Rotations Lab Write-Up

By John Ying

**Introduction**

Within the gaming world, rotations is quite key to nearly all interactions, especially in the 3D world. Without it, how could you effectively turn your character, spaceship, *anything!* Even stationary objects like turrets all use rotation. This program simulates the coordinate point rotations of such required movements.

**Methods**

The matrix multiplication required for rotation on a 3D plane can be cut down simply because of multiple multiples of zeros that end up adding to nothing. In the long full version, the rotation matrix is dot product to the position vector (given shape points). Below are the rotation matrices.

This is simplified for the x-axis rotation point by point based on a given degrees.

And for the y-axis rotation…..

And for the z-axis rotation…...

**Results**

These represent the results of the actual program. I began with a rotation off of the x-axis, then y-axis, and finally the z-axis. All calculations are based upon the same input coordinates that were input at the beginning. The w point can be ignored since it is not used within the scope or rotations (but it is used for scaling and translating).

------------------------------------------

Please input the coordinates for point 0: {x, y, z, w}

1 0 0 0

Please input the coordinates for point 1: {x, y, z, w}

3 0 0 0

Please input the coordinates for point 2: {x, y, z, w}

2 1 0 0

Please input the coordinates for point 3: {x, y, z, w}

2 0 1 0

What kind of transformation would you like to do?

t = translation

rs = raw scaling

sc = scaling about a center

r = rotation

r

What would you like to rotate? {degrees axis}

30 x

The transformation output is:

x: 1

y: 0

z: 0

vPoint(w): 0

x: 3

y: 0

z: 0

vPoint(w): 0

x: 2

y: 0.866026

z: 0.5

vPoint(w): 0

x: 2

y: -0.5

z: 0.866026

vPoint(w): 0

What would you like to rotate? {degrees axis}

50 y

The transformation output is:

x: 0.642788

y: 0

z: -0.766044

vPoint(w): 0

x: 1.92836

y: 0

z: -2.29813

vPoint(w): 0

x: 1.28558

y: 1

z: -1.53209

vPoint(w): 0

x: 2.05162

y: 0

z: -0.8893

vPoint(w): 0

What would you like to rotate? {degrees axis}

130 z

The transformation output is:

x: -0.642786

y: 0.766046

z: 0

vPoint(w): 0

x: -1.92836

y: 2.29814

z: 0

vPoint(w): 0

x: -2.05162

y: 0.889305

z: 0

vPoint(w): 0

x: -1.28557

y: 1.53209

z: 1

vPoint(w): 0

All results came out as expected (correctly) based on a 3D coordinate plane. These can be used correctly up to float c++ digits in efficiency and accuracy.

**Conclusion**

This rotation program will solve for probably about 90% of all needed rotational game calculations needed very effectively and accurately. With numbers dealing in floats, all rotations will be easily applied to an in-game coordinate system’s game engine environment. Although a rotation through time is not applied, a simple calculation only is needed based on the time frame and using the forward euler method.