Comparative Study of 3 Dimensional Rotation Methods in Computer Graphics

Vijay Kumar Singh
Assistant Professor, Department of I.T
L.N. Mishra College of Business Management, Muzaffarpur

S. P. Varma
Ex-Faculty, P.G. Department of Maths (also a centre for education of I.T)
B.R.A.Bihar University, Muzaffarpur

Abstract: Rotation in 3 Dimensional geometry is more complex because of the fact that here we consider the angle of rotation as well as the axis of rotation. It is proposed to study the different methods of rotation on different objects in 3Dimension. We shall use the matrix multiplication method, orthogonal matrix multiplication, Euler's method of rotation and also use the Quaternion's method of rotation as well. Also all these will be authenticated through by programming using java.

Keywords: Rotation, Quaternion, Euler, Matrix Multiplication, 3D graphics

Introduction: It was in the 19th century that an Iris mathematician and physicist W.R.Hamilton got attracted to notice the role of complex numbers in 2 Dimensional geometry. Then for years he tried to invent algebra of triplets to play the same role in 3 Dimensions. In 1843 Hamilton discovered a 4Dimensional Division algebra known as Quaternion's. This made many rotations in space easier. In what follows different methods of rotations will be discussed and a comparative analysis will be presented.

Rotation Using Matrix Multiplication

The general rotation is performed using matrix multiplication. The rotation matrix for rotation about the X-Axis is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The points after rotation can be calculated from

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Similarly Rotation about Y-Axis can be calculated from the equation written in matrix form as below:-

$$R_y(\theta) = 0 \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 1 & 0 & 0 & \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

And Rotation about Z-axis

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Euler's Method for rotation

The most commonly used definition in graphics described a rotation R by Euler angles (Φ, θ, ψ) as a product of three rotations.

$R=R_Z(\Phi)\ R_Y(\theta)\ R_X(\psi)$

Thus the objects are first rotated by angle Φ in XY- Plane then by θ in ZX- Plane, and third by angle ψ in YZ-Plane

This is represented by matrix

$$\begin{pmatrix} \cos\theta*\cos\Phi & \sin\Phi*\sin\theta*\cos\theta - \cos\psi*\sin\Phi & \cos\psi*\sin\theta*\cos\Phi + \sin\psi*\sin\Phi & 0 \\ \cos\theta*\cos\Phi & \sin\psi*\sin\theta*\cos\Phi + \cos\psi*\sin\Phi & \cos\psi*\sin\theta*\cos\Phi - \sin\psi*\sin\Phi & 0 \\ -\sin\theta & \sin\psi*\cos\theta & \cos\psi*\cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

With the proper choice of **Φ, θ** and **ψ** such products represent all possible rotations.

Orthogonal Matrix for Rotation

The real orthogonal square matrix of order 3 represents a rotation in3 dimension through

an angle **θ** about a general line through the origin with direction cosine **l, m,** and **n.** The rotation matrix is given by

$$R_{XYZ} = \begin{pmatrix} l^{2*}(1-\cos\theta)+\cos\theta & l*m(1-\cos\theta)- n*\sin\theta & (1-\cos\theta)+m*\sin\theta & 0 \\ m*l(1-\cos\theta)+n*\sin\theta & m^2(1-\cos\theta)+\cos\theta & m*n(1-\cos\theta)- l*\sin\theta & 0 \\ n*l(1-\cos\theta) - m*\sin\theta & n*m(1-\cos\theta)+l*\sin\theta & n^2(1-\cos\theta) - \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

To rotate through **X-axis** put ***l*=1**, ***m*=0** and ***n*=0** for **Y-axis** rotation put ***l*=1, *m*=0** and ***n*=0** and for the **Z-axis** rotation put ***l*=0**, ***m*=0** and ***n*=1.**

Quaternion's for rotation

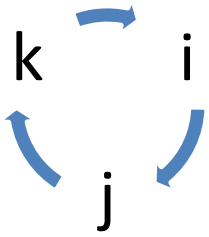
Hamilton discovered a four dimensional algebra known as quaternions. Where

$\mathbf{Q = q_1 + iq_2 + jq_3 + kq_4}$

Where q_1 is the real part q_2, q_3 and q_4 are the imaginary part. And i , j and k are the imaginary units and they meet the relationships.

1) $i^2 = j^2 = k^2 = -1$
2) $ij = -ik = k$
3) $jk = -kj = i$
4) $ki = -ik = j$

The ***i ,j and k*** are all square roots of -1. It is to be noticed that quaternions do not commute and quaternion product is same as cross product.



of vectors where

$\vec{i} \times \vec{j} = \vec{k}$
$\vec{j} \times \vec{k} = \vec{k}$
$\vec{i} \times \vec{k} = \vec{j}$

Except for the cross product

$\vec{i} \times \vec{i} = \vec{j} \times \vec{j} = \vec{k} \times \vec{k} = 0$

Where for quaternions this is -1.

Program code of x-axis rotation

```
for(int j = 60; j <= 60; j+=60)
                {
```

```
double angle=(j*(Math.PI/180));
System.out.println("Theta   = "+j);
double matrix_b[][]={
{ 1,0,0,0 } ,
{ 0,Math.cos(angle),Math.sin(angle),0 } ,
{ 0,-(Math.sin(angle)),Math.cos(angle),0 } ,
{ 0,0,0,1 }
};
int s=0;
for(int i = 0; i < 36; i++)
{
pt[0]=p[i].x;
pt[1]=p[i].y;
pt[2]=p[i].z;
pt[3]=1;
for(int ij=0;ij<4;ij++)
{
s=0;
for(int
count_2=0;count_2<4;count_2++)
{
s+=(pt[count_2]*matrix_b[count_2][ij]);
}
pt[ij]=s;
}
pdraw[i].x=(pt[0])+240;
```

```
                                        pdraw[i].y=(pt[1])+180;
                                        pdraw[i].z=pt[2];
                                }
```

Program Code for Y-Axis Rotation

```
        for(int j = 60; j <= 60; j+=60)
        {
                double angle=(j*(Math.PI/180));
                System.out.println("Theta   = "+j);
                double matrix_b[][]={
                                        {  Math.cos(angle),0,-(Math.sin(angle)),0
} ,
                                        { 0,1,0,0 } ,
                                        { Math.sin(angle),0,Math.cos(angle),0 } ,
                                        { 0,0,0,1 }
                                        };
                                         int s=0;
                        for(int i = 0; i < 36; i++)
                        {
                                        pt[0]=p[i].x;
                                        pt[1]=p[i].y;
                                        pt[2]=p[i].z;
                                        pt[3]=1;

                                for(int ij=0;ij<4;ij++)
                                        {       s=0;
                                                for(int
count_2=0;count_2<4;count_2++)
```

```
                                        {

s+=(pt[count_2]*matrix_b[count_2][ij]);

                                        }

                                            pt[ij]=s;

                                        }

                                        pdraw[i].x=(pt[0])+240;

                                        pdraw[i].y=(pt[1])+180;

                                        pdraw[i].z=pt[2];

                                }
```

Program Code for Z-Axis Rotation

```
        for(int j = 60; j <=60; j+=60)

        {

                double angle=(j*(Math.PI/180));

                System.out.println("Theta   = "+j);

                double matrix_b[][]={

                                        { Math.cos(angle),Math.sin(angle),0,0 } ,

                                        { -(Math.sin(angle)),Math.cos(angle),0,0 } ,

                                        { 0,0,1,0 } ,

                                        { 0,0,0,1 }

                                };

                        int s=0;

                        for(int i = 0; i < 36; i++)

                        {

                                        pt[0]=p[i].x;

                                        pt[1]=p[i].y;
```

```
pt[2]=p[i].z;


pt[3]=1;
for(int ij=0;ij<4;ij++)
{
        s=0;
        for(int
count_2=0;count_2<4;count_2++)
        {
        s+=(pt[count_2]*matrix_b[count_2][ij]);
        }
        pt[ij]=s;
}
pdraw[i].x=(pt[0])+240;
pdraw[i].y=(pt[1])+180;
pdraw[i].z=pt[2];
}
```

Program Code for Quaternion Rotations

```
  private FourPoint qm(FourPoint q, FourPoint p)
 {
     FourPoint l = new FourPoint();
     l.x = q.w * p.x - q.z * p.y + q.y * p.z + q.x * p.w;
     l.y = q.z * p.x + q.w * p.y - q.x * p.z + q.y * p.w;
     l.z =-q.y * p.x + q.x * p.y + q.w * p.z + q.z * p.w;
     l.w =-q.x * p.x - q.y * p.y - q.z * p.z + q.w * p.w;
     return l;
```

```
    }
public void rotate()
{
      for(int j = 60; j <= 60; j+=60)
         {
         o.x =(a*0.5)*Math.sin(Math.PI*(j)/180);
         neo.x =-o.x;
         o.y =-(a)*Math.sin(Math.PI*j/180);
         neo.y =-o.y;
                  o.z =(a/3)*Math.sin(Math.PI*j/180);
         neo.z =-o.z;
                  o.w =Math.cos(Math.PI*j/180);
         neo.w =o.w;
         for(int i = 0; i < 36; i++)
         {
            m = qm(o, p[i]);
            h = qm(m, neo);
            pdraw[i].x = (int)Math.round(h.x)+ 240;
            pdraw[i].y = (int)Math.round(h.y)+ 180;
         }
```

Program Code for Orthogonal Matrix Rotation Direction Cosine is X-Axis

```
public void rotateN(int dir)
         {
         int l=0,m=0,n=0;
         if(dir==1)
```

```
l=1;

else if(dir==2)

m=1;

else if(dir==3)

        n=1;

        for(int j =60; j <= 60; j+=60)

        {

        double angle=(j*(Math.PI/4));

        System.out.println("Theta   = "+j);

        double matrix_b[][]={

        {l*l*(1-Math.cos(angle))+Math.cos(angle),l*m*(1-Math.cos(angle))-
        n*Math.sin(angle),l*n*(1-Math.cos(angle))+m*Math.sin(angle),0},

        {m*l*(1-Math.cos(angle))+n*Math.sin(angle),m*m*(1-
        Math.cos(angle))+Math.cos(angle),m*n*(1-Math.cos(angle))-
        l*Math.sin(angle),0},

        {n*l*(1-Math.cos(angle))-m*Math.sin(angle),n*m*(1-
        Math.cos(angle))+l*Math.sin(angle),n*n*(1-
        Math.cos(angle))+Math.cos(angle),0},

        {0,0,0,1}

        };

         int s=0;

          for(int i = 0; i < 36; i++)

          {

                pt[0]=p[i].x;

                pt[1]=p[i].y;

                pt[2]=p[i].z;

                pt[3]=p[i].w;

                for(int ij=0;ij<4;ij++)
```

```
{
        s=0;
        for(int count_2=0;count_2<4;count_2++)
        {
            s+=(pt[count_2]*matrix_b[count_2][ij]);
        }
                    pt[ij]=s;
                }
                pdraw[i].x=(pt[0])+240;
                pdraw[i].y=(pt[1])+180;
                pdraw[i].z=pt[2]+400;
                pdraw[i].w=pt[3]+300;
            }
```

Program Code for Euler Rotation

```
for(int j = 60; j <= 60; j+=60)
{
        double angle=(j*(Math.PI/180));
        double angle1=((j+2)*(Math.PI/180));
        double angle2=((j+3)*(Math.PI/180));
        System.out.println("Theta   = "+j);
        double matrix_eul[][]={
    {
Math.cos(angle2)*Math.cos(angle1),Math.sin(angle)*Math.sin(angle2)*Math
.cos(angle1)-
Math.cos(angle)*Math.sin(angle1),Math.cos(angle)*Math.sin(angle2)*Math.
cos(angle1)+Math.sin(angle)*Math.sin(angle1),0 } ,
```

```
{
Math.cos(angle2)*Math.cos(angle1),Math.sin(angle)*Math.sin(angle2)*Math
.cos(angle1)+Math.cos(angle)*Math.sin(angle1),Math.cos(angle)*Math.sin(a
ngle2)*Math.cos(angle1)-Math.sin(angle)*Math.sin(angle1),0 } ,

{ -
(Math.sin(angle2)),Math.sin(angle)*Math.cos(angle2),Math.cos(angle)*Math.
cos(angle2),0 } ,

{ 0,0,0,1 }

                                                    };

        int s=0;

        for(int i = 0; i < 36; i++)

        {

                                pt[0]=p[i].x;

                                pt[1]=p[i].y;

                                pt[2]=p[i].z;

                                pt[3]=1;

                                for(int ij=0;ij<4;ij++)

                                {

                                        s=0;

                                for(int count_2=0;count_2<4;count_2++)

                                {

                                s+=(pt[count_2]*matrix_eul[count_2][ij]);

                                }

                                        pt[ij]=s;

                                }

                                pdraw[i].x=(pt[0])+240;

                                pdraw[i].y=(pt[1])+180;
```

```
                                        pdraw[i].z=pt[2];

                                    }
```

Program Code for Ellipse Rotation

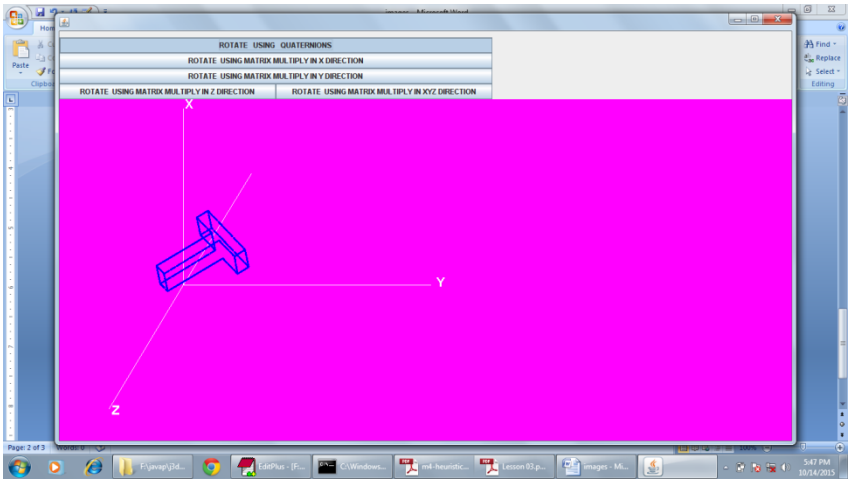
```
   for(int j = 0; j <=60; j+=60)
                    {
                            if(an>5)an=3;
                            o.x =(a*0.5)*Math.sin(Math.PI*(j)/180);
                            neo.x =-o.x;
                            o.y =-(a)*Math.sin(Math.PI*j/180);
                            neo.y =-o.y;


                                        o.z =(a/3)*Math.sin(Math.PI*j/180);
                                        neo.z =-o.z;
                                            o.w =Math.cos(Math.PI*j/180);
                                        neo.w =o.w;
        pdraw[3].sa+=(Math.abs(Math.cos(Math.PI*j/180)))+(Math.abs(Math.sin(Math.PI*j/
180)));
        pdraw[3].ea=(Math.abs(Math.sin(Math.PI*j/180)))+(Math.abs(Math.cos(Math.PI*j/1
80)));
        pdraw[4].sa+=Math.round(Math.cos(Math.PI*j/180));
        pdraw[4].ea+=Math.round(Math.sin(Math.PI*j/180));
        pdraw[5].sa+=Math.round(Math.cos(Math.PI*j/180));
        pdraw[5].ea+=Math.round(Math.sin(Math.PI*j/180));
        an++;
        for(int i = 0; i < 6; i++)
```

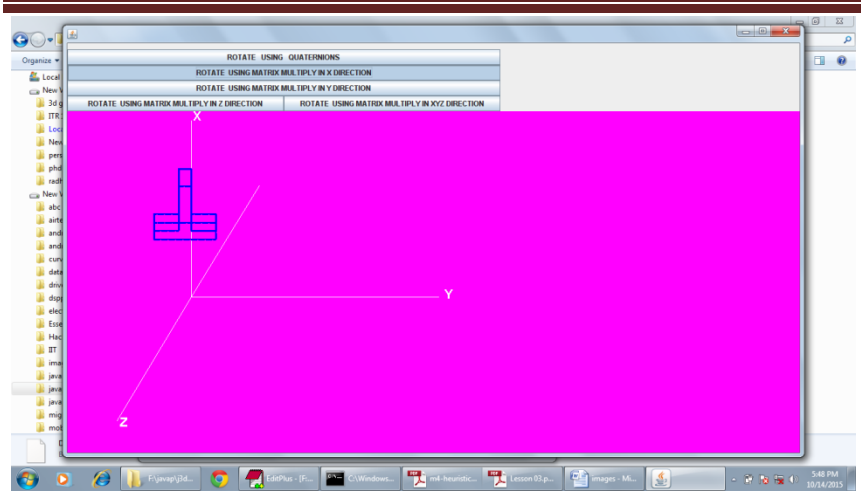
```
{
        m = qm(o, p[i]);
        h = qm(m, neo);
        pdraw[i].z = (int)Math.round(h.z)+ 400;
        pdraw[i].w = (int)Math.round(h.w)+ 300;
}
```

Images

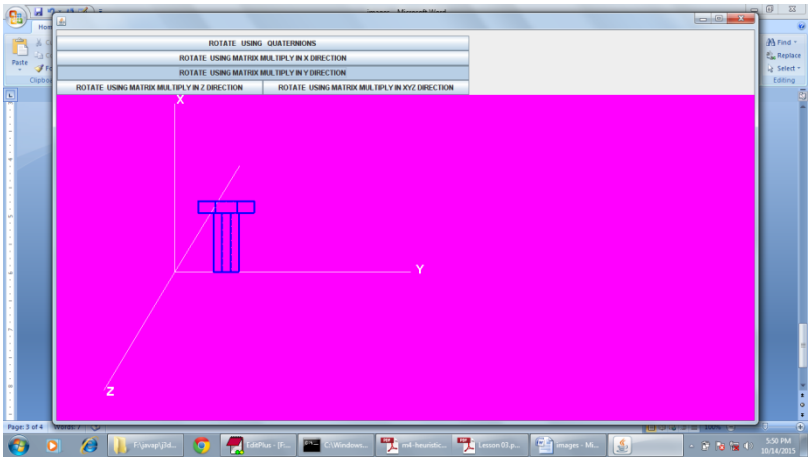
Rotation using Quaternion



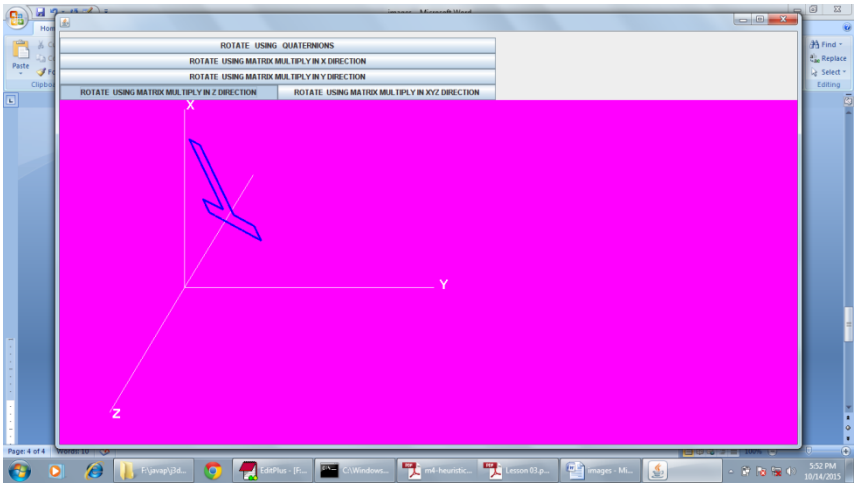
Rotation Through X axis



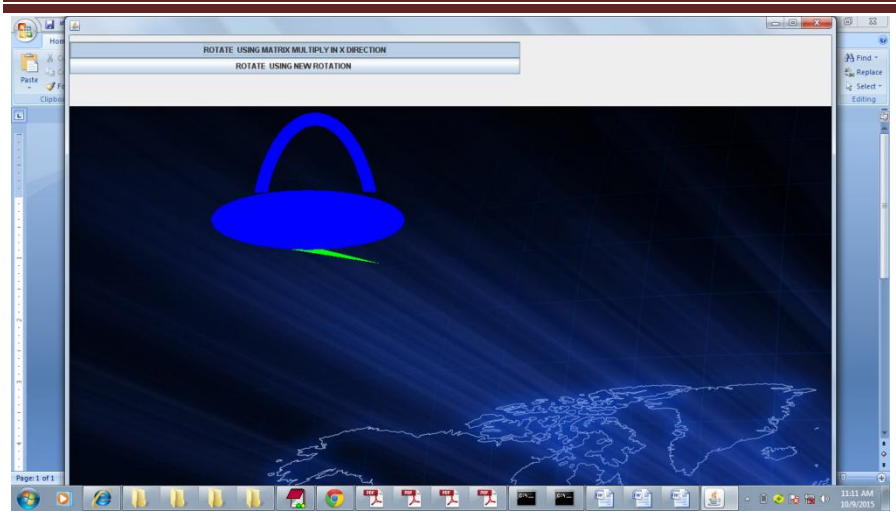
Rotation Through Y-Axis

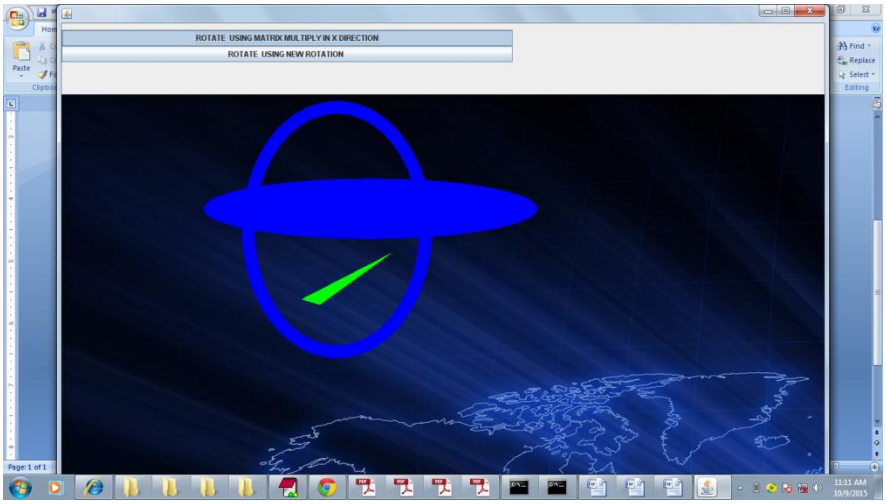


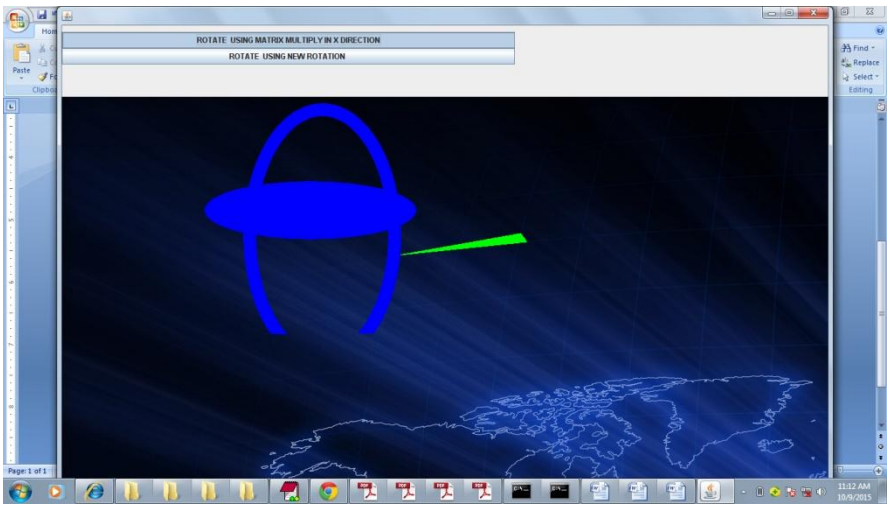
Rotation Through Z-Axis



Images Of Ellipse Shape after Rotation


ROTATE USING MATRIX MULTIPLY IN X DIRECTION
ROTATE USING NEW ROTATION


ROTATE USING MATRIX MULTIPLY IN X DIRECTION
ROTATE USING NEW ROTATION


ROTATE USING MATRIX MULTIPLY IN X DIRECTION
ROTATE USING NEW ROTATION

Results

Values of X and Y on different Rotation methods of Simple Image Drawn with Lines. Where Angle of Rotation Is 60^0

Quaternion Rotation Values:

X0 To X10	Y0 To Y10
240	180
264	215
274	221
298	256
264	215
245	228
221	194
240	180
221	194
234	210
258	245

Euler Rotation Values:

X0 To X10	Y0 To Y10
275	163
240	180
219	181
254	164
275	163
279	178
244	196
240	180
244	196
237	196
272	179

Orthogonal Rotation Values (Direction Cosine X-Axis for 60^0) :

X0 To X10	Y0 To Y10
240	180
240	180
140	180
140	180
240	180
240	160
240	160
240	180

240	160
200	160
200	160

Rotation Values of Matrix Multiplication Method of X-Axis:

X0 To X10	Y0 To Y10
240	214
240	180
140	180
140	214
240	214
240	224
240	190
240	180
240	190
200	190
200	224

Orthogonal Rotation Values (Direction Cosine Z-Axis for 60^0) :

X0 To X10	Y0 To Y10
240	180
240	180
340	180
340	180
240	180
240	160
240	160
240	180
240	160
280	160
280	160

Rotation Values of Matrix Multiplication Method of Z-Axis Where Angle is 60^0:

X0 To X10	Y0 To Y10
240	180
240	180
190	137
190	137
240	180
223	177

223	177
240	180
223	177
203	158
203	158

Orthogonal Rotation Values (Direction Cosine X-Axis for 60^0) :

X0 To X10	Y0 To Y10
240	180
240	180
140	180
140	180
240	180
240	160
240	160
240	180
240	160
200	160
200	160

Rotation Values of Elliptical Arc :

X0 To X10	Y0 To Y10
290	150
211	383
220	130
290	20
214	383
290	150
211	383
220	130
159	182
290	20
214	383

Conclusion and Summary:

In this paper we have perform the geometric rotations of two different types of objects. One is ‘T’ shaped while the other is an elliptical object. We have noticed that rotations by the use

of Quaternions method give the better result than the results obtained orthogonal method and rotation by matrix multiplication method.

We have authenticated our results by Java using Swing programming as well.

Bibliography

1) Donald Hearn and M. Pauline Baker (2006) Computer graphics C version second Edition Pearson Education
2) Harrington ,S. (1983) Computer Graphics- A Programming Approach , McGraw-Hill
3) Rogers, D.F. (2001) Procedural Elements for Computer Graphics Tata McGraw-Hill
4) Rogers, D.F and Adams ,J.A (1990) Mathematical Elements for Computer Graphics McGraw-Hill
5) Simon L. Altmann (2005) Rotations, Quaternions and Double Groups first edition Dover Publications, New York
6) Unleashed (1999) Java 2 Techmedia