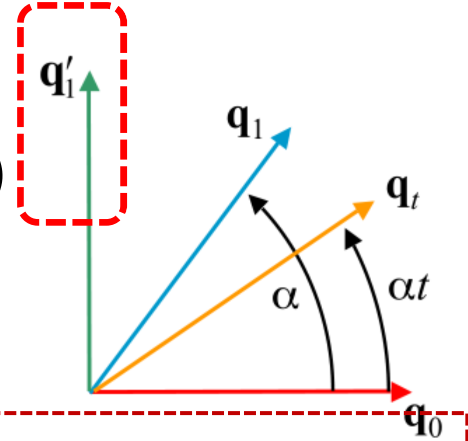
- 새로운 q_t 를 이 새 축을 이용해서 생성 할 수 있음
  - SLERP
  - $\mathbf{q_t} = cos\alpha t\mathbf{q_0} + sin\alpha t\mathbf{q}_1'$



- 어떻게 하면 $\mathbf{q}_1'$를 구할 수 있나?
  - $\mathbf{q_0}$를 90도로 회전하면?

2D: 복소수 Complex Numbers 보간

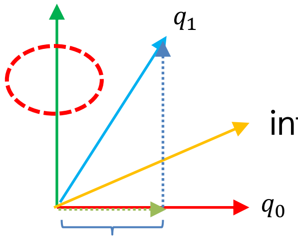
- $\mathbf{q}_1' = \frac{\mathbf{q}_1 - (\mathbf{q}_0 \cdot \mathbf{q}_1)\mathbf{q}_0}{\|\mathbf{q}_1 - (\mathbf{q}_0 \cdot \mathbf{q}_1)\mathbf{q}_0\|}$ (유도과정 다음 슬라이드)
- 이를 단순화 시키면 $\boxed{\mathbf{q}_1' = \frac{\mathbf{q}_1 - \cos\alpha\,\mathbf{q}_0}{\sin^2\alpha}}$ (유도과정 다음)
- Slerp 공식을 유도하면
  - $\mathbf{q}_t = \cos\alpha t\,\mathbf{q}_0 + \sin\alpha t\,\boxed{\mathbf{q}_1'}$에 대입하면
  - $\mathbf{q}_1' = \frac{\mathbf{q}_1 - \cos\alpha\,\mathbf{q}_0}{\sin^2\alpha}$



$$\boxed{\mathbf{q}_t = \frac{\sin(1-t)\alpha}{\sin\alpha}\mathbf{q}_0 + \frac{\sin t\alpha}{\sin\alpha}\mathbf{q}_1}$$

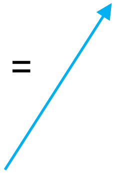
$\mathbf{q}_t = \mathbf{q}_0(\mathbf{q}_0^{-1}\mathbf{q}_1)^t$ 〈- 위 식과 동일

직각 축 구하기



 $= (q_0 \cdot q_1) * q_0$

 $= q_1 - (q_0 \cdot q_1) * q_0$ normalize $\Rightarrow \dfrac{q_1 - (q_0 \cdot q_1) * q_0}{\|q_1 - (q_0 \cdot q_1) * q_0\|}$

=  – 

$$\frac{q_1 - (q_0 \cdot q_1)q_0}{\|q_1 - (q_0 \cdot q_1)q_0\|}$$

$$= \frac{q_1 - (cos\alpha)q_0}{\sqrt{(q_1 - (cos\alpha)q_0) \cdot (q_1 - (cos\alpha)q_0)}}$$

$$= \frac{q_1 - (cos\alpha)q_0}{\sqrt{(q_1 \cdot q_1) + (q_0 \cdot q_0)cos^2\alpha - 2(q_0 \cdot q_1)cos\alpha}}$$

$$= \frac{q_1 - (cos\alpha)q_0}{\sqrt{1 + cos^2\alpha - 2(q_0 \cdot q_1)cos\alpha}}$$

$$= \frac{q_1 - (cos\alpha)q_0}{\sqrt{(1 - cos\alpha)^2}}$$

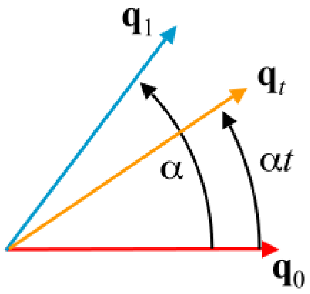
$$= \frac{q_1 - (cos\alpha)q_0}{\sqrt{sin\alpha^2}}$$

$$= \frac{q_1 - (cos\alpha)q_0}{sin\alpha}$$

2D: 복소수 Complex Numbers 보간

- SLERP 도 이상적인 것은 아님..

$$\mathbf{q_t} = \frac{\sin(1-t)\alpha}{sin\alpha}\mathbf{q_o} + \frac{sint\alpha}{sin\alpha}\mathbf{q_1}$$



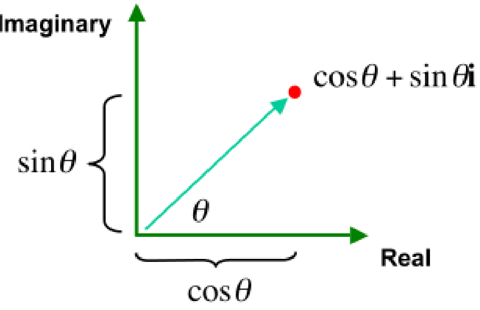
- 이 공식 또한 교환법칙이 성립하지 않음
- 또한 α, sin α, sin α t 를 구하는 것은 계산 량이 많음
- α → 0 갈 수록 에러가 커짐

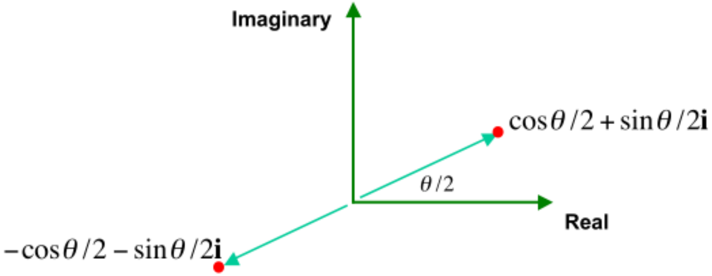
복소수의 half-angle폼

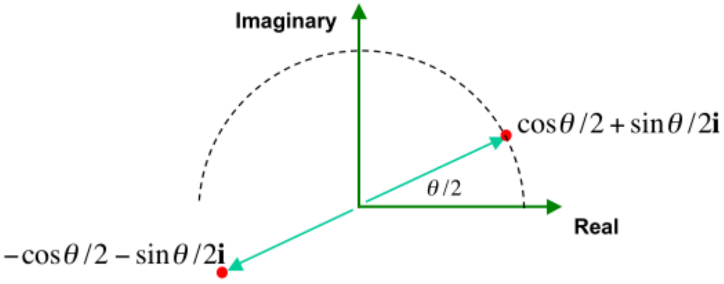
- 이제 복소수의 Angle을 반으로 나누어서 적용하는 경우를 생각해 보자.
- Half-angle form
  - 원래 Form : $q = (cos\theta + sin\theta\mathbf{i})$
  - Half-angle Form : $q = \left(\cos\boxed{(\frac{\theta}{2})} + \sin\boxed{(\frac{\theta}{2})}\mathbf{i}\right)$
- 벡터 p의 q에 대한 rotation은 아래와 같이 표현됨
  - Rot(p,θ) = q*p*q
    - 반 절 씩 나누어서 rotation한다고 생각하면 됨

복소수의 half-angle폼의 특징

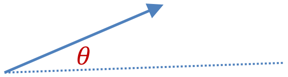
- Negative가 같은 회전을 의미함 (왜? 다음 슬라이드)

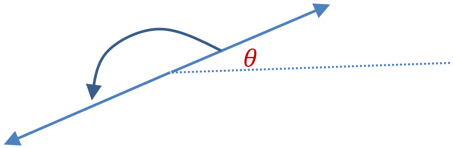






보통은 원래 half-angle과 negative half-angle사이의 반원을 이용해 모든 회전을 표현함
(이 둘 사이가 180도 같지만, 사실은 360도)





$$\color{red}{cos\theta + \mathbf{i}sin\theta} = \cos(\theta + \color{red}{180}) + \mathbf{i}(\sin(\theta + 180))$$
$$\Rightarrow \cos(\theta + 180) = cos\theta\cos(180) - sin\theta\sin(180) = -cos\theta$$
$$\Rightarrow \mathbf{i}(\sin(\theta + 180)) = \mathbf{i}(sin\theta\cos(180) + cos\theta\sin(180)) = \mathbf{i}(-sin\theta)$$
$$\Rightarrow \color{red}{-cos\theta - \mathbf{i}sin(\theta)}$$

$\cos(180) = -1 \quad \sin(180) = 0$

$$\therefore \sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \pm \cos(\alpha)\sin(\beta)$$
$$\therefore \cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \mp \sin(\alpha)\sin(\beta)$$

사원 수(Quaternion)

- 기존의 복소수에 대한 확장 개념
  - *a+bi* 가 확장되어서 w+xi+*y*j+zk
- 4 개의 값이 필요
  - (x, y, z,w)
- 이를 하나의 scalar와 3차원 벡터 pair로 생각할 수 있음
  - (w, (**v**))

- 사원수는 복소수의 half-angle를 3차원으로 확장한 개념으로 이해하는것이 쉬움

Unit Quaternion = Axis Angle +
Half-angle Complex number

사원 수 회전이란?

- 길이가 1은 사원수를 Unit quaternion라고 하며 이 unit quaternion만이 Orientation을 표현 함
- Unit Quaternion (w, x, y, z) ⇒ $\sqrt{w^2 + x^2 + y^2 + z^2} = 1$
- Unit Quaternion으로 바꾸기 위해서는 길이로 나누어야 함 (벡터 normalize와 유사)
- w 는 회전각 θ 을 의미하며 실제 들어가는 값은 아래와 같다
  - w = cos(θ/2)
- x, y, z 는 normalize된 회전축(rotation axis) r 을 의미하며 실제 들어가는 값은 아래와 같다
  - (x,y,z) = v = sin(θ/2) * r
- 사원수 표현을 (w, v) 로 표현 함
- 다른 표현으로는 수정된 axis-angle라고 할 수 있음

Example) Creating Rotation Quaternion

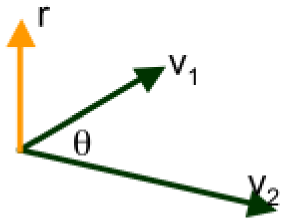
- z축으로 90도 회전하는 unit quaternion을 표현한다면..
  - w = cos(45) = $\sqrt{2}/2$
  - x = 0 * sin(45) = 0
  - y = 0 * sin(45) = 0
  - z = 1 * sin(45) = $\sqrt{2}/2$
  - q = ($\sqrt{2}/2$, 0, 0, $\sqrt{2}/2$)

w = cos(θ/2)

(x,y,z) = v = sin(θ/2) * r

Example) Creating Rotation Quaternion

- 벡터 v1이 있을 때 또 다른 벡터 v2로 회전하고 싶다고 하자. 이 회전을 unit quaternion으로 나타내면.. (즉, v1과 v2 사이 각에 대한 quaternion은?)
  - 회전 축 *r* 과 각도 θ을 구해야 함
    - r = v1 x v2 (외적 cross product)
    - θ = acos(v1 · v2) (내적 dot product, v1, v2 은 반드시 normalize되어야 함)
  - 위 공식에 넣으면
    - w = cos(θ/2)
    - (x,y,z) = v = sin(θ/2) r

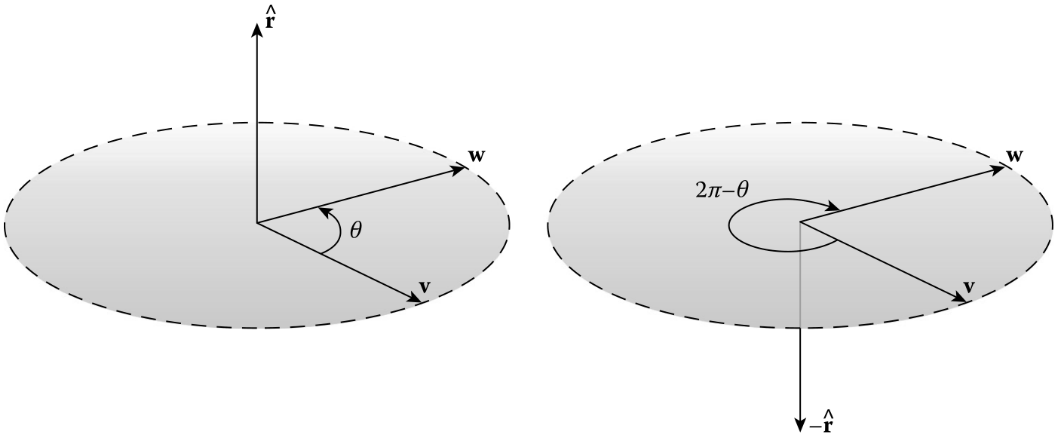


Special cases

- 만일 두 벡터 v1과 v2가 거의 평행 하다면 위 공식에 오류가 있음. 이를 해결하고자..
  - v1 과 v2 normalize
    - r = v1 x v2
    - s = $\sqrt{2(1 + v1 \cdot v2)}$ (내적)
  - $q = (2s, r/s)$, 만일 v1, v2가 서로 거의 평행 하다면 이 표현이 더 안정적임

Negating 사원수

- q = -q
- 원래 사원수와 모든 값을 –1을 곱한 사원수는 같음



r을 이용해서 θ만큼 로테이션 하는 것은 - r에서 $2\pi-\theta$만큼 로테이션 하는것과 동일

Quaternion multiplication (결합)

- 일반적으로 복소수 곱셈보다는 복잡 함
- Take $q_0 = (w_0, \mathbf{v}_0)$ $q_1 = (w_1, \mathbf{v}_1)$
  - $q_0 q_1 = (w_1 w_0 - \mathbf{v}_1 \cdot \mathbf{v}_0, w_1 \mathbf{v}_0 + w_0 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_0)$ //어디서 많은 본 듯한?
- 교환 법칙이 성립 안됨 (왜? 식에 X(외적) 이 있으므로)
  - $q_0 q_1 \neq q_1 q_0$
- 결합
  - 두 개의 quaternion이 있을 때, 결합은 단순히 곱하면 됨. 하지만 순서가 중요함 (이유는 곱셈에 외적이 존재 하므로)

Multiplication 구현 예..

$$q_0 q_1 = (w_1 w_0 - \mathbf{v}_1 \cdot \mathbf{v}_0, \; w_1 \mathbf{v}_0 + w_0 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_0)$$

```
q_w = q1_w *q0_w - q1_x *q0_x - q0_y*q0_y - q1_z*q0_z;

q_x = q1_w *q0_x + q1_x *q0_w + (q1_y *q0_z - q1_z*q0_y);
q_y = q1_w *q0_y + q1_y *q0_w + (q1_z *q0_x - q1_y*q0_z);
q_z = q1_w *q0_z + q1_z *q0_w + (q1_x *q0_y - q1_y*q0_x);
```

```
q1_x  q1_y  q1_z  q1_x  q1_y

q0_x  q0_y  q0_z  q0_x  q0_y
```

Identity and Inverse

- 항등 사원수(Identity quaternion) ->(1, 0, 0, 0) : 일종의 identity matrix 같은의미
  - no rotation
- Conjugate
  - 각도는 같고, 회전벡터가 반대인 quaternion
  - q = (w,v) -> q* = (w, -v)
- q^{-1} 는 역 사원수를 의미.
  - $q\ q^{-1}$ 는 항등 사원수(1,0,0,0) 를 반환 함
- |q| = q의 길이 =$\sqrt{qq^*} = \sqrt{w^2 + x^2 + y^2 + z^2}$
- 역사원수는 같은 각도이나 opposite vector
  - Inverse of $q^{-1} = (w, \mathbf{v})^{-1} = \frac{q^*}{|q|^2}$ (만일 unit quaternion이라면 (q^{-1} = q^*))

Example) 다음을 계산 하시 오

- $q = (2, 2, 0, 1)$: (w, x, y, z)
- $|q| = ?$
- normalize q = ? (q' = normalized q)
- $q'^* = ?$ (Conjugate)
- $q'^{-1} = ?$ (역 사원수)
- $q' * q'^{-1} =$

벡터를 사원수를 이용해 회전시키기

- vector p, 와 quaternion q 있을 경우
- p 를 quaternion으로 표현 할 수 있음 (0, p)
- p 의 q 에 대한 회전은 $\mathbf{q}\ \mathbf{p}\ \mathbf{q}^{-1}$로 나타낼 수 있음 (이것도 어디서 많은 본 듯한?)
  - half-angle 복소수와 유사
- Vector p 의 unit quaternion (w, v)에 대한 회전은 아래 수식으로 정리 됨
  - q = (w, **v**)
  - $\mathbf{p}' = (1 - w^2)\mathbf{p} + 2(\mathbf{v} \cdot \mathbf{p})\mathbf{v} + 2w(\mathbf{v} \times \mathbf{p})$

Example) 다음을 계산 하시 오

- 벡터 p(1,1,1)을 회전 축 (1,0,-1)을 이용해 60도 만큼 회전하고 싶다. 회전 후 벡터 p'를 계산 하시오
  - 1. 먼저 unit quaternion q을 계산
    - r = (1,0,-1), θ=60
    - q =?
  - 2. p' = q p q^{-1} (사원 수 곱셈방법 : $\mathbf{p}' = (1 - w^2)\mathbf{p} + 2(\mathbf{v} \cdot \mathbf{p})\mathbf{v} + 2w(\mathbf{v} \times \mathbf{p})$)

벡터를 연속적으로 회전하려면..

- 결합
  - 백터 p를 q0에 대해 먼저 적용 한 후에, q1를 나중에 적용하는 것은 q1q0를 적용한 것과 동일한 효과
  - $q_1 \cdot (q_0 \cdot p \cdot q_0^{-1}) \cdot q_1^{-1} = \boxed{(q_1 \cdot q_0)} \cdot p \cdot \boxed{(q_1 \cdot q_0)}^{-1}$
- 순서가 중요 : right-to-left

Quaternion 보간

- 복소수 slerp과 동일

$$\mathbf{q_t} = \frac{\sin(1-t)\alpha}{sin\alpha}\mathbf{q_0} + \frac{sint\alpha}{sin\alpha}\mathbf{q_1}$$

- 주의 할 점
  - q 와 −q 는 같은 회전을 의미 함
  - 보간 할 때 주의 해야 함.
    - 보간 대상이 되는 두 quaternion에 대한 내적을 구해서 <0 라면 q대신에 -q를 이용

Interpolation 함수 추가

```
static Quaternion Quaternion::slerp(const Quaternion& q1, const Quaternion& q2, float u) {
Quaternion result;

float dotProd = q1.r * q2.r + q1.i * q2.i + q1.j * q2.j + q1.k * q2.k;
float theta;

if (dotProd < 0) {
   theta = acos(-dotProd);
}
else {
   theta = acos(dotProd);
}

float sinTheta = sin(theta);

if (fabs(sinTheta) < TRIG_ANGLE_TOL) {
   result=q1;
   return(result);
}
```

두 quaternion에 대한 내적을 구해서 <0 라면 q대신에 -q를 이용

$$\mathbf{q_t} = \frac{\sin(1-t)\alpha}{sin\alpha}\mathbf{q_0} + \frac{sint\alpha}{sin\alpha}\mathbf{q_1}$$

sin(theta)가 0에 가까워질 만큼 작으면 0으로 나누는 것이기 때문에
에러가 생김 이를 방지

```
float coeff1 = sin((1.0 - u) * theta) / sinTheta;
float coeff2 = sin(u * theta) / sinTheta;
if (dotProd < 0) {
        result.r = -coeff1 * q1.r + coeff2 * q2.r;
        result.i = -coeff1 * q1.i + coeff2 * q2.i;
        result.j = -coeff1 * q1.j + coeff2 * q2.j;
        result.k = -coeff1 * q1.k + coeff2 * q2.k;

}
else {
        result.r = coeff1 * q1.r + coeff2 * q2.r;
        result.i = coeff1 * q1.i + coeff2 * q2.i;
        result.j = coeff1 * q1.j + coeff2 * q2.j;
        result.k = coeff1 * q1.k + coeff2 * q2.k;
}

return(result);
}
```

두 quaternion에 대한 내적을 구해서 <0 라면 q대신에 -q를 이용

$$\mathbf{q_t} = \boxed{\frac{\sin(1-t)\alpha}{sin\alpha}}\mathbf{q_0} + \boxed{\frac{sint\alpha}{sin\alpha}}\mathbf{q_1}$$

변환 공식 (사원수 -> 행렬)

$$\mathbf{q} = (\cos(\theta/2), \sin(\theta/2)\vec{a}) = (w, (x, y, z))$$

$$R_{\mathbf{q}} = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL이나 DirectX, 혹은 게임엔진에서는 빠른 처리를 위해서 행렬로 내부적으로 변경해서 사용 함 (행렬처리를 GPU에서 할 경우 굉장히 빠름)

Quaternion 프로그래밍 연습

제공해준 소스코드를 이용한 Quaternion연습

cyclone을 이용한 quaternion 연습

- 행렬을 이용한 오리엔테이션 표현
  - cyclone::Matrix4
- 3차원 임에도 불구하고 3x3 행렬이 아닌 4x4을 사용하는 이유는?
  - Homogeneous 좌표 계 때문
    - 3차원 좌표 -> 4차원 좌표
      - 3차원 포인트 : (x,y,z,1)
      - 3차원 벡터 : (x,y,z,0)
    - 2차원 좌표 -> 3차원 좌표
      - 2차원 포인트 : (x,y,1)
      - 2차원 벡터 : (x,y,0)

위치이동 행렬

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

회전 행렬 (만일 사원수에서 행렬로 바꾸었다면)

$$R_{\mathbf{q}} = \begin{pmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy & 0 \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx & 0 \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Mover 클래스에 변환을 위한 행렬 추가

```
class Mover
{
public:

Mover(cyclone::Vector3 &p);
~Mover();

cyclone::Matrix4 transformMatrix;

…
```

※이 변환 행렬은 위치이동 뿐 아니라 회전도 모두 가능..

cyclone

```
Mover * a = new Mover(cyclone::Vector3(3, size, 3));

cyclone::Quaternion q;

//사원수에 q의 오리엔테이션과 vector3의 위치이동으로 통해 mat4를 구성
a->transformMatrix.setOrientationAndPos(q, cyclone::Vector3(0, 6, 0));
m_movers.push_back(a);
```

$$M = \begin{bmatrix} & & & 0 \\ & & & 6 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

위치이동과 로테이션 결합된 4x4행렬

```
void Mover::draw(int shadow)
{
GLfloat mat[16];
getGLTransform(mat);  //transformMatrix로 부터 opengl용 행렬로 변경

if (!shadow) {
glPushMatrix();
glMultMatrixf(mat);

glLineWidth(3.0f);
glBegin(GL_LINES);  //오브젝트에 3개축 그림
glColor3f(1, 0, 0);
glVertex3f(0, 0.1, 0);
glVertex3f(0, 10, 0);
glColor3f(0, 1, 0);
glVertex3f(0, 0.1, 0);
glVertex3f(10, 0.1, 0);
glColor3f(0, 0, 1);
glVertex3f(0, 0.1, 0);
glVertex3f(0, 0.1, 10);
glEnd();

glPopMatrix();
glLineWidth(1.0f);
}
```

```
if (shadow) {  //그림자 구분
glColor3f(0.2f, 0.2f, 0.2f);
}
else {

glColor3f(1, 0., 0);
}


glPushMatrix();
glMultMatrixf(mat);

glutSolidCube(3.0f);  //박스그리기
glPopMatrix();


}
```

cyclone의 quaternion 클래스

```
class Quaternion
    {
    public:
       union {
          struct {
             real r; real i; real j; real k;    // r -> w, i,j,k -> (x, y, z)
          };
          real data[4];  //or data[0] -> w, data[1]=x, data[2]=y, data[3]=z

       };
      Quaternion() : r(1), i(0), j(0), k(0) {}  //기본 생성자
      Quaternion(const real r, const real I, const real j, const real k)  //w,x,y,z를 입력하는 생성자
      Quaternion(const Vector3 a, const Vector3 b)  //vector a에서 vector b로 가는 사원 수

      void normalise()  //정규화
      void operator *=(const Quaternion &multiplier) //quaternion곱셈
      void addScaledVector(const Vector3& vector, real scale)  //미분 시 사용
      void rotateByVector(const Vector3& vector) //벡터 vector를 회전
      void inverse() //역사원수
```

Y축으로 60도로 회전하는 사원 수

- (w, x, y, z)
  - w= cos(θ/2)
  - v = (x,y,z) = sin(θ/2) * r (r=회전축)

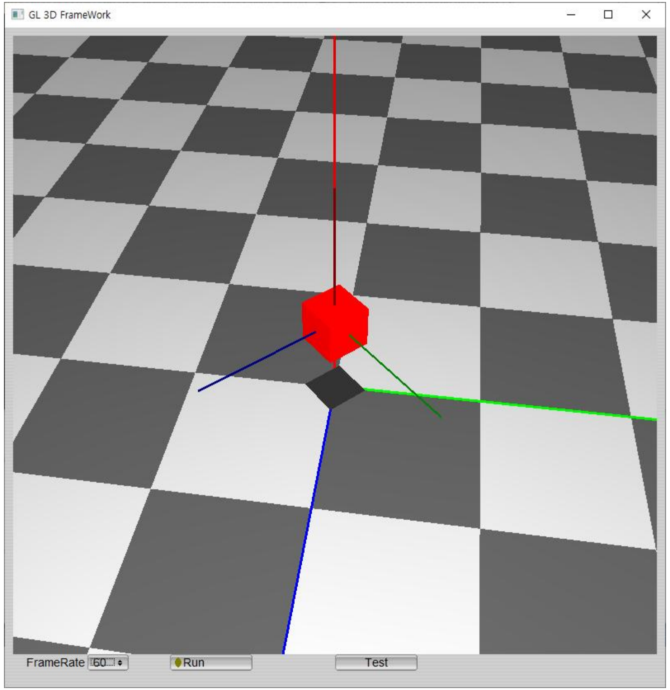
- w = ?
- x,y,z = ?

- 중요한 점 : 반드시 normalize해야 함

θ	0°	30°	45°	60°	90°
$\sin\theta$	0	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$	1
$\cos\theta$	1	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$	0
$\tan\theta$	0	$\frac{\sqrt{3}}{3}$	1	$\sqrt{3}$	undef.

Y축으로 45도로 회전하는 사원 수

```
cyclone::Quaternion q;
const float degrees2Radians = 3.141592f / 180;

//원하는 각도의 1/2값을 넣어야 함
q.r = cos(degrees2Radians * 45.0f*0.5);
cyclone::Vector3 v =
cyclone::Vector3(0,1,0)*sin(degrees2Radians * 45.0f*0.5);
q.i = v.x;
q.j = v.y;
q.k = v.z;

q.normalise();

a->transformMatrix.setOrientationAndPos(q,
cyclone::Vector3(0, 6, 0));
```



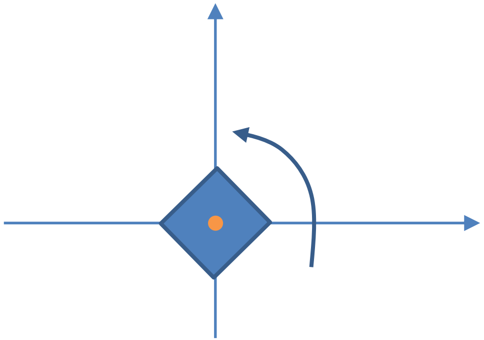
Quaternion 클래스에 생성자 추가

```
Quaternion(const float angle, const Vector3 axis)  //angle : 각도, axis : 회전축
{
    double cosAng = cos(angle / 2.0);
    double sinAng = sin(angle / 2.0);
    double norm = sqrt(axis[0] * axis[0] + axis[1] * axis[1] + axis[2] * axis[2]);

    i = axis[0] / norm;
    j = axis[1] / norm;
    k = axis[2] / norm;

    r = cosAng; //w

    i *= sinAng; //x
    j *= sinAng; //y
    k *= sinAng; //z
}
```

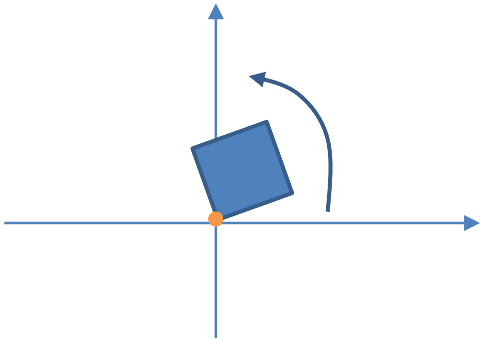
회전 벡터 axis로 angle만큼 회전 하는 사원 수
주의 : angle는 라디안 이어야 함 (아래 상수 이용)

```
const double DEGREES_TO_RADIAN = 3.141592 / 180.0f;
const double RADIANS_TO_DEGREES = 180.0f / 3.141592;
```

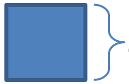
특정 점을 중심으로 회전 하려면?



원래는 원점을 중심으로 회전

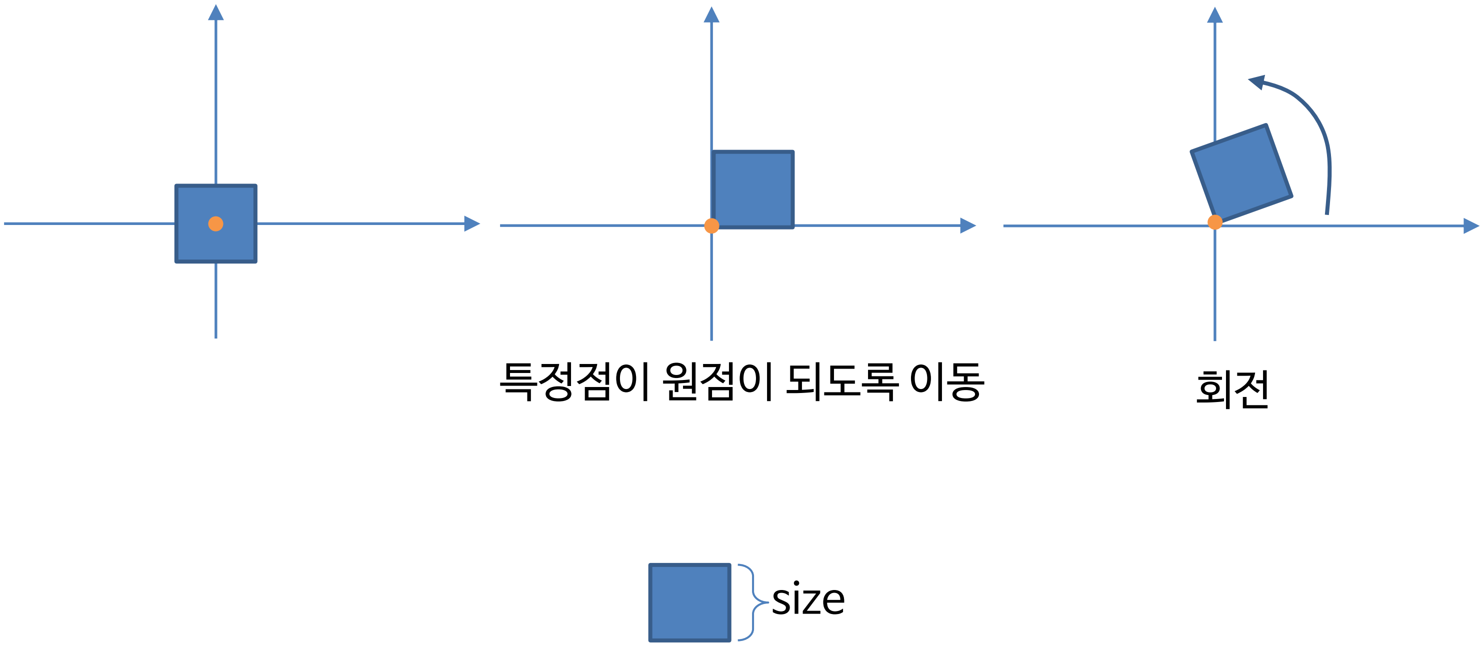


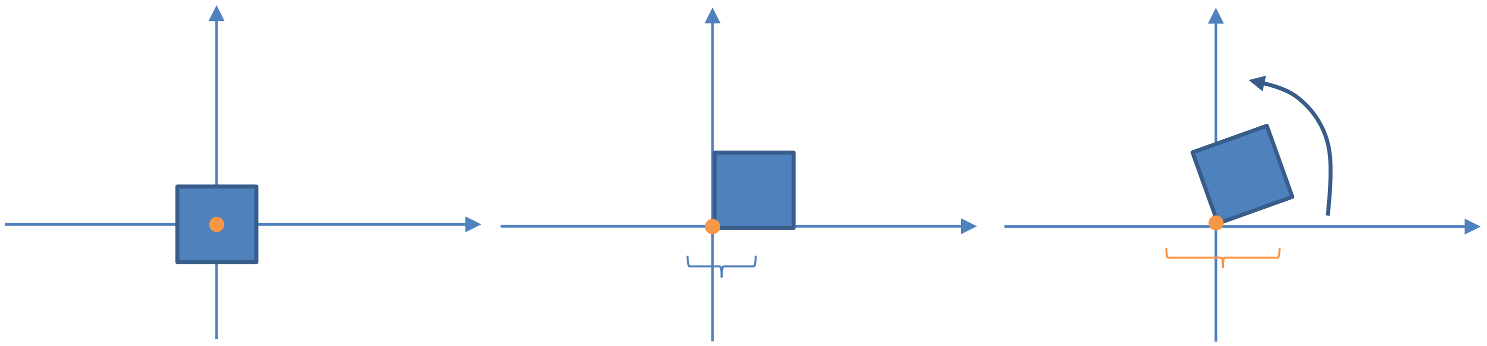
왼쪽 아래 꼭지점을 중심으로 회전하려면



size : 3

특정 점을 중심으로 회전 하려면?

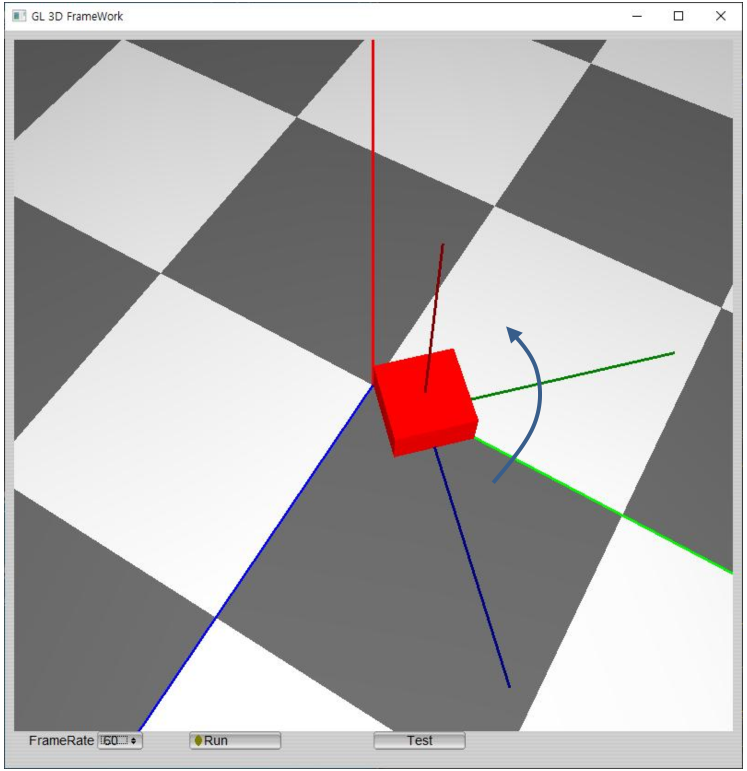




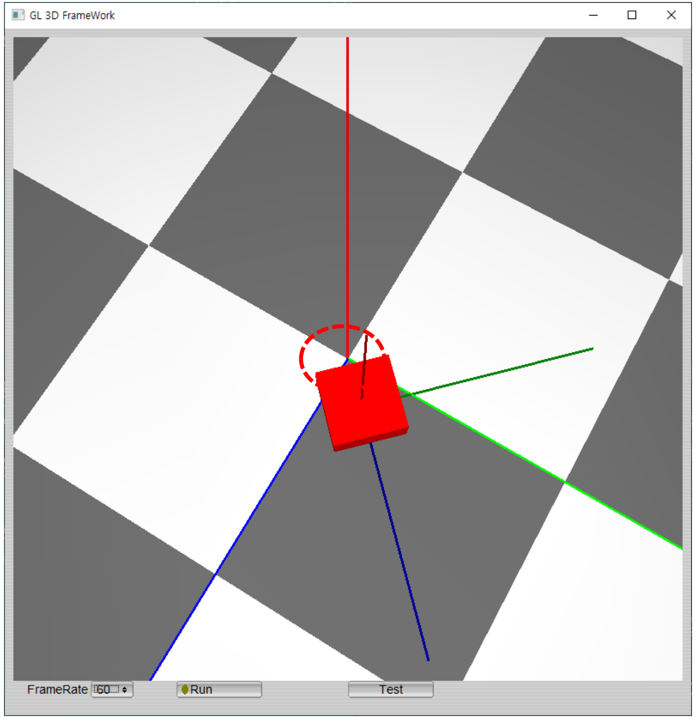
특정점이 원점이 되도록 이동

회전

```
cyclone::Matrix4 m;
m.setOrientationAndPos(cyclone::Quaternion(1, 0, 0, 0), cyclone::Vector3(1.5, 0, 1.5));

cyclone::Quaternion c(-45 * DEGREES_TO_RADIAN, cyclone::Vector3(0, 1, 0));
c.normalise();
cyclone::Matrix4 m2;
m2.setOrientationAndPos(c, cyclone::Vector3(0, 0, 0));

m_moverconnection->m_movers[0]->transformMatrix = m2 * m;
```


GL 3D FrameWork
FrameRate
60
Run
Test

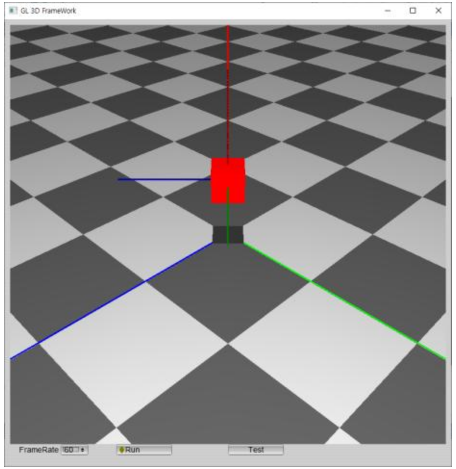
한꺼번에 하면 안되나요?



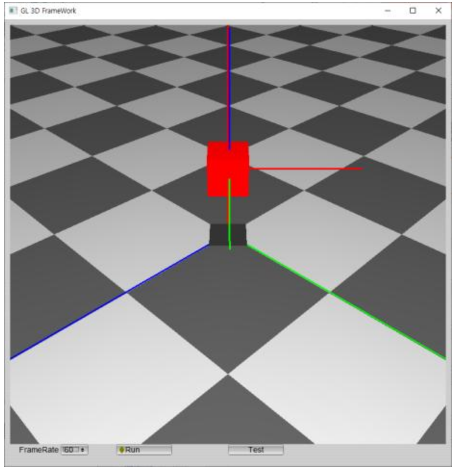

```
cyclone::Quaternion c(-45 * DEGREES_TO_RADIAN, cyclone::Vector3(0, 1, 0));
c.normalise();
cyclone::Matrix4 m2;
m2.setOrientationAndPos(c, cyclone::Vector3(1.5, 0, 1.5));

m_moverconnection->m_movers[0]->transformMatrix = m2;
```

Example) Y축으로 45도로 회전한 후에 x축으로 90도 회전

- Quaternion 두 개를 생성 후에 곱하면 됨
  - 곱해지는 순서가 중요
  - cyclone quaternion클래스의 *= operator를 사용



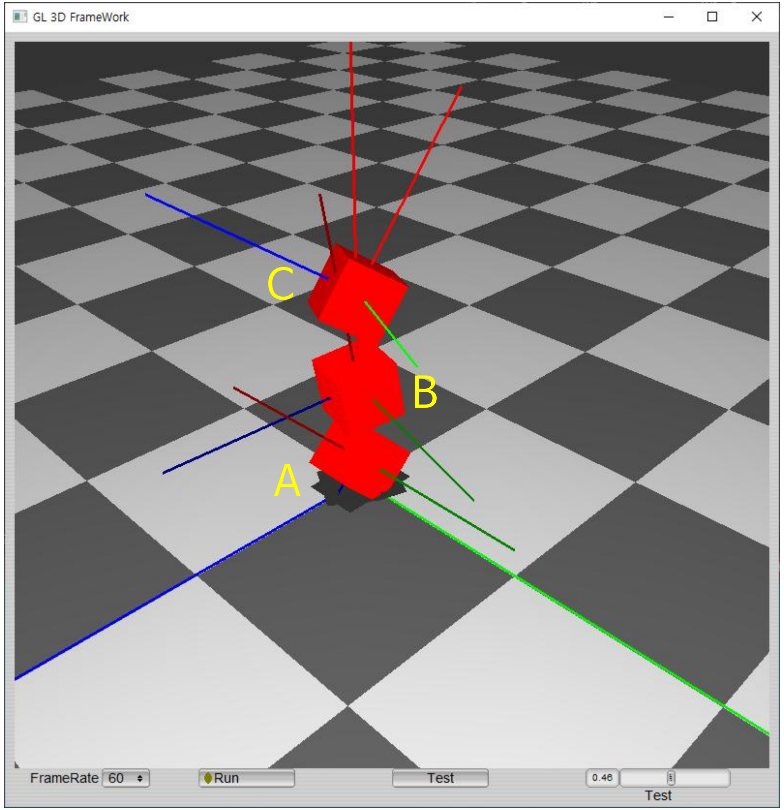
y축(빨간 축)으로 45도



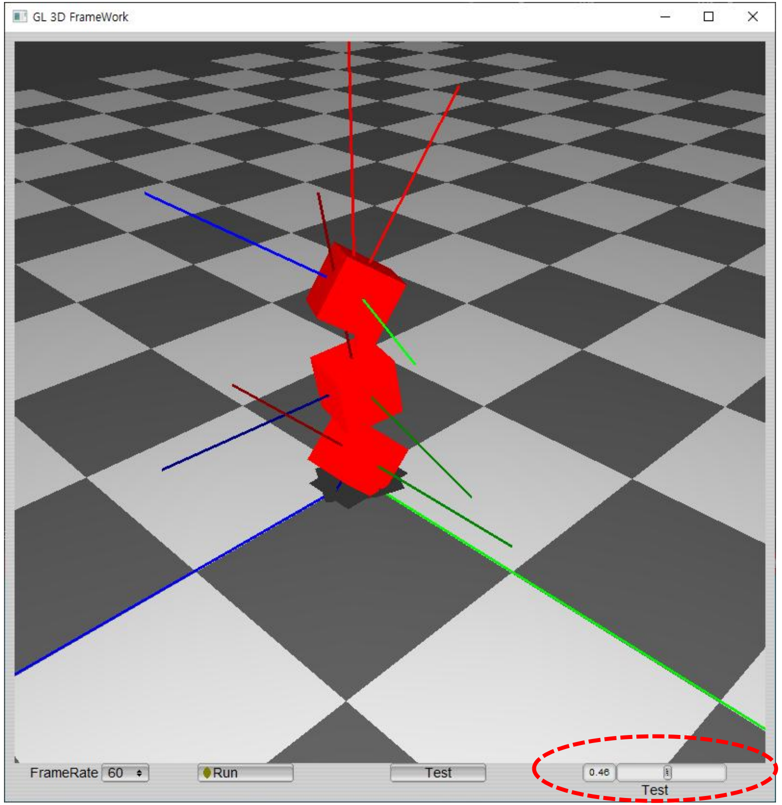
x축(녹색 축)으로 90도

example) Slerp 연습

- 총 3개의 Mover 생성
  - A (고정)
  - B (A와 B를 보간)
  - C (고정)
- B의 오리엔테이션 및 위치
  - 오리엔테이션은 Slerp(A,C)이용
  - 위치는 lerp(A,C) 이용



슬라이더 UI




```
void MyGlWindow::testValue(float v)
```

값이 변경될 때 마다 자동호출 (0<= v <=1)

v = 0 -> A와 동일
v = 1 -> C와 동일

0과 1사이 -> slerp & lerp

Slerp 연습 : 초기값

- 오리엔테이션
  - A, B = -45도 x축(1,0,0) 으로 오리엔테이션
  - C = 45도 (1,1,0)축으로 오리엔테이션
  - cyclone::Quaternion c2 = cyclone::Quaternion::slerp(c1, c3, v)
- 위치
  - A,B = (0, 1.5, 0)
  - C = (0, 10.5,0)
  - Lerp 힌트 :   B = C * t + A(1.0-t)

위치 lerp 이 best 일까?

- 부드러운 움직임을 위해서는 선형보간 보다 커브를 이용하는 편이 좋음
  - 커브 처리 방법 : 스플라인(Spline) 커브 이용
    - Cubic B-Spline
    - Bezier Spline
    - Hermit Spline