

```

/* ITTextCG, Chapter 7, Exercise 4, Triangle Mesh */
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>

void MakeMeshData(void);
void MakePolygonalMesh(void);
void DisplayPolygonalMesh(void);
void VectorProduct(GLfloat p1x,GLfloat ply,GLfloat plz,
                   GLfloat p2x,GLfloat p2y,GLfloat p2z,
                   GLfloat p3x,GLfloat p3y,GLfloat p3z,
                   GLfloat *nx,GLfloat *ny,GLfloat *nz);

/* 頂点ブロックのための構造体 */
struct Vertex_block
{
    GLfloat x;
    GLfloat y;
    GLfloat z;
};

/* 稜線ブロックのための構造体 */
struct Edge_block
{
    struct Vertex_block *start_point;
    struct Vertex_block *end_point;
    struct Edge_block *edge_next;
};

```

```

/* 多角形ブロックのための構造体 */
struct Polygon_block
{
    struct Edge_block *edge_list;
    struct Polygon_block *polygon_next;

};

/* ポリゴンメッシュへのポインター */
static struct Polygon_block *polygonal_mesh;
static struct Vertex_block vertices[25];
static int edge_vertex[32][3][2];

/* ポリゴンメッシュのためのデータを作成する関数 */
void MakeMeshData(void)
{
    float x1, y1, z1, pitchx1, pitchz1;
    int i1, i2, i3;
    int v10, v11, v12, v20, v21, v22;

    pitchx1 = 30.0;
    pitchz1 = 30.0;
    x1 = 0.0;
    y1 = 0.0;

    i3 = 0;
    for(i1 = 1; i1 <= 5; ++i1)
    {
        z1 = pitchz1 * 4.0;
        for(i2 = 1; i2 <= 5; ++i2)
        {

```

```

vertices[i3].x = x1;
vertices[i3].y = y1;
vertices[i3].z = z1;

z1 = z1 - pitchz1;
i3 = i3 + 1;

} /* for i2 */

x1 = x1 + pitchx1;

} /* for i1 */

v10 = 0;
v11 = 5;
v12 = 6;
v20 = 0;
v21 = 6;
v22 = 1;

i3 = 0;
for(i1 = 1; i1 <= 4; ++i1)
{
    for(i2 = 1; i2 <= 4; ++i2)
    {
        edge_vertex[i3][0][0] = v10;
        edge_vertex[i3][0][1] = v11;
        edge_vertex[i3][1][0] = v11;
        edge_vertex[i3][1][1] = v12;
        edge_vertex[i3][2][0] = v12;
        edge_vertex[i3][2][1] = v10;
    }
}

```

```
v10 = v10 + 1;  
v11 = v11 + 1;  
v12 = v12 + 1;
```

```
i3 = i3 + 1;
```

```
edge_vertex[i3][0][0] = v20;  
edge_vertex[i3][0][1] = v21;  
edge_vertex[i3][1][0] = v21;  
edge_vertex[i3][1][1] = v22;  
edge_vertex[i3][2][0] = v22;  
edge_vertex[i3][2][1] = v20;
```

```
v20 = v20 + 1;  
v21 = v21 + 1;  
v22 = v22 + 1;
```

```
i3 = i3 + 1;
```

```
} /* for i2 */
```

```
v10 = v10 + 1;  
v11 = v11 + 1;  
v12 = v12 + 1;
```

```
v20 = v20 + 1;  
v21 = v21 + 1;  
v22 = v22 + 1;
```

```
} /* for i1 */
```

```
vertices[12].y = 40.0;
```

```

    return;

}

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(200.0, 80.0, 200.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    DisplayPolygonalMesh();

    glFlush();

}

/* ポリゴンメッシュのデータ構造を作成する関数 */
void MakePolygonalMesh(void)
{
    struct Vertex_block *ppv;
    struct Edge_block *ppe1, *ppe2;

```

```

struct Polygon_block *ppp1, *ppp2;
int    i, j;
char   *malloc();

for(i = 31; i >= 0; --i)
{
    for(j = 2; j >= 0; --j)
    {
        ppe1 = (struct Edge_block *)malloc
                (sizeof(struct Edge_block));

        ppe1 ->start_point =
            &vertices[(int)edge_vertex[i][j][0]];
        ppe1 ->end_point =
            &vertices[(int)edge_vertex[i][j][1]];

        if(j == 2)
        {
            ppe1->edge_next = NULL;

        }
        else
        {
            ppe1->edge_next = ppe2;

        } /* if */
        ppe2 = ppe1;

    } /* for j */

    ppp1 = (struct Polygon_block *)malloc
            (sizeof(struct Polygon_block));

```

```

    ppp1->edge_list = ppe1;

    if(i == 31)
    {
        ppp1->polygon_next = NULL;

    }
    else
    {
        ppp1->polygon_next = ppp2;

    } /* if */
    ppp2 = ppp1;

} /* for i */

/* ポリゴンメッシュへのポインター */
polygonal_mesh = ppp1;

}

/* ポリゴンメッシュのデータ構造を表示する関数 */
void DisplayPolygonalMesh(void)

{
    struct Vertex_block *ppv;
    struct Edge_block *ppe1, *ppe2;
    struct Polygon_block *ppp1, *ppp2;
    GLfloat x1, y1, z1;
    int     i1;

```

```
float px1,py1,pz1,px2,py2,pz2,px3,py3,pz3;  
float nx1,ny1,nz1,pax1[3],pay1[3],paz1[3];
```

```
ppp1 = polygonal_mesh;
```

```
while(1)  
{  
    ppe1 = ppp1->edge_list;  
  
    i1 = 0;  
    while(1)  
    {  
        ppv = ppe1->start_point;  
        x1 = ppv->x;  
        y1 = ppv->y;  
        z1 = ppv->z;  
        /* glVertex3f(x1, y1, z1); */  
  
        pax1[i1] = x1;  
        pay1[i1] = y1;  
        paz1[i1] = z1;  
        i1 = i1 + 1;  
  
        if(ppe1->edge_next ==NULL)  
        {  
            break;  
        }  
        else  
        {  
            ppe1 = ppe1->edge_next;
```



```

        } /* if */

    } /* while */

    /* 多角形の表示 */
    glBegin(GL_LINE_LOOP);

    px1 = pax1[0];
    py1 = pay1[0];
    pz1 = paz1[0];
    px2 = pax1[1];
    py2 = pay1[1];
    pz2 = paz1[1];
    px3 = pax1[2];
    py3 = pay1[2];
    pz3 = paz1[2];

    VectorProduct(px1,py1,pz1,px2,py2,pz2,px3,py3,pz3,
                  &nx1,&ny1,&nz1);
    glNormal3f(nx1, ny1, nz1);
    glVertex3f(px1, py1, pz1);
    glVertex3f(px2, py2, pz2);
    glVertex3f(px3, py3, pz3);

    glEnd();

    i1 = 0;

    if(ppp1->polygon_next == NULL)
    {
        break;
    }

```

```

    }
    else
    {
        ppp1 = ppp1->polygon_next;

    } /* if */

} /* while */

}

/* 法線ベクトルのためのベクトルの外積を計算する関数 */
void VectorProduct(GLfloat p1x,GLfloat ply,GLfloat plz,
                   GLfloat p2x,GLfloat p2y,GLfloat p2z,
                   GLfloat p3x,GLfloat p3y,GLfloat p3z,
                   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[] = { 0.0, 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);

}

/* キーボードのエスケープ・キーによりプログラムを終了する関数 */
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("TriangleMesh");

```

```
glClearColor(0.0, 0.0, 0.0, 0.0);

LightSource();

MakeMeshData();
MakePolygonalMesh();

glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
glutMainLoop();

}
```