Quaternions for 3D Rotation Features and Its Application in Unity Game Engine

Geraldi Dzakwan 13514065
Informatics Undergraduate Program
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Ganesha Avenue Number 10 Bandung 40132, Indonesia
geraldzakwan@students.itb.ac.id

Abstract*—Linear and vector algebra are some of the basic for computer graphic. It is commonly used to transform an object (such as rotating) in computer graphics as what most of people know. But, there is a fairly new concept relative to linear and vector algebra which becomes more popular recently. That would be the concept of quaternion. Quaternion (frequently called as quaternion algebra) basicly is the 3D equivalent of complex numbers which eases many problems in 3D computer graphics especially object transformation. Many game developers start to use quaternion to develop some features for their game such as 3D rotation feature. Hence, this paper will focus on 3D computer graphics and review its 3D rotation feature. There are reasons why they leave linear and vector algebra and prefer to choose quaternion (these will also be reviewed in this paper). This paper will also include implementations of quaternion for 3D rotation in unity game engine as a popular game development tool with describing illustrations in the last section.

***Keywords*—computer graphic, linear algebra, vector algebra, quaternion algebra, 3D rotation, game development, unity game engine**

I. Introduction

Computer graphic is one of important branch in computer science as it supports software development, mainly for games development. Good graphic design for software can help to attract more users as users would always look for best interface.

Computer graphic has wide variety of applications in computer science. Every tools, software, and games are supported by good graphic design. For example, gamers would prefer a game which has better computer graphics. It applies to other computer science products as well.

The discussion in this paper will be restricted just for 3D computer graphic features, especially 3D rotation. It is more complicated than 2D rotation since we must specify an axis of rotation. In 2D, the axis of rotation is always perpendicular to the xy plane, i.e., the Z axis, but in 3D the axis of rotation can have any spatial orientation.

As stated before in the abstract, there are many approaches on how to rotate 3D objects. People tend to use linear algebra (such as rotation matrix) or vector algebra to be applied to 3D rotation. But, since the new concept of quaternion is introduced, developers start to use quaternion as it has some advantages. Further explanation about quaternion and what advantages quaternion has will not be discussed here but instead later in quaternion section.

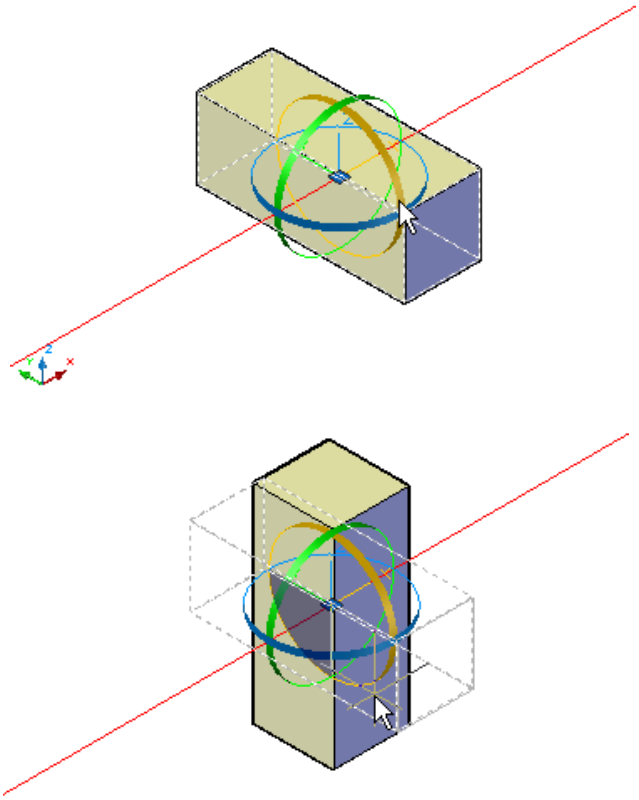


Fig 1.1 An illustration for rotating 3D object
Source :
http://docs.autodesk.com/ACD/2010/ENU/AutoCAD%202010%20User%20Documentation/images/PTDCPM/Gator-All/English/ill_23_rotate_GT_vector.png

Hence, I would say that this paper holds the whole thing about 3D rotation. This paper starts with basic theories of quaternion algebra which underlies 3D rotation feature. Then, how quaternion supports 3D rotation and what makes it more versatile tools than linear or vector algebra will be reviewed in the middle part of the paper. Last but not least, there will be an insight of a unity game engine as a popular game development tool among game developers. There will be illustrations on how to rotate a game object in unity. Their concept of rotating object is based on quaternion algebra so I think this paper's last section will correspond to the previous sections preceding.

II. QUATERNION THEORIES

The whole explanations about quaternion theories are taken from Geometric Algebra for Computer Graphics book written by John Vince. Some contents are modified by my own writing and by other references as well. There will be as well illustrations, some are taken from the book, some other are taken from other references.

A. Quaternion Overview

Quaternions are the result of one man's determination to find the 3D equivalent of complex numbers. Sir William Rowan Hamilton was the man, and in 1843 he revealed to the world his discovery which had taken him over a decade to resolve.

Quaternion is the form of 4-tuple as stated below :

$$z = a + ib + jc + kd$$

where i, j, and k are unit imaginaries which obey Hamilton's rules.

The Hamilton's rules of unit imaginaries :

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

$$ij = k \quad jk = i \quad ki = j \quad ji = -k \quad kj = -i \quad ik = -j$$

However, when vector algebra became the preferred system over quaternion algebra, the *i*, *j* and *k* terms became the Cartesian unit vectors i, j and k.

One very important feature of quaternion algebra is its anticommuting rules. Maintaining order between the unit imaginaries is vital for the algebra to remain consistent, which is also a feature of GA (geometric algebra).

B. Quaternion Operation and Properties

1. Sum of Quaternions

Two quaternions $q1$ and $q2$

$$q_1 = s_1 + i\,x_1 + j\,y_1 + k\,z_1$$
$$q_2 = s_2 + i\,x_2 + j\,y2 + k\,z_2$$

are equal if, and only if, their corresponding terms are equal. Furthermore, like vectors, they can be added or subtracted as follows:

$$q_1 \pm q_2 = [(s_1 \pm s_2) + i(x_1 \pm x_2) + j(y_1 \pm y_2) + k(z_1 \pm z_2)].$$

For example, given two quaternions

$$q_1 = 1 + i2 + j3 + k4$$
$$q_2 = 2 - i + j5 - k2$$

their sum is given by

$$q_1 + q_2 = 3 + i + j8 + k2$$

2. Product of Quaternions

Given two quaternions

$$q_1 = s_1 + \mathrm{v}_1 = s_1 + i\,x_1 + j\,y_1 + k\,z_1$$
$$q_2 = s_2 + \mathrm{v}_2 = s_2 + i\,x_2 + j\,y2 + k\,z_2$$

their product is given by

$$q_1\,q_2 = s_1\,s_2 - \mathrm{v}_1 \cdot \mathrm{v}_2 + s_1\,\mathrm{v}_2 + s_2\,\mathrm{v}_1 + \mathrm{v}_1 \times \mathrm{v}_2$$

which is still a quaternion and ensures closure.

However, the quaternion product anticommutes, which we can prove by computing $q2q1$:

$$q_2\,q_1 = s_1\,s_2 - \mathrm{v}_2 \cdot \mathrm{v}_1 + s_2\,\mathrm{v}_1 + s_1\,\mathrm{v}_2 + \mathrm{v}_2 \times \mathrm{v}_1$$

The pure scalar terms $s_2\,s_1$, $\mathrm{v}_2 \cdot \mathrm{v}_1$ and the products $s_2\,\mathrm{v}_1$ and $s_1\,\mathrm{v}_2$ commute, but the cross product $\mathrm{v}_2 \times \mathrm{v}_1$ anticommutes, therefore $q_1\,q_2$ is not equal to $q_2\,q_1$.

For example, given the quaternions

$$q_1 = 1 + i2 + j3 + k4$$
$$q_2 = 2 - i + j5 - k2$$

their product $q1q2$ is

$$\begin{aligned} q_1\,q_2 &= (1 + i2 + j3 + k4)(2 - i + j5 - k2) \\ &= [1 \times 2 - (2 \times (-1) + 3 \times 5 + 4 \times (-2)) \\ &\quad +1(-i + j5 - k2) + 2(i2 + j3 + k4) \\ &\quad +i(3 \times (-2) - 4 \times 5) + j(4 \times (-1) - (-2) \times 2) + k(2 \times 5 - (-1) \times 3)] \\ &= -3 + i3 + j11 + k6 - i26 + k13 \end{aligned}$$

$$q_1\,q_2 = -3 - i23 + j11 + k19$$

which is a quaternion.

Whereas the product $q2q1$ is

$$\begin{aligned} q_2\,q_1 &= (2 - i + j5 - k2)(1 + i2 + j3 + k4) \\ &= [2 - ((-1) \times 2 + 5 \times 3 + (-2) \times 4) \\ &\quad +2(i2 + j3 + k4) + 1(-i + j5 - k2) \\ &\quad +i(5 \times 4 - 3 \times (-2)) + j((-2) \times 2 - 4 \times (-1)) + k((-1) \times 3 - 2 \times 5)] \end{aligned}$$

$$q_2\,q_1 = -3 + i29 + j11 - k7$$

which is also a quaternion, but $q2q1$ is not equal to $q_1\,q_2$.

3. Magnitude of Quaternion

Given the quaternion

$$q = s + ix + jy + kz$$

its magnitude is defined as

$$||q|| = (s^2 + x^2 + y^2 + z^2)^{1/2}$$

For example, given the quaternion

$$q = 1 + i2 + j3 + k4$$
$$||q|| = (1^2 + 2^2 + 3^2 + 4^2)^{1/2} = \sqrt{30}$$

4. Unit of Quaternion

Like vectors, quaternions have a unit form where the magnitude equals unity. For example, the magnitude of the quaternion

$$q = 1 + i2 + j3 + k4$$

is

$$||q|| = (1^2 + 2^2 + 3^2 + 4^2)^{1/2} = \sqrt{30}$$

therefore, the unit quaternion q^u equals

$$q^u = 1/30\,(1 + i2 + j3 + k4)$$

5. Pure Quaternion

Hamilton named a quaternion with a zero scalar term a *pure quaternion*. For example,

$$q_1 = s_1 + i\,x_1 + j\,y_1 + k\,z_1$$
$$\text{and } q_2 = s_2 + i\,x_2 + j\,y2 + k\,z_2$$

are pure quaternions. Let's see what happen when we multiply them together:

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

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

which is no longer a pure quaternion, as a negative scalar term has emerged. Thus the algebra of pure quaternions is not closed.

6. Conjugate of Quaternion

Given the quaternion

$$q = s + \mathrm{v}$$
$$q = s + ix + jy + kz$$

by definition, its conjugate is

$$q^c = s - \mathrm{v} = s - (ix + jy + kz)$$

For example, the quaternion

$$q = 1 + i2 + j3 + k4$$

its conjugate is

$$q^c = 1 - i2 - j3 - k4$$

7. Inverse of Quaternion

Given the quaternion

$$q = s + ix + jy + kz$$

the inverse quaternion q^{-1} is

$$q^{-1} = s - ix - jy - kz / ||q||^2$$

because this satisfies the product

$$qq^{-1} = (s + ix + jy + kz)(s - ix - jy - kz) / ||q||^2 = 1$$

We can show that this is true by expanding the product as follows:

$$qq^{-1} = (s_2 - isx - jsy - ksz + isx + x_2 - ijxy - ikxz + jsy - jixy + y_2 - jkyz + ksz - kixz - kjyz + z_2) / ||q||^2$$
$$= (s_2 + x_2 + y_2 + z_2 - ijxy - ikxz - jixy - jkyz - kixz - kjyz) / ||q||^2$$

$$qq^{-1} = (s_2 + x_2 + y_2 + z_2) / ||q||^2 = 1$$

and confirms that the inverse quaternion q^{-1} is

$$q^{-1} = q / ||q||^2$$

Because the unit imaginaries do not commute, we need to discover whether

$$qq{-}1 = q{-}1q$$

Expanding this product

$$q^{-1}q = (s - ix - jy - kz)(s + ix + jy + kz) / ||q||^2$$
$$= (s_2 + isx + jsy + ksz - isx + x_2 - ijxy - ikxz - jsy - jixy + y_2 - jkyz - ksz - kixz - kjyz + z_2) / ||q||^2$$
$$= (s_2 + x_2 + y_2 + z_2 - ijxy - ikxz - jixy - jkyz - kixz - kjyz) / ||q||^2$$

$$q^{-1}q = (s^2 + x^2 + y^2 + z^2) / ||q||^2 = 1$$

therefore,

$$qq^{-1} = q^{-1}q$$

C. Quaternion Algebra

The axioms associated with quaternions are as follows:

Given

$$q, q_1, q_2, q_3 \in \mathrm{C}:$$

1. Closure

For all q_1 and q_2

addition $q_1 + q_2 \in \mathrm{C}$

multiplication $q_1 q_2 \in \mathrm{C}$

2. Identity

For each q there is an identity element 0 and 1 such that:

addition $q+0 = 0+q = q$ $(0 = 0 + i0 + j0 + k0)$

multiplication $q(1) = (1)q = q$ $(1 = 1 + i0 + j0 + k0)$

3. Inverse

For each q there is an inverse element $-q$ and q^{-1} such that:

addition $q + (-q) = -q + q = 0$

multiplication $qq^{-1} = q^{-1}q = 1$ (q is not zero).

4. Associativity

For all q_1, q_2 and q_3

addition $q_1 + (q_2 + q_3) = (q_1 + q_2) + q_3$

multiplication $q_1(q_2\, q_3) = (q_1\, q_2)q_3$

5. Commutativity

For all q_1 and q_2

addition $q_1 + q_2 = q_2 + q_1$

multiplication $q_1\, q_2$ is not equal to $q_2\, q_1$

6. Distributivity

For all q_1, q_2 and q_3

$$q_1(q_2 + q_3) = q_1\, q_2 + q_1\, q_3$$
$$(q_1 + q_2)q_3 = q_1\, q_3 + q_2\, q_3$$

C. Conclusion for Quaternion Theories

Out of all the algebras we have so far considered, quaternion algebra paves the way to geometric algebra. In fact, as we will soon discover, geometric algebra shows that quaternions are a left-handed system and employ the concepts of geometric algebra. The good news is that if you understand quaternions, you will find it much easier to understand geometric algebra.

III. 3D ROTATION USING QUATERNION

A. Why Quaternion

As stated before in the introduction, there would be other methods to rotate an object, such as using linear algebra (rotation matrix) and vector algebra. Then, why we bother considering to rotate object using quaternion? Of course, there are reasons behind it. Rotating object using quaternion does have some advantages compared to other methods. Not only for rotating object problems, but quaternions are used in computer graphics a lot for these reasons :

1. They are much more efficient to store than rotation matrices (4 floats rather than 16)
2. They are much easier to interpolate than euler angle rotations (spherical interpolation or normalized liner interpolation)
3. They avoid gimbal lock
4. It's more sophisticating to tell that your rotation is described as a great circle on the surface of a unit 4 dimensional hypersphere

To understand more about the benefits of using quaternions you have to consider different ways to represent rotations.

Here are few ways with a summary of the pros and cons:

- Euler angles
- Rotation matrices
- Axis angle
- Quaternions
- Rotors (normalized Spinors)

Euler angles are the best choice if you want a user to specify an orientation in a intuitive way. They are also space efficient (three numbers). However, it is more difficult to linear interpolate values. Consider the case where you want to interpolate between 359 and 0 degrees. Linearly interpolating would cause a large rotation, even though the two orientations are almost the same. Writing shortest path interpolation, is easy for one axis, but non-trivial when considering the three Euler angles (for instance the shortest route between (240, 57, 145) and (35, -233, -270) is not immediately clear).

Rotation matrices specify a new frame of reference using three normalized and orthogonal vectors (Right, Up, Out, which when multiplied become the new x, y, z). Rotation matrices are useful for operations like strafing (side way movement), which only requires translating along the Right vector of the camera's rotation matrix. However, there is no clear method of interpolating between them. They are also expensive to normalize which is necessary to prevent scaling from being introduced.

Axis angle, as the name suggests, are a way of specifying a rotation axis and angle to rotate around that axis. You can think of Euler angles, as three axis angle rotations, where the axis is the x, y, z axis respectively. Linearly interpolating the angle in a axis angle is pretty straight forward (if you remember to take the shortest path), however linearly interpolating between different axis is not.

Quaternions are a way of specifying a rotation through a axis and the cosine of half the angle. They main advantage is I can pick any two quaternions and smoothly interpolate between them.

Rotors are another way to perform rotations. Rotors are basically quaternions, but instead of thinking of them as 4D complex numbers, rotors are thought of as real 3D multivectors. This makes their visualization much more understandable (compared to quaternions), but requires fluency in geometric algebra to grasp their significance.

B. Quaternion Rotation Concept

One excellent application for quaternions is rotating vectors. If you require an introduction to this topic or are willing to learn more you can see **[2]**.

It can be shown that a position vector p can be rotated about an axis $\hat{u}$ by an angle θ to p′ using the following operation:

$$p' = qpq^{-1}$$

where

$$p = xi + yj + zk$$
$$p = 0 + ix + jy + kz$$
$$q = \cos(\theta/2) + \sin(\theta/2)\,\hat{u}$$
$$q^{-1} = \cos(\theta/2) - \sin(\theta/2)\,\hat{u}$$

and the axis of rotation is

$$\hat{u} = [x_u i + y_u j + z_u k] \quad (\|\hat{u}\| = 1)$$

This is best demonstrated through an example. Let the point to be rotated be

$$P(0, 1, 1)$$

Let the axis of rotation be

$$\hat{u} = j$$

Let the angle of rotation be

$$\theta = 90^0$$

Therefore,

$$p = 0 + i0 + j + k$$
$$q = \cos 45^0 + \sin 45^0 (i0 + j + k0)$$
$$q = \sqrt{2}/2\ (1 + i0 + j + k0)$$
$$q^{-1} = \cos 45^0 - \sin 45^0 (i0 + j + k0)$$
$$q^{-1} = \sqrt{2}/2\ (1 - i0 - j - k0)$$

The rotated point is given by

$$p' = qpq^{-1}$$
$$p' = \sqrt{2}/2\ (1 + i0 + j + k0)\ (0 + i0 + j + k)\ \sqrt{2}/2\ (1 - i0 - j - k0)$$

This is best expanded in two steps, and zero imaginary terms are included for clarity.
qp followed by $(qp)q{-}1$.

Step 1

$$qp = \sqrt{2}/2\ (1 + i0 + j + k0)\ (0 + i0 + j + k)$$
$$qp = \sqrt{2}/2\ (-1 + i + j + k)$$

Step 2

$$(qp)q^{-1} = \sqrt{2}/2\ (-1 + i + j + k)\ \sqrt{2}/2\ (1 - i0 - j - k0)$$
$$= ½\ (-1 + 1 + j + i + j + k + i - k)$$
$$= ½\ (0 + i2 + j2 + k0)$$
$$(qp)q{-}1 = 0 + i + j + k0$$

The coordinates of the rotated point are stored in the pure part of the quaternion: (1, 1, 0).

IV. 3D ROTATION IMPLEMENTATION FOR UNITY AS A GAME ENGINE

A. Unity Overview

Unity is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. First announced only for OS X, at Apple's Worldwide Developers Conference in 2005, it has since been extended to target more than fifteen platforms. It is now the default software development kit (SDK) for the Wii U. Five versions of Unity have been released, its latest is the Unity 5. At the 2006 WWDC trade show, Apple, Inc. named Unity as the runner up for its Best Use of Mac OS X Graphics category.

With an emphasis on portability, the engine targets the following APIs : Direct3D on Windows and Xbox 360; OpenGL on Mac and Windows; OpenGL ES on Android and iOS; and proprietary APIs on video game consoles. Unity allows specification of texture compression and resolution settings for each platform the game engine supports, and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects. Unity's graphics engine's platform diversity can provide a shader with multiple variants and a declarative fallback specification, allowing Unity to detect the best variant for the current video hardware; and if none are compatible, fall back to an alternative shader that may sacrifice features for performance.

Unity is notable for its ability to target games to multiple platforms. Within a project, developers have control over delivery to mobile devices, web browsers, desktops, and consoles. Supported platforms include BlackBerry 10, Windows Phone 8, Windows, OS X, Android, iOS, Unity Web Player (including Facebook), PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii U, Nintendo 3DS line and Wii. It includes an asset server and Nvidia's PhysX physics engine. Unity Web Player is a browser plugin that is supported in Windows and OS X only. Unity is the default software development kit (SDK) for Nintendo's Wii U video game console platform, with a free copy included by Nintendo with each Wii U developer license. Unity Technologies calls this bundling of a third-party SDK an "industry first".

In 2012, VentureBeat said, "Few companies have contributed as much to the flowing of independently produced games as Unity Technologies."

For the Apple Design Awards at the 2006 WWDC trade show, Apple, Inc. named Unity as the runner up for its Best Use of Mac OS X Graphics category, a year after Unity's launch at the same trade show. Unity Technologies says this is the first time a game design tool has ever been nominated for this award. A May 2012 survey by Game Developer magazine indicated Unity as its top game engine for mobile platforms. In July 2014, Unity won the "Best Engine" award at the UK's annual Develop Industry Excellence Awards.

Unity 5 has been met with similar praise, with The Verge stating "Unity started with the goal of making game development universally accessible. Unity 5 is a long-awaited step towards that future."

There are so many popular games which are built by this powerful and versatile game engine since the very first time this engine was introduced. Some of them are *Dead Frontier, Three Kingdoms Online, Temple Run, Plague Inc, Slender, Game of Thrones, Space Hulk, World Series of Poker, Angry Birds, Crossy Road, Wasteland, Heroes of Warcraft,* and *Pathfinder Online*. As you may see, Unity works on wide variety of platform, such as console, PC, and mobile.

Fig 4.1 Unity Working Environment/User Interface on Mac OS platform
Source : http://3.bp.blogspot.com/-O6Ut6OMivBQ/UTRZzEgsFNI/AAAAAAAAAKk/fLa2cklVQLA/s1600/358365%5B1%5D.png

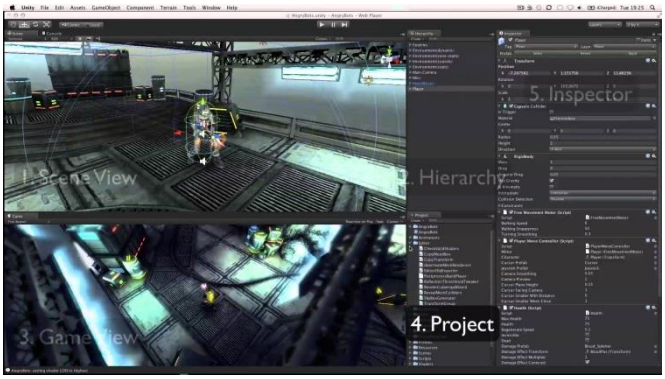


Fig 4.2 Unity Scene View and Game View on Mac OS platform
Source : http://3.bp.blogspot.com/-O6Ut6OMivBQ/UTRZzEgsFNI/AAAAAAAAAKk/fLa2cklVQLA/s1600/358365%5B1%5D.png

B. Unity 3D Rotation Feature using Quaternion

It is best to explain this section with example and illustrations which describe it. Let us take an example of a human character object that is programmed to always look at an orb wherever the orb goes.

Fig 4.2 Scene with a human character object and an orb object

Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

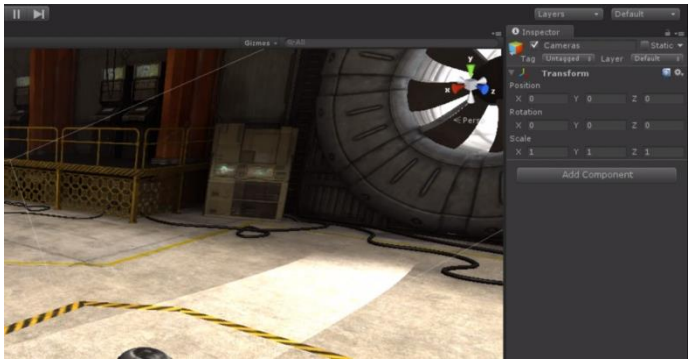


Fig 4.3 There are panel in the top right side in which you can set position, rotation, and scale component. To specify rotation, you must add the quaternion component which corresponds to the rotation you want
Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

Fig 4.4 The orb is attached to a Motion script so later it can rotate to move around/orbit the character
Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

```
using UnityEngine;
using System.Collections;

public class MotionScript : MonoBehaviour
{
    public float speed = 3f;

    void Update ()
    {
        transform.Translate(-Input.GetAxis("Horizontal") * speed * Time.deltaTime, 0, 0);
    }
}
```

Fig 4.5 The motion script in JS which is attached to the orb. The motion is set based on time and axis.
Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

Fig 4.6 The character is attached to a LookAt script so it can look at the orb wherever the orb goes
Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

```
using UnityEngine;
using System.Collections;

public class LookAtScript : MonoBehaviour
{
    public Transform target;

    void Update ()
    {
        Vector3 relativePos = target.position - transform.position;
        transform.rotation = Quaternion.LookRotation(relativePos);
    }
}
```

Fig 4.7 The LookAt script which take benefits of quaternion to make the character rotate relative to the position of the orb
Source :
https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

Fig 4.8 When the orb goes to the right, the character's face will rotate to the right so it can look at the orb
Source : https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

Fig 4.9 When the orb goes to the left, the character's face will rotate to the left so it can look at the orb
Source : https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions

To see more comprehensive example, visit this link : https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions. You can watch the Unity Official Tutorials about how to utilize the Quaternion System to manage the rotation of game object.

V. Conclusion

In conclusion, quaternion algebra is a beneficial system to use for rotating 3D object and for other feature related to computer graphics. It is because quaternion algebra has some advantages over linear algebra (rotation matrices) and vector algebra which has been reviewed before in the middle part of the paper.

Quaternion system is not only implemented in 3D rotation, but also for computer graphics generally. So, beside game development, there are a lot of fields in computer graphics which will be using quaternion system to improve furthermore. Quaternion is a versatile tool to be implemented in computer graphics besides the other methods such as Euler angle, rotation matrices, axis angle, and rotors.

Although the implementation of quaternion in computer graphics is not as far as the linear or vector algebra, but I believe the use of quaternion will grow bigger as many developers realize the advantage of using this system for developing their product, not only game, but also software in general.

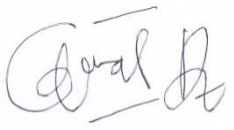
References

[1] Vince, John. *Geometric Algebra for Computer Graphics*.
[2] Vince, John. *Mathematics for Computer Graphics*.
[3] http://math.stackexchange.com/questions/71/real-world-uses-of-quaternions.
Accessed on December 12th 2015 at 16.00.
[4] https://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/3drota.htm.
Accessed on December 12th 2015 at 16.00.
[5] https://www.researchgate.net/publication/255594303_Quaterions_and_their_Applications_to_Rotation_in_3D_Space.
Accessed on December 12th 2015 at 17.00
[6] http://www.gamasutra.com/view/feature/131686/rotating_objects_using_quaternions.php
Accessed on December 12th 2015 at 17.00
[7] http://docs.unity3d.com/Manual/UnityOverview.html
Accessed on December 15th 2015 at 10.00
[8] http://docs.unity3d.com/Manual/UnityManual.html
Accessed on December 15th 2015 at 10.00
[9] http://docs.unity3d.com/learn/tutorials/topics/graphics
Accessed on December 15th 2015 at 11.00
[10] https://unity3d.com/learn/tutorials/modules/intermediate/scripting/quaternions
Accessed on December 15th 2015 at 11.00

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 16 Desember 2015



Geraldi Dzakwan 13514065