

```

/* ITTextCG, Chapter 7, Exercise 5, Intersection Line */
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>

void MakeMeshData(void);
void MakePolygonalMesh(void);
void DisplayPolygonalMesh(void);
void VectorProduct(GLfloat plx,GLfloat ply,GLfloat plz,
                   GLfloat p2x,GLfloat p2y,GLfloat p2z,
                   GLfloat p3x,GLfloat p3y,GLfloat p3z,
                   GLfloat *nx,GLfloat *ny,GLfloat *nz);
void MinMaxTest(GLfloat plx,GLfloat ply,GLfloat plz,
                GLfloat p2x,GLfloat p2y,GLfloat p2z,
                GLfloat p3x,GLfloat p3y,GLfloat p3z,
                GLfloat q1x,GLfloat q1y,GLfloat q1z,
                GLfloat q2x,GLfloat q2y,GLfloat q2z,
                GLfloat q3x,GLfloat q3y,GLfloat q3z,int *flag);
double DetermS(double xa, double ya, double za, double wa,
               double x0, double y0, double z0, double w0,
               double x1, double y1, double z1, double w1,
               double x2, double y2, double z2, double w2);
void GetIntLine(GLfloat plx,GLfloat ply,GLfloat plz,
                GLfloat p2x,GLfloat p2y,GLfloat p2z,
                GLfloat p3x,GLfloat p3y,GLfloat p3z,
                GLfloat q1x,GLfloat q1y,GLfloat q1z,
                GLfloat q2x,GLfloat q2y,GLfloat q2z,
                GLfloat q3x,GLfloat q3y,GLfloat q3z,
                int *IntLineFlag,
                GLfloat *r1x,GLfloat *r1y,GLfloat *r1z,
                GLfloat *r2x,GLfloat *r2y,GLfloat *r2z);

```

```
/* 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];
```

```
GLfloat Tr1x = 50.0, Tr1y = -30.0, Tr1z = 70.0,
```

```
Tr2x = 80.0, Tr2y = 80.0, Tr2z = 70.0,  
Tr3x = 20.0, Tr3y = 80.0, Tr3z = 70.0;
```

```
/* 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[6].y = 20.0;
vertices[12].y = 40.0;
vertices[16].y = 30.0;
vertices[18].y = 30.0;

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, minmaxflag1, intlineflag1;
    GLfloat px1,py1,pz1,px2,py2,pz2,px3,py3,pz3;
    GLfloat ispx1, ispy1, ispz1, ispx2, ispy2, ispz2;
    GLfloat 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 */

```

```

glColor3f(1.0, 1.0, 1.0);
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();

/* 三角形の表示 */
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_LOOP);
    VectorProduct(Tr1x, Tr1y, Tr1z, Tr2x, Tr2y, Tr2z,
                  Tr3x, Tr3y, Tr3z, &nx1,&ny1,&nz1);
    glNormal3f(nx1, ny1, nz1);
    glVertex3f(Tr1x, Tr1y, Tr1z);
    glVertex3f(Tr2x, Tr2y, Tr2z);
    glVertex3f(Tr3x, Tr3y, Tr3z);
glEnd();

```

```

/* ミニ・マックステスト */
MinMaxTest(px1, py1, pz1, px2, py2, pz2, px3, py3, pz3,
           Tr1x, Tr1y, Tr1z, Tr2x, Tr2y, Tr2z,
           Tr3x, Tr3y, Tr3z, &minmaxflag1);

if(minmaxflag1 != -1)
{
    /* 多角形 (三角形) と三角形の交線の計算 */
    GetIntLine(px1, py1, pz1, px2, py2, pz2, px3, py3, pz3,
              Tr1x, Tr1y, Tr1z, Tr2x, Tr2y, Tr2z,
              Tr3x, Tr3y, Tr3z, &intlineflag1,
              &ispx1, &ispy1, &ispz1,
              &ispx2, &ispy2, &ispz2);

    if(intlineflag1 == 1)
    {
        /* 多角形 (三角形) と三角形の交線の表示 */
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINES);
            glVertex3f(ispx1, ispy1, ispz1);
            glVertex3f(ispx2, ispy2, ispz2);
        glEnd();

    } /* if */

} /* if */

il = 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;

}

/* 多角形のミニ・マックステストを行う関数 */
void MinMaxTest(GLfloat p1x,GLfloat p1y,GLfloat p1z,
                GLfloat p2x,GLfloat p2y,GLfloat p2z,
                GLfloat p3x,GLfloat p3y,GLfloat p3z,
                GLfloat q1x,GLfloat q1y,GLfloat q1z,
                GLfloat q2x,GLfloat q2y,GLfloat q2z,
                GLfloat q3x,GLfloat q3y,GLfloat q3z,int *flag)

{
    GLfloat arx1[3], ary1[3], arz1[3];

```

```
GLfloat maxx1, maxy1, maxz1;  
GLfloat minx1, miny1, minz1;  
GLfloat arx2[3], ary2[3], arz2[3];  
GLfloat maxx2, maxy2, maxz2;  
GLfloat minx2, miny2, minz2;  
int i1;
```

```
*flag = 0;
```

```
arx1[0] = p1x;  
arx1[1] = p2x;  
arx1[2] = p3x;  
ary1[0] = p1y;  
ary1[1] = p2y;  
ary1[2] = p3y;  
arz1[0] = p1z;  
arz1[1] = p2z;  
arz1[2] = p3z;
```

```
arx2[0] = q1x;  
arx2[1] = q2x;  
arx2[2] = q3x;  
ary2[0] = q1y;  
ary2[1] = q2y;  
ary2[2] = q3y;  
arz2[0] = q1z;  
arz2[1] = q2z;  
arz2[2] = q3z;
```

```
maxx1 = arx1[0];  
minx1 = arx1[0];  
maxy1 = ary1[0];
```

```

miny1 = ary1[0];
maxz1 = arz1[0];
minz1 = arz1[0];

maxx2 = arx2[0];
minx2 = arx2[0];
maxy2 = ary2[0];
miny2 = ary2[0];
maxz2 = arz2[0];
minz2 = arz2[0];
for(i1 = 0; i1 <= 2; ++i1)
{
    if(maxx1 <= arx1[i1])
    {
        maxx1 = arx1[i1];

    } /* if */

    if(minx1 >= arx1[i1])
    {
        minx1 = arx1[i1];

    } /* if */

    if(maxy1 <= ary1[i1])
    {
        maxy1 = ary1[i1];

    } /* if */

    if(miny1 >= ary1[i1])
    {

```

```

    miny1 = ary1[i1];

} /* if */

if(maxz1 <= arz1[i1])
{
    maxz1 = arz1[i1];

} /* if */

if(minz1 >= arz1[i1])
{
    minz1 = arz1[i1];

} /* if */

if(maxx2 <= arx2[i1])
{
    maxx2 = arx2[i1];

} /* if */

if(minx2 >= arx2[i1])
{
    minx2 = arx2[i1];

} /* if */

if(maxy2 <= ary2[i1])
{
    maxy2 = ary2[i1];

```



```

    } /* if */

    if(miny2 >= ary2[i1])
    {
        miny2 = ary2[i1];
    } /* if */

    if(maxz2 <= arz2[i1])
    {
        maxz2 = arz2[i1];
    } /* if */

    if(minz2 >= arz2[i1])
    {
        minz2 = arz2[i1];
    } /* if */

} /* for i1 */


if((maxx1 < minx2 || maxy1 < miny2 || maxz1 < minz2) ||
    (maxx2 < minx1 || maxy2 < miny1 || maxz2 < minz1))
{
    *flag = -1;
}
else
{
    *flag = 1;
}

```

```

    } /* if */

    return;

}

/* 交線計算のために 4 × 4 の行列式を計算する関数 */
double DetermS(double xa, double ya, double za, double wa,
               double x0, double y0, double z0, double w0,
               double x1, double y1, double z1, double w1,
               double x2, double y2, double z2, double w2)
{
    double s1;

    s1 = (xa*y0 - ya*x0)*(z1*w2 - w1*z2)
        - (xa*z0 - za*x0)*(y1*w2 - w1*y2)
        + (xa*w0 - wa*x0)*(y1*z2 - z1*y2)
        + (ya*z0 - za*y0)*(x1*w2 - w1*x2)
        - (ya*w0 - wa*y0)*(x1*z2 - z1*x2)
        + (za*w0 - wa*z0)*(x1*y2 - y1*x2);

    return s1;
}

/* 三角形同士の交線を計算する関数 */
void GetIntLine(GLfloat p1x,GLfloat p1y,GLfloat p1z,
               GLfloat p2x,GLfloat p2y,GLfloat p2z,

```

```

        GLfloat p3x, GLfloat p3y, GLfloat p3z,
        GLfloat q1x, GLfloat q1y, GLfloat q1z,
        GLfloat q2x, GLfloat q2y, GLfloat q2z,
        GLfloat q3x, GLfloat q3y, GLfloat q3z,
        int *IntLineFlag,
        GLfloat *r1x, GLfloat *r1y, GLfloat *r1z,
        GLfloat *r2x, GLfloat *r2y, GLfloat *r2z)

{
    GLfloat triax1[2][3][2], triay1[2][3][2], triaz1[2][3][2];
    GLfloat nx1, ny1, nz1;
    GLfloat ispx1[2], ispy1[2], ispz1[2];
    double sab1, sa012, sb012, sp01n, spl2n, sp20n;
    double pax1, pay1, paz1;
    int i1, i2, i3, i4, cnt1;

    *IntLineFlag = 0;
    cnt1 = 0;

    triax1[0][0][0] = p1x;
    triay1[0][0][0] = p1y;
    triaz1[0][0][0] = p1z;
    triax1[0][0][1] = p2x;
    triay1[0][0][1] = p2y;
    triaz1[0][0][1] = p2z;

    triax1[0][1][0] = p2x;
    triay1[0][1][0] = p2y;
    triaz1[0][1][0] = p2z;
    triax1[0][1][1] = p3x;
    triay1[0][1][1] = p3y;
    triaz1[0][1][1] = p3z;

```

```
triax1[0][2][0] = p3x;  
triay1[0][2][0] = p3y;  
triaz1[0][2][0] = p3z;  
triax1[0][2][1] = p1x;  
triay1[0][2][1] = p1y;  
triaz1[0][2][1] = p1z;
```

```
triax1[1][0][0] = q1x;  
triay1[1][0][0] = q1y;  
triaz1[1][0][0] = q1z;  
triax1[1][0][1] = q2x;  
triay1[1][0][1] = q2y;  
triaz1[1][0][1] = q2z;
```

```
triax1[1][1][0] = q2x;  
triay1[1][1][0] = q2y;  
triaz1[1][1][0] = q2z;  
triax1[1][1][1] = q3x;  
triay1[1][1][1] = q3y;  
triaz1[1][1][1] = q3z;
```

```
triax1[1][2][0] = q3x;  
triay1[1][2][0] = q3y;  
triaz1[1][2][0] = q3z;  
triax1[1][2][1] = q1x;  
triay1[1][2][1] = q1y;  
triaz1[1][2][1] = q1z;
```

```
for(i1 = 0; i1 <= 1; ++i1)  
{  
    if(i1 == 0)
```

```

{
    i3 = 1;
    i4 = 0;

} /* if */

if(i1 == 1)
{
    i3 = 0;
    i4 = 1;

} /* if */

VectorProduct((GLfloat)triax1[i1][0][0],
              (GLfloat)triay1[i1][0][0],
              (GLfloat)triaz1[i1][0][0],
              (GLfloat)triax1[i1][1][0],
              (GLfloat)triay1[i1][1][0],
              (GLfloat)triaz1[i1][1][0],
              (GLfloat)triax1[i1][2][0],
              (GLfloat)triay1[i1][2][0],
              (GLfloat)triaz1[i1][2][0], &nx1, &ny1, &nz1);

for(i2 = 0; i2 <= 2; ++i2)
{
    sa012 = DetermS((double)triax1[i3][i2][0],
                   (double)triay1[i3][i2][0],
                   (double)triaz1[i3][i2][0], 1.0,
                   (double)triax1[i4][0][0],
                   (double)triay1[i4][0][0],
                   (double)triaz1[i4][0][0], 1.0,
                   (double)triax1[i4][1][0],

```

```

        (double)triay1[i4][1][0],
        (double)triaz1[i4][1][0], 1.0,
        (double)triax1[i4][2][0],
        (double)triay1[i4][2][0],
        (double)triaz1[i4][2][0], 1.0);

sb012 = DetermS((double)triax1[i3][i2][1],
                (double)triay1[i3][i2][1],
                (double)triaz1[i3][i2][1], 1.0,
                (double)triax1[i4][0][0],
                (double)triay1[i4][0][0],
                (double)triaz1[i4][0][0], 1.0,
                (double)triax1[i4][1][0],
                (double)triay1[i4][1][0],
                (double)triaz1[i4][1][0], 1.0,
                (double)triax1[i4][2][0],
                (double)triay1[i4][2][0],
                (double)triaz1[i4][2][0], 1.0);

sab1 = sa012 - sb012;
if(sab1 == 0.0)
{
    sab1 = 0.0000001;

} /* if */

/* 線分と三角形の交点の計算 */
pax1 = (double)triax1[i3][i2][0] +
        sa012/sab1*((double)triax1[i3][i2][1] -
        (double)triax1[i3][i2][0]);
pay1 = (double)triay1[i3][i2][0] +
        sa012/sab1*((double)triay1[i3][i2][1] -

```

```

        (double)triay1[i3][i2][0]);
paz1 = (double)triaz1[i3][i2][0] +
        sa012/sab1*((double)triaz1[i3][i2][1] -
        (double)triaz1[i3][i2][0]);

sp01n = DetermS(pax1, pay1, paz1, 1.0,
        (double)triax1[i4][0][0],
        (double)triay1[i4][0][0],
        (double)triaz1[i4][0][0], 1.0,
        (double)triax1[i4][1][0],
        (double)triay1[i4][1][0],
        (double)triaz1[i4][1][0], 1.0,
        (double)nx1, (double)ny1, (double)nz1,
        0.0);

sp12n = DetermS(pax1, pay1, paz1, 1.0,
        (double)triax1[i4][1][0],
        (double)triay1[i4][1][0],
        (double)triaz1[i4][1][0], 1.0,
        (double)triax1[i4][2][0],
        (double)triay1[i4][2][0],
        (double)triaz1[i4][2][0], 1.0,
        (double)nx1, (double)ny1, (double)nz1,
        0.0);

sp20n = DetermS(pax1, pay1, paz1, 1.0,
        (double)triax1[i4][2][0],
        (double)triay1[i4][2][0],
        (double)triaz1[i4][2][0], 1.0,
        (double)triax1[i4][0][0],
        (double)triay1[i4][0][0],
        (double)triaz1[i4][0][0], 1.0,

```

```

        (double)nx1, (double)ny1, (double)nz1,
        0.0);

/* 線分と三角形の交点の三角形への包含判定 */
if(sp01n <= 0.0 && sp12n <= 0.0 && sp20n <= 0.0)
{
    cnt1 = cnt1 + 1;
    ispx1[cnt1-1] = (GLfloat)pax1;
    ispy1[cnt1-1] = (GLfloat)pay1;
    ispz1[cnt1-1] = (GLfloat)paz1;

} /* if */

if(cnt1 >= 2)
{
    *IntLineFlag = 1;
    *r1x = ispx1[0];
    *r1y = ispy1[0];
    *r1z = ispz1[0];
    *r2x = ispx1[1];
    *r2y = ispy1[1];
    *r2z = ispz1[1];

    break;

} /* if */

} /* for i2 */

if(cnt1 >= 2)
{
    break;
}

```



```

    } /* if */

} /* for i1 */

}

/* 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();
```

```
}
```