Quaternion Rotation:

A Magical Journey to the Fourth Dimension

MA 398: Senior Seminar

Linnea Caraballo

Supervised by Prof. Tina Romansky

Fall 2021

Abstract

Quaternions are an extension of the complex number system and have a large presence in various applied fields. The most common use of quaternions is to model the rotation of three-dimensional objects. In this paper we will explore what quaternions are, discuss some quaternion history and establish the fundamental concepts of quaternion algebra. This will lead us into a discussion of quaternion rotation, how it works, and why it is the preferred method for rotation by highlighting some common issues with other forms of rotation. We will then analyze original data depicting the speed at which quaternions can be calculated versus other forms of rotation. So let us go on a magical journey to the fourth dimension as we explore quaternions.

1 Introduction

Quaternions are an extension of the complex number system and consist of one real part and three imaginary parts. This means that quaternions exist in the fourth dimension and were first described by William Rowan Hamilton, an Irish mathematician. Hamilton was struck with the idea of adding a fourth dimension in order to multiply triples while on a walk with his wife crossing the Broom Bridge and into the bridge Hamilton carved the fundamental property of quaternions.

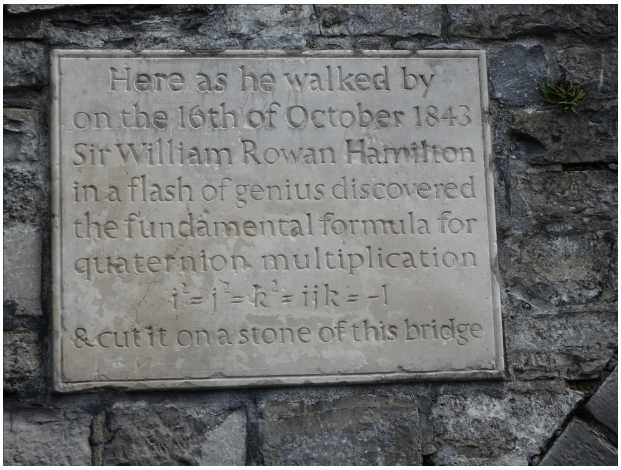


Figure 1: The Broom Bridge [4]

Definition 1.1. *A quaternion is a number of the form:*

$$q = s + a\vec{i} + b\vec{i} + c\vec{k}$$

where $s, a, b, c \in \mathbb{R}$ *and* $\vec{i}, \vec{j}, \vec{k}$ *are imaginary. We call* s *the* ***scalar*** *part and* i, j, k *the* ***vector*** *part. We denote the set of quaternions as* $\mathbb{H}$.

Definition 1.2. *The fundemental property of quaternions is:*

$$i^2 = j^2 = k^2 = -1$$

Because quaternions have a real and imaginary part it is important to go over complex numbers, especially their relation to vectors. A **complex number** is a combination of a real number and an imaginary number in the form of $a + bi$, where a is the real part and i is the imaginary unit. We know that $i = \sqrt{-1}$, so $\sqrt{-4} = 2i$, and we define the set of complex numbers as $\mathbb{C}$. Since we express a complex number in terms of a real and imaginary part, we can also describe it in terms of a two-dimensional vector, i.e. $4+2i$ can be represented as a vector $\langle 4, 2\rangle$. Essentially we take the real parts a, b and put them into a vector described in 2D space. Thus we can treat complex numbers as vectors. However, not all vector properties translate over to complex numbers, which is also the case with quaternions. Thus we need to use the rules of quaternion algebra.

Quaternions are primarily used to model the orientation and rotation of three-dimensional objects. What this means is that quaternions lie in $\mathbb{R}^4$, but operate on objects and vectors in $\mathbb{R}^3$. This seems strange; how does something in 4D, a quaternion, affect something in 3D? One cannot even visualize something in 4D. The easiest way to understand how quaternions affect 3D space is to go down a dimension and instead of considering a 4D object projecting onto a 3D space,

think of projecting a 3D solid object onto a 2D plane (something we experience with shadows). With shadows the sun hits an object like a tree at a certain angle and then a 2D image, the shadow, is projected onto the ground. We can also see this with stereographic projection which can actually extend into the 4D space with the use of a hypersphere [2]. When we accept the projection of shadows and stereographic projection, the idea of a higher dimension-object being projected onto a lower dimension becomes a little less abstract.

2 What are Quaternions?

Quaternions, just like complex numbers, are similar to vectors. With vectors $\vec{i}, \vec{j}, \vec{k}$ are the unit vectors on the x, y, and z axis respectively. With quaternions we are however viewing $\vec{i}, \vec{j}, \vec{k}$ as imaginary numbers because they represent the unit quaternions on the x, y, and z axis respectively. But, because we have $\vec{i}$ as well as $\vec{j}$ and $\vec{k}$ and a scalar s we have been extended into the fourth dimension.

We can then treat the $\vec{i}, \vec{j}, \vec{k}$ component as a vector allowing us to split up a quaternion into two different parts, a scalar and a vector. Therefore we can say a quaternion $q = s + \vec{v}$ or $q = [s, \vec{v}]$ such that $s \in \mathbb{R}$ and $\vec{v} = a\vec{i} + b\vec{j} + c\vec{k}$ [9]. The ability to split up quaternions like this, allows us to consider quaternions with the scalar or vector part equal to zero which will result in two other forms

of quaternions.

Definition 2.1. *A **pure quaternion** is a quaternion with a scalar part of 0, meaning $s = 0$ when referring to Definition 1.1.*

Example 1. Given the quaternions q_1, and q_2 below, only q_2 is a pure quaternion.

$$q_1 = 5 + 2\vec{i} + 3\vec{j} - 8\vec{k} \qquad q_2 = 4\vec{i} - 2\vec{j}$$

Definition 2.2. *A **scalar quaternion** is a quaternion with a vector part of 0, meaning $q = s$.*

Example 2. Given the quaternions q_3, and q_4 below, only q_4 is a scalar quaternion.

$$q_3 = 8\vec{i} + 3\vec{j} + 4\vec{k} \qquad q_4 = 4$$

Theorem 2.1. *We can relate the imaginary terms $\vec{i}, \vec{j}, \vec{k}$ as follows:*

$$\vec{k}\vec{i} = \vec{j},\ \vec{i}\vec{j} = \vec{k},\ \vec{j}\vec{k} = \vec{i}$$

$$\vec{i}\vec{k} = -\vec{j},\ \vec{j}\vec{i} = -\vec{k},\ \vec{k}\vec{j} = -\vec{i}$$

$$\vec{i}\vec{j}\vec{k} = \vec{j}\vec{k}\vec{i} = \vec{k}\vec{i}\vec{j} = -1$$

Remark: We can see that when we multiply the imaginary terms, they are **not** commutative. This is important to keep in mind as these relations between the imaginary parts are the reason why quaternion multiplication is not commutative.

Proof. We know from Definition 1.2 that $\vec{i}^2 = \vec{j}^2 = \vec{k}^2 = \vec{i}\vec{j}\vec{k} = -1$. Thus we can begin with $\vec{i}\vec{j}\vec{k} = -1$ and use it to prove the relations.

We will first show that $\vec{i}\vec{j} = \vec{k}$. We know that $\vec{i}\vec{j}\vec{k} = -1$, thus we can left multiply both sides by $\vec{i}$ to obtain:

$$\begin{aligned}
\vec{i}(\vec{i}\vec{j}\vec{k}) &= \vec{i}(-1) \\
\vec{i}\vec{i}\vec{j}\vec{k} &= -\vec{i} \\
-(\vec{j}\vec{k}) &= -\vec{i} \\
\vec{j}\vec{k} &= \vec{i}
\end{aligned}$$

We can now show that $\vec{j}\vec{i} = -\vec{k}$. Recall $\vec{i}\vec{j}\vec{k} = -1$, so we can left multiply both sides by $\vec{j}\vec{i}$, such that:

$$\begin{aligned}
\vec{j}\vec{i}(\vec{i}\vec{j}\vec{k}) &= -\vec{j}\vec{i} \\
\vec{j}(-1)\vec{j}\vec{k} &= \vec{j}\vec{i} \\
-(-1)\vec{k} &= -\vec{j}\vec{i} \\
\vec{k} &= -\vec{j}\vec{i} \\
\vec{j}\vec{i} &= -\vec{k}
\end{aligned}$$

The desired results can be obtained for the other relations using a similar method.

□

3 Quaternion Algebra

Now that some background information on quaternions has been established and we understand how the imaginary parts of a quaternion relate to each other; we can now think about adding, subtracting, multiplying, and dividing quaternions. This is called quaternion algebra and it is important to gain an understanding of this as many of the formulas and techniques of quaternion rotation build off of quaternion algebra.

3.1 Quaternion Multiplication

Quaternion multiplication is not the same as vector multiplication in the same way that multiplying complex numbers is not the same as multiplying vectors. Recall a complex number $a+bi$ can be expressed as a two-dimensional vector, but multiplication of complex numbers is different because the properties of vectors and complex numbers are different. A complex number has an imaginary part while a vector does not, and quaternions have three imaginary parts, so we must preserve the relations between $\vec{i}, \vec{j}, \vec{k}$.

Remark: In general $q_1 q_2 \neq q_2 q_1$ for $q_1, q_2 \in \mathbb{H}$

Example 1. *Given* $q_1 = 2 + 3\vec{i} + 5\vec{k},\ q_2 = 1 - 2\vec{j} + 3\vec{k}$

$$q_1 q_2 = (2 + 3\vec{i} + 5\vec{k})(1 - 2\vec{j} + 3\vec{k})$$

$$q_1q_2 = 2(1 - 2\vec{j} + 3\vec{k}) + 3\vec{i}(1 - 2\vec{j} + 3\vec{k}) + 5\vec{k}(1 - 2\vec{j} + 3\vec{k})$$

$$= 2 - 4\vec{j} + 6\vec{k} + 3\vec{i} - 6\vec{i}\vec{j} + 9\vec{i}\vec{k} + 5\vec{k} - 10\vec{k}\vec{j} + 15\vec{k}^2$$

$$= 2 - 4\vec{j} + 6\vec{k} + 3\vec{i} - 6\vec{k} + 9(-\vec{j}) + 5\vec{k} - 10(-\vec{i}) + 15(-1)$$

$$= -13 + (3 + 10)\vec{i} + (-4 - 9)\vec{j} + (6 - 6 + 5)\vec{k}$$

$$q_1q_2 = -13 + 13\vec{i} - 13\vec{j} + 5\vec{k}$$

Remark: $q_2q_1 = -13 - 7\vec{i} + 5\vec{j} + 17\vec{k}$

Theorem 3.1. *Given two quaternions* $q_1 = s_1 + a_1\vec{i} + a_2\vec{j} + a_3\vec{k}$ *and* $q_2 = s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k}$ *multiplying them together gets us the general formula*

$$q_1q_2 = (s_1s_2 - a_1b_1 - a_2b_2 - a_3b_3) + (s_1b_1 + a_1s_2 + a_2b_3 - a_3b_2)\vec{i}$$

$$(s_1b_2 - a_1b_3 + a_2s_2 + a_3b_1)\vec{j} + (s_1b_3 + a_1b_2 - a_2b_1 + a_3s_2)\vec{k}.$$

Proof. Given two quaternions $q_1 = s_1 + a_1\vec{i} + a_2\vec{j} + a_3\vec{k}$ and $q_2 = s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k}$ we want to establish the general formula for quaternion multiplication.

$$q_1q_2 = (s_1 + a_1\vec{i} + a_2\vec{j} + a_3\vec{k})(s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k})$$

$$= s_1(s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k}) + a_1\vec{i}(s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k}) +$$

$$a_2\vec{j}(s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k}) + a_3\vec{k}(s_2 + b_1\vec{i} + b_2\vec{j} + b_3\vec{k})$$

$$= s_1s_2 + s_1b_1\vec{i} + s_1b_2\vec{j} + s_1b_3\vec{k} + a_1s_2\vec{i} + a_1b_1(\vec{i})^2 + a_1b_2\vec{i}\vec{j} + a_1b_3\vec{i}\vec{k} +$$

$$a_2s_2\vec{j} + a_2b_1\vec{j}\vec{i} + a_2b_2(\vec{j})^2 + a_2b_3\vec{j}\vec{k} + a_3s_2\vec{k} + a_3b_1\vec{k}\vec{i} + a_3b_2\vec{k}\vec{j} + a_3b_3(\vec{k})^2$$

$$= s_1s_2 + s_1b_1\vec{i} + s_1b_2\vec{j} + s_1b_3\vec{k} + a_1s_2\vec{i} - a_1b_1 + a_1b_2\vec{k} + a_1b_3(-\vec{j}) +$$

$$a_2s_2\vec{j} + a_2b_1(-\vec{k}) - a_2b_2 + a_2b_3\vec{i} + a_3s_2\vec{k} + a_3b_1\vec{j} + a_3b_2(-\vec{i}) - a_3b_3$$

$$q_1q_2 = (s_1s_2 - a_1b_1 - a_2b_2 - a_3b_3) + (s_1b_1 + a_1s_2 + a_2b_3 - a_3b_2)\vec{i} +$$

$$(s_1b_2 - a_1b_3 + a_2s_2 + a_3b_1)\vec{j} + (s_1b_3 + a_1b_2 - a_2b_1 + a_3s_2)\vec{k}$$

Thus using Theorem 2.1 and regular algebra we have found the general solution.

□

We can also use what we know from linear algebra to simplify quaternion multiplication through the use of dot and cross products.

Theorem 3.2. *For quaternions $q_1 = s_1 + \vec{v}_1$ and $q_2 = s_2 + \vec{v}_2$ we have:*

$$q_1q_2 = (s_1s_2 - \vec{v}_1 \cdot \vec{v}_2) + (s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2) \quad (1)$$

$$q_2q_1 = (s_1s_2 - \vec{v}_1 \cdot \vec{v}_2) + (s_1\vec{v}_2 + s_2\vec{v}_1 - \vec{v}_1 \times \vec{v}_2) \quad (2)$$

Proof. Let there be two quaternions defined as $q_1 = s_1 + \vec{v}_1$ and $q_2 = s_2 + \vec{v}_2$ where $\vec{v}_1 = a_1\vec{i} + a_2\vec{j} + a_3\vec{k}$ and $\vec{v}_2 = b_1\vec{i} + b_2\vec{j} + b_3\vec{k}$. Observe that the dot product of $\vec{v}_1 \cdot \vec{v}_2 = a_1b_1 + a_2b_2 + a_3b_3$ and the cross product of $\vec{v}_1 \times \vec{v}_2 = (a_2b_3 - a_3b_2)\vec{i} - (a_1b_3 - a_3b_1)\vec{j} + (a_1b_2 - a_2b_1)\vec{k}$. We will now substitute everything into Equation (1) to obtain

$$q_1q_2 = (s_1s_2 - (a_1b_1 + a_2b_2 + a_3b_3)) + [s_1b_1\vec{i} + s_1b_2\vec{j} + s_1b_3\vec{k} +$$

$$s_2a_1\vec{i} + s_2a_2\vec{j} + s_2a_3\vec{k} + (a_2b_3 - a_3b_2)\vec{i} - (a_1b_3 - a_3b_1)\vec{j} + (a_1b_2 - a_2b_1)\vec{k}].$$

Simplifying and combining like terms we get the following result

$$\begin{aligned} q_1q_2 =&(s_1s_2 - a_1b_1 - a_2b_2 - a_3b_3) + (s_1b_1 + s_2a_1 + a_2b_3 - a_3b_2)\vec{i}+ \\ &(s_1b_2 + s_2a_2 - a_1b_3 + a_3b_1)\vec{j} + (s_1b_3 + s_2a_3 + a_1b_2 - a_2b_1)\vec{k}. \end{aligned}$$

Observe that after moving around a few terms we obtain the general equation for quaternion multiplication established in Theorem 3.1. You can prove Equation (2) in the same way. □

Theorem 3.3. *The only quaternions we graphically visualize are pure quaternions. So, for* ***pure quaternions*** $\vec{v}$ *and* $\vec{w}$ *we have:*

$$\vec{v}\vec{w} = \vec{v} \times \vec{w} - \vec{v} \cdot \vec{w}$$

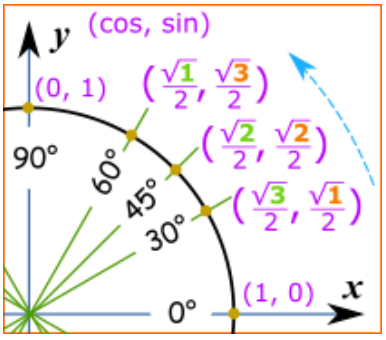
$$\vec{w}\vec{v} = \vec{w} \times \vec{v} - \vec{w} \cdot \vec{v} = -(\vec{v} \times \vec{w}) - \vec{v} \cdot \vec{w}$$

We can see that we have a cross product and dot product in Theorem 3.3, so let us get a visual understanding of how two pure quaternions can interact using what we know from linear algebra. First let us evaluate what happens to the quaternions when the cross product of the two quaternions is zero. In this case, they are parallel to each other. However, with pure quaternions we still have the dot product part of the equation from Theorem 3.3. Looking at these two parts

of the equation we can gain some insight on how these quaternions will change and how they relate to each other graphically once we multiply them together. Let us first recall the unit circle.

Utilizing the unit circle as shown in Figure 2, we can see the values that sine and cosine take at different angles. Observe that as the angle gets larger, cosine gets smaller and sine gets larger when we are in quadrant one. With this intuition we are able to understand that as the two pure quaternions, $\vec{v}$ and $\vec{w}$, turn so that they start to become more perpendicular the angle θ between the two quaternions is heading to 90°. Observe that the dot product, $\vec{v} \cdot \vec{w} = |\vec{v}||\vec{w}| \sin(\theta)$, will then grow because of the sine component. What we can conclude from this is that as the cross product of the two quaternions shrinks, the dot product of the two quaternions grows so that they become more perpendicular. Essentially, what we lose with $\vec{v} \times \vec{w}$ we gain with $\vec{v} \cdot \vec{w}$, and the opposite can be said when the quaternions become more parallel.

Figure 2: The Unit Circle



3.2 Conjugates

Another important aspect of quaternion algebra is the conjugate. The conjugate is especially important when using quaternions for rotation. We will explore

the role of conjugates for the rotation of quaternions in Section 4.

Definition 3.1. *The **conjugate** of a quaternion* $q = s + a\vec{i} + b\vec{j} + c\vec{k}$ *is denoted* $\boldsymbol{q^*}$ *where* $q^* = s - a\vec{i} - b\vec{j} - c\vec{k}$

Theorem 3.4. *If* $q_1, q_2 \in \mathbb{H}$ *then* $(q_1 q_2)^* = q_1^* q_2^*$

3.3 Norm

Definition 3.2. *The **norm** (magnitude) of a quaternion* $q = s + a\vec{i} + b\vec{j} + c\vec{k}$ *is:* $|q| = \sqrt{s^2 + a^2 + b^2 + c^2}$

Note: A unit quaternion is a quaternion with a norm of 1:

$$|q| = \sqrt{s^2 + a^2 + b^2 + c^2} = 1$$

Theorem 3.5. *The norm of a quaternion is multiplicative so for* $q_1,\ q_2 \in \mathbb{H}$*:* $|q_1 q_2| = |q_1||q_2|$

4 Quaternions and Rotation of 3D Objects

Now that we have established a basic understanding of quaternions we can explore quaternion rotation. Quaternions are used to represent the rotation of 3D objects, however quaternions are not the only method to rotate these objects. It is important to note that many of these alternative methods are easier to understand

on a conceptual level and are also more intuitive, yet there are issues that come up with these other methods which make quaternions favorable. So, what are these issues, and why does that make quaternions better to use despite their abstract nature?

Recall from Theorem 3.1 that the multiplication of quaternions is not computationally difficult and this translates over to calculations being used for quaternion rotation. The most difficult part about quaternions is the idea of having something in 4D affect something in 3D, which becomes less trivial when thinking of shadows and stereographic projection as described at the end of Section 1. Because calculations with quaternions are not computationally difficult it makes them ideal to use in software that is already computationally heavy, such as those used to model 3D objects. This is because there are less processes that need to be carried out by the computer which leads to less memory and processing power being needed.

Theorem 4.1. *Given a unit vector $\vec{u}$ and some vector $\vec{v}$, where $\vec{u}, \vec{v} \in \mathbb{R}^3$, we want to rotate $\vec{v}$ around $\vec{u}$ by an angle θ. To do this we must first find a quaternion*

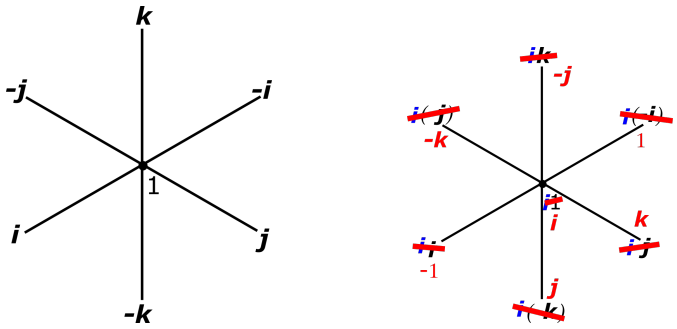
$$q = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\vec{u}.$$

To find the quaternion that represents the rotation:

$$R_q(\vec{v}) = q\vec{v}q^*$$

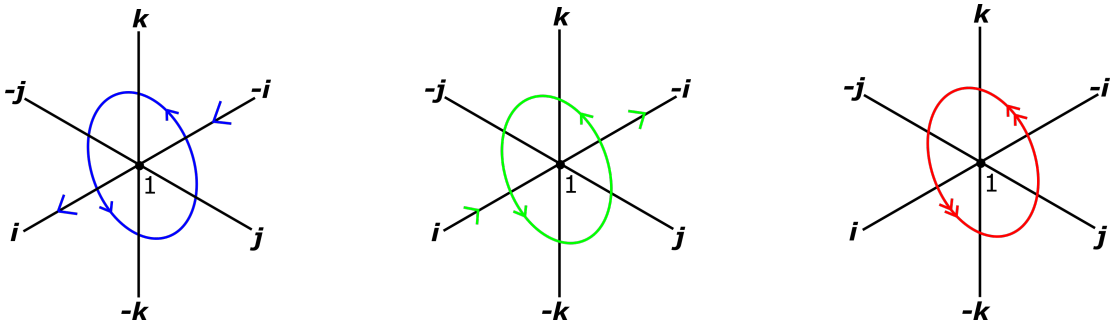
Observe that we use the quaternion, q, and the same quaternions' conjugate, q^*. Keep in mind that quaternion multiplication is not commutative. This means that $q\vec{v}q^* \neq q(q^*)\vec{v}$. The order in particular is extremely important and can be understood more when we try to visualize it. This visualization will help show how the quaternion rotation formula works and also explain why we use $\frac{\theta}{2}$ instead of θ.

Example 1. Given a quaternion q = $\vec{i}$ and some vector $\vec{v}$, we want to rotate $\vec{v}$ around our quaternion q. This means that our quaternion representing the rotation is $R_q(\vec{v}) = \vec{i}\vec{v}(-\vec{i})$. Now, let us get some insight into how this rotation looks and works by breaking down $R_q(\vec{v}) = q\vec{v}q^*$.



(a) Original coordinate system

(b) **Left** multiply by $\vec{i}$



(c) $q\vec{v} = \vec{i}\vec{v}$ (d) $\vec{v}q^* = \vec{v}(-\vec{i})$ (e) $q\vec{v}q^* = \vec{i}\vec{v}(-\vec{i})$

Figure 3: Visualization of Example 1

In Figure 3b we are **left** multiplying all coordinate values by $\vec{i}$. Doing this causes the coordinate system to rotate clockwise on the jk-axis. Observe that when we then carry out the first part of our rotation formula, $q\vec{v}$, in Figure 3c, that this results in a counter-clockwise rotation of our object on the jk-axis (see Appendix B for explanation), as well as a rotation on the i-axis. When we then carry out the second part of the rotation formula $\vec{v}q^*$, in Figure 3d, we then have the same counter-clockwise rotation on the jk-axis, but the opposite rotation of direction on the i-axis.

This creates a very interesting result when we combine Figure 3c and Figure 3d which shows what the whole formula looks like as a whole in Figure 3e. The rotation on the i-axis cancels out because they were going opposing directions in Figure 3c and Figure 3d. However, observe that since we had the same counter-

clockwise rotation on the jk-axis, we end up having a double rotation on the jk-axis in Figure 3e. This is called double coverage, and is why in Theorem 4.1 we have to divide θ by 2. When we rotate something in 4D by 180°, if we do not account for double coverage, our object in 3D will end up rotating 360° which means that the object is unaffected. To counter-act this double coverage we must then use $\frac{\theta}{2}$.

It is also important to point out that the use of the conjugate allows us to focus on the rotation of the two perpendicular axes, instead of all three. This is also why the order of our rotation formula, $R_q(\vec{v}) = q\vec{v}q*$, is so important. It allows for the cancellation of one axis because of the opposing rotation. Figure 4 is a generalization of the rotation formula.

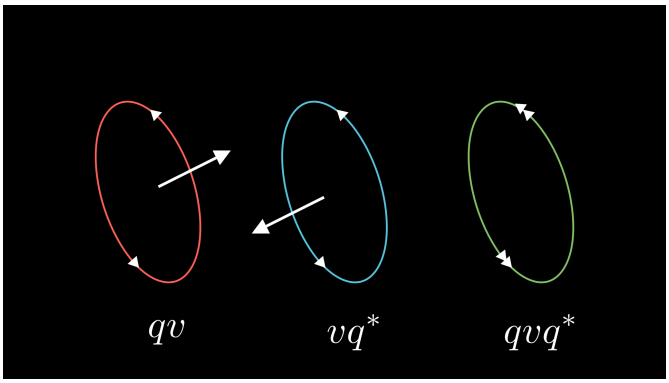


Figure 4: General visual representation of quaternion rotation, Theorem 4.1

Let us now look specifically at two of the most common ways to rotate 3D objects and compare them to the use of quaternions. We will analyze rotation

using Euler angles and Rodrigues' rotation formula for the rotation of a vector around a unit vector. First let's analyze the use of Euler angles with rotation matrices and quaternions to see why quaternions are superior in this case.

4.1 Euler Angles vs. Quaternions

The traditional technique used to rotate points and frames of reference is to use Euler rotations which utilize Euler angles. Euler rotations allow for rotation about one of the three Cartesian axes, a combination of two rotations about two different axes, or a combination of any three rotations. Euler rotations use rotation matrices in order to describe the rotation.

Definition 4.1. *A rotation around the x, y, or z axis can be defined by R_x, R_y, and R_z respectively. These matrices describe the rotation by an angle θ around each respective axis. All three matrices can then be combined to find the general rotation matrix.*

$$Rx = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \qquad Ry = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix},$$

$$Rz = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrices are easy to visualize and utilize, so why not use them? Well, when we need to combine more than one rotation, the calculations can get long and tedious, especially if doing the calculations out by hand. This also means that there are more processes that need to be completed by the computer making it more computationally demanding. However, there is one very large problem with using rotation matrices and that has to do with something called gimbal lock.

When looking at something in three-dimensions there are three axes each of which we are free to move around without necessarily affecting the others. Each axis is a degree, and being able to move around the axis means that it has a degree of freedom. Essentially this means that each axis is an independent parameter, so a 3D object can have up to three degrees of freedom. Gimbal lock happens in very specific cases when the angles of rotation on each axis causes two of them to become parallel and lock. Once this happens we cannot unlock the axes and one degree of freedom is lost, so instead of having three degrees of freedom we only have two.

Gimbal lock is a huge issue in computer graphics, and even animators will run

into gimbal lock irrespective of the software being used. Gimbal lock will always be a potential issue and while there are ways of mitigating its effect, it is inevitable when using Euler angles for rotation. The only way to avoid gimbal lock is to use another form of rotation; this is one reason why we use quaternions.

Another big advantage of using quaternions over rotation matrices is the compute time. To see this for ourselves let us analyze some real world data that takes three random Euler angles and uses them to compare the compute time of the rotation matrix and quaternion.

In order to conduct this comparison, we are going to need to use a different equation to find the quaternion. The general way of finding quaternions is using two vectors, one being a unit vector. In this case our axis will act as that unit vector. Recall back in Linear Algebra that $\vec{i} = \langle 1, 0, 0 \rangle$, a unit vector. Thus using the first part of Theorem 4.1 we will have

$$q_\theta = \cos\frac{\theta}{2} + \sin\frac{\theta}{2}\vec{i}$$
$$q_\phi = \cos\frac{\phi}{2} + \sin\frac{\phi}{2}\vec{j}$$
$$q_\psi = \cos\frac{\psi}{2} + \sin\frac{\psi}{2}\vec{k}.$$

In order to combine these three quaternions we will simply multiply them in the

order $q_\psi q_\phi q_\theta$ to obtain the formula

$$\begin{aligned} q_\psi q_\phi q_\theta \quad &= \cos\tfrac{\psi}{2}\cos\tfrac{\phi}{2}\cos\tfrac{\theta}{2} + \sin\tfrac{\psi}{2}\sin\tfrac{\phi}{2}\sin\tfrac{\theta}{2} \\ &+(\sin\tfrac{\psi}{2}\cos\tfrac{\phi}{2}\cos\tfrac{\theta}{2} - \cos\tfrac{\psi}{2}\sin\tfrac{\phi}{2}\sin\tfrac{\theta}{2})\vec{i} \\ &+(\cos\tfrac{\psi}{2}\sin\tfrac{\phi}{2}\cos\tfrac{\theta}{2} + \sin\tfrac{\psi}{2}\cos\tfrac{\phi}{2}\sin\tfrac{\theta}{2})\vec{j} \\ &+(\cos\tfrac{\psi}{2}\cos\tfrac{\phi}{2}\sin\tfrac{\theta}{2} - \sin\tfrac{\psi}{2}\sin\tfrac{\phi}{2}\cos\tfrac{\theta}{2})\vec{k} \end{aligned}$$

Knowing how to compute the rotation matrix and quaternion we can now directly compare their compute times. The hypothesis is that quaternions are a faster way to compute the rotation of a 3D object because there are only four numbers needed for a quaternion representing the rotation versus nine numbers needed for a rotation matrix. To test this we will use a Python script that will compare the compute times of the rotation matrix and quaternion given three random Euler angles for each of the three axes. What we want is for the bars on the graph to be as short as possible as that indicates a faster compute time.

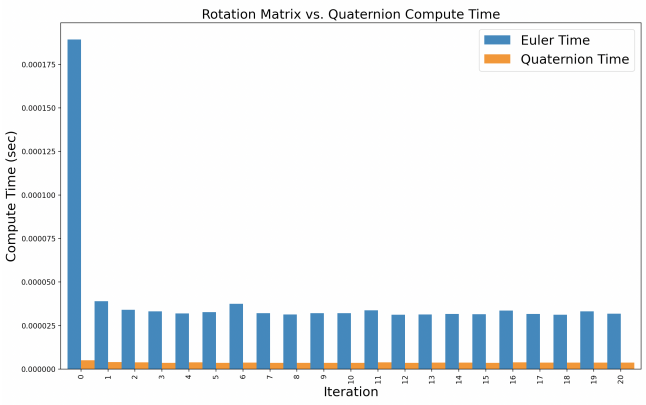


Figure 5: Graph to compare the compute time of Euler angles and quaternions.

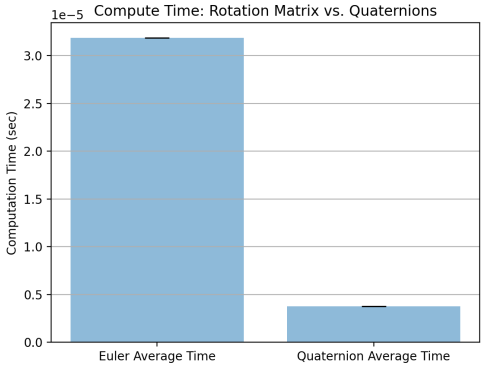


Figure 6: Graph to compare the average compute time of Euler angles and quaternions.

Observing the bars of the graphs it is clear to see that the Euler compute time is always greater than the quaternion compute time. Thus it always takes longer to compute the rotation matrix given three random Euler angles than the quaternion given the same three random Euler angles. The spikes in the graph indicated that there are times where the rotation matrix is more complicated to compute, but we can see that the quaternion compute times are always steady, so they have a more consistent compute time.

4.2 Rodrigues' Formula vs. Quaternions

In 1840 three years before Hamilton came up with quaternions, another mathematician named Benjamin Olinde Rodrigues published a paper describing how one

can use a single rotation about a third axis to represent two successive rotations about two different axes. Rodrigues proved that if we have a unit vector $\vec{u}$ and $\vec{v}$ is some other vector, we can rotate $\vec{v}$ around $\vec{u}$ by an angle θ in a counterclockwise direction with respect to the right hand rule.

$$Rot(\vec{v}) = (1 - \cos(\theta))(\vec{u} \cdot \vec{v})\vec{u} + \cos(\theta)\vec{v} + \sin(\theta)(\vec{u} \times \vec{v})$$

Rodrigues' formula is easy to understand, as it contains many components that we see in classes such as Linear Algebra. This seems like a nice solution. We have a formula and we do not have to make a jump to the fourth dimension. However, the issue with this formula lies in the complexity of the computations, especially with the cross product as well as the angle of rotation. Recall that multiplying two quaternions gets out a quaternion, so if we are strictly using quaternions there is also no need to find the quaternion like in Theorem 4.1, we can just skip to the rotation formula which was not computationally difficult making quaternions more ideal.

Again we will directly compare the compute times of Rodrigues' rotation formula and quaternions, using Theorem 4.1. This Python script generates two random vectors and an angle of rotation, and will convert one of the vectors into a unit vector. Shorter bars on the graph indicate faster compute times, so lets see how the compute times of Rodrigues' rotation formula and quaternions compare.

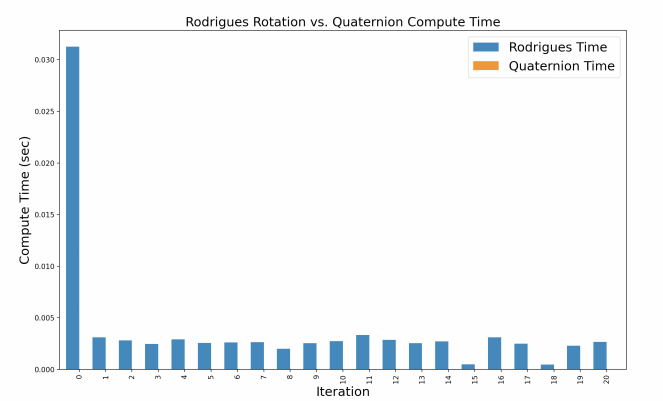


Figure 7: Graph to compare the compute time of Rodrigues and quaternions.

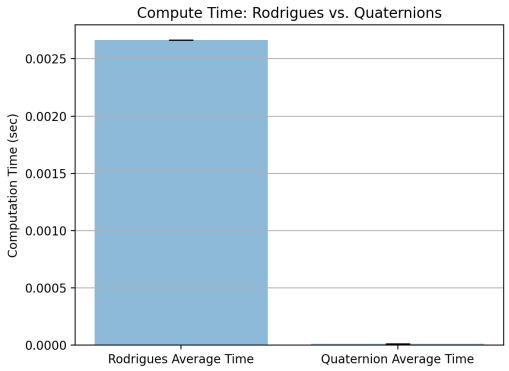


Figure 8: Graph to compare the average compute time of Rodrigues and quaternions.

Again we can see that there is a significant difference between the compute times of Rodrigues' rotation formula and quaternions. This indicates that computing quaternions is faster than two of the most popular forms of rotations.

5 More Applications and Further Research Being Conducted

There are many different applications for quaternions the most common being their use in 3D software such as Unity. But there are many more ways to use quaternions from health and fitness to drones and robots.

One of the most interesting uses of quaternions in recent years is the use of quaternions in smart watches to detect the movement of runners. This is done by the creation of a Wear OS app that "receives orientation (quaternion) data and matches the sensors to the arm or leg segments using a flexible and simple procedure" [6]. Being able to use quaternions allows for more accurate, rapid, and less problematic data that can be analyzed and used in order to benefit the wearer. Smart accessories are increasing in popularity, and many companies rave about the health insights and benefits that they can provide. However, smart watches can only do so much with the sensors they are currently equipped with, and for elite athletes a lot of money can be spent on extensive testing used to improve performance. Utilizing quaternions as a way to get additional data from current smart watches shows us how we can get more health benefits out of our technology and allows for more data to be received at a more reasonable cost.

Quaternions are even being used to create watermark images as described in

[5] and [7]. With the internet and digital media becoming an even more integrated part of everyone's daily lives issues of cybersecurity like duplicating and editing multimedia content have become more important to address. This has made copyrights both more important and complicated, and this is where watermarks come in. Watermarking is when one imposes a logo or pattern that can be seen when held up to the light. This helps to avoid fraud, which means it must be done in a way that is not easily duplicated which is extremely important when it comes to sensitive medical documents. This is where generalized Fourier descriptor, QR code, and quaternion discrete Fourier transforms come into play. These allow for higher quality and harder to reproduce watermarks protecting the legitimacy and copyright of important documents and images.

There are also applications in the field of robotics, from using quaternions with robotic arms to drones. We are able to get more fluid arm movements and ensure that our drones are able to keep on flying even after being disrupted in the air. Even NASA uses quaternions! Overall quaternions make up an extremely useful area of math that should be further explored, as their applications extend far outside of computer graphics and into the innovation of the most cutting edge technology we use.

References

[1] David. (2019) "How Quaternions Produce 3D Rotations." Penguin Maths, https://penguinmaths.blogspot.com/2019/06/

[2] Eater, B. and Sanders G. (2018). "Visualizing Quaternions, an Explorable Video Series." Ben Eater, https://eater.net/quaternions

[3] Nelli, F. (2020). "Hamilton's quaternions and 3D rotation with Python" Meccanismo Complesso, https://www.meccanismocomplesso.org/en/hamiltons-quaternions-and-3d-rotation-with-python/

[4] Murphy, W. (2018). Broom Bridge or is it Broombridge or Even Broome Bridge [Photograph]. Flicker. https://flic.kr/p/2aEfCR5

[5] Li, M., Yuan, X., Chen H., and Li, J., "Quaternion Discrete Fourier Transform-Based Color Image Watermarking Method Using Quaternion QR Decomposition," in IEEE Access, vol. 8, pp. 72308-72315, 2020, doi: 10.1109/ACCESS.2020.2987914.

[6] Seuter, M., Pollock, A., Bauer, G., and Kray, C.(2020). *Recognizing Running Movement Changes with Quaternions on a Sports Watch.* Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 4, 4, Article 151 (December 2020). DOI, https://doi.org/10.1145/3432197

[7] Wang, B., Wang, W., and Zhao, P. (2021). *Multiple color medical images zero-watermark scheme based on quaternion generalized Fourier descriptor and QR code*. ACM Turing Award Celebration Conference - China (ACM TURC 2021). Association for Computing Machinery, New York, NY, USA, 248–253. DOI, https://doi.org/10.1145/3472634.3474079

[8] Webmaster. (2020). "3D rotations and Euler angles in Python." Meccanismo Complesso, https://www.meccanismocomplesso.org/en/3d-rotations-and-euler-angles-in-python/

[9] Wyss-Gallifent, J. (2021a). *MA 431: Quaternions*. UMD, https://www.math.umd.edu/~immortal/MATH431/book/ch_quaternions.pdf

[10] Wyss-Gallifent, J. (2021b). *MA 431: Gimbal Lock*. UMD, https://www.math.umd.edu/~immortal/MATH431/book/ch_gimballock.pdf

A Comparison of Euler Angles and Quaternions

```
import numpy

import math

import timeit

import pandas as pd

import matplotlib.pyplot as plt


from random import *

from prettytable import PrettyTable

from EulerAngle import X, Y, Z

from Quaternions import euler_to_quaternion

from Graphs import graph_all_euler_quaternions,

    graph_avg_euler_quaternions


ITERATIONS = 20


def main():

    final = []

    time = []
```

```
headers = ["phi", "theta", "psi", "Euler Rotation Matrix", "Time
    for Euler", "Time for Quaternion", "Quaternion Result"] #
    Headers for table
table = PrettyTable(headers)     # Generate table


i = 0      # Initialize iteration


while i<=ITERATIONS:                          # Loop (depends on
    ITERATIONS constant at top)
    #Generate random angles
    phi = math.pi / randint(1, 6)
    theta = math.pi / randint(1, 6)
    psi = math.pi / randint(1, 6)


    startEuler = timeit.default_timer()    # Start timer for Euler
        calculation run time


    euler = X(phi) * Y(theta) * Z(theta)   # Calculate rotation


    stopEuler = timeit.default_timer()     # Stop timer for euler
        calculation run time
```

```
startQuaternion = timeit.default_timer()

quaternion = euler_to_quaternion(phi, theta, psi)

stopQuaternion = timeit.default_timer()        # Stop timer for
    run time

add = table.add_row([phi, theta, psi, numpy.round(euler,
    decimals = 2), stopEuler - startEuler, stopQuaternion -
    startQuaternion, numpy.round(quaternion, decimals = 2)])
    #Add information to table

result = [phi, theta, psi, euler, stopEuler - startEuler,
    stopQuaternion - startQuaternion, quaternion]  # Add
    information to first array

final.append(result)         #Append first array to final array

eQTime = [i, stopEuler - startEuler, stopQuaternion -
    startQuaternion] # List to hold iteration number and run
```

```
        time data


    eTime = [stopEuler - startEuler]           # List to hold run
        time data for Euler
    qTime = [stopQuaternion - startQuaternion] # List to hold run
        time data for quaternions


    time.append(eQTime)


    i += 1         # Move to next iteration


print(table)


dictionary = dict(zip(headers, zip(*final))) # Convert final array
    to a dictionary


dfTable = pd.DataFrame.from_dict(dictionary)                    #
    Define the dataframes
dfTable.to_csv("Comparison.csv", index = False, header = True)  #
    Write dataframes to csv to be saved
```

```
    graph_all_euler_quaternions(time)           # Graph data for all
        iterations
    graph_avg_euler_quaternions(eTime, qTime)   # Graph data for average
        of all iterations


    plt.show()


main()
```

GitHub page with all the code.

B Visual of Counter-Clockwise Rotation

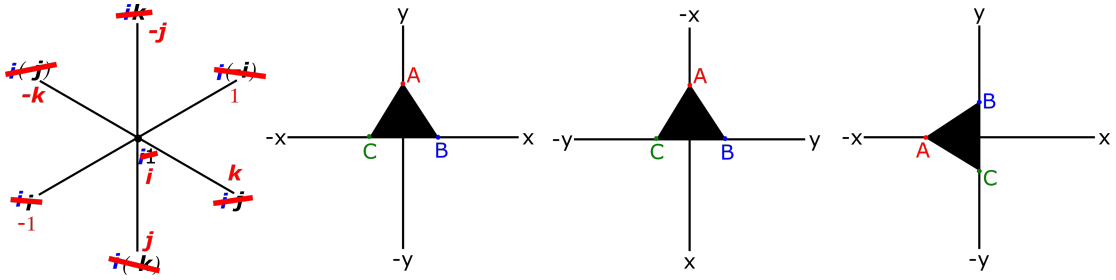


Figure 9: Understanding why the object rotates counter-clockwise using the xy-plan.