

```

/* IT Text CG, Chapter 8, Exercise 4, B Spline Surface */
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>

```

```

void DisplayBSplineSurface(void);
float BN(int j, int m, float t);
void VectorProduct(GLfloat p1x,GLfloat p1y,GLfloat p1z,
                   GLfloat p2x,GLfloat p2y,GLfloat p2z,
                   GLfloat p3x,GLfloat p3y,GLfloat p3z,
                   GLfloat *nx,GLfloat *ny,GLfloat *nz);

```

```

/* Bスプライン曲面の制御点 */

```

```

GLfloat controlpoints[4][4][3] = {
    {
        {0.0, 0.0, 120.0},
        {0.0, 40.0, 80.0},
        {0.0, 40.0, 40.0},
        {0.0, 0.0, 0.0}
    },
    {
        {40.0, 40.0, 120.0},
        {40.0, 80.0, 80.0},
        {40.0, 80.0, 40.0},
        {40.0, 40.0, 0.0}
    },
    {
        {80.0, 40.0, 120.0},
        {80.0, 80.0, 80.0},
        {80.0, 80.0, 40.0},
        {80.0, 40.0, 0.0}
    },
    {
        {120.0, 0.0, 120.0},
        {120.0, 40.0, 80.0},
        {120.0, 40.0, 40.0},
        {120.0, 0.0, 0.0}
    }
};

```

```

/* Bスプライン曲面のノットベクトル */

```

```

/* T = [t0 t1 ... tn+M] = [-(M-1) -(M-2) ... n+1] */

```

```

float knotvec[8] = {-3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0};

```

```

void display(void)

```

```

{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-50.0, 50.0, -50.0, 50.0, 50.0, 1000.0);
    gluLookAt(130.0, 100.0, 130.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    DisplayBSplineSurface();

    glFlush();
}

```

/* Bスプライン曲面を計算し表示する関数 */

```

void DisplayBSplineSurface(void)
{
    int    i1, i2, j1, j2;
    float  px01,py01,pz01,px02,py02,pz02,px03,py03,pz03;
    float  px11,py11,pz11,px12,py12,pz12,px13,py13,pz13;
    float  u1, v1, pu1, pv1;
    float  nx1,ny1,nz1;

    pu1 = 1.0/30.0;
    pv1 = 1.0/30.0;
    u1 = 0.0;
    for(i1 = 0; i1 <= 30; ++i1)
    {
        v1 = 0.0;
        for(j1 = 0; j1 <= 30; ++j1)
        {
            px01 = 0.0;
            py01 = 0.0;
            pz01 = 0.0;

```

```
px02 = 0.0;  
py02 = 0.0;  
pz02 = 0.0;
```

```
px03 = 0.0;  
py03 = 0.0;  
pz03 = 0.0;
```

```
px11 = 0.0;  
py11 = 0.0;  
pz11 = 0.0;
```

```
px12 = 0.0;  
py12 = 0.0;  
pz12 = 0.0;
```

```
px13 = 0.0;  
py13 = 0.0;  
pz13 = 0.0;
```

```
/* Bスプライン曲面の計算 */
```

```
for(i2 = 0; i2 <= 3; ++i2)  
{
```

```
    for(j2 = 0; j2 <= 3; ++j2)  
    {
```

```
        /* triangle #0 */
```

```
        px01 = px01 + BN(i2, 4, u1) * BN(j2, 4, v1) *  
            controlpoints[i2][j2][0];
```

```
        py01 = py01 + BN(i2, 4, u1) * BN(j2, 4, v1) *  
            controlpoints[i2][j2][1];
```

```
        pz01 = pz01 + BN(i2, 4, u1) * BN(j2, 4, v1) *  
            controlpoints[i2][j2][2];
```

```
        px02 = px02 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *  
            controlpoints[i2][j2][0];
```

```
        py02 = py02 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *
```

```

        controlpoints[i2][j2][1];
    pz02 = pz02 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][2];

    px03 = px03 + BN(i2, 4, u1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][0];
    py03 = py03 + BN(i2, 4, u1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][1];
    pz03 = pz03 + BN(i2, 4, u1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][2];

    /* triangle #1 */
    px11 = px11 + BN(i2, 4, u1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][0];
    py11 = py11 + BN(i2, 4, u1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][1];
    pz11 = pz11 + BN(i2, 4, u1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][2];

    px12 = px12 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][0];
    py12 = py12 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][1];
    pz12 = pz12 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1) *
        controlpoints[i2][j2][2];

    px13 = px13 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][0];
    py13 = py13 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][1];
    pz13 = pz13 + BN(i2, 4, u1+pu1) * BN(j2, 4, v1+pv1) *
        controlpoints[i2][j2][2];

} /* for j2 */

} /* for i2 */

```

```

/* Bスプライン曲面 (三角面) の表示 */
glBegin(GL_POLYGON);
    VectorProduct(px01,py01,pz01,px02,py02,pz02,
                  px03,py03,pz03,&nx1,&ny1,&nz1);
    glNormal3f(nx1, ny1, nz1);
    glVertex3f(px01, py01, pz01);
    glVertex3f(px02, py02, pz02);
    glVertex3f(px03, py03, pz03);
glEnd();

/* Bスプライン曲面 (三角面) の表示 */
glBegin(GL_POLYGON);
    VectorProduct(px11,py11,pz11,px12,py12,pz12,
                  px13,py13,pz13,&nx1,&ny1,&nz1);
    glNormal3f(nx1, ny1, nz1);
    glVertex3f(px11, py11, pz11);
    glVertex3f(px12, py12, pz12);
    glVertex3f(px13, py13, pz13);
glEnd();

v1 = v1 + pv1;

} /* for j1 */

u1 = u1 + pul;

} /* for i1 */

}

/* Bスプライン関数 $N_{j,M}(t)$ の計算をする関数 */
float BN(int j, int m, float t)
{
    float s1, r1, r2, r3, r4, r12, r34;

```

```

if(m == 1)
{
    if(t >= knotvec[j] && t < knotvec[j+1])
    {
        s1 = 1.0;

    }
    else
    {
        s1 = 0.0;

    } /* if */

    return s1;

} /* if */

```

```

if(m > 1)
{
    r1 = (t - knotvec[j]);
    r2 = (knotvec[j+m-1] - knotvec[j]);
    r3 = (knotvec[j+m] - t);
    r4 = (knotvec[j+m] - knotvec[j+1]);

    if(r1 == 0.0 && r2 == 0.0)
    {
        r12 = 0.0;

    }
    else
    {

        if(r2 == 0.0)
        {
            r2 = r2 + 0.0000001;

```



```

        GLfloat *nx, GLfloat *ny, GLfloat *nz)
{
    GLfloat v1x, v1y, v1z, v2x, v2y, v2z;
    GLfloat s1, sx1, sy1, sz1;

    v1x = p2x - p1x;
    v1y = p2y - p1y;
    v1z = p2z - p1z;
    v2x = p3x - p2x;
    v2y = p3y - p2y;
    v2z = p3z - p2z;

    sx1 = v1y * v2z - v1z * v2y;
    sy1 = v1z * v2x - v1x * v2z;
    sz1 = v1x * v2y - v1y * v2x;

    s1 = sqrt(sx1 * sx1 + sy1 * sy1 + sz1 * sz1);

    if(s1 == 0.0)
    {
        s1 = 0.000000001;
    } /* if */

    *nx = sx1/s1;
    *ny = sy1/s1;
    *nz = sz1/s1;

    return;
}

/* 材質と照明を設定する関数 */
void LightSource()
{
    GLfloat mat_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };

```



```

GLfloat mat_specular[] = { 0.3, 0.3, 0.3, 1.0 };
GLfloat mat_shininess[] = { 6.0 };
GLfloat light_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat light_specular[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat light_ambient[] = { 0.3, 0.3, 0.3, 1.0 };
GLfloat light_position[] = { 0.5, 0.5, 0.5, 0.0 };

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_DEPTH_TEST);

}

void keyboard(unsigned char key, int x, int y)
{
    switch(key) {
        case 27: exit(0); break;

    }
}

int main(int argc, char** argv)
{
    glutInitWindowSize(1000,1000);

```

```
glutInitWindowPosition(0,0);  
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);  
glutCreateWindow("BSplineSurface");  
  
glClearColor(0.0, 0.0, 0.0, 0.0);  
  
LightSource();  
  
glutDisplayFunc(display);  
glutKeyboardFunc(keyboard);  
glutMainLoop();  
  
}
```