Xilinx Zynq FPGA, TI DSP, MCU 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

Quaternion Mathematics

Quaternions

과거 3D 게임 프로그래밍 교육을 진행하면서 교육에 사용했던 쿼터니온 자료다.
연산 속도 측면에서 이득을 볼 수 있기 때문에 일부 내용을 임베디드 과정에서도 사용하도록 한다.

사원수는 하나의 스칼라와 하나의 3차원 Vector를 하나로 묶어서 4개의 요소로 구성한 복소수의 일종이다.

$$\hat{q} = w + ai + bj + ck = S_q + V_q$$
$$= (S_q, V_q)$$
$$= (w, a, b, c)$$

$\hat{p} = d_p + a_p i + b_p j + c_p k$와
$\hat{q} = d_q + a_q i + b_q j + c_q k$에 대한 덧셈은 아래와 같다.

$$
\begin{aligned}
\hat{p} + \hat{q} &= (S_p, V_p) + (S_q, V_q) \\
&= (S_p + S_q, V_p + V_q) \\
&= (d_p + d_q) + (a_p + a_q)i + (b_p + b_q)j + (c_p + c_q)k
\end{aligned}
$$

뺄샘은 부호만 바뀌면 되며,
스칼라 곱은 모든 항이 스칼라 값만큼의 배수가 되고,
이들의 곱셈은 복소수의 성질을 이용하여 수행됨

$$
\begin{aligned}
\hat{p}\,\hat{q} &= (d_p + a_p i + b_p j + c_p k)(d_q + a_q i + b_q j + c_q k) \\
&= (d_p d_q - a_p a_q - b_p b_q - c_p c_q) \\
&\quad + d_p(a_q i + b_q j + c_q k) + d_q(a_p i + b_p j + c_p k) \\
&\quad + \{a_p i(b_q j + c_q k) + b_p j(a_q i + c_q k) + c_p k(a_q i + b_q j)\}
\end{aligned}
$$

$$
\begin{aligned}
\hat{p}\,\hat{q} &= (d_p + a_p i + b_p j + c_p k)(d_q + a_q i + b_q j + c_q k) \\
&= (d_p d_q - a_p a_q - b_p b_q - c_p c_q) \\
&\quad + d_p(a_q i + b_q j + c_q k) + d_q(a_p i + b_p j + c_p k) \\
&\quad + \{a_p i(b_q j + c_q k) + b_p j(a_q i + c_q k) + c_p k(a_q i + b_q j)\}
\end{aligned}
$$

수식의 4번째 줄의 Vector 부분에 대한 전개가 남음

$ij = k,\ jk = i,\ ki = j$

$ji = -k,\ kj = -i,\ ik = -j$

$i^2 = -1,\ j^2 = -1,\ k^2 = -1$

최종적으로 아래와 같아짐

$$
\begin{aligned}
\hat{p}\,\hat{q} &= (d_p + a_p i + b_p j + c_p k)(d_q + a_q i + b_q j + c_q k) \\
&= (d_p d_q - a_p a_q - b_p b_q - c_p c_q) \\
&\quad + d_p(a_q i + b_q j + c_q k) + d_q(a_p i + b_p j + c_p k) \\
&\quad + \{(a_p b_q - c_p b_q)i + (c_p a_q - a_p c_q)j + (a_p b_q - b_p a_q)k\}
\end{aligned}
$$

$$\hat{p}\,\hat{q} = (d_p + a_p i + b_p j + c_p k)(d_q + a_q i + b_q j + c_q k)$$
$$= (d_p d_q - a_p a_q - b_p b_q - c_p c_q)$$
$$+ d_p(a_q i + b_q j + c_q k) + d_q(a_p i + b_p j + c_p k)$$
$$+ \{(a_p b_q - c_p b_q)i + (c_p a_q - a_p c_q)j + (a_p b_q - b_p a_q)k\}$$

최종적으로 구했던 식은 위와 같은데

$$\hat{p}\,\hat{q} = (S_p S_q - V_p . V_q,\ S_q V_p + S_p V_q + V_p \text{ x } V_q)$$

여기서 Scalar 부분은 $S_p = d_p$, $S_q = d_q$이며,
Vector 부분은 $V_p = a_p i + b_p j + c_p k$, $V_q = a_q i + b_q j + c_q k$

A Unit Quaternions

단위 사원수란 아래와 같다.

$$q = \cos\frac{\theta}{2} + e \sin\frac{\theta}{2}$$

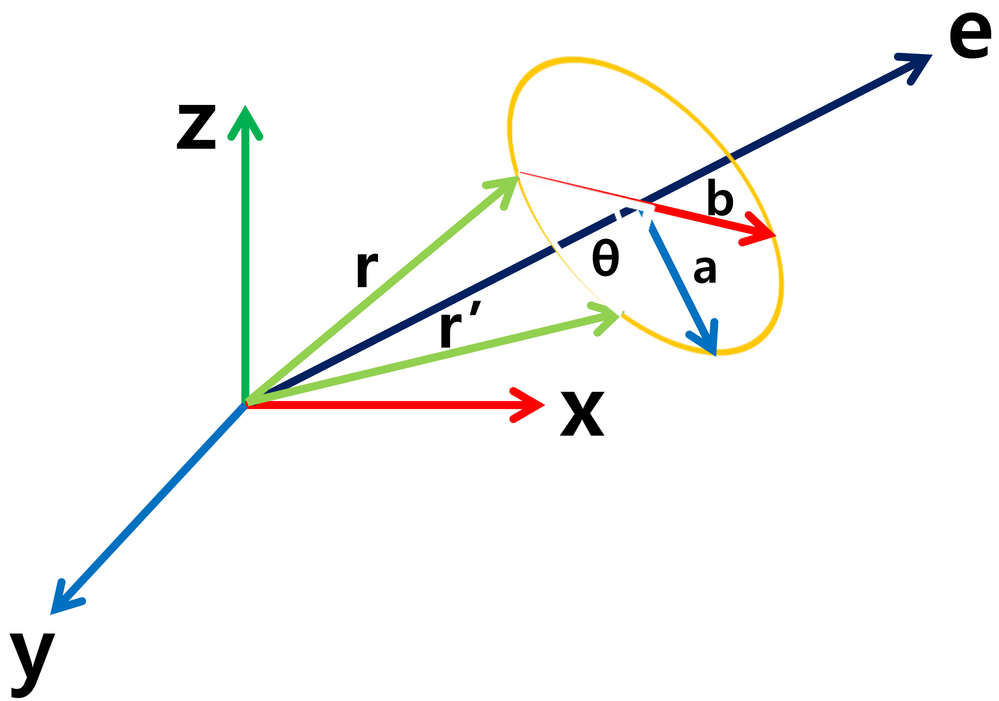
공간상의 임의의 회전축을 표현하는 회전축의 방향 Vector는 단위 Vector로 지정해야 단위 사원수가 된다.

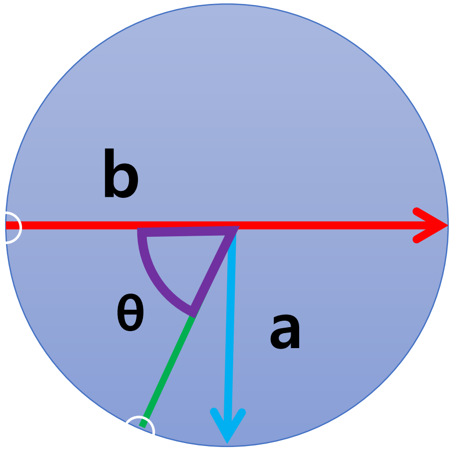
$e = (e_x, e_y, e_z)$로 구성된 단위 Vector일 때, 단위 벡터는 아래와 같다.

$$q = q_0 + q_1 i + q_2 j + q_3 k = \cos\frac{\theta}{2} + e_x \sin\frac{\theta}{2} i + e_y \sin\frac{\theta}{2} j + e_z \sin\frac{\theta}{2} k$$

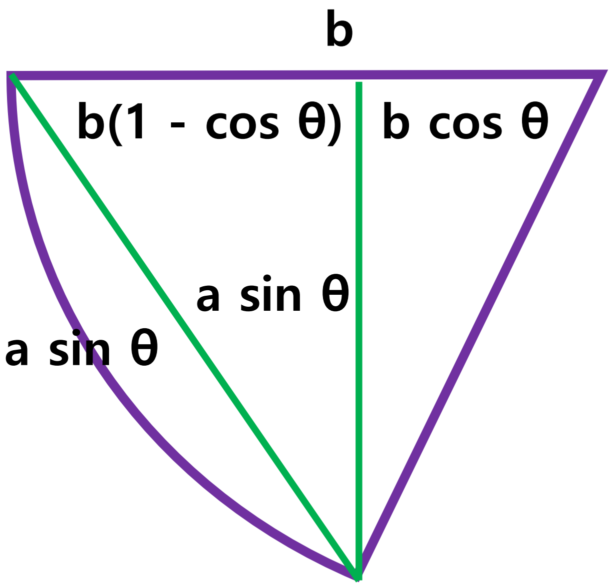
Matrix Expression of Quaternions

하나의 Vector를 임의의 회전축에 대해 회전시키는
공간 회전을 시작으로 사원수 변환의 행렬 표현을 알아보자!





Zoom!



$$r' = r + b(1 - \cos\theta) + a\sin\theta$$

$$a = e \times r, \qquad b = e(e \times r)$$

$$r' = r + (1 - \cos\theta)e \times (e \times r) + (e \times r)\sin\theta$$

$$\sin^2\frac{\theta}{2} = \frac{1 - \cos\theta}{2} \qquad \sin\theta = 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}$$

$$r' = r + (2\sin^2\frac{\theta}{2})\{e \times (e \times r)\} + 2\sin\frac{\theta}{2}\cos\frac{\theta}{2}(e \times r)$$

$$r' = r + 2(\sin\frac{\theta}{2}e) \times (\sin\frac{\theta}{2}e \times r) + (2\cos\frac{\theta}{2})(\sin\frac{\theta}{2}(e \times r))$$

$$r' = r + \left(2\sin^2 \frac{\theta}{2}\right)\{e \times (e \times r)\} + 2\sin \frac{\theta}{2} \cos \frac{\theta}{2} (e \times r)$$

$$r' = r + 2\left(\sin \frac{\theta}{2} e\right) \times \left(\sin \frac{\theta}{2} e \times r\right) + \left(2\cos \frac{\theta}{2}\right)\left(\sin \frac{\theta}{2} (e \times r)\right)$$

$$q = \cos \frac{\theta}{2} + e \sin \frac{\theta}{2} = S_q + V_q$$

$$r' = r + 2V_q \times V_q \times r + 2S_q V_q \times r$$

$$V_q \times = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} = Q$$

$$r' = (I + 2QQ + 2S_q Q)r$$

$$A = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_3 q_2 + q_0 q_1) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix}$$

Example

```
#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>

#define bool int
#define false 0
#define true 1
#define M_PI 3.14159

int winWidth, winHeight;
float angle = 0.0, axis[3], trans[3];
bool trackingMouse = false;
bool redrawContinue = false;
bool trackballMove = false;

GLfloat vertices[][3] = {
        {-1.0,-1.0,-1.0},
        {1.0,-1.0,-1.0},
        {1.0,1.0,-1.0},
        {-1.0,1.0,-1.0},
        {-1.0,-1.0,1.0},
        {1.0,-1.0,1.0},
        {1.0,1.0,1.0},
        {-1.0,1.0,1.0}
};
```

```
GLfloat colors[][3] = {
        {0.0,0.0,0.0},
        {1.0,0.0,0.0},
        {1.0,1.0,0.0},
        {0.0,1.0,0.0},
        {0.0,0.0,1.0},
        {1.0,0.0,1.0},
        {1.0,1.0,1.0},
        {0.0,1.0,1.0}
};

void polygon(int a, int b, int c , int d, int face)
{
        glBegin(GL_POLYGON);
        glColor3fv(colors[a]);
        glVertex3fv(vertices[a]);
        glColor3fv(colors[b]);
        glVertex3fv(vertices[b]);
        glColor3fv(colors[c]);
        glVertex3fv(vertices[c]);
        glColor3fv(colors[d]);
        glVertex3fv(vertices[d]);
        glEnd();
}
```

```
void colorcube(void)
{
		polygon(1,0,3,2,0);
		polygon(3,7,6,2,1);
		polygon(7,3,0,4,2);
		polygon(2,6,5,1,3);
		polygon(4,5,6,7,4);
		polygon(5,4,0,1,5);
}

float lastPos[3] = {0.0F, 0.0F, 0.0F};
int curx, cury;
int startX, startY;

void trackball_ptov(int x, int y, int width, int height, float v[3])
{
		float d, a;
		v[0] = (2.0F*x - width) / width;
		v[1] = (height - 2.0F*y) / height;
		d = (float) sqrt(v[0]*v[0] + v[1]*v[1]);
		v[2] = (float) cos((M_PI/2.0F) * ((d < 1.0F) ? d : 1.0F));
		a = 1.0F / (float) sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
		v[0] *= a;
		v[1] *= a;
		v[2] *= a;
}
```

```
void mouseMotion(int x, int y)
{
        float curPos[3], dx, dy, dz;
        trackball_ptov(x, y, winWidth, winHeight, curPos);
        if(trackingMouse)
        {
                dx = curPos[0] - lastPos[0];
                dy = curPos[1] - lastPos[1];
                dz = curPos[2] - lastPos[2];
                if (dx || dy || dz)
                {
                        angle = 90.0F * sqrt(dx*dx + dy*dy + dz*dz);
                        axis[0] = lastPos[1]*curPos[2] - lastPos[2]*curPos[1];
                        axis[1] = lastPos[2]*curPos[0] - lastPos[0]*curPos[2];
                        axis[2] = lastPos[0]*curPos[1] - lastPos[1]*curPos[0];
                        lastPos[0] = curPos[0];
                        lastPos[1] = curPos[1];
                        lastPos[2] = curPos[2];
                }
        }
        glutPostRedisplay();
}
```

```
void startMotion(int x, int y)
{
	trackingMouse = true;
	redrawContinue = false;
	startX = x;
	startY = y;
	curx = x;
	cury = y;
	trackball_ptov(x, y, winWidth, winHeight, lastPos);
	trackballMove=true;
}

void stopMotion(int x, int y)
{
	trackingMouse = false;
	if (startX != x || startY != y)
	{
		redrawContinue = true;
	}
	else
	{
		angle = 0.0F;
		redrawContinue = false;
		trackballMove = false;
	}
}
```

```
void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        if (trackballMove)
        {
                glRotatef(angle, axis[0], axis[1], axis[2]);
        }
        colorcube();
        glutSwapBuffers();
}

void mouseButton(int button, int state, int x, int y)
{
        if(button==GLUT_RIGHT_BUTTON)
                exit(0);
        if(button==GLUT_LEFT_BUTTON)
                switch(state)
                {
                        case GLUT_DOWN:
                                y=winHeight-y; startMotion( x,y);
                                break;
                        case GLUT_UP:
                                stopMotion( x,y);
                                break;
                }
}
```

```
void myReshape(int w, int h)
{
		glViewport(0, 0, w, h);
		winWidth = w;
		winHeight = h;
}

void spinCube()
{
		if (redrawContinue)
				glutPostRedisplay();
}
```

```
int main(int argc, char **argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(500, 500);
        glutCreateWindow("colorcube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutIdleFunc(spinCube);
        glutMouseFunc(mouseButton);
        glutMotionFunc(mouseMotion);
        glEnable(GL_DEPTH_TEST);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-2.0, 2.0, -2.0, 2.0, -2.0, 2.0);
        glMatrixMode(GL_MODELVIEW);
        glutMainLoop();
        return 0;
}
```