gimbal lock
보완

- x, y, z 축의 회전 순서에 기반을 두고 있으며 3개의 축은 서로 종속 관계
  - 오일러각의 각 축들은 계층구조를 가짐
    - xyz의 경우, x>y>z의 계층구조를 가짐
      - x가 회전하는 경우, y, z 축과 같이 회전
      - y가 회전하는 경우, z축과 같이 회전
      - z가 회전하는 경우, z축만 회전
- 짐벌락 발생
  - 오일러각에서의 행렬곱 ‘결과’(물체의 정점벡터)가 틀린게 아니라 회전하는 ‘과정’에서 원하는 방향으로 회전하지 못하는 경우를 가리킴
  - 그 원인이 짐벌락이고, 짐벌락이란 두 축이 평행하면서 한 개의 차원이 소실되어 엉뚱한 방향으로 회전하는 경우를 말함
  - (예시) 그림에서 x, z 축이 겹치면서, 회전할 축을 하나 잃어버림. y축을 회전시킬 때, 제대로 된 방향으로 회전하지 못함
- 짐벌락은 오일러 각에서 항상 발생하고 회전 순서와 상관없이 반드시 발생함
- 짐벌락 없이 회전할 수 있는 방법으로 쿼터니언을 이용하는 것이 있음

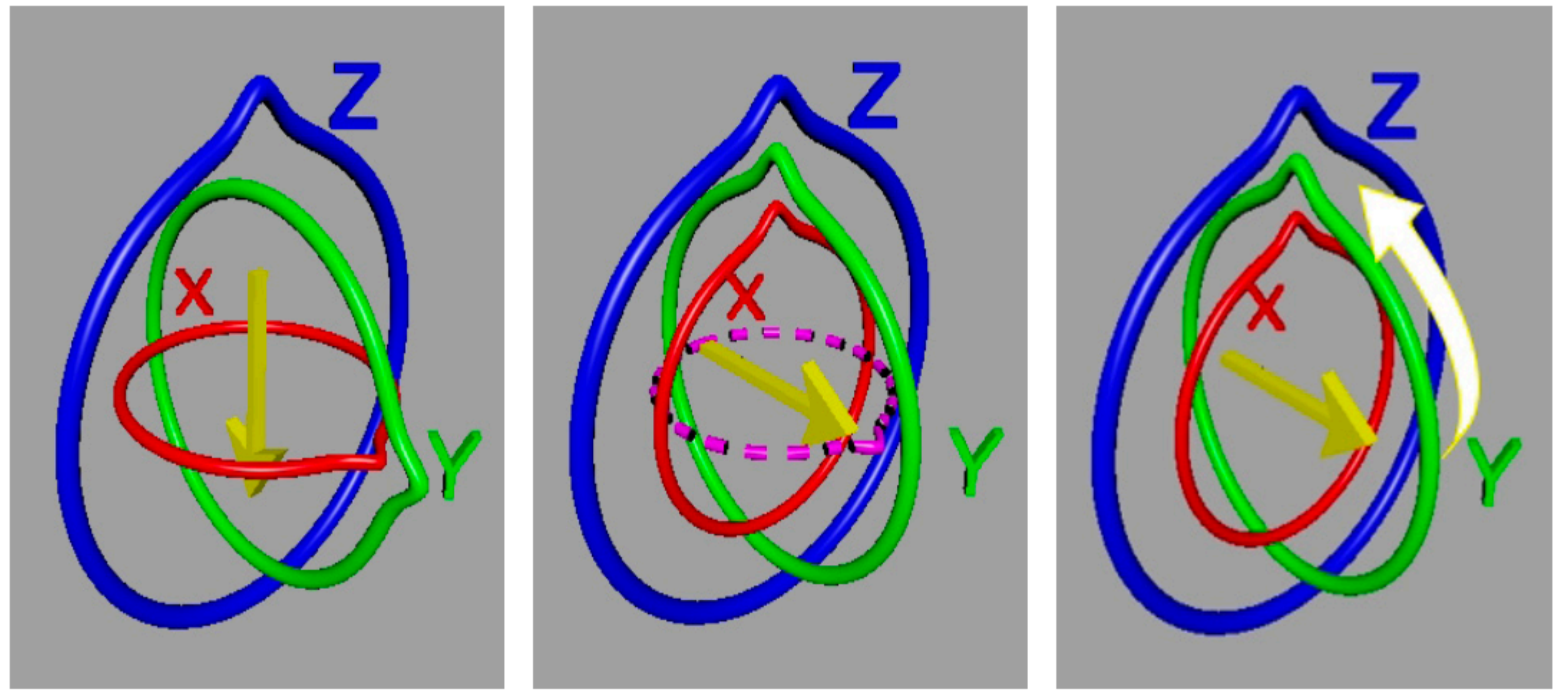


그림 2. z 〉 y 〉 x 계층구조 그림 3. x축과 z축이 겹쳐짐 그림 4. 점선은 사라진 축

짐벌락의 예

쿼터니언 회전을 변환행렬로 변환

(쿼터니언 곱 -> 4x4 행렬)

p, q는 사원수

$$pq = (p_x i + p_y j + p_z k + p_w)(q_x i + q_y j + q_z k + q_w)$$
$$= (p_x q_w + p_y q_z - p_z q_y + p_w q_x)i$$
$$+ (-p_x q_z + p_y q_w + p_z q_x + p_w q_y)j$$
$$+ (p_x q_y - p_y q_x + p_z q_w + p_w q_z)k$$
$$+ (-p_x q_x - p_y q_y - p_z q_z + p_w q_w)$$

① $pq = \begin{pmatrix} q_w & q_z & -q_y & q_x \\ -q_z & q_w & q_x & q_y \\ q_y & -q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} = M_q p$

② $pq = \begin{pmatrix} p_w & -p_z & p_y & p_x \\ p_z & p_w & -p_x & p_y \\ -p_y & p_x & p_w & p_z \\ -p_x & -p_y & -p_z & p_w \end{pmatrix} \begin{pmatrix} q_x \\ q_y \\ q_z \\ q_w \end{pmatrix} = N_p q$

두 쿼터니언 곱을 행렬의 곱으로 표현 (두 개의 버전)

쿼터니언 곱을 행렬로 표현

- 이 과정이 필요한 이유
  - 변환행렬(크기, 이동, 회전 - 오일러각)에서 모두 4x4 행렬로 표현
  - 쿼터니언 회전을 사용해도, 변환행렬로 표현할 수 있어야 함
- 두 쿼터니언 곱을 행렬의 곱으로 표현 가능 (그림 참고)
  - (4×4 행렬 * 4×1 행렬) 꼴
  - 두 가지 버전으로 표현 가능: $M_q p$ or $N_p q$
    - 곱셈의 교환법칙이 성립 ($p_x q_x = q_x p_x$)
    - 행렬의 곱셈은 교환법칙 성립 안함
- 쿼터니언 회전을 행렬의 곱으로 표현하기
  - qpq*를 행렬의 곱으로 표현하기
  - (전개는 뒷장에서)

쿼터니언 회전을 변환행렬로 표현

$$
\begin{aligned}
qpq^* &= (qp)q^* \\
&= M_{q^*}(qp) \\
&= M_{q^*}(N_q p) \\
&= (M_{q^*} N_q) p \\
&= \begin{pmatrix} q_w & -q_z & q_y & -q_x \\ q_z & q_w & -q_x & -q_y \\ -q_y & q_x & q_w & -q_z \\ q_x & q_y & q_z & q_w \end{pmatrix}
\begin{pmatrix} q_w & -q_z & q_y & q_x \\ q_z & q_w & -q_x & q_y \\ -q_y & q_x & q_w & q_z \\ -q_x & -q_y & -q_z & q_w \end{pmatrix}
\begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \\
&= \boxed{\begin{pmatrix} q_w^2+q_x^2-q_y^2-q_z^2 & 2q_xq_y-2q_zq_w & 2q_zq_x+2q_xq_z & 0 \\ 2q_xq_y+2q_zq_w & q_w^2-q_x^2+q_y^2-q_z^2 & 2q_yq_z-2q_xq_w & 0 \\ 2q_xq_z-2q_yq_w & 2q_yq_z+2q_xq_w & q_w^2-q_x^2-q_y^2+q_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}
\begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix}
\end{aligned}
$$

- 쿼터니언 회전
  - qpq* (q는 단위벡터)
    - p벡터를 회전벡터 q를 이용해 회전함
    - 이때, q는 단위벡터임
- 쿼터니언 회전을 행렬의 곱으로 표현 (그림 참고)
  - 전개한 결과, 변환행렬과 똑같은 꼴인 4x4 행렬이 됨
  - q는 단위 쿼터니언이므로

$$(qw^2 + qx^2 + qy^2 + qz^2 = 1).$$

최종 정리하면, 아래와 같음

$$
\boxed{\begin{pmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y-q_wq_z) & 2(q_xq_z+q_wq_y) & 0 \\ 2(q_xq_y+q_wq_z) & 1-2(q_x^2+q_z^2) & 2(q_yq_z-q_wq_x) & 0 \\ 2(q_xq_z-q_wq_y) & 2(q_yq_z+q_wq_x) & 1-2(q_x^2+q_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}
$$

쿼터니언 보간

slerp

- 오일러각을 통해 선형보간을 하는 경우

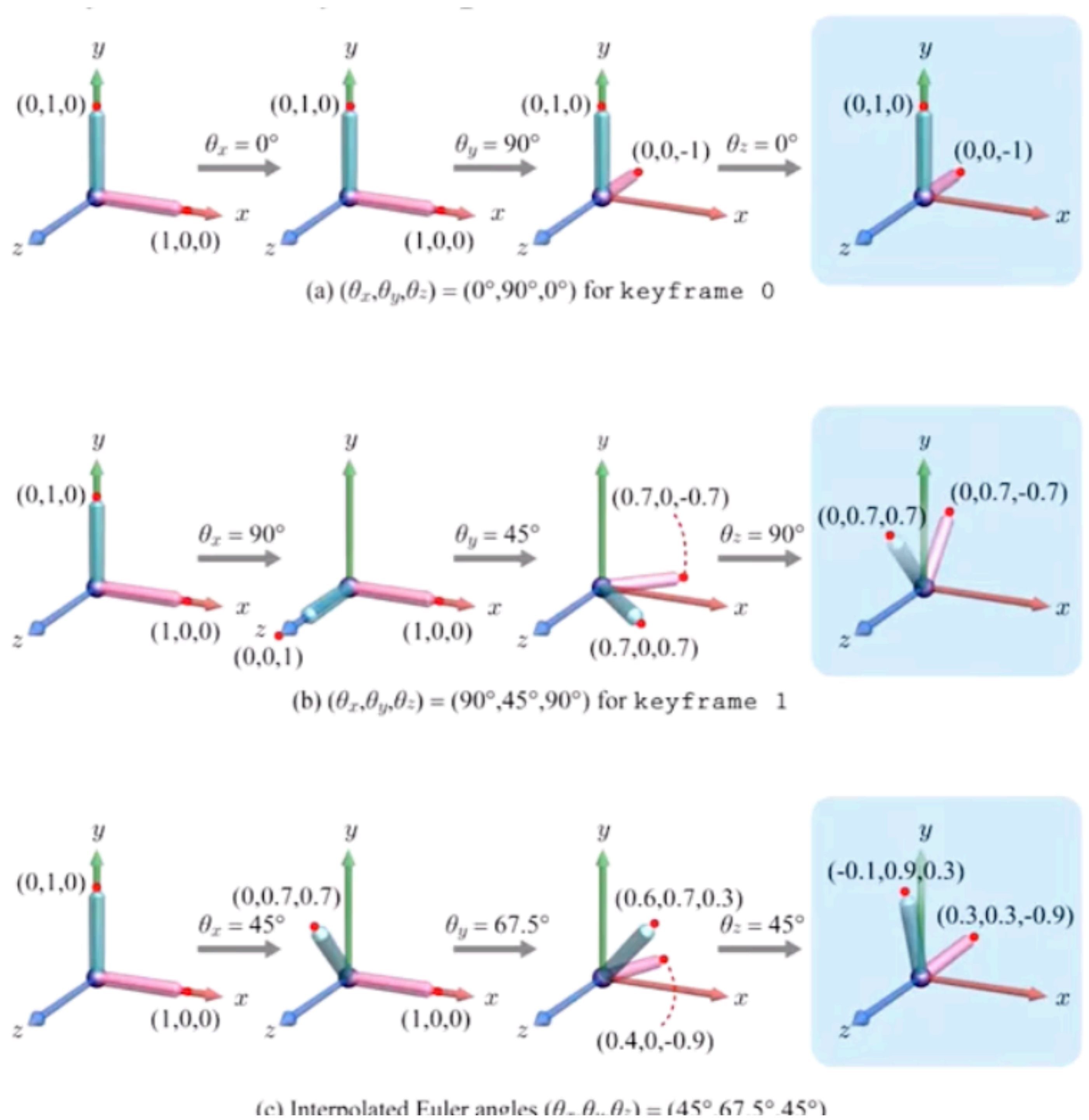
  <그림1>의 세번째 그림은 선형보간의 결과임. x축이 0에서 벗어나는 것을 원하지 않지만, 실제 계산 결과, x축이 각각 -0.1, 0.3으로 0에서 벗어나 있음.

  선형 보간 특성상 직선과 곡선의 속도오차 발생 (그림3)

- 쿼터니언을 통해 정확한 보간 가능
  - 하나의 구로 표현할 수 있음
  - Slerp: '구면'을 이용해 선형보간

    두 점을 잇는 호에 따라서 보간 -> 두 점 사이의 보간을 일정하게 할 수 있음.

그림1



첫번째 그림: 회전하기 전
두번째 그림: 회전한 후
세번째 그림: 앞의 두 그림의 1/2 지점을 선형보간함

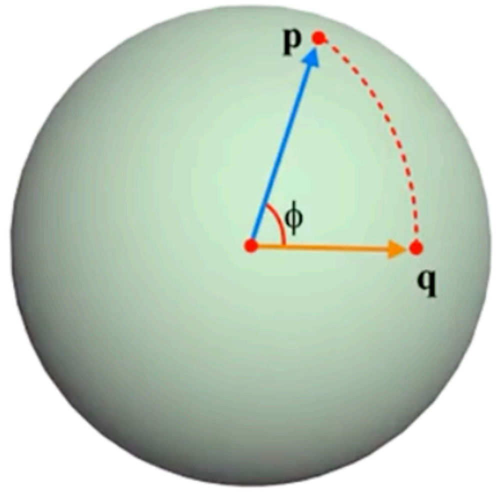


그림2

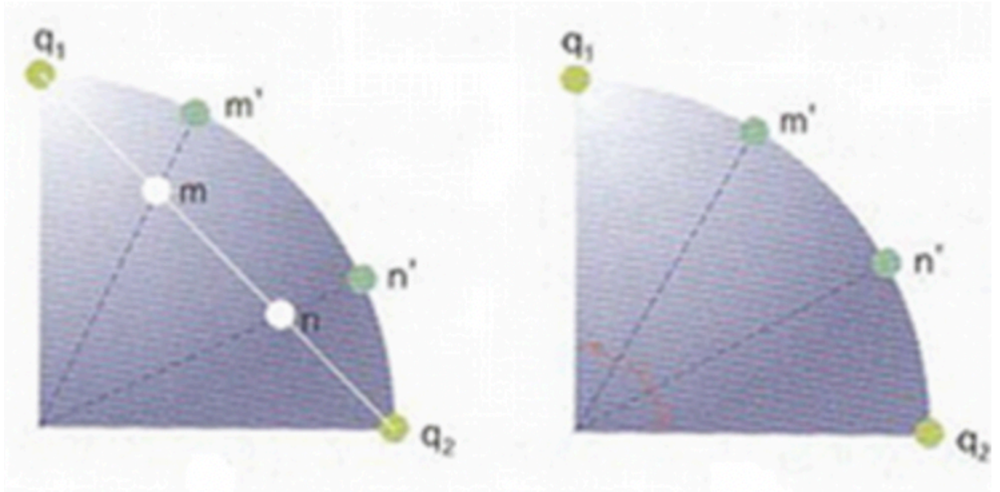


그림3

쿼터니언 정리

한계 및 장단점

- 180도 이상의 회전을 알 수 없음.
  - 원인: 두 벡터의 사이각만 알 수 있기 때문에 180도 이상의 회전을 파악할 수 없음.
  - 해결방법(임시): 쿼터니언으로 180도 이상의 회전을 하는 경우, 179도까지 회전하고 이후 추가 회전하는 방법으로 임시 해결할 수 있음

장단점

- 직관적 x
- 정확한 보간이 가능.
- 짐벌락 문제 해결
- 연산속도 빠름
- 오일러와 다르게 오차가 누적되지 않아 오차가 작음(부동소수점 관점)

오일러-쿼터니언 정리

- 쿼터니언의 경우, 짐벌락 문제를 해결하지만, 오일러각과 다르게 직관적이지 못함.
- 유니티 Transform UI의 경우
  - 우리가 직접 오브젝트의 기울기를 설정할 때는 오일러각을 이용
  - 하지만, 실제 내부에서는 회전을 쿼터니언으로 정리
- 쿼터니언을 오일러의 회전행렬(4x4행렬)꼴로 변환 가능
  - 행렬의 이점을 버리지 않고, 계산 가능
  - 다른 변환행렬과 같이 사용할 수 있음