Mathematical Exploration: Emulating Computer Calculations in Order to Manipulate a Cube's On-Screen Appearance

May 2020 Session.
2002 Words.

Contents

1 Problem

The aim of this exploration is to emulate what a computer would calculate when it manipulates the appearance of an object on a screen. For simplicity, I am going to be exploring what these calculations would be for a cube, as opposed to complicated models used in the real world. More specifically, I will explore what mathematical steps are required for the following on-screen transformation: Firstly, a twicefold increase in the length of the edges (increasing the cube's volume by 8). Secondly, a rightward $\frac{\pi}{4}$ radian rotation of the cube. Thirdly, a downward $\frac{\pi}{4}$ radian rotation of the cube. This is shown graphically in figure 1.

By exploring these steps, I will touch upon various concepts. I will be using basic vector algebra, which is the only topic covered in the scope of the syllabus. Additionally, I will mainly be working with matrices and performing matrix arithmetic. The rotations will cover Euler angles as well as some aspects of quaternions.

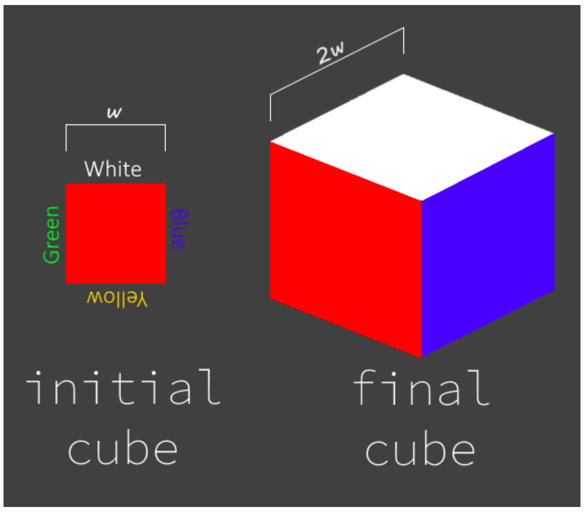

Figure 1: Visual representation of the problem

2 Context

OpenGL is a software which is a tool that allows programmers to perform mathematical calculations and display graphics on the screen to the user [3]. Graphical processing units are hardware chips that allow points, lines, and triangles to be rendered on a display [2]. Whilst quadrilaterals can also be rendered, triangles are preferred as they are able to model complicated surfaces easily, and consume less power. Simply put, OpenGL allows programmers to forward instructions directly to the graphical processing unit. As such, this exploration will be me emulating multiple of these instructions (scaling and rotating).

As I am emulating the mathematical steps, in order to stay consistent with what the computer would do, there need to be certain restrictions. There are a few deviations from standard cartesian mathematics, but these will be elaborated upon later. The steps must also be achievable on a Windows computer, with a programming language that supports OpenGL, such as C++.

3 Coordinates and Meshes

3.1 The Coordinate System

OpenGL's coordinate system differs from the three dimensional cartesian coordinate system. The X axis goes rightward, the Y axis goes upward, and the Z axis out of the page. This is shown on figure 2.

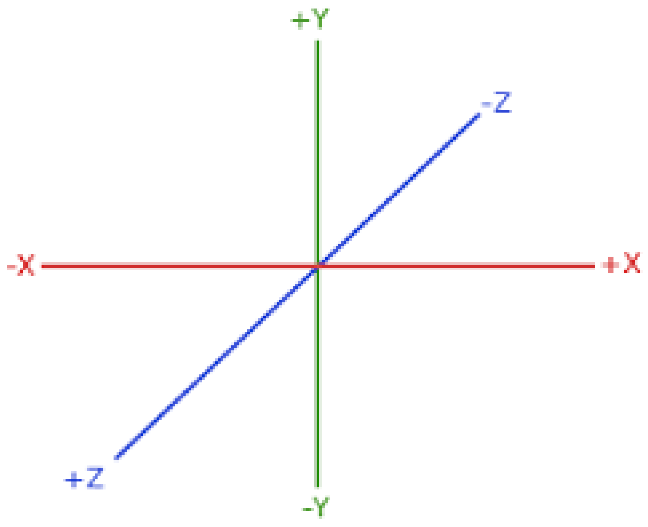

Figure 2: OpenGL coordinate system (from https://learnopengl.com)

3.2 Meshes and Vertices

A mesh is a collection of vertices connected by edges which then form the faces of an object. The vertices are defined as position vectors represented as 1x4 matrices. Matrices are numbers/expressions that are arranged into columns and rows. $V_{1,1}$ to $V_{1,3}$ represent the position vector, the x, y, and z values, respectively. $V_{1,4} = 1$ is used to tell OpenGL that V is a position, rather than a direction ($P_{1,4} = 0$). Direction matrices are beyond the scope of this investigation, and as such the fourth row only exists as compatibility for the aforementioned emulation.

$$V = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \; ; \; x, y, z \in \mathbb{R}$$

3.3 Cube Vertices

Because graphics processing units work with two dimensional triangles, the cube needs to be split up into its faces. A cube has six faces, each of which can be split up again into two equal-sized triangles. As such, a cube consists of 36 individual vertices, duplicates of 8 unique vertices. Only the unique vertices will be calculated for performance reasons, but all 36 need to be specified in OpenGL.

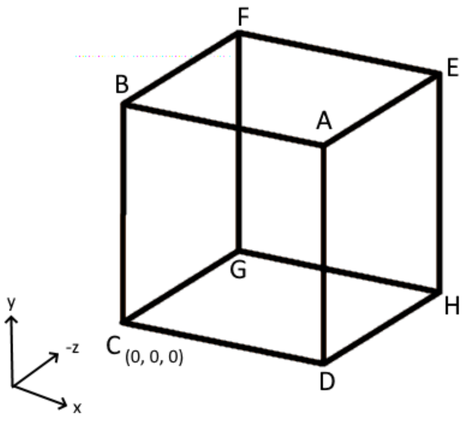

Figure 3: Diagram representing the vertices in a 3D OpenGL coordinate system

Figure 3 shows the 8 unique verices A to H graphically. The vertex C is located at the origin. As such, the matrices I defined for the vertices are shown in equations 1 and 2.

$$A = \begin{bmatrix}1\\1\\0\\1\end{bmatrix} \quad B = \begin{bmatrix}0\\1\\0\\1\end{bmatrix} \quad C = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \quad D = \begin{bmatrix}1\\0\\0\\1\end{bmatrix} \tag{1}$$

$$E = \begin{bmatrix}1\\1\\-1\\1\end{bmatrix} \quad F = \begin{bmatrix}0\\1\\-1\\1\end{bmatrix} \quad G = \begin{bmatrix}0\\0\\-1\\1\end{bmatrix} \quad H = \begin{bmatrix}1\\0\\-1\\1\end{bmatrix} \tag{2}$$

4 Scaling

4.1 The Scale Matrix

In order to make one of the cube's edges twice as long, at least one of the two vertices needs to move such that the displacement between the two increases twicefold. More specifically, if one represents the displacement from one vertex to another (i.e. an edge) as a vector, the vector's magnitude would need to be twicefold. This can be achieved by simply multiplying the vertices (as position matrices) by a scale matrix.

A matrix that scales by a factor of a on the x axis, a factor of b on the y axis, and a factor of c on the z axis takes the following form:

$$S = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ;\ a, b, c \in \mathbb{R}$$

Because the cube will be scaled uniformly (such that it remains a cube, rather than becoming a rectangle), $a = b = c$. The column S_4 is required because of the nature of matrix multiplication (the position matrix will be multiplied by the scale matrix). The number of columns on the scale matrix must be equal to the number of rows on the position matrix.

4.2 Application to Problem

The problem requires the cube's edges to be twice the length, therefore the scale matrix looks as follows:

$$S = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This can then be applied to the vertices that were defined in the previous section. The new position matrices are shown in equations 3 and 4.

$$\begin{aligned}
A &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 1 \end{bmatrix} \\
B &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} \\
C &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
D &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix}
\end{aligned} \tag{3}$$

$$\begin{aligned}
E &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ -2 \\ 1 \end{bmatrix} \\
F &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ -2 \\ 1 \end{bmatrix} \\
G &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -2 \\ 1 \end{bmatrix} \\
H &= \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \\ 1 \end{bmatrix}
\end{aligned} \tag{4}$$

5 Rotations

Rotations can be done in two ways: Euler angles and quaternions. I will be elaborating upon both of these, but only apply quaternions to solve the problem.

5.1 Euler Angles

Euler angles are a set of three angles that describe the orientation of a mesh, and by extension, each vertex [8]. An example can be seen in figure 4, but it should be noted that these are for a standard Cartesian coordinate system, whereas this exploration uses OpenGL's system. It is important to distinguish between orientations and rotations. Orientations describe the state of the object, whereas rotations are the process of orientating an object to be in this state [5]. As the Euler angles describe an orientation, a respective rotation matrix needs to be defined in order to translate (rotate) the mesh into the particular orientation described by the Euler angles.

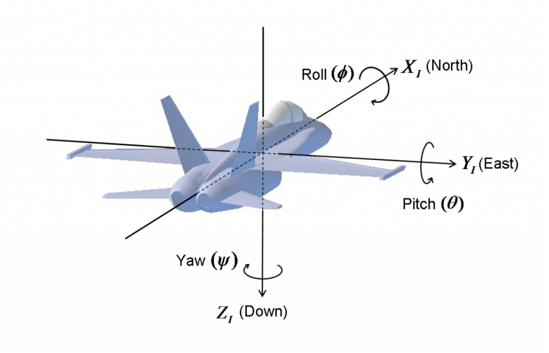

Figure 4: Example of how Euler angles can portray orientation on a model plane [6]

Euler angles can be specified in different orders, and this order will change the final orientation of the mesh [8]. This is because each successive rotation contains all the previous rotations. This will be relevant later on. Different orders of angles are given separate names. Proper euler angles describe *z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y* orientations [8]. Tait-Bryan angles describe *x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z* orientations [8]. Within graphics processing, the order of angles is generally specified as y-z-x, which are Tait-Bryan angles. However, it should be noted that these are nonetheless referred to as Euler angles in graphics jargon.

Every one of the three components of Euler angles can be expressed as matrices. Assuming a, b and c are the angles for X, Y and Z respectively, the following formulae can be used [7]:

$$Y = \begin{bmatrix} \cos(b) & 0 & \sin(b) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(b) & 0 & \cos(b) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad b \in \mathbb{R} \qquad (6)$$

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ;\ a \in \mathbb{R} \qquad (5)$$

$$Z = \begin{bmatrix} \cos(c) & -\sin(c) & 0 & 0 \\ \sin(c) & \cos(c) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad c \in \mathbb{R} \qquad (7)$$

Again, the fourth column is included to allow the multiplication with the position matrix. The final rotation matrix can then be found by multiplying these components. For a y-z-x order, $R = Y \times Z \times X$, where R is the final rotation matrix that can be applied to all vertices.

However, Euler (or Tait-Bryan) angles can be problematic as they can create a gimbal lock. These can be explained using an object, such as a plane, in a gimbal. As aforementioned, each rotation also includes the previous rotations. It some cases, this can lead to two "gimbals" aligning with each other, often when angles approach $\frac{\pi}{2}$ radians. This is shown in figure 5.

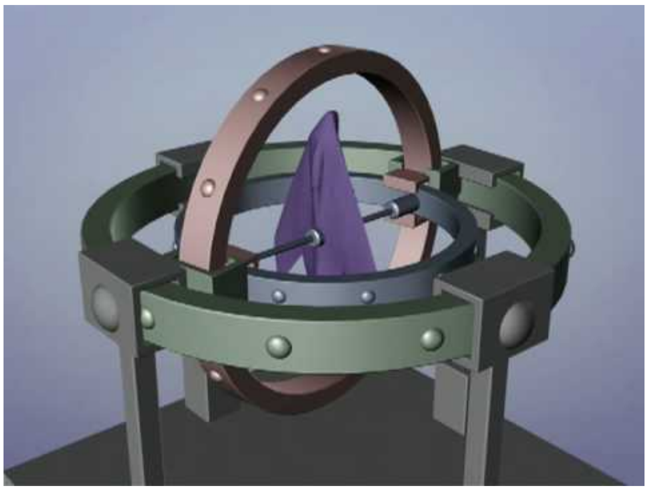

Figure 5: A gimbal lock on an aeroplane [4]

Here, two gimbals are aligned horizontally. As such, one gimbal is "lost", they are "stuck together", and rather than being able to move three gimbals, only two can be used. Mathematically speaking, this means that in this scenario one dimension is lost. This is undesireable, as rotations often need to be three dimensional. While it is possible to solve my problem with Euler angles, I will be using quaternions instead, as these are guaranteed to work for anyting without creating gimbal locks.

5.2 Quaternions

Quaternions are number systems that extend that of complex numbers [9]. Complex numbers take the form of a real part and an imaginary part, e.g.:

$$z = a + b\mathbf{i};\ \ a, b \in \mathbb{R}, z \in \mathbb{C}$$

Quaternions take the form of:

$$q = w + a\mathbf{i} + b\mathbf{j} + c\mathbf{k};\ \ w, a, b, c \in \mathbb{R}, q \in \mathbb{H}$$

Where $w, a, b, c \in \mathbb{R}$ and $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are the fundamental quaternion units. In this case, w refers to the real part, and $ai + bj + ck$ the vector part. Quaternions are much more complicated, but in this investigation I will not go in depth into them, and only apply what is neccessary to solve my problem.

5.2.1 Rotation Angle and Axis

There are two components to quaternion rotations [5]. Firstly, a rotation axis needs to be defined. This is a unit vector representing the direction of the rotation axis. Then, an angle describing the magnitude of the rotation about the axis needs to be defined. This is shown in figure 6

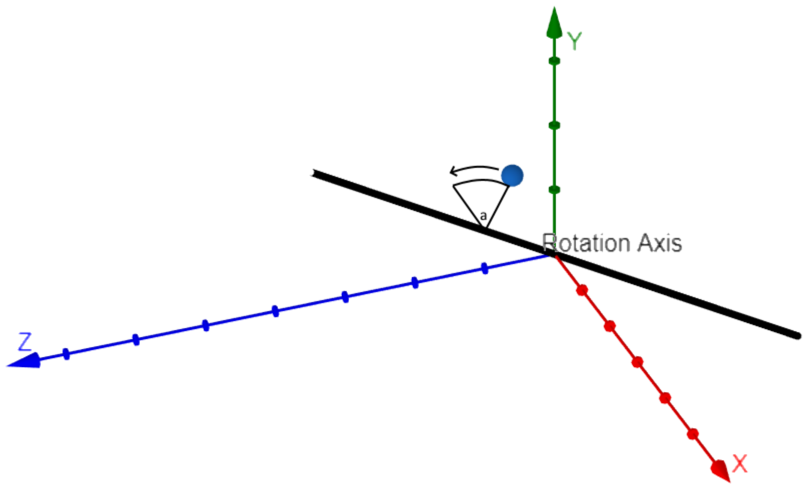

Figure 6: A rotation with angle a of a point around a rotation axis

Due to both the angles being $\frac{\pi}{2}$ radians, this is also the rotation angle. I found the axis graphically. Since the rotations are done on the X and Y axes, I found the four possible direction vectors. These are:

$$\vec{a} = \begin{pmatrix}1\\1\\0\end{pmatrix}, \vec{b} = \begin{pmatrix}-1\\1\\0\end{pmatrix}, \vec{c} = \begin{pmatrix}1\\-1\\0\end{pmatrix}, \vec{d} = \begin{pmatrix}-1\\-1\\0\end{pmatrix}$$

By graphing these vectors and performing (imaginary) roations around the axis they create, I deduced that the correct axis was $\vec{b}$. Next, I normalized the vector to get a unit vector (this is important later on):

$$\begin{aligned}\vec{b} &= -\mathbf{i} + \mathbf{j}\\ \hat{b} &= \frac{\vec{b}}{|\vec{b}|}\\ &= \frac{-\mathbf{i}+\mathbf{j}}{\sqrt{2}}\\ &= -\frac{1}{\sqrt{2}}\mathbf{i} + \frac{1}{\sqrt{2}}\mathbf{j}\end{aligned} \tag{8}$$

5.2.2 Quaternion Normalization

Before moving on, another concept is vital for quaternion rotations. In order to painlessly convert quaternion rotations into a matrix form, the quaternion needs to be normalized [10]. Normalized/unit quaternions have magnitude 1. They are normalized exactly the same way as vectors are [1], just with an additional component, shown in equation 9

$$
\begin{aligned}
q &= w + a\mathbf{i} + b\mathbf{j} + c\mathbf{k};\;\; w,a,b,c \in \mathbb{R}, q \in \mathbb{H} \\
q_{norm} &= \frac{q}{||q||} \\
q_{norm} &= \frac{w + a\mathbf{i} + b\mathbf{j} + c\mathbf{k}}{\sqrt{w^2 + a^2 + b^2 + c^2}}
\end{aligned} \tag{9}
$$

5.2.3 Axis and Angle to Quaternion

The rotation axis (unit vector, $\hat{v}$) and angle (θ) can be converted into quaternions using the following formulae [5]:

$$
\begin{aligned}
w &= \cos\left(\frac{\theta}{2}\right) \\
a &= v_1 \cdot \sin\left(\frac{\theta}{2}\right) \\
b &= v_2 \cdot \sin\left(\frac{\theta}{2}\right) \\
c &= v_3 \cdot \sin\left(\frac{\theta}{2}\right)
\end{aligned} \tag{10}
$$

This makes sense because quaternions consist of a scalar and vector component. With a unit vector $a^2\mathbf{i} + b^2\mathbf{j} + c^2\mathbf{k} = 1$, this can be combined with Pythagorean's identity of $\cos^2(\theta) + \sin^2(\theta) = 1$ [7]. While Wolfram MathWorld introduces this idea [7], it is not very clear and I would like to offer a more concrete proof.

$$
\begin{aligned}
\cos^2\left(\frac{\theta}{2}\right) + \left(1 \cdot \sin^2\left(\frac{\theta}{2}\right)\right) &= 1 \\
\cos^2\left(\frac{\theta}{2}\right) + \left(\left(a^2\mathbf{i} + b^2\mathbf{j} + c^2\mathbf{k}\right) \cdot \sin^2\left(\frac{\theta}{2}\right)\right) &= 1 \\
\cos^2\left(\frac{\theta}{2}\right) + \left(a \cdot \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(b \cdot \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(c \cdot \sin\left(\frac{\theta}{2}\right)\right)^2 &= 1 \\
\sqrt{\cos^2\left(\frac{\theta}{2}\right) + \left(a \cdot \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(b \cdot \sin\left(\frac{\theta}{2}\right)\right)^2 + \left(c \cdot \sin\left(\frac{\theta}{2}\right)\right)^2} &= 1
\end{aligned} \tag{11}
$$

This is a form of $\sqrt{w^2 + a^2 + b^2 + c^2} = 1$, a unit quaternion. Here, w, a, b and c are the individual components laid out in equation 10.

5.2.4 Finding the Quaternion

Using the previously determined unit vector $\hat{b}$:

$$
\begin{aligned}
w &= \cos\left(\frac{\frac{\pi}{4}}{2}\right)\\
&\approx 0.924\\
x &= -\frac{1}{\sqrt{2}} \cdot \sin\left(\frac{\frac{\pi}{4}}{2}\right)\\
&\approx -0.271\\
y &= \frac{1}{\sqrt{2}} \cdot \sin\left(\frac{\frac{\pi}{4}}{2}\right)\\
&\approx 0.271
\end{aligned}
\tag{12}
$$

Therefore, the quaternion can be expressed as:

$$q = 0.924 - 0.271\mathbf{i} + 0.271\mathbf{j};\ q \in \mathbb{H}$$

This quaternion, q, essentially describes the rotation that the cube will undergo in order to be in the desired orientation. It will be applied to each individual vertex in order to create a new vertex. These new vertices will form the final mesh of the rotated cube. However, in order to be able to do this, the quaternion needs to be converted to a rotation matrix.

5.2.5 Converting to a Rotation Matrix

Converting to a rotation matrix can be done using a formula [10]. The mathematics behind it is beyond my capabilities and the scope of the investigation. Yet again, I've added another column to allow multiplication with position matrices.

$$R = \begin{bmatrix} 1-2(q_y^2+q_z^2) & 2(q_xq_y - q_zq_w) & 2(q_xq_z + q_yq_w) & 0 \\ 2(q_xq_y + q_zq_w) & 1-2(q_x^2+q_z^2) & 2(q_yq_z - q_xq_w) & 0 \\ 2(q_xq_z - q_yq_w) & 2(q_yq_z + q_xq_w) & 1-2(q_x^2+q_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix};\ \mathit{where}\ q \in \mathbb{H}, ||q|| = 1$$

When plugging in the previously found quaternion, q, into the equation, it yields the following rotation matrix:

$$R = \begin{bmatrix} 0.854 & -0.146 & 0.5 & 0 \\ -0.146 & 0.854 & 0.5 & 0 \\ -0.5 & -0.5 & 0.707 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is the rotation matrix that can be applied to each individual vertex.

5.3 Rotating the Vertices

Finally, all the vertices can be transformed by multiplying the position matrices by the rotation matrix in order to form the new vertex positions. The new position matrix will again be denoted as *, and are shown in equations 13 and 14.

$$
\begin{aligned}
A* = R \times A &= \begin{bmatrix} 1.416 \\ 1.416 \\ -2 \\ 1 \end{bmatrix} \\
B* = R \times B &= \begin{bmatrix} -0.292 \\ 1.708 \\ -1 \\ 1 \end{bmatrix} \\
C* = R \times C &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
D* = R \times D &= \begin{bmatrix} 1.708 \\ -0.292 \\ -1 \\ 1 \end{bmatrix}
\end{aligned} \tag{13}
$$

$$
\begin{aligned}
E* = R \times E &= \begin{bmatrix} 0.416 \\ 0.416 \\ -3.414 \\ 1 \end{bmatrix} \\
F* = R \times F &= \begin{bmatrix} -1.292 \\ 0.708 \\ -2.414 \\ 1 \end{bmatrix} \\
G* = R \times G &= \begin{bmatrix} -1 \\ -1 \\ -1.414 \\ 1 \end{bmatrix} \\
H* = R \times H &= \begin{bmatrix} 0.708 \\ -1.292 \\ -2.414 \\ 1 \end{bmatrix}
\end{aligned} \tag{14}
$$

Now that the vertices of a scaled and rotated cube have been found, the mesh is complete. The final step is to group vertices together to form 12 equal area triangles.

6 Conclusion

With the new vertices of the mesh, the investigation is complete. The twelve triangles can now be fed into OpenGL with their respective vertices. These triangles are shown in table 1, the vertices of each triangle are simply in alphabetical order.

Triangle	Vertex 1	Vertex 2	Vertex 3
T_1	A*	B*	C*
T_2	A*	C*	D*
T_3	E*	F*	G*
T_4	E*	G*	H*
T_5	A*	B*	E*
T_6	B*	E*	F*
T_7	C*	D*	G*
T_8	D*	G*	H*
T_9	B*	C*	G*
T_{10}	B*	F*	G*
T_{11}	A*	D*	H*
T_{12}	A*	E*	H*

Table 1: The twelve triangles to be fed into OpenGL

There need to be several more steps before the result is actually visible on-screen. The colour (gradient) of each triangle needs to be specified, in this case two triangles will share one solid colour to form a cube face. Then, everything would have to go through a shader program before it is actually rendered on a display.

Under normal circumstances, a programmer will define the vertices of the mesh, and do one of two things. Either, apply a movement matrix to the "camera", such that the camera moves around the object, making it appear as if there was movement, when in fact, the camera is the only thing moving. Alternatively, they will give the program a rotation axis with an angle or Euler/Tait-Bryan angles, after which the program calculates the rotation quaternion and matrix. This matrix can then be applied to the coordinates. However, mathematically this is not particularly interesting. As such that is why I chose to emulate the second option manually, up to creating the final mesh, for my exploration. This mesh I found can then be inserted, as aforementioned, into OpenGL, to continue the rest of the computation.

Nonetheless, a cube is an excellent example of how vectors, matrices, quaternions and trigonometry work together to solve a problem. Taking this investigation further, one could find more complicated meshes (e.g. of a curved surface) and apply rotations to those.

References

[1] Peter Petrov (https://math.stackexchange.com/users/116591/peter-petrov). *Normalizing a quaternion.* URL: `https://math.stackexchange.com/q/1703467` (visited on 02/10/2020).

[2] jalf (https://stackoverflow.com/users/33213/jalf). *Are triangles a gpu restriction or are there other rendering pathways?* URL: `https://stackoverflow.com/a/12495652` (visited on 03/04/2020).

[3] *About OpenGL.* URL: `https://www.opengl.org/about/` (visited on 03/04/2020).

[4] *Euler (gimbal lock) Explained.* URL: `https://www.youtube.com/watch?v=zc8b2Jo7mno` (visited on 02/10/2020).

[5] *OpenGL Tutorial.* URL: `https://www.opengl-tutorial.org/` (visited on 12/19/2019).

[6] *Understanding Euler Angles.* URL: `http://www.chrobotics.com/library/understanding-euler-angles` (visited on 02/10/2020).

[7] Eric Weisstein. *Euler Angles.* URL: `http://mathworld.wolfram.com/EulerAngles.html` (visited on 02/10/2020).

[8] Wikipedia contributors. *Euler angles — Wikipedia, The Free Encyclopedia.* 2020. URL: `https://en.wikipedia.org/wiki/Euler_angles` (visited on 02/10/2019).

[9] Wikipedia contributors. *Quaternions — Wikipedia, The Free Encyclopedia.* 2019. URL: `https://en.wikipedia.org/wiki/Quaternion` (visited on 12/19/2019).

[10] Wikipedia contributors. *Quaternions and spatial rotation – Wikipedia, The Free Encyclopedia.* 2019. URL: `https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation` (visited on 12/19/2019).