作業：

期中考替代作業(108 上)

參數化線性軸曲面設計

資工三乙-黃品翰 406262163

2019/12/20

● 程式架構

- 左側視窗(Curve Drawer)

  為平面貝茲曲線的呈現，分為上、下半部的貝茲曲線，各有兩個控制點與固定點，白線部分為七個點連線，固定點為青藍色，控制點則為綠色，選取時則會有紅色變化。白線為七個點連線，紅線則為貝茲曲線，渲染方式則是利用 glMap1f，劃出貝茲曲線。右鍵按下則有選單，可以選擇模式(WIRE, FILL, choose_color, auto_rotate, auto_changecolor)

- 右側視窗(Vase)

  將左側視窗的貝茲曲線，將其視角繞 y 軸旋轉，則能呈現直線繞圈的效果。橫線繞圈則是在 xz 平面上，將原本二維的貝茲曲線的(x, y)，轉換成三維空間的(x,y,z)，畫圓圈(GL_LINE_STRIP)處理，塗色部分則用(GL_QUAD_STRIP)處理，上下左右按鍵則能調整視角

● 討論

- 環境

  利用 VScode 做文字編輯器，opengl 有專屬給 Mingw 的套件，將其 Lib 匯入則可以使用，這部分研究了一下子

- 二維

  二維圖形部分其實相較來說比較簡單，opengl 的 glMap1f 就能實作出來，一開始採到坑的部分應該是 reshape 及 orthogonal，網路上看了一下才大概，及比對學長姐們的 code 才知道在幹嘛

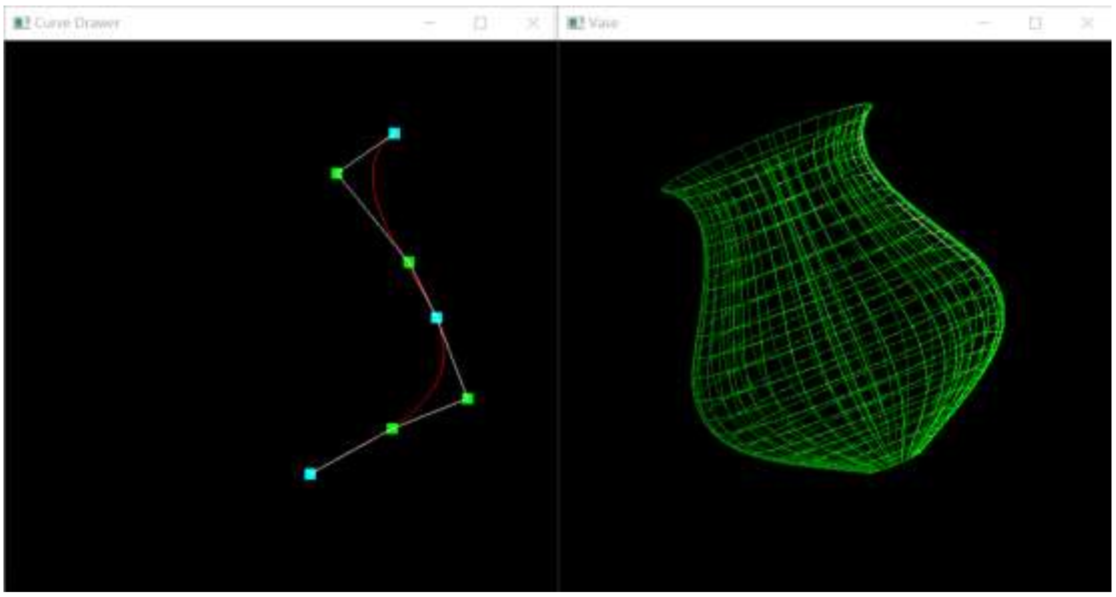
- 三維及打光

  一開始至實作出直線繞圈部分，橫線繞圓部分比較難去理解，因為需要把座標進行轉換，繞圓部分則是用圓的參數式來實作，利用多段直線來逼近圓，打光則是參考網路上的教學實作，不過跟預想的不太一樣，但因時間關係則沒有在另行調整

- 開關(轉動、變色)

  利用狀態機做處理，因為 opengl 的建構都是一直持續，因此要轉換狀態需用狀態機去實作，變色則只須改調色參數即可

● 執行畫面



● 程式碼

```cpp
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <windows.h>
#include <limits>
#include <iostream>
#include <string>
#include "GL/glut.h"

#define LINE_MODE 1
#define FILL_MODE 2
#define PI acos(-1)

// 初始化設定
int windowHeight = 500;
int windowWidth = 500;
int numOfPoints = 7;
int selectedPoint = -1;
int cid = 0;
int mode = LINE_MODE;
```

```
float min[] = {-1.0, -1.0, -1.0};
float max[] = {1.0, 1.0, 1.0};
GLfloat light_ambient[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat light_diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat light_specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat light_position[] = {1.0f, 1.0f, 0.0f, 1.0f};
GLfloat mat_ambient[] = {0.8f, 0.8f, 0.8f, 1.0f};
GLfloat mat_diffuse[] = {0.8f, 0.8f, 0.8f, 1.0f};
GLfloat mat_specular[] = {0.8f, 0.8f, 0.8f, 1.0f};
GLfloat high_shininess[] = {100.0f};
GLfloat points[7][3] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}, {0, 0, 0}};
GLfloat rpoint[7][3] = {{140, 90, 0}, {300, 120, 0}, {365, 200, 0}, {390, 250, 0}, {380, 270, 0}, {350, 350, 0}, {250, 370, 0}};
float orthoMax[3], orthoMin[3];
float draw_point[21][2];

//真實座標換成 opengl 平面視窗座標，利用 Orthographic projection
void real_point2ortho(int x, int y, GLfloat &fx, GLfloat &fy)
{
    fx = orthoMin[0] + (float)x / (float)windowWidth * (orthoMax[0] - orthoMin[0]);
    fy = orthoMin[1] + (float)(windowHeight - y) / (float)windowHeight * (orthoMax[1] - orthoMin[1]);
}

// 算距離
GLfloat dist(GLfloat x1, GLfloat y1, GLfloat z1, GLfloat x2, GLfloat y2, GLfloat z2)
{
    int near_pt = -1;
    GLfloat d;
    d = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) + (z1 - z2) * (z1 - z2);
    return d;
}
// 找離滑鼠最近的點
int check_near_point(GLfloat x, GLfloat y, GLfloat z)
```

```
{
    int idx = -1;
    GLfloat near_dis = std::numeric_limits<GLfloat>::max();
    for (int i = 0; i < 7; i++)
    {
        if (near_dis > dist(points[i][0], points[i][1], points[i][2], x, y, z))
        {
            near_dis = dist(points[i][0], points[i][1], points[i][2], x, y, z);
            idx = i;
        }
    }
    return idx;
}

// mouse postion
void motion(int x, int y)
{
    GLfloat fx, fy;
    real_point2ortho(x, y, fx, fy);
    // std::cout << x << " " << y << '\n';
    if (selectedPoint >= 0)
    {
        rpoint[selectedPoint][0] = x, rpoint[selectedPoint][1] = y;
        glutPostRedisplay();
    }
}
//mouse state
void Mouse(int button, int state, int x, int y)
{
    GLfloat fx, fy;
    real_point2ortho(x, y, fx, fy);
    if (button == GLUT_LEFT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            selectedPoint = check_near_point(fx, fy, 0.0);
            glutMotionFunc(motion);
        }
        else
            selectedPoint = -1;
    }
    glutPostRedisplay();
    glutSwapBuffers();
}

//畫控制點
void draw_points()
{
    glPointSize(10.0f);
    glBegin(GL_POINTS);
    {
        for (int i = 0; i < numOfPoints; i++){
            if (i == 0 || i == 3 || i == 6)
            {
                glColor3f(0.0, 1.0, 1.0);
                glVertex3f(points[i][0], points[i][1], points[i][2]);
            }
            else{
                glColor3f(0.0, 1.0, 0.0);
                glVertex3f(points[i][0], points[i][1], points[i][2]);
            }
            if (i == selectedPoint)
            {
                glColor3f(1.0, 0.0, 0.0);
                glVertex3f(points[i][0], points[i][1], points[i][2]);
            }
        }
    }
    glEnd();
    // 將七個點連線, 利用 GL_LINE_STRIP
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
    {
        for (int i = 0; i < numOfPoints; i++)
```

```
            glVertex3f(points[i][0], points[i][1], points[i][2]);
    }
    glEnd();
}


void Reshape3D(int width, int height)
{
    // Find the largest and smallest values for all coordinates
    float max3D = 1.0f, min3D = -1.0f;

    GLfloat aspect;
    windowWidth = width, windowHeight = height;

    // Set the viewport
    // 把視景體截取的圖像按照怎樣的高和寬顯示到 screen
    glViewport(0, 0, (GLsizei)width, (GLsizei)height);
    // Make the projection matrix current
    glMatrixMode(GL_PROJECTION);
    // Clear the projection matrix
    glLoadIdentity();

    // Set the projection matrix (based on the aspect ratio)
    // 因應視窗的大小變化，需要做長寬調整
    if (width <= height)
    {
        //y 座標需要被拉長
        aspect = (GLfloat)height / (GLfloat)width;
        orthoMin[0] = min3D;
        orthoMin[1] = min3D * aspect;
        orthoMin[2] = min3D;
        orthoMax[0] = max3D;
        orthoMax[1] = max3D * aspect;
        orthoMax[2] = max3D;
    }
    else
    {
        aspect = (GLfloat)width / (GLfloat)height;
```

```
            // x 座標需要被拉長
            orthoMin[0] = min3D * aspect;
            orthoMin[1] = min3D;
            orthoMin[2] = min3D;
            orthoMax[0] = max3D * aspect;
            orthoMax[1] = max3D;
            orthoMax[2] = max3D;
        }

        //glOrtho(left,right,up,down,near,far)
        //利用 Orthographic projection
        //將立體座標壓成平面
        glOrtho(orthoMin[0], orthoMax[0],
                orthoMin[1], orthoMax[1],
                orthoMin[2], orthoMax[2]);

        // Make the Model-View matrix active
        glMatrixMode(GL_MODELVIEW);
    }

    void bezier_curve()
    {
        glClear(GL_COLOR_BUFFER_BIT);
        //x = f(u); y = g(u); z = h(u);
        //將 7 個點換成 ortho 座標
        glPushMatrix();
        for (int i = 0; i < 7; i++)
        {
            GLfloat fx, fy;
            real_point2ortho(rpoint[i][0], rpoint[i][1], fx, fy);
            points[i][0] = fx;
            points[i][1] = fy;
            // std::cout << fx << " " << fy << '\n';
        }
        glMap1f(GL_MAP1_VERTEX_3, //生成的數據類型
                0.0f,             //u 值的下界
                100.0f,           //u 值的上界
                3,                //每個頂點在數據中的間隔，每一個頂點資訊都有x,y,z，所以長度為3
                4,                //控制點的個數
                &points[0][0]);   //其他點指向該控制點的 pointer

        //利用劃線方式將點連成貝茲曲線
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINE_STRIP);
        for (int i = 0; i < 100; i++)
        {
            glEvalCoord1f((GLfloat)i);
        }
        glEnd();

        glMap1f(GL_MAP1_VERTEX_3, //生成的數據類型
                0.0f,             //下界
                100.0f,           //上界
                3,                //每個頂點在數據中的間隔，每一個頂點資訊都有x,y,z，所以長度為3
                4,                //控制點的個數
                &points[3][0]);   //其他點指向該控制點的 pointer

        glBegin(GL_LINE_STRIP);
        for (int i = 0; i < 100; i++)
        {
            glEvalCoord1f((GLfloat)i);
        }
        glEnd();
        glPopMatrix();

        //將 evalCoord 中的點全部連線
        glEnable(GL_MAP1_VERTEX_3);

        //畫 control point
        draw_points();
        // for mouse
        glutMouseFunc(Mouse);
        glutSwapBuffers();
        glutPostRedisplay();

    }

    //在 xz 平面畫圓，以 x 為半徑繞 y 軸旋轉畫圓
    void draw_circle(float x, float y, float z, float radius)
    {
        // glColor3f(0, 1, 0);
        int sections = 100;
        GLfloat TWOPI = 2.0f * 3.14159f;
        glBegin(GL_LINE_STRIP);
        for (int count = 0; count <= sections; count++)
        {
            glVertex3f(x + radius * cos(count * TWOPI / sections), y, z + radius * sin(count * TWOPI / sections));
        }
        glEnd();
    }



    //xz 平面的圓
    void Horizontal_circle()
    {
        //貝茲曲線三次公式：P0 * (1-t)^3 + 3 * P1 * t(1-t)^2 + 3 * P2 * t^2(1-t) + P3 * t^3
        GLfloat P0_X = points[0][0], P0_Y = points[0][1];
        GLfloat P1_X = points[1][0], P1_Y = points[1][1];
        GLfloat P2_X = points[2][0], P2_Y = points[2][1];
        GLfloat P3_X = points[3][0], P3_Y = points[3][1];
        GLfloat P4_X = points[4][0], P4_Y = points[4][1];
        GLfloat P5_X = points[5][0], P5_Y = points[5][1];
        GLfloat P6_X = points[6][0], P6_Y = points[6][1];

        for (GLfloat t = 0; t <= 1.1; t += 0.1)
        {
            GLfloat x = P0_X * pow((1 - t), 3) + 3 * P1_X * t * pow((1 - t), 2) + 3 * P2_X * pow(t, 2) * (1 - t) + P3_X * pow(t, 3);
            GLfloat y = P0_Y * pow((1 - t), 3) + 3 * P1_Y * t * pow((1 - t), 2) + 3 * P2_Y * pow(t, 2) * (1 - t) + P3_Y * pow(t, 3);
            draw_point[(int)(t * 10)][0] = x;
            draw_point[(int)(t * 10)][1] = y;
            if(mode == LINE_MODE)
                draw_circle(0, y, 0, x);
        }

        for (GLfloat t = 0; t <= 1.1; t += 0.1)
        {
            GLfloat x = P3_X * pow((1 - t), 3) + 3 * P4_X * t * pow((1 - t), 2) + 3 * P5_X * pow(t, 2) * (1 - t) + P6_X * pow(t, 3);
            GLfloat y = P3_Y * pow((1 - t), 3) + 3 * P4_Y * t * pow((1 - t), 2) + 3 * P5_Y * pow(t, 2) * (1 - t) + P6_Y * pow(t, 3);
            draw_point[10+(int)(t * 10)][0] = x;
            draw_point[10+(int)(t * 10)][1] = y;
            if (mode == LINE_MODE)
                draw_circle(0, y, 0, x);
        }
    }

    void draw_surface_color()
    {
        int sections = 10;
        GLfloat TWOPI = 2.0f * 3.14159f;
        glBegin(GL_QUAD_STRIP);
        for (int i = 0; i < 20; i++)
        {
            for (int count = 0; count <= sections; count++)
            {
                glVertex3f(draw_point[i][0] * cos(count * TWOPI / sections), draw_point[i][1], draw_point[i][0] * sin(count * TWOPI / sections));
                glVertex3f(draw_point[i + 1][0] * cos(count * TWOPI / sections), draw_point[i + 1][1], draw_point[i + 1][0] * sin(count * TWOPI / sections));
            }
        }
        glEnd();
```

```

}



int x_view = 0;
int y_view = 0;
int z_view = 0;
GLfloat color_r = 0;
GLfloat color_g = 0;
GLfloat color_b = 0;

void KeyBoards(unsigned char key, int mx, int my)
{
    switch (key)
    {
        case 'l':
            mode = LINE_MODE;
            break;
        case 'L':
            mode = LINE_MODE;
            break;
        case 'o':
            mode = FILL_MODE;
            break;
        case 'O':
            mode = FILL_MODE;
            break;
    }
}
int save_view = 0;
void SpecialKey(GLint key, int mx, int my)
{
    if(x_view >= 360 || x_view <= -360)
        x_view = 0, save_view = x_view;

    if (key == GLUT_KEY_UP)
    {
        x_view -= 1;
        save_view = x_view;
    }
    if (key == GLUT_KEY_LEFT)
    {
        z_view -= 1;
    }
    if (key == GLUT_KEY_DOWN)
    {
        x_view += 1;
        save_view = x_view;
    }
    if (key == GLUT_KEY_RIGHT)
    {
        z_view += 1;
    }
}

void vase()
{
    // Initialize the Model-View matrix
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // glPolygonMode 正反面都是用線或填滿
    if (mode == LINE_MODE)
    {
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    }
    else
    {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    }

    //在 x,z 平面，繞 y 軸旋轉
    glutKeyboardFunc(KeyBoards);
    glutSpecialFunc(SpecialKey);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    int idx = 0;

    if (mode == LINE_MODE)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

    //隨 x_view/z_view 做視角調整
    glRotatef(x_view, 1, 0, 0);
    glRotatef(z_view, 0, 0, 1);

    for (int i = 0; i < 360; i += 3)
    {
        glColor3f(color_r, color_g, color_b);
        glRotatef(y_view + i, 0, 1, 0);
        GLfloat fx, fy;
        for (int j = 0; j < 7; j++)
        {
            real_point2ortho(rpoint[j][0], rpoint[j][1], fx, fy);
            points[j][0] = fx;
            points[j][1] = fy;
        }
        if (mode == LINE_MODE)
            Horizontal_circle();
        else{
            Horizontal_circle();
            draw_surface_color();
        }


        glMap1f(GL_MAP1_VERTEX_3, //生成的數據類型
                0.0f,             //u 值的下界
                500.0f,           //u 值的上界
                3,                //每個頂點在數據中的間隔，每一個頂點資訊都有
x,y,z，所以長度為 3
                4,                //控制點的個數
                &points[0][0]);   //其他點指向該控制點的 pointer

        //利用劃線方式將點連成貝茲曲線
        // glColor3f(0.0, 1.0, 0.0);
        glBegin(GL_LINE_STRIP);
        for (int i = 0; i < 500; i++)
            glEvalCoord1f((GLfloat)i);
        glEnd();

        glMap1f(GL_MAP1_VERTEX_3, //生成的數據類型
                0.0f,             //下界
                500.0f,           //上界
                3,                //每個頂點在數據中的間隔，每一個頂點資訊都有
x,y,z，所以長度為 3
                4,                //控制點的個數
                &points[3][0]);   //其他點指向該控制點的 pointer

        glBegin(GL_LINE_STRIP);
        for (int i = 0; i < 500; i++)
            glEvalCoord1f((GLfloat)i);
        glEnd();

        //將 evalCoord 中的點全部連線
        glEnable(GL_MAP1_VERTEX_3);
    }

    glutPostRedisplay();
    glutSwapBuffers();
}



bool auto_rotate = false;
bool auto_change_color = false;
int save_color = 6;

GLfloat colors[7][3] = {
    {0.0, 0.0, 0.0},
    {1.0, 0.0, 0.0},
    {0.0, 1.0, 0.0},
    {0.0, 0.0, 1.0},
    {0.0, 1.0, 1.0},
    {1.0, 0.0, 1.0},
    {1.0, 1.0, 0.0},
};

void auto_color()
{
    if (auto_change_color)
    {
        if (save_color == 0)
            save_color = 1;
        else if (save_color == 1)
            save_color = 2;
        else if (save_color == 2)
            save_color = 3;
        else if (save_color == 3)
            save_color = 4;
        else if (save_color == 4)
            save_color = 5;
        else if (save_color == 5)
            save_color = 6;
        else if (save_color == 6)
            save_color = 0;
        color_r = colors[save_color][0];
        color_g = colors[save_color][1];
        color_b = colors[save_color][2];
    }
    else
    {
        color_r = colors[save_color][0];
        color_g = colors[save_color][1];
        color_b = colors[save_color][2];
    }
}

void auto_rot()
```

```
{
    if (auto_rotate)
    {
        if(save_view >= 360 || save_view <= -360)
            x_view = 0;
        x_view += 1;
        save_view = x_view;
    }
    else
    {
        x_view = save_view;
    }
}

int vase_window;

void idle()
{
    glutSetWindow(vase_window);
    auto_color();
    auto_rot();
}


void main_menu(int index)
{
    switch (index)
    {
        case 0:
            mode = LINE_MODE;
            break;
        case 1:
            mode = FILL_MODE;
            break;
        case 2:
            break;
        case 3:
            auto_change_color ^= 1;
```

```
                break;
            case 4:
                auto_rotate ^= 1;
                break;
        }
}

static void color_menu(int index)
{
    if ((index <= 6) && (index >= 0))
    {
        save_color = index;
    }
}

//MENU
void menu()
{
    int cm = glutCreateMenu(color_menu);
    glutAddMenuEntry("Black", 0);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 3);
    glutAddMenuEntry("Cyan", 4);
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("Yellow", 6);
    glutCreateMenu(main_menu);
    glutAddMenuEntry("WIRE", 0);
    glutAddMenuEntry("FILL", 1);
    glutAddSubMenu("choose_color", cm);
    glutAddMenuEntry("auto_rotate on/off", 4);
    glutAddMenuEntry("auto_change color on/off", 3);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

int main(int argc, char *argv[])
{
    //opengl 基礎設定
    glutInit(&argc, argv);
    glEnable(GL_MAP1_VERTEX_3);
    glEnable(GL_DEPTH_TEST);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);


    //視窗設定
    glutInitWindowPosition(100, 100); // 設定視窗位置
    glutInitWindowSize(windowWidth, windowHeight); // 設定視窗大小

    //視窗顏色設定
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glColor3f(1.0f, 0.0f, 1.0f);

    // render point/curve
    // reshape 為視窗更動時，圖形長寬會更著變動
    glutCreateWindow("Curve Drawer"); // 設定視窗標題
    glutDisplayFunc(bezier_curve);
    glutReshapeFunc(Reshape3D);

    //MENU
    menu();


    // render vase
    // reshape 為視窗更動時，圖形長寬會更著變動
    glutInitWindowPosition(600, 100); // 設定視窗位置
    vase_window = glutCreateWindow("Vase"); // 設定視窗標題
    glutDisplayFunc(vase);
    glutReshapeFunc(Reshape3D);
    glutIdleFunc(vase);
    glutIdleFunc(idle);


    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_COLOR_MATERIAL);

    GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat mat_shininess[] = {100.0};
    GLfloat light_position[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat white_light[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat Light_Model_Ambient[] = {0.5, 0.5, 1, 1};

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white_light);
    glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);            //镜面反射光
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, Light_Model_Ambient); //环境光参数



    glutMainLoop();
    return 0;
}
```