CS461 - Computer Graphics

Assignment 3 - Quaternions

Submitted By: Kartik Gupta (170101030)

Introduction

In computer graphics, we use transformation matrices to express a position in space (translation) as well as its orientation in space (rotation). Optionally, a single transformation matrix can also be used to express the scale or “shear” of an object. We can think of this transformation matrix as a “basis space” where if you multiply a vector or a point (or even another matrix) by a transformation matrix you “transform” that vector, point or matrix into the space represented by that matrix.

Complex Numbers

The Complex Number system introduces a new set of numbers called imaginary numbers. Imaginary numbers were invented to solve certain equations that had no solutions such as:

$$x^2 + 1 = 0$$

To solve this expression, we must state that

$$x^2 = -1$$

which we know is not possible because the square of any number (positive or negative) is always positive.

Mathematicians generally can’t accept that an expression does not have a solution so a new term was invented called the imaginary number that can be used to solve such equations.

The imaginary number has the form:

$$i^2 = -1$$

Don’t try to actually understand this term as there is no logical reason why it exists. We just have to accept that i is just something that squares to −1.

The set of complex numbers is the sum of a real number and an imaginary number and has the form:

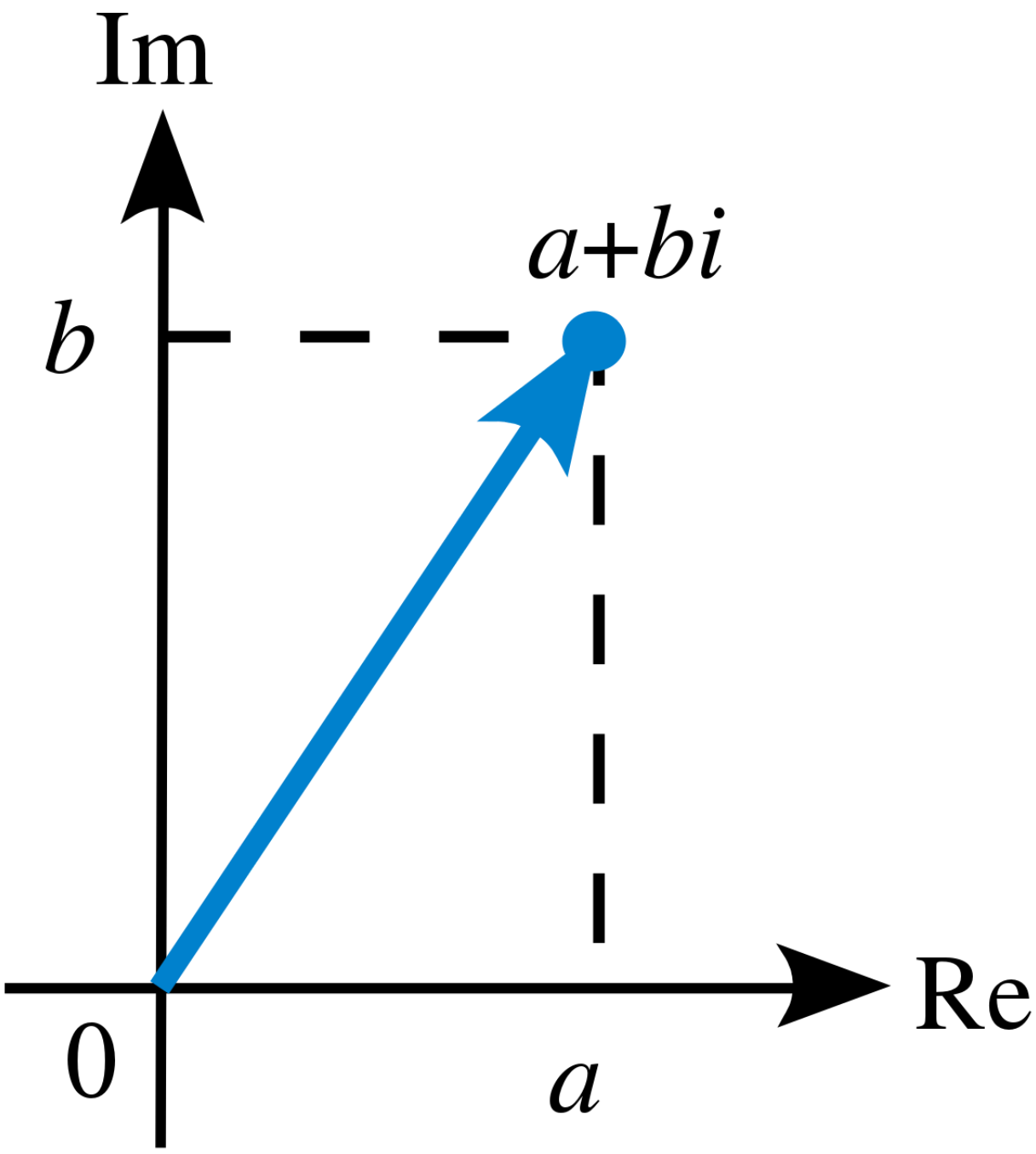
$$z = a+bi \quad a,b \in \mathbb{R}$$

It could also be stated that all Real numbers are complex numbers with b = 0 and all imaginary numbers are complex numbers with a = 0

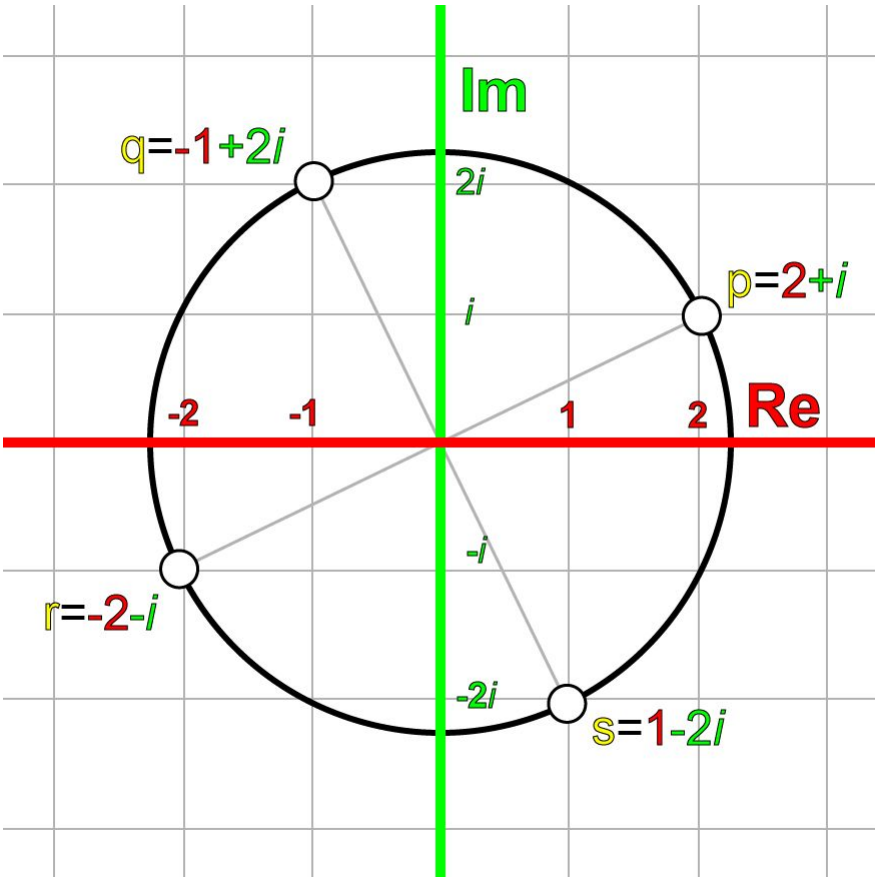
Complex Plane

We can also map complex numbers in a 2D grid called the **Complex Plane** by mapping the **Real** part on the horizontal axis and the **Imaginary** part on the vertical axis

.



if we multiply a complex number by i, we can rotate the complex number through the complex plane at 90° increments



We can also perform arbitrary rotations in the complex plane by defining a complex number of the form:

$$q = \cos\theta + i \sin\theta$$

Multiplying any complex number by the rotor q produces the general formula:

$$\begin{aligned} p &= a + bi \\ q &= \cos\theta + i\sin\theta \\ pq &= (a + bi)(\cos\theta + i\sin\theta) \\ a' + b'i &= a\cos\theta - b\sin\theta + (a\sin\theta + b\cos\theta)i \end{aligned}$$

Which is the method to rotate an arbitrary point in the complex plane counter-clockwise about the origin.

Quaternions

With this knowledge of the complex number system and the complex plane, we can extend this to 3-dimensional space by adding two imaginary numbers to our number system in addition to i.

The general form to express quaternions is

$$q = s + xi + yj + zk \quad s, x, y, z \in \mathbb{R}$$

Where, according to Hamilton's famous expression:

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

and

$$\begin{array}{lll} ij = k & jk = i & ki = j \\ ji = -k & kj = -i & ik = -j \end{array}$$

You may have noticed that the relationship between i, j, and k are very similar to the cross product rules for the unit cartesian vectors:

$$\begin{array}{lll} \mathbf{x} \times \mathbf{y} = \mathbf{z} & \mathbf{y} \times \mathbf{z} = \mathbf{x} & \mathbf{z} \times \mathbf{x} = \mathbf{y} \\ \mathbf{y} \times \mathbf{x} = -\mathbf{z} & \mathbf{z} \times \mathbf{y} = -\mathbf{x} & \mathbf{x} \times \mathbf{z} = -\mathbf{y} \end{array}$$

Hamilton also recognized that the i,j and k imaginary numbers could be used to represent three cartesian unit vectors $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ with the same properties of imaginary numbers, such that

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1.$$

Quaternions as an Ordered Pair

We can also represent quaternions as an ordered pair:

$$q = [s, \mathbf{v}] \quad s \in \mathbb{R}, \mathbf{v} \in \mathbb{R}^3$$

Where **v** can also be represented by its individual components:

$$q = [s, x\mathbf{i} + y\mathbf{j} + z\mathbf{k}] \quad s, x, y, z \in \mathbb{R}$$

Using this notation, we can more easily show the similarities between quaternions and complex numbers.

It should be possible to express a quaternion that can be used to rotate a point in 3D-space as such:

$$q = [\cos\theta, \sin\theta\mathbf{v}]$$

Quaternions vs Euler Angles

Quaternions	Euler Angles
Easier to **compute** for the computer and more efficient.	More **human** understandable.
Need to make three rotations and multiply them together when rotating by Euler angles, whereas a Quaternion is only **one rotation**.	Good for **decomposing rotations** into individual degrees of freedom (for kinematic joints and the like).
Using SLERP Quaternion interpolation causes good results in interpolating between two keyframes. SLERP selects the **shortest arc** in all possible paths in the rotation space to rotate one point to another which is beneficial to character animation.	Euler angles suffer from **gimbal lock**, which prevents them from measuring orientation when the pitch angle approaches +/-90°.
Quaternions don't suffer from ambiguity, since they only represent a single rotation, with a **well-defined axis**.	There's an **ambiguity** in the specification of Euler angles: which angle applies to which axis? Code that uses one convention cannot interoperate with code that assumes another convention.

Quaternion Interpolation

One of the most important reasons for using quaternions in computer graphics is that quaternions are very good at representing rotations in space. Quaternions overcome the issues that plague other methods of rotating points in 3D space such as Gimbal lock which is an issue when you represent your rotation with euler angles.

Using quaternions, we can define several methods that represent a rotational interpolation in 3D space. The first method we will examine is called **SLERP** which is used to smoothly interpolate a point between two orientations. The second method is an extension of **SLERP** called **SQUAD** which is used to interpolate through a sequence of orientations that define a path.

SLERP

SLERP stands for **S**pherical **L**inear Int**erp**olation. **SLERP** provides a method to smoothly interpolate a point about two orientations.
We will represent the first orientation as $q1$ and the second orientation as $q2$. The point that is interpolated will be represented by $\mathbf{p}$ and the interpolated point will be represented by $\mathbf{p}$. The interpolation parameter t will interpolate $\mathbf{p}$ from $q1$ when t = 0 to $q2$ when t=1.
The standard linear interpolation formula is:

$$\mathbf{p}' = \mathbf{p_1} + t(\mathbf{p_2} - \mathbf{p_1})$$

The general steps to apply this equation are:

1. Compute the difference between $\mathbf{p}1$ and $\mathbf{p}2$
2. Take the fractional part of that difference.
3. Adjust the original value by the fractional difference between the two points.

The first step dictates that we must compute the difference between two quaternions. This is equivalent to computing the angular difference between the two quaternions.

$$\Delta q = q_1^{-1} q_2$$

The next step is to take the fractional part of that difference. We can compute the fractional part of a quaternion by raising it to a power whose value is in the range [0….1]

The general formula for quaternion exponentiation is:

$$q^t = \exp(t \log q)$$

Where the exponential function for quaternions is given by:

$$\begin{aligned} \exp(q) &= \exp\big([0, \theta\hat{\mathbf{v}}]\big) \\ &= [\cos\theta, \sin\theta\hat{\mathbf{v}}] \end{aligned}$$

And the logarithm of a quaternion is given by:

$$\begin{aligned} \log q &= \log(\cos\theta + \sin\theta\hat{\mathbf{v}}) \\ &= \log\big(\exp(\theta\hat{\mathbf{v}})\big) \\ &= \theta\hat{\mathbf{v}} \\ &= [0, \theta\hat{\mathbf{v}}] \end{aligned}$$

To compute the interpolated angular rotation, we adjust the original orientation q_1 by the fractional part of the difference between q_1 and q_2.

$$q' = q_1\left(q_1^{-1} q_2\right)^t$$

Which is the general form of spherical linear interpolation using quaternions. However, this is not the form of the **SLERP** equation that is commonly used in practice.

We can apply a similar formula for performing a spherical interpolation of vectors to quaternions. The general form of a spherical interpolation for vectors is defined as:

$$\mathbf{v}_t = \frac{\sin(1-t)\theta}{\sin\theta}\mathbf{v}_1 + \frac{\sin t\theta}{\sin\theta}\mathbf{v}_2$$

This formula can be applied unmodified to quaternions:

$$q_t = \frac{\sin(1-t)\theta}{\sin\theta}q_1 + \frac{\sin t\theta}{\sin\theta}q_2$$

And we can obtain the angle θ by computing the dot-product between $q1$ and $q2$.

SQUAD

Just as a **SLERP** can be used to compute an interpolation between two quaternions, a **SQUAD** (**S**pherical and **Quad**rangle) can be used to smoothly interpolate over a path of rotations.

If we have the sequence of quaternions:

$$q_1, q_2, q_3, \cdots, q_{n-2}, q_{n-1}, q_n$$

And we also define the "helper" quaternion (s_i) which we can consider an intermediate control point:

$$s_i = \exp\left(-\frac{\log\left(q_{i+1}q_i^{-1}\right) + \log\left(q_{i-1}q_i^{-1}\right)}{4}\right)q_i$$

Then the orientation along the sub-curve defined by:

$$q_{i-1}, q_i, q_{i+1}, q_{i+2}$$

at time **t** is given by:

$$\text{squad}(q_i, q_{i+1}, s_i, s_{i+1}, t) = \text{slerp}(\text{slerp}(q_i, q_{i+1}, t), \text{slerp}(s_i, s_{i+1}, t), 2t(1-t))$$

Conclusion

Despite being extremely difficult to understand, quaternions provide a few obvious advantages over using matrices or Euler angles for representing rotations.

- Quaternion interpolation using SLERP and SQUAD provides a way to interpolate smoothly between orientations in space.
- Rotation concatenation using quaternions is faster than combining rotations expressed in matrix form.
- For unit-norm quaternions, the inverse of the rotation is taken by subtracting the vector part of the quaternion. Computing the inverse of a rotation matrix is considerably slower if the matrix is not orthonormalized (if it is, then it's just the transpose of the matrix).
- Converting quaternions to matrices is slightly faster than for Euler angles.

- Quaternions only require 4 numbers (3 if they are normalized. The Real part can be computed at run-time) to represent a rotation where a matrix requires at least 9 values.

However, for all of the advantages in favor of using quaternions, there are also a few disadvantages.

- Quaternions can become invalid because of floating-point round-off error however this “error creep” can be resolved by re-normalizing the quaternion.
- And probably the most significant deterrent for using quaternions is that they are very hard to understand. I hope that this issue is resolved after reading this article.