Experiment: 4

Student Name: Shaurya Gulati **UID: 18BCS6092**
Branch: AIML **Lab Group: A**
Semester: 8th **Date of Performance: March 4th**
Subject Name: AR LAb **Subject Code: CSF- 484**

1. Aim/Overview of the practical:

3D Rotations; Quaternions; Conversions

2. Task to be done:

Discuss about the 3D Rotations in detail. In Addition, implement as well as write down the steps to perform the 3D Rotations-Quaternions and Conversions.

3. Theory:

3D Rotation: 3D Rotation is a process of rotating an object with respect to an angle in a three dimensional plane.

We learned that a 3x3 orthogonal matrix with determinant 1 can describe a rotation in R3 about an axis through the origin. The matrix representation, on the other hand, appears redundant because only four of the nine elements are independent. Also, until we perform numerous steps of calculation to extract the rotation axis and angle, the geometric inter- pretation of such a matrix is unclear. Furthermore, we must compute the product of the two matching matrices to create two rotations, which necessitates twenty-seven multiplications and eighteen additions.
When it comes to understanding scenarios involving rotations in R3, quaternions are extremely useful. A quaternion is a four-tuple representation that is more compact than a rotation matrix. It has a geometric connotation.

Two items are represented by a quaternion. It contains three components: x, y, and z, which represent the axes around which the rotation will take place. It also has a w component, which indicates how much rotation will occur around this axis. In a nutshell, it's a vector and a float. With these four integers, you can create a matrix that exactly represents all rotations and eliminates the possibility of gimbal lock.

A quaternion is a set of four numbers, three of which are imaginary numbers. It is defined as sqrt, as many of you may recall from math education (-1). In the case of quaternions, I = j = k = sqrt (-1). q = w + xi + yj + zk is the definition of the quaternion. All of the numbers w, x, y, and z are real. If you ever have a math class involving quaternions, the imaginary components are crucial, but they aren't really important in programming. This is why: A quaternion will be stored in a class with four member variables: float w, x, y, and z. We'll start by defining our quaternions (w, x, y, z).

Normalizing a Quaternion:

magnitude = sqrt($w^2 + x^2 + y^2 + z^2$)
w = w / magnitude
x = x / magnitude
y = y / magnitude
z = z / magnitude

Let Q_1 and Q_2 be two quaternions, which are defined, respectively, as (w_1, x_1, y_1, z_1) and (w_2, x_2, y_2, z_2).
$(Q_1 * Q_2).w = (w_1w_2 - x_1x_2 - y_1y_2 - z_1z_2)$
$(Q_1 * Q_2).x = (w_1x_2 + x_1w_2 + y_1z_2 - z_1y_2)$
$(Q_1 * Q_2).y = (w_1y_2 - x_1z_2 + y_1w_2 + z_1x_2)$
$(Q_1 * Q_2).z = (w_1z_2 + x_1y_2 - y_1x_2 + z_1w_2$

```
// generate local_rotation
total = local_rotation * total //multiplication order matters on this line
```

It is very important to note that Q1 * Q2 is NOT equal to Q2 * Q1. Quaternion multiplication is not commutative.

Generating the Local_rotation:

```
//axis is a unit vector
local_rotation.w = cosf( fAngle/2)
local_rotation.x = axis.x * sinf( fAngle/2 )
local_rotation.y = axis.y * sinf( fAngle/2 )
local_rotation.z = axis.z * sinf( fAngle/2 )
```

Since we'll be multiplying two unit quaternions together, the result will be a unit quaternion.

All that is left for this tutorial is generating the matrix from the quaternion to rotate our points.

$w^2+x^2-y^2-z^2$	$2xy-2wz$	$2xz+2wy$	0
$2xy+2wz$	$w^2-x^2+y^2-z^2$	$2yz+2wx$	0
$2xz-2wy$	$2yz-2wx$	$w^2-x^2-y^2+z^2$	0
0	0	0	1

And, since we're only dealing with unit quaternions, that matrix can be optimized a bit down to this:

$1-2y^2-2z^2$	$2xy-2wz$	$2xz+2wy$	0
$2xy+2wz$	$1-2x^2-2z^2$	$2yz+2wx$	0
$2xz-2wy$	$2yz-2wx$	$1-2x^2-2y^2$	0
0	0	0	1

We'll regenerate these matrices each frame. Most importantly, we won't get gimbal lock. We'll need to initialize your total quaternion to the value (1,0,0,0). This represents the "initial" state of your object - no rotation.

Interpolating between two orientations using quaternions is also the smoothest way to interpolate angles. It's done via a method known as SLERP, or Spherical Linear intERPolation.