

# 사원수(쿼터니언, Quaternion)

> 2018년8월6일, 서진택

## Step1

$$(\cos\beta + i\sin\beta)(\cos\alpha + i\sin\alpha)$$

$$\cos(\alpha + \beta) + i\sin(\alpha + \beta)$$

$$\cos\theta + i\sin\theta$$

$$\cos\theta + (0,0,1)\sin\theta$$

$$q = a + bi + cj + dk$$

$$q = w + xi + yj + zk$$

$$q = (w, x, y, z)$$

$$ijk = -1$$

$$ij = k, ji = -k$$

$$jk = i, kj = -i$$

$$ki = j, ik = -j$$

×	1	i	j	k
1	1	i	j	k
i	i	-1	k	-j
j	j	-k	-1	i
k	k	j	-i	-1

표. 쿼터니언 곱셈(Quaternion Multiplication)

$$\hat{u} = x i + y j + z k = (x, y, z)$$

$$q = (w, \hat{u})$$

순수 사원수는 실수 부분  $w$ 가 0이 사원수입니다.

$$\cos \theta + i \sin \theta$$

$$\hat{u} = x i + y j + z k = (0, 0, 1)$$

$$\cos \theta + (0i + 0j + 1k) \sin \theta$$

쿼터니언의 곱셈을 이해하기 위해, 먼저 순수 사원수의 곱셈을 계산해 봅시다.

$$q_1 = x_1 i + y_1 j + z_1 k$$

$$q_2 = x_2 i + y_2 j + z_2 k$$

$$q_1 q_2 = x_1 x_2 i i + x_1 y_2 i j + x_1 z_2 i k + y_1 x_2 j i + y_1 y_2 j j + y_1 z_2 j k + z_1 x_2 k i + z_1 y_2 k j + z_1 z_2 k k$$

$$q_1 q_2 = -x_1 x_2 + x_1 y_2 k - x_1 z_2 j - y_1 x_2 k - y_1 y_2 + y_1 z_2 i + z_1 x_2 j - z_1 y_2 i - z_1 z_2 k k$$

$$q_1 q_2 = (y_1 z_2 - z_1 y_2) i + (z_1 x_2 - x_1 z_2) j + (x_1 y_2 - y_1 x_2) k - (x_1 x_2 + y_1 y_2 + z_1 z_2)$$

$$q_1 q_2 = q_1 \times q_2 - q_1 \cdot q_2$$

사실 벡터의 외적과 내적은 사원수에서 유도된 것입니다.

$$Q_1 = (w_1, q_1)$$

$$Q_2 = (w_2, q_2)$$

$$Q_1 Q_2 = w_1 w_2 + w_1 q_2 + w_2 q_1 + q_1 \times q_2 - q_1 \cdot q_2$$

$$\begin{aligned}
Q_1 Q_2 &= (w_1 + x_1 i + y_1 j + z_1 k)(w_2 + x_2 i + y_2 j + z_2 k) \\
&= (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) + \\
&\quad (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) i + \\
&\quad (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) j + \\
&\quad (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) k
\end{aligned}$$

## 켈레복소수(Conjugate)

켈레복소수는 복소수의 역(Inverse)을 정의하기 위해서 필요합니다. 일반적인 복소수의 놈(Norm), 켈레복소수와 그 성질은 다음과 같습니다.

$$c = a + bi$$

$$|c| = \sqrt{a^2 + b^2}$$

$$c^* = a - bi$$

$$c \cdot c^* = a^2 + b^2 = |c|^2$$

$$c^{-1} = \frac{c^*}{|c|^2} = \frac{c^*}{c \cdot c^*} = \frac{1}{c}$$

$$q = (w, x, y, z)$$

$$q^* = (w, -x, -y, -z)$$

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

$$q \cdot q^* = |q|^2$$

$$q^{-1} = \frac{q^*}{|q|^2} = \frac{q^*}{q \cdot q^*} = \frac{1}{q}$$

켈레복소수는 아래와 같은 성질이 있습니다.

$$(q_1 q_2)^* = q_2^* q_1^*$$

## KQuaternion의 구현

```

#pragma once
#include <cmath>

class KQuaternion
{
public:
    const static KQuaternion Zero;
    const static KQuaternion One;
    const static KQuaternion xHat;
    const static KQuaternion yHat;
    const static KQuaternion zHat;

public:
    double w;
    double x;
    double y;
    double z;

public: // Constructors
    KQuaternion( );
    KQuaternion( const KQuaternion& Q );
    KQuaternion( const double w0, const double x0, const double y0,
const double z0 );

    inline KQuaternion operator/( const double t ) const { return
KQuaternion( w / t, x / t, y / t, z / t ); }
    KQuaternion operator*( const KQuaternion& Q ) const;
    inline KQuaternion operator/( const KQuaternion& Q ) const { return
( *this )*( Q.inverse( ) ); }

    inline KQuaternion inverse( ) const { return conjugate( ) /
normsquared( ); }
    inline KQuaternion conjugate( ) const { return KQuaternion( w, -x,
-y, -z ); }
    inline double      normsquared( ) const { return ( w*w + x*x + y*y +
z*z ); }
};

```

## 쿼터니언의 곱셈

```
KQuaternion KQuaternion::operator*( const KQuaternion& Q ) const
{
    return KQuaternion( w*Q.w - x*Q.x - y*Q.y - z*Q.z
        , w*Q.x + x*Q.w + y*Q.z - z*Q.y
        , w*Q.y - x*Q.z + y*Q.w + z*Q.x
        , w*Q.z + x*Q.y - y*Q.x + z*Q.w );
}
```

임의의 3D 벡터를 쿼터니언과 곱하면 3D 벡터의 회전을 의미합니다. 요소가 3개인 벡터를 쿼터니언과 곱할 수 없으므로, 입력 벡터  $v$ 를  $w$ 값이 0인 순수 쿼터니언으로  $v_q$  변환합니다.

$$v = (x, y, z)$$

$$v_q = (0, x, y, z)$$

이제  $v_q$ 를 임의의 쿼터니언과 곱하는 것이 가능합니다.

2차원의 회전을 다룰 때, 모든 점들이 하나의 복소평면에 위치했으므로, 육면체를 이루는 모든 점들이 같은  $xy$ -평면상에 위치한다고 가정해 보겠습니다. 그래서 다각형의 버텍스 버퍼를 설정하는 함수에서  $z$ 값을 모두 0으로 설정합니다.

```
void KPolygon::SetVertexBuffer()
{
    m_vertexBuffer[0] = KVector3(-5.f, -5.f, 0.f);
    m_vertexBuffer[1] = KVector3(-5.f, 5.f, 0.f);
    m_vertexBuffer[2] = KVector3(5.f, 5.f, 0.f);
    m_vertexBuffer[3] = KVector3(5.f, -5.f, 0.f);
    m_vertexBuffer[4] = KVector3(-5.f, -5.f, -0.f);
    m_vertexBuffer[5] = KVector3(-5.f, 5.f, -0.f);
    m_vertexBuffer[6] = KVector3(5.f, 5.f, -0.f);
    m_vertexBuffer[7] = KVector3(5.f, -5.f, -0.f);
    m_sizeVertex = 8;
} //KPolygon::SetVertexBuffer()
```

이제 쿼터니언의 곱셈을 멤버 함수로 추가하고, 다각형을  $z$ 축 (0,0,1)을 중심으로 회전하는 코드를 작성해 보겠습니다.

```

void KPolygon::Rotate( const KQuaternion& q )
{
    for(int i = 0; i<m_sizeVertex; ++i)
    {
        KQuaternion qPoint = KQuaternion( 0, m_vertexBuffer[i].x,
        m_vertexBuffer[i].y, m_vertexBuffer[i].z );
        KQuaternion qResult = q * qPoint;
        m_vertexBuffer[i] = KVector3( (float)qResult.x,
        (float)qResult.y, (float)qResult.z );
    } //for
} //Rotate()

```

실행 결과는 다음과 같습니다.

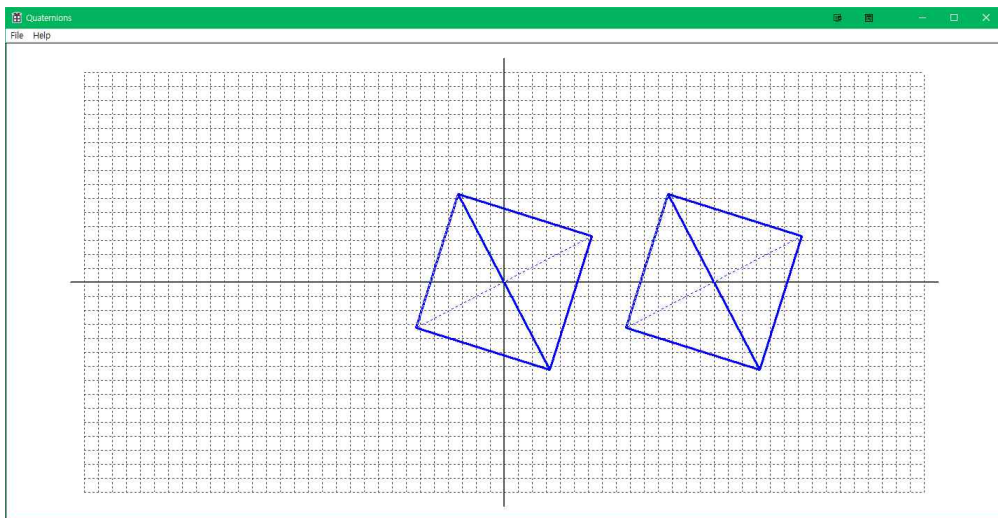


그림. 쿼터니언의 곱셈: 벡터를 순수 쿼터니언으로 변환한 후에, 다른 쿼터니언과 곱하면 회전된 새로운 벡터를 얻을 수 있습니다.

## vq\*의 의미

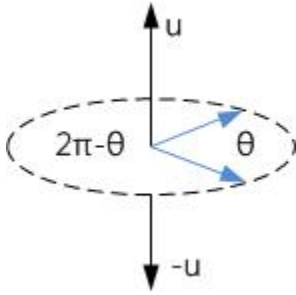


그림.  $vq^*$ 의 의미:  $vq^*$ 는 원래  $q$ 의 방향 벡터 성분에 대해서  $2\pi - \theta$ 만큼의 회전을 의미합니다.

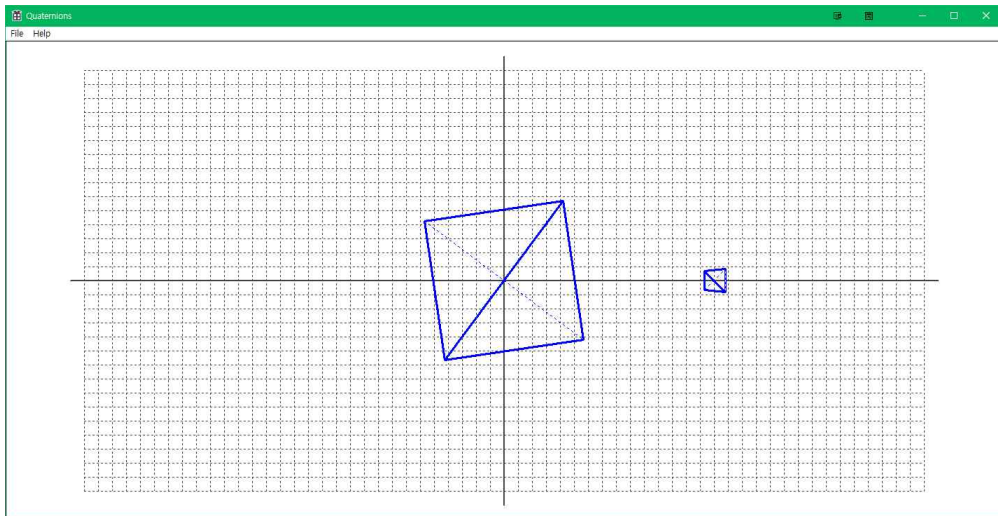


그림. 쿼터니언의 곱셈: 쿼터니언과 벡터의 단순한 곱셈은 회전을 표현하지 못합니다. 그 이유는 곱셈의 결과로 얻은 쿼터니언이 순수 쿼터니언이 되지 않기 때문입니다.

## Step2

쿼터니언과 벡터의 곱셈에서 실수항을 0으로 만드는 방법이 있을까요?

$$v' = qvq^*$$

선택 함수  $w()$ 를 아래와 같이 정의합니다.

$$W(q) = W(w + xi + yj + zk) = w$$

v'의 실수 부분이 0임을 아래와 같이 증명할 수 있습니다.

$$\begin{aligned} W(v') &= W(qvq^*) \\ &= [(qvq^*) + (qvq^*)^*]/2 \quad \leftarrow (qvq^*) + (qvq^*)^* = 2w \\ &= [qvq^* + qv^*q^*]/2 \quad \leftarrow (ab)^* = b^*a^* \\ &= q[(v + v^*)/2]q^* \\ &= qW(v)q^* \quad \leftarrow qrq^* = r|q|^2, \text{ where } r \in R \\ &= W(v) \\ &= 0 \end{aligned}$$

이제 q를 이용해서  $\theta$ 만큼 회전하기 위해서는, q를 설정할 때  $\cos\theta + \hat{u}\sin\theta$ 로 설정해서는 안 됩니다. 왜냐하면 qv가  $\theta$ 만큼 회전하고,  $vq^*$ 이  $\theta$ 만큼 회전시키기 때문에,  $qvq^*$ 은  $2\theta$ 만큼 회전하게 되기 때문입니다.

그래서 회전을 나타내는 쿼터니언을 설정할 때,  $\cos(\theta/2) + \hat{u}\sin(\theta/2)$ 가 되게 설정해 주어야 합니다.

$qvq^*$ 의 동작을 처음으로 이해했을 때, 참으로 놀라웠습니다.  $q = \cos(\theta/2) + \hat{u}\sin(\theta/2)$ 로 정의된 쿼터니언에 대해서,  $qvq^*$ 를 해석하는 한 가지 방법은 다음과 같습니다.

처음 qv는 v를  $\theta/2$ 만큼 회전한 값을 4차원 쿼터니언 공간에 배치시킵니다. 뒷부분의  $vq^*$ 는  $\theta/2$ 만큼 회전된 쿼터니언을  $\theta/2$ 만큼 더 회전시키면서, 3차원 공간에 배치시킨다고 이해하는 것입니다.

이제 쿼터니언을 이용해서 큐브를 회전시키는 코드를 다음과 같이 작성할 수 있습니다.

```
{
    KPolygon      poly2;
    const float    cos1_2theta = cosf( s_fTheta / 2.0f );
    const float    sin1_2theta = sinf( s_fTheta / 2.0f );
    KQuaternion    qRot = KQuaternion( cos1_2theta, 0.0 *

```



```
sin1_2theta, 1.0*sin1_2theta, 0.0*sin1_2theta );
```

```
    poly2.SetIndexBuffer( );  
    poly2.SetVertexBuffer( );  
    poly2.Rotate( qRot );  
    poly2.Transform( matTrans );  
    poly2.Transform( matProjection );  
    poly2.Render( hdc );  
}
```

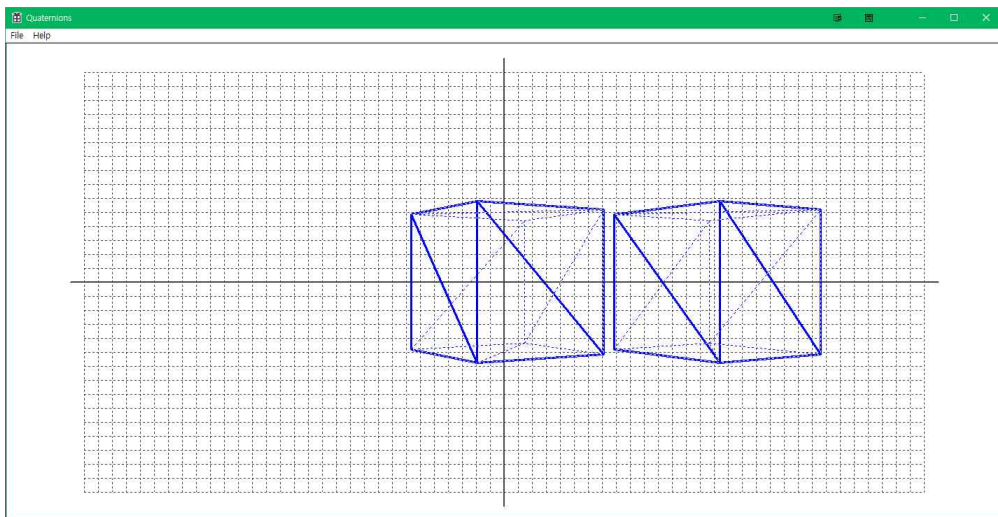


그림. 쿼터니언이  $qvq^*$ 를 사용하여 회전을 나타내는 이유

@