Quaternions in Computer Graphics

***Abstract*—Objects in applications that are represented in 3D space must have their orientation described in a way that is both efficient and easy to manipulate. Quaternions are a coordinate system, that can be used for this purpose, that has several advantages over other methods. This article, serves as an introduction to quaternions, explaining how they can be used to describe 3D rotations, and provides examples of their use in real-world applications.**

***Index Terms*—Quaternions, Computer Graphics, 3D Rotations**

I. Introduction

In the world of computer graphics, especially 3D graphics and animation, precision and efficiency are critical for creating lifelike simulations, smooth animations, and immersive experiences in real-time.

Quaternions are a mathematical construct first described by William Rowan Hamilton in 1843 and have been used more and more over the years in this field and many others, in comparison to other options such as Euler Angles, Rotation Matrices and many more, due to their special advantages which will be explored further later in this article.

II. Quaternions 101

Before understanding Quaternions, it's helpful to first understand complex numbers, since at their core Quaternions are an extension of them into the three dimensions.

A. Complex Numbers

Complex numbers are a number system that extends the real numbers, with an Imaginary unity element designated by i, they have their use in mathematics that we won't go in depth here, but the key takeaway is that they are defined by satisfying the following equation:

$$i^2 = -1 \tag{1}$$

This allows the complex number system to have solutions to all polynomial equations, even those that have no solutions in real numbers.

B. Quaternions

Quaternions are obtained similarly by adding three imaginary elements, i, j, and k, to the real numbers, making their representation in the form of:

$$q = w + xi + yj + zk \tag{2}$$

Where w, x, y, and z are real numbers, and i, j, and k are the imaginary elements that allow quaternions to encode three-dimensional rotations and reflect orientation changes associated with spatial transformations. The rules governing the imaginary elements of quaternions are defined by the following equations [1]:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{3}$$

$$ij = -ji = k \tag{4}$$

$$ki = -ik = j \tag{5}$$

$$jk = -kj = i \tag{6}$$

These rules are much the same like the Complex number fundamental equation, as they generalize the algebra of quaternions which will help represent and simplify 3D rotations.

C. Rules & Properties of Quaternions

The previous equations define the rules of quaternions, which are used to perform operations such as addition, subtraction, multiplication, and division. The following are key properties of quaternions:

- **Non-commutativity:** $q_1 * q_2 \neq q_2 * q_1$
- **Conjugate:** $q^* = w - xi - yj - zk$
- **Norm:** $|q| = \sqrt{w^2 + x^2 + y^2 + z^2}$
- **Multiplicative inverse:** $q^{-1} = \frac{q^*}{|q|^2}$, $/|q| \neq 0$

Most of these are the same as the real number system we're all acquainted with, the ones that must have special attention are the Non-commutativity rule and the multiplicative inverse, which are leveraged heavily when representing rotations which well see in chapter 3.

D. Examples of Quaternion Calculations

The addition of quaternions is performed component-wise, similarly to real and complex numbers, as such there will be no example, due to its simplicity and uselessness in representing 3D rotations.

Meanwhile, quaternion multiplication is performed using the distributive property like normal multiplication, but simplifying them requires more work; as such the following example will show a quaternion example with the respective rules from Subsection B when they are used.

Let:

$$q_1 = w_1 + x_1 i + y_1 j + z_1 k \tag{7}$$

$$q_2 = w_2 + x_2 i + y_2 j + z_2 k \tag{8}$$

The product is :

$$\begin{aligned} q_1 * q_2 = {} & w_1 w_2 + w_1 x_2 i + w_1 y_2 j + w_1 z_2 k \\ & + x_1 w_2 i + x_1 x_2 i^2 (3) + x_1 y_2 ij (4) + x_1 z_2 ik (5) \\ & + y_1 w_2 j + y_1 x_2 ji (4) + y_1 y_2 j^2 (3) + y_1 z_2 jk (6) \\ & + z_1 w_2 k + z_1 x_2 ki (5) + z_1 y_2 kj (6) + z_1 z_2 k^2 (3) \end{aligned} \tag{9}$$

$$\begin{aligned} q_1 * q_2 = {} & (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) \\ & + (w_1 x_2 + x_1 w_2 + y_1 z_2 - z_1 y_2) i \\ & + (w_1 y_2 - x_1 z_2 + y_1 w_2 + z_1 x_2) j \\ & + (w_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 w_2) k \end{aligned} \tag{10}$$

These examples demonstrate the foundation of quaternion calculations, the multiplication in particular will be useful in the following sections where the rotation of a vector in three-dimensional space is discussed.

III. Pros and Cons

One of the main reasons of the rise in Quaternion usage over the years, is due to many of their advantages compared to their counterparts, but they are not without faults as we will see in this chapter.

A. Pros

The strengths of quaternions lie in their efficiency and stability, among other unique features.

1) Compact Representation: Quaternions compared to Rotation matrices, an industry standard, occupy much less space in memory, requiring only 4 scalars compared to the needed 9 scalars in a 3x3 matrix. This compactness is very useful in limited-memory environments such as embedded systems.

A side effect from this is another advantage, since there are fewer variables overall, the performance is usually better compared to other representations which is helpful in real-time applications such as gaming and virtual reality, where performance issues may lead to adverse symptoms to the user, like motion sickness.

2) Gimbal Lock: Gimbal lock is the loss of a degree of freedom when certain alignments of the axes occur [2]. For example in three dimensions, gimbal lock occurs when two axes of the gimbals are in a parallel configuration making it impossible to distinguish between certain rotational movements. A visual example of this happening can be seen in Fig.1 and Fig.2.

Quaternions avoid this issue in its entirety because they do not rely on sequential rotations on multiple axes (yaw, pitch and roll) and instead represent the rotation as a single, unified entity. This representation encodes the rotation as a combination of an axis of rotation and an angle, which we will see in the following chapter, ensures smooth and continuous orientation changes without loosing freedom.

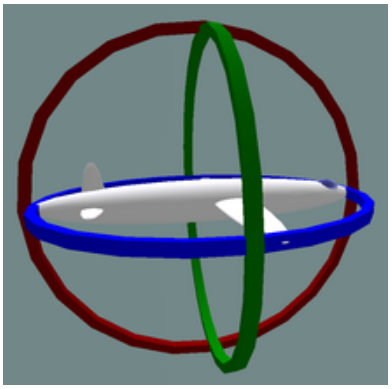

Fig. 1. 3 Independent Gimbals.

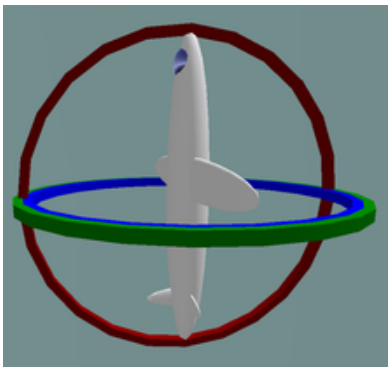

Fig. 2. Roll (Green) and Yaw (Blue) are locked, loosing one degree of freedom.

3) Efficient Interpolation: A technique often used in 3D graphics is Spherical Linear Interpolation (SLERP) [2] [3], because it is crucial in continuous motion, such as character animation and camera movements.

While Euler anglers and rotation matrices require additional calculations or conversions to interpolate, quaternions allow for direct interpolation with a lower overhead.

4) Numerical Stability: The ability to maintain accuracy in repeated calculations, such as when performing multiple operations such as rotations, combinations, or interpolations, is crucial for real-time systems where small numerical errors can accumulate very rapidly.

Since quaternions are typically used in a normalized form, they always represent a valid rotation without scaling, even through multiplication, because the multiplication of two unit quaternions is also a unit quaternion, they can avoid errors that other representations must correct over time.

B. Cons

1) Intuitiveness: Due to quaternions requiring an understanding of concepts like complex number and rotations in higher dimensional space, they are less accessible to people when compared to Euler Angles or rotation matrices which can be easily be translated into visual manner.

2) Debugging: A knock on effect of intuitiveness is making debugging quaternions a more challenging task. Due to the Non-Commutative rule and abstract representation, quater-

nions implementations can very easily have bugs that take work to find out [2].

3) Compatibility: Compatibility is a big problem with quaternions. If conversions between representations are required, such as using legacy systems, it can introduce a lot of overhead depending on the target representation. While Quaternion to rotation matrices are relatively straightforward, converting into Euler angles can be more complex and computationally expensive, which is a problem for real-time systems where speed is important.

IV. 3D Rotations

A. Theory

To represent a rotation using quaternions, first choose an axis of rotation and represent it with a unit vector, which is a quaternion with only the imaginary components, $xi+yj+zk$, normalized so that, $x^2+y^2+z^2=1$.

Next choose an angle, phi, which will be how much we want to rotate, and construct a quaternion like the following [1],

$$q = cos(phi/2) + sin(phi/2) * (xi + yj + zk) \quad (11)$$

with the imaginary part being the axis of rotation that was chosen, the reason the angle inside the trigonometry functions is $phi/2$ will be explained later when we reach the practical example.

Now we choose a specific 3D point for which we want to know the coordinates after the rotation of phi around the specified axis, which we will write with the imaginary components and we will perform the following operation

$$p' = q * p * q^{-}1 \quad (12)$$

which will give us the rotated version of the point.

B. Practical Example

To further help understand the concept, we will look into a concrete example.

Imagine we have a sphere and we want it to rotate around the X axis. We must first define our vector, and since the imaginary components are each assigned to a plane in the three dimensions, the vector can be seen in Fig.3 as the purple arrow, will be the following equation:

$$V = 1i + 0j + 0k \quad (13)$$

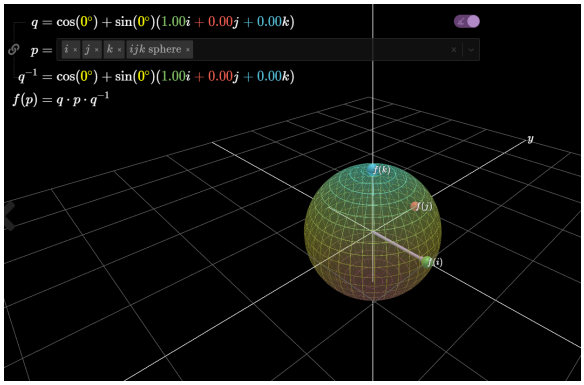

Fig. 3. Initial state, no rotations

Next we must decide the angle of rotation we want, for this example we will want to rotate the sphere 90 degrees. This means that the rotation will be represented by the following equation:

$$p' = (cos(45) + sin(45) * (1i + 0j + 0k)) * p \quad (14)$$
$$*(cos(-45) + sin(-45) * (1i + 0j + 0k))$$

The reason we use half of the desire angle on each side, is due to quaternions representing a hyper sphere [4], where it rotates while stretching out as seen in Fig.4, q, and rotates while returning to original size $q^{-}1$. The result is a sphere that was rotated 90 degrees around the x axis, Fig.5.

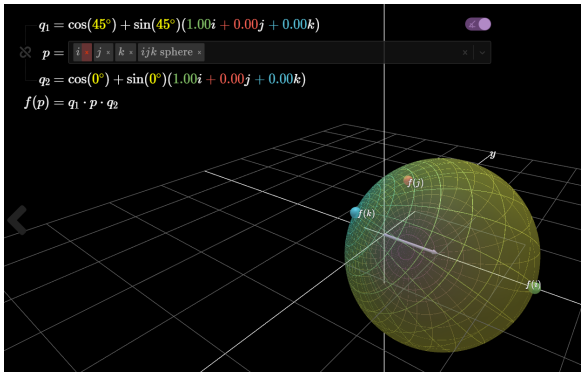

Fig. 4. Middle state, rotation applied only to q

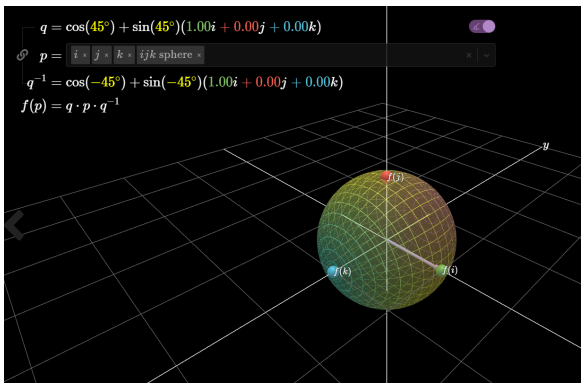

Fig. 5. Final state, rotation completed

Since we require only half of the angle to reach our 90 degree example, there's actually another angle that we could use to reach the same rotation, Fig.6, this is referred to as Double coverage, and allows quaternions to interpolate between a short path, for snappy movement or a longer path for smoother movement.

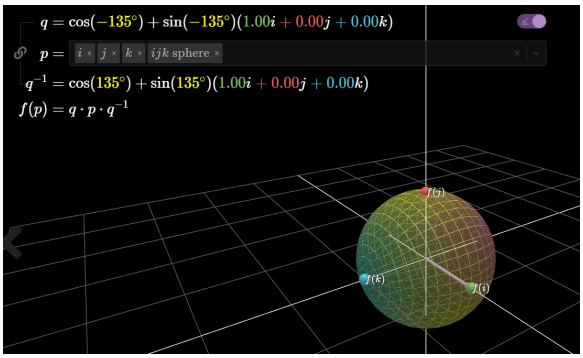

Fig. 6. Alternative Final state, rotation completed

V. Conclusion

As shown throughout this article, quaternions are a powerful tool when used correctly, their usage goes beyond 3D Graphics, appearing also in other fields like Robotics and Aerospace navigation. They're hard to understand do to their abstractness but this is outweighed by the advantages they offer over other more traditional methods, and with the increasing computational demand, quaternions are an useful system to keep in mind.

VI. Extra Resources

This sections includes some tools and videos, that are helpful to understand further concepts of quaternions.

- **Videos:**
  - How quaternions produce 3D rotation by Penguin-Maths
  - Visualizing quaternions (4d numbers) with stereographic projection by 3Blue1Brown
  - Quaternions and 3d rotation, explained interactively by 3Blue1Brown
- **Website:**
  - Website with interactive videos by 3Blue1Brown and Ben Eater

References

[1] J. C. Hart, G. K. Francis, and L. H. Kauffman, "Visualizing quaternion rotation," *ACM Trans. Graph.*, vol. 13, no. 3, p. 256–276, Jul. 1994. [Online]. Available: https://doi.org/10.1145/195784.197480

[2] M. Toso, E. Pennestri, and V. Rossi, "Esa multibody simulator for spacecrafts' ascent and landing in a microgravity environment," *CEAS Space Journal*, vol. 7, 03 2015.

[3] K. Shoemake, "Animating rotation with quaternion curves," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, p. 245–254, Jul. 1985. [Online]. Available: https://doi.org/10.1145/325165.325242

[4] R. Goldman, "Understanding quaternions," *Graphical Models*, vol. 73, pp. 21–49, 03 2011.