Rotation with Vectors and Quaternions

Basic Rules

First, we have to go through some of the basic setup here.

Quaternions are defined as the following format (similar to imaginary numbers):

$$x\mathbf{i} + y\mathbf{j} + z\mathbf{k} + w \tag{1}$$

or written in axis angle format:

$$sin(\frac{\alpha}{2})u_x\mathbf{i} + sin(\frac{\alpha}{2})u_y\mathbf{j} + sin(\frac{\alpha}{2})u_z\mathbf{k} + cos(\frac{\alpha}{2}) \tag{2}$$

where α is the rotation angle around a certain axis (right hand rule, that is, pointing your thumb to the axis's direction, so the rest of your fingers pointing to the rotation direction), and u_x , u_y and u_z are the normalized coordinates of the axis.

To discuss the rotation by quaternion, we don't have to go through all the mathematical attributes of quaternion, nor the complex regulations in calculation. But I still have to list the following attributes for the convenience of our later discussion:

For a normalized quaternion, which is given by $x^2 + y^2 + z^2 + w^2 = 1$ we have:

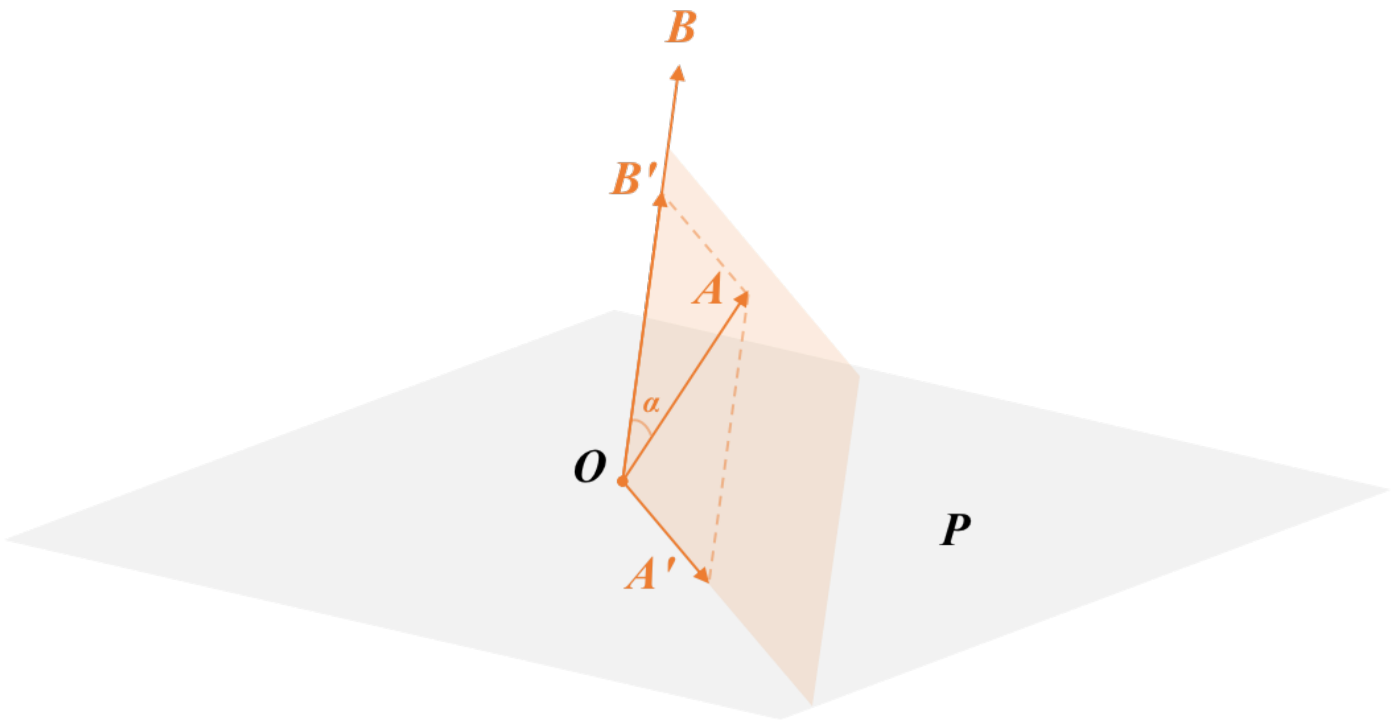
$$\begin{gathered} sin(\frac{\alpha}{2})^2u_x^2 + sin(\frac{\alpha}{2})^2u_y^2 + sin(\frac{\alpha}{2})^2u_z^2 + cos(\frac{\alpha}{2})^2 = 1 \\ \Rightarrow sin(\frac{\alpha}{2})^2(u_x^2 + u_y^2 + u_z^2) + cos(\frac{\alpha}{2})^2 = 1 \\ \Rightarrow sin(\frac{\alpha}{2})^2(u_x^2 + u_y^2 + u_z^2) = 1 - cos(\frac{\alpha}{2})^2 \\ \Rightarrow sin(\frac{\alpha}{2})^2(u_x^2 + u_y^2 + u_z^2) = sin(\frac{\alpha}{2})^2 \end{gathered} \tag{3}$$

Therefore, if a quaternion is normalized, we can get either an identity quaternion ($0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k} + 1$), or its rotation axis is also normalized ($u_x^2 + u_y^2 + u_z^2 = 1$).

Axis Rotation for Vector

Let's consider a general case for a vector $\overrightarrow{OA}$ rotate around an axis $\overrightarrow{OB}$, here, for convenience the origin of the vector and axis are aligned together.

First, we want to get the projection of $\overrightarrow{OA}$ to the plane P defined by $\overrightarrow{OB}$, that is, $\overrightarrow{OB}$ being a normal vector of P.



Suppose A' is the projection of A to P, and B' is the projection of A to $\overrightarrow{OB}$, α being the angle between $\overrightarrow{OA}$ and $\overrightarrow{OB}$. We have:

$$\begin{aligned} \overrightarrow{OB'} &= \overrightarrow{OB} \cdot ||\overrightarrow{OA}|| \cdot cos(\alpha) \\ \Rightarrow \overrightarrow{OB'} &= \overrightarrow{OB} \cdot ||\overrightarrow{OA}|| \frac{\overrightarrow{OA} \cdot \overrightarrow{OB}}{||\overrightarrow{OA}|| ||\overrightarrow{OB}||} \\ \Rightarrow \overrightarrow{OB'} &= \overrightarrow{OB} \frac{\overrightarrow{OA} \cdot \overrightarrow{OB}}{||\overrightarrow{OB}||} \end{aligned} \tag{4}$$

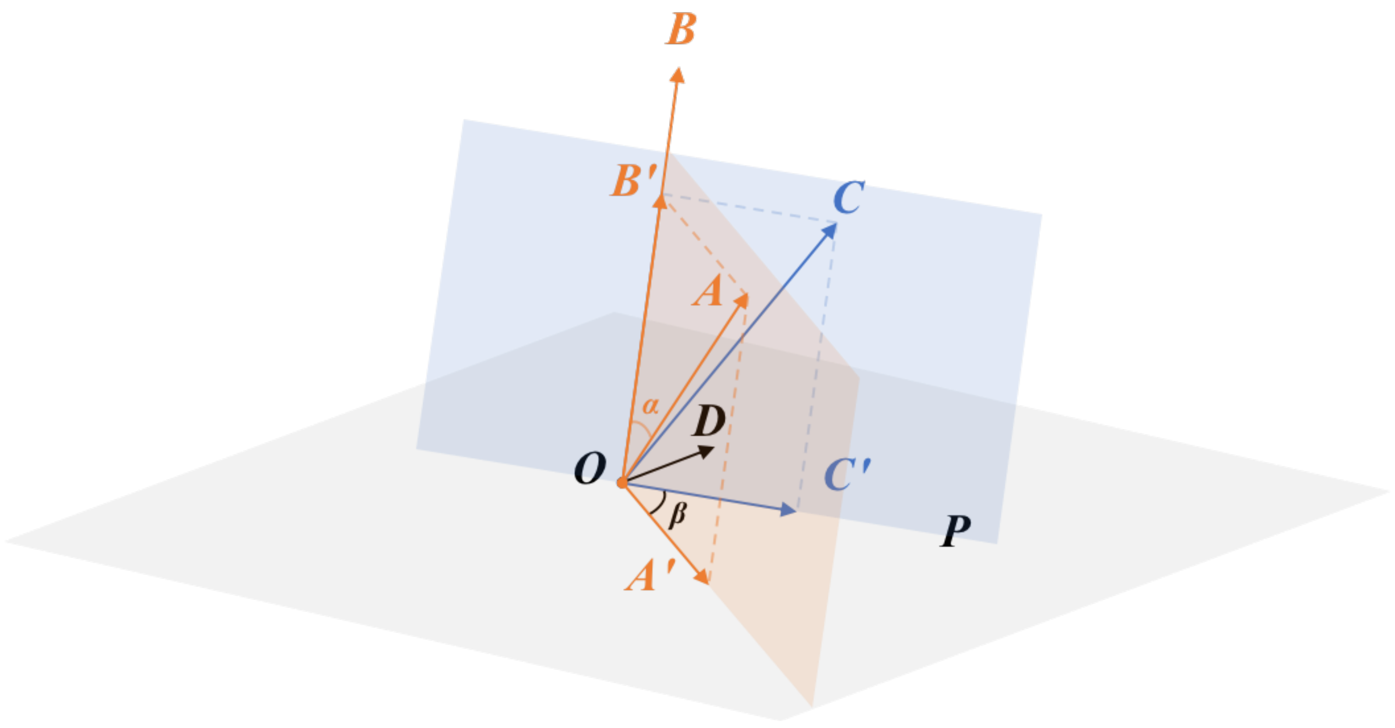
To simplify the equation, let's denote the normalized axis $\overrightarrow{OB}$ by $\vec{n}$, we have:

$$\overrightarrow{OB'} = (\overrightarrow{OA} \cdot \vec{n}) \cdot \vec{n} \tag{5}$$

So, we can easily get the projection of $\overrightarrow{OA}$ to the plane P By:

$$\overrightarrow{OA'} = \overrightarrow{OA} - \overrightarrow{OB'} = \overrightarrow{OA} - (\overrightarrow{OA} \cdot \vec{n}) \cdot \vec{n} \tag{6}$$

Next, we rotate the vector $\overrightarrow{OA'}$ by β on the plane P To get vector $\overrightarrow{OC'}$ by adopting the right hand rule:



To calculate $\overrightarrow{OC'}$, we obtain a orthogonal coordinates system using $\overrightarrow{OA'}$, $\vec{n}$ and $\overrightarrow{OD}$ which is obtained by cross product of $\vec{n}$ and $\vec{a}$ (For convenience, here we denote normalized vector $\overrightarrow{OA'}$ by $\vec{a}$). Therefore $\vec{n} \times \vec{a} = \vec{d} = \frac{\overrightarrow{OD}}{||\overrightarrow{OD}||}$

$$\overrightarrow{OC'} = (\vec{a} \cdot cos(\beta) + \vec{n} \times \vec{a} \cdot sin(\beta)) \cdot ||\overrightarrow{OA'}|| = \overrightarrow{OA'} \cdot cos(\beta) + \vec{n} \times \overrightarrow{OA'} \cdot sin(\beta) \tag{7}$$

Finally, to get the rotation result, we have:

$$\begin{gathered}
\overrightarrow{OC} = \overrightarrow{OC'} + \overrightarrow{A'A} = \overrightarrow{OC'} + \overrightarrow{OA} - \overrightarrow{OA'} \\
\Rightarrow \overrightarrow{OC} = \overrightarrow{OA'} \cdot cos(\beta) + \vec{n} \times \overrightarrow{OA'} \cdot sin(\beta) + \overrightarrow{OA} - \overrightarrow{OA'} \\
\Rightarrow \overrightarrow{OC} = \overrightarrow{OA} + \overrightarrow{OA'} \cdot (cos(\beta) - 1) + \vec{n} \times \overrightarrow{OA'} \cdot sin(\beta) \\
\Rightarrow \overrightarrow{OC} = \overrightarrow{OA} + (\overrightarrow{OA} - (\overrightarrow{OA} \cdot \vec{n}) \cdot \vec{n}) \cdot (cos(\beta) - 1) + \vec{n} \times (\overrightarrow{OA} - (\overrightarrow{OA} \cdot \vec{n}) \cdot \vec{n}) \cdot sin(\beta) \\
\Rightarrow \overrightarrow{OC} = \overrightarrow{OA} \cdot cos(\beta) + (\overrightarrow{OA} \cdot \vec{n}) \cdot \vec{n} \cdot (1 - cos(\beta)) + \vec{n} \times \overrightarrow{OA} \cdot sin(\beta)
\end{gathered} \tag{8}$$

Describe Axis Rotation by Quaternion

From the basic rule part, we have the quaternion described by the form $sin(\frac{\alpha}{2})u_x\mathbf{i} + sin(\frac{\alpha}{2})u_y\mathbf{j} + sin(\frac{\alpha}{2})u_z\mathbf{k} + cos(\frac{\alpha}{2})$, and for convenience, we split the quaternion into one vector $\vec{u}$ and a number w :

$$\vec{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \cdot sin(\frac{\alpha}{2}) = \vec{n} \cdot sin(\frac{\alpha}{2}), w = cos(\frac{\alpha}{2}) \tag{9}$$

Notice that $\vec{u}$ can be regard as a scaled version of $\vec{n}$ from the previous chapter.

And suppose the vector to rotate by quaternion is $\vec{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$

We get the following supposing the quaternion is not an identity quaternion:

$$w^2 - \vec{u} \cdot \vec{u} = cos^2(\frac{\alpha}{2}) - sin^2(\frac{\alpha}{2})(u_x^2 + u_z^2 + u_z^2) = cos^2(\frac{\alpha}{2}) - sin^2(\frac{\alpha}{2}) = cos(\alpha) \tag{10}$$

And also:

$$\begin{gathered}
(\vec{a} \cdot \vec{u}) \cdot \vec{u} = (a_x u_x + a_y u_y + a_z u_z) sin(\frac{\alpha}{2}) \cdot \vec{n} \cdot sin(\frac{\alpha}{2}) \\
= (a_x u_x + a_y u_y + a_z u_z) \cdot \vec{n} \cdot sin^2(\frac{\alpha}{2}) = (\vec{a} \cdot \vec{n}) \cdot \vec{n} \cdot \frac{1 - cos(\alpha)}{2}
\end{gathered} \tag{11}$$

And finally:

$$\vec{u} \times \vec{a} \cdot w = \vec{n} \times \vec{a} \cdot sin(\frac{\alpha}{2}) cos(\frac{a}{2}) = \vec{n} \times \vec{a} \cdot \frac{sin(\alpha)}{2} \tag{12}$$

Then, as we combine the (10), (11) and (12), we get:

$$\begin{gathered}
\vec{a} \cdot (w^2 - \vec{u} \cdot \vec{u}) + (\vec{a} \cdot \vec{u}) \cdot \vec{u} \cdot 2 + \vec{u} \times \vec{a} \cdot w \cdot 2 \\
= \vec{a} \cdot cos(\alpha) + (\vec{a} \cdot \vec{n}) \cdot \vec{n} \cdot (1 - cos(\alpha)) + \vec{n} \times \vec{a} \cdot sin(\alpha)
\end{gathered} \tag{13}$$

Which is exactly in the same form as (8), so, we have the method to rotate a vector by quaternion.

But don't forget that we assume that this quaternion is identity, so we also have to check for an identity quaternion, whether this equation stands. An identity quaternion indicates that $sin(\frac{\alpha}{2}) = 0$, so this rotation is either degree 0 or 180 (meaning after the rotation, the vector is either parallel or opposite to the origin vector).

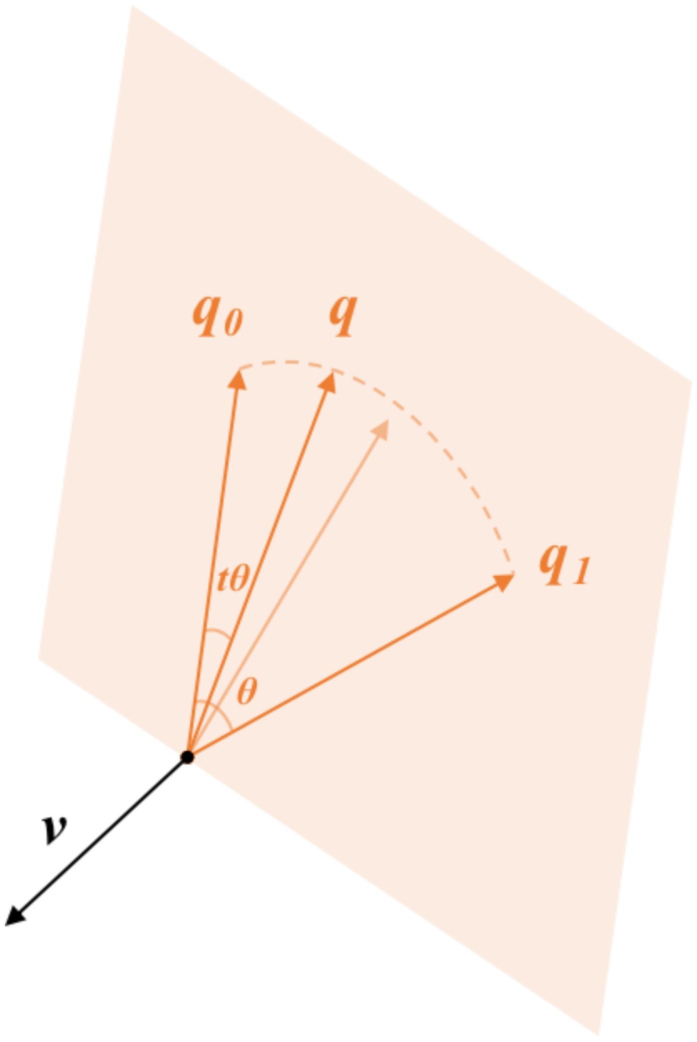
For degree 0 or 360, we have:

$$\begin{gathered}
\vec{a'} = \vec{a} \cdot (w^2 - \vec{u} \cdot \vec{u}) + (\vec{a} \cdot \vec{u}) \cdot \vec{u} \cdot 2 + \vec{u} \times \vec{a} \cdot w \cdot 2 \\
= \vec{a} \cdot (1^2 - \vec{0} \cdot \vec{0}) + (\vec{a} \cdot \vec{0}) \cdot \vec{0} \cdot 2 + \vec{0} \times \vec{a} \cdot 1 \cdot 2 = \vec{a}
\end{gathered} \tag{14}$$

Which indicates no rotation is applied to the vector.

Spherical Linear Interpolation of Quaternion

The ideal interpolation between two quaternions q_0 and q_1 can be visualized as follow, where q is the ideal interpolation, θ being the angle between two quaternions and v being the rotation axis of the relative rotation from q_0 to q_1 .



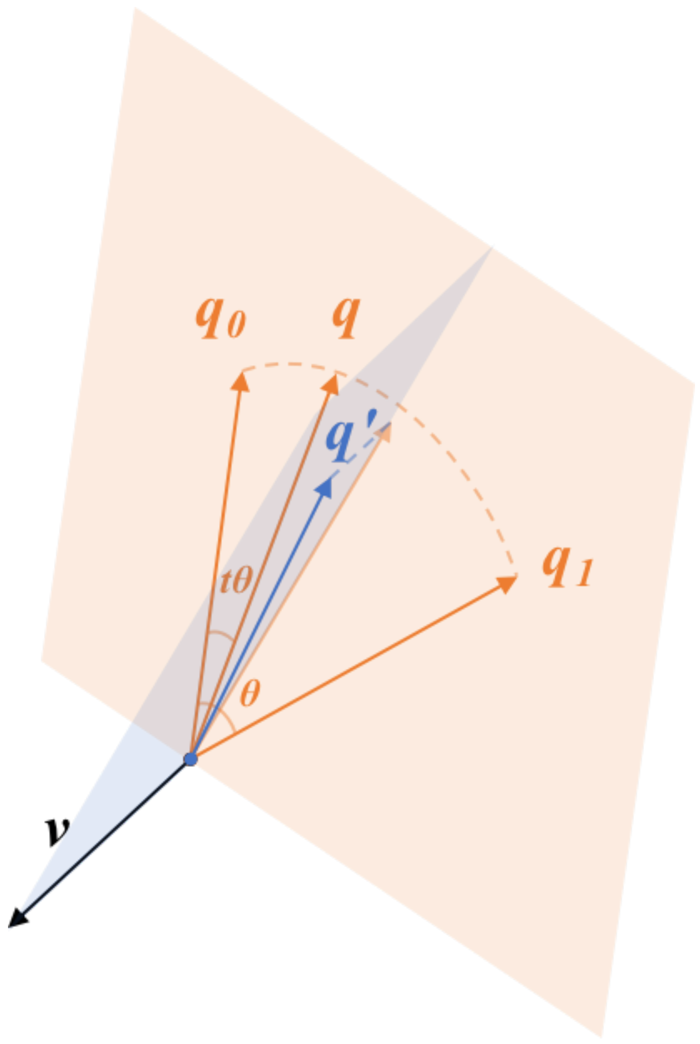
Relative rotation of two quaternions can be described as:

$$\begin{aligned} q_0 \Delta q &= q_1 \\ \Rightarrow \Delta q &= q_0^{-1} q_1 \end{aligned} \tag{15}$$

To interpolate quaternions, a most efficient and strait way is to do linear interpolation, simply put:

$$q' = (1-\alpha) \cdot q_0 + \alpha \cdot q_1 = \begin{bmatrix} (1-\alpha)x_0 + \alpha x_1 \\ (1-\alpha)y_0 + \alpha y_1 \\ (1-\alpha)z_0 + \alpha z_1 \\ (1-\alpha)w_0 + \alpha w_1 \end{bmatrix} \tag{16}$$

But this interpolation method creates a rotation that does not land on the very plane determined by the rotation axis of the relative rotation of q_0 and q_1 :



A better way of doing interpolation is to adopt the relative rotation between input quaternions. To get the optimum quaternion, we simply formulate:

$$q = q_0 \cdot \Delta q^t = q_0 \cdot (q_0^{-1} q_1)^t \tag{17}$$

where q^t is defined as:

$$\begin{aligned} q = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} &= \begin{bmatrix} u_x \cdot sin(\frac{\theta}{2}) \\ u_y \cdot sin(\frac{\theta}{2}) \\ u_z \cdot sin(\frac{\theta}{2}) \\ cos(\frac{\theta}{2}) \end{bmatrix} \\ q^t &= \begin{bmatrix} u_x \cdot sin(t \cdot \frac{\theta}{2}) \\ u_y \cdot sin(t \cdot \frac{\theta}{2}) \\ u_z \cdot sin(t \cdot \frac{\theta}{2}) \\ cos(t \cdot \frac{\theta}{2}) \end{bmatrix} \end{aligned} \tag{18}$$

Therefore, here we already obtained a feasible method of acquiring interpolation between two quaternions, and the implementation in c++ is given as below:

```cpp
// Code sample from project : Lu Renderer
Quaternion slerp(const Quaternion & from, const Quaternion & to, float alpha)
{
    alpha = clamp(alpha, 0.0, 1.0);

    float cos_angle = from.dot(to);
    // handle parallel quaternions
    // handle cos_angle < 0
    // ...
    Quaternion delta_q = from.inverse() * to;
    float angle = acosf(cos_angle);

    Quaternion quat(
        q.x * asinf(alpha * angle) / asinf(angle),
        q.y * asinf(alpha * angle) / asinf(angle),
        q.z * asinf(alpha * angle) / asinf(angle),
        acosf(alpha * angle) / cos_angle
    );
    return from * quat;
}
```

But the above implementation involves at least two quaternion multiplication which is quite time consuming. Therefore, a much light-burden methods will be shown below:

First, we rewrite the spherical linear interpolation equation as follow:

$$q = Slerp(q_0, q_1, t) = c_0(t)q_0 + c_1(t)q_1 \tag{19}$$

Where $c_0(t)$ and $c_1(t)$ are real-valued functions, and $c_0(0) = 1, c_0(1) = 0$, $c_1(0) = 0, c_1(1) = 1$ and $0 \leq t \leq 1$.

And to get an ideal interpolation, suppose here the angle between two quaternions is θ we have:

$$q \cdot q_0 = (c_0(t)q_0 + c_1(t)q_1) \cdot q_0 = c_0(t)q_0 \cdot q_0 + c_1(t)q_1 \cdot q_0 = c_0(t) + c_1(t)cos(\theta) = cos(t\theta) \tag{20}$$

$$q \cdot q_1 = (c_0(t)q_0 + c_1(t)q_1) \cdot q_1 = c_0(t)q_0 \cdot q_1 + c_1(t)q_1 \cdot q_1 = c_0(t)cos(\theta) + c_1(t) = cos((1-t)\theta) \tag{21}$$

With two equations (20) and (21) and two unknown variables $c_0(t)$ and $c_1(t)$, we can solve the above equations:

$$\begin{aligned} c_0(t) &= \frac{cos(t\theta) - cos((1-t)\theta)cos(\theta)}{1 - cos^2(\theta)} \\ c_1(t) &= \frac{cos((1-t)\theta) - cos(t\theta)cos(\theta)}{a - cos^2(\theta)} \end{aligned} \tag{22}$$

$$c_0(t) = \frac{cos(\theta)cos((1-t)\theta) + sin(\theta)sin((1-t)\theta) - cos((1-t)\theta)cos(\theta)}{1 - cos^2(\theta)} = \frac{sin(\theta)sin((1-t)\theta)}{sin^2(\theta)} = \frac{sin((1-t)\theta)}{sin(\theta)} \tag{23}$$

$$c_1(t) = \frac{cos(\theta)cos(t\theta) + sin(\theta)sin(t\theta) - cos(t\theta)cos(\theta)}{1 - cos^2(\theta)} = \frac{sin(\theta)sin(t\theta)}{sin^2(\theta)} = \frac{sin(t\theta)}{sin(\theta)} \tag{24}$$

Finally, we have reached a new method of calculating quaternion interpolation:

$$Slerp(q_0, q_1, t) = \frac{q_0 sin((1-t)\theta) + q_1 cos(t\theta)}{sin(\theta)} \tag{25}$$

And an implementation in c++ is given below, which takes only half the time of the previous algorithm.

```cpp
// Code sample from project : Lu Renderer
Quaternion slerp(const Quaternion & from, const Quaternion & to, float alpha)
{
    alpha = clamp(alpha, 0.0, 1.0);

    float cos_angle = from.dot(to);
    // handle parallel quaternions
    // handle cos_angle < 0
    // ...
    float angle = acosf(cos_angle);
    float sin_angle = sinf(angle);
    float angle_from = (1 - alpha) * angle;
    float angle_to = alpha * angle;
    float factor_from = sinf(angle_from) / sin_angle;
    float factor_to = sinf(angle_to) / sin_angle;

    Quaternion quat(
        from.x * factor_from + to.x * factor_to,
        from.y * factor_from + to.y * factor_to,
        from.z * factor_from + to.z * factor_to,
        from.w * factor_from + to.w * factor_to
    );
    return quat;
}
```

References:

[1] https://www.geometrictools.com/Documentation/Quaternions.pdf

[2] https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/slerp/index.htm

[3] https://github.com/zauonlok/renderer/blob/master/renderer/core/maths.c

[4] https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/slerp/index.htm