8.3 背面剔除消隐.cpp

```cpp
#define GLUT_DISABLE_ATEXIT_HACK
#include <List>
#include <vector>
#include "GLUT.H"
#include <math.h>
#include <string.h>
#include "My_3DVector.h"
using namespace std;

int nearplane_width = 600; //视景体宽度
int nearplane_height = 600; //视景体高度
int nearplane_distance = 500; //视景体近平面与视点距离
int farplane_distance = nearplane_distance + 300; //视景体远平面与视点距离

float eye_x = 20;
float eye_z = 20;
float theta = 0.1; //视点偏转角度

//定义面的结构体，包含该面的各个顶点和法向量
struct my_face_homogeneous
{
	list<my_homogeneous_point> mList;//各个顶点按照逆时针的顺序储存
	my_3Dvector n; //定义面法向量
};

vector<my_face_homogeneous> model; //用来存储三维图形的10个面
my_homogeneous_point facepoint, a, b, c;//用来存储临时点
my_3Dvector N1, N2, N3;//用来存储向量以及计算面的法向量
my_3Dvector v(eye_x, 1, eye_x);//用向量v存储视点方向
my_face_homogeneous tempt; //用于存储面的信息

//初始化三维图形顶点坐标
void init(void)
{
	//面一
	facepoint.x = 0;
	facepoint.y = 0;
	facepoint.z = 0;
	tempt.mList.push_back(facepoint);
	a = facepoint;//a用来存储当前平面上的一个点

	facepoint.x = 0;
	facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
b = facepoint;//b用来存储当前平面上的一个点

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
c = facepoint;//c用来存储当前平面上的一个点

facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 0;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面二
facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
c = facepoint;
```

```
facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面三
facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面四
facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面五
facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
```

```
a = facepoint;

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面六
facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 90;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面七
facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 0;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 0;
facepoint.y = 0;
facepoint.z = 90;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
```

```
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面八
facepoint.x = 0;
facepoint.y = 0;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 0;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 0;
facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 0;
facepoint.y = 0;
facepoint.z = 0;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面九
```

```
facepoint.x = 0;
facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 0;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 90;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 45;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

facepoint.x = 60;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 20;
tempt.mList.push_back(facepoint);

facepoint.x = 80;
facepoint.y = 50;
facepoint.z = 0;
tempt.mList.push_back(facepoint);

N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
```

```
N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
tempt.n.dx = N3.dx;
tempt.n.dy = N3.dy;
tempt.n.dz = N3.dz;

model.push_back(tempt);
tempt.mList.clear();

//面十
facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 0;
tempt.mList.push_back(facepoint);
a = facepoint;

facepoint.x = 80;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
b = facepoint;

facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 20;
tempt.mList.push_back(facepoint);
c = facepoint;

facepoint.x = 60;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 45;
tempt.mList.push_back(facepoint);

facepoint.x = 45;
facepoint.y = 0;
facepoint.z = 90;
tempt.mList.push_back(facepoint);

facepoint.x = 0;
```

```
    facepoint.y = 0;
    facepoint.z = 90;
    tempt.mList.push_back(facepoint);

    facepoint.x = 0;
    facepoint.y = 0;
    facepoint.z = 0;
    tempt.mList.push_back(facepoint);

    N1 = my_3Dvector(a, b);//N1用来存储当前平面上的一个向量
    N2 = my_3Dvector(a, c);//N2用来存储当前平面上的一个向量
    N3 = N1.cross_multiply(N2);//N3为当前平面的法向量
    tempt.n.dx = N3.dx;
    tempt.n.dy = N3.dy;
    tempt.n.dz = N3.dz;

    model.push_back(tempt);
    tempt.mList.clear();
}

//绘制坐标系
void draw_coordinate()
{
    glBegin(GL_LINES);
    glColor3f(1.0, 0.0, 0.0); //红色x轴
    glVertex3f(nearplane_width, 0.0, 0.0);
    glVertex3f(-nearplane_width, 0.0, 0.0);

    glColor3f(0.0, 1.0, 0.0);//绿色y轴
    glVertex3f(0.0, nearplane_height, 0.0);
    glVertex3f(0.0, -nearplane_height, 0.0);

    glColor3f(0.0, 0.0, 1.0);//蓝色z轴
    glVertex3f(0.0, 0.0, nearplane_height);
    glVertex3f(0.0, 0.0, -nearplane_height);
    glEnd();
}

//绘制内容
void display(void)
{
    glClearColor(1.f, 1.f, 1.f, 0.f);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    draw_coordinate(); //绘制坐标系

    glColor3f(157.0 / 256, 195.0 / 256, 230.0 / 256);

    //背部剔除消隐算法的实现
    for (int i = 0; i < 10; i++)
    {
        float num = model[i].n.dot_multiply(v);//模型第i个面法向量与视向量点乘
        if (num > 0)//绘制可见面
        {
            //以下代码是为了让不同平面呈现不同颜色
            if (i % 5 == 0)
            {
                glColor3f(1,0,0);
            }
            else if (i % 5 == 1)
            {
                glColor3f(0,0,0);
            }
            else if (i % 5 == 2)
            {
                glColor3f(0,1,0);
            }
            else if(i % 5 == 3)
            {
                glColor3f(1,1,0);
            }
            else if (i % 5 == 4)
            {
                glColor3f(0,0,1);
            }

            glBegin(GL_POLYGON);
            list<my_homogeneous_point>::iterator iter = model[i].mList.begin();
            for (; iter != model[i].mList.end(); iter++)
            {
                glVertex3f((*iter).x, (*iter).y, (*iter).z);
            }
            glEnd();
        }
    }
    glutSwapBuffers();
}
```

```
//键盘交互事件
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 27:
        exit(0);
        break;
    }
}

//投影方式、modelview方式设置
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if (w <= h)
        glOrtho(-0.5 * nearplane_width, 0.5 * nearplane_width, -0.5 * nearplane_height *
(GLfloat)nearplane_height / (GLfloat)nearplane_width, 0.5 * nearplane_height *
(GLfloat)nearplane_height / (GLfloat)nearplane_width,
            -nearplane_distance, farplane_distance); //相对于视点
    else
        glOrtho(-0.5 * nearplane_width, 0.5 * nearplane_width, -0.5 * nearplane_height *
(GLfloat)nearplane_width / (GLfloat)nearplane_height, 0.5 * nearplane_height *
(GLfloat)nearplane_width / (GLfloat)nearplane_height,
            -nearplane_distance, farplane_distance);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    gluLookAt(eye_x, 10, eye_z, 0, 0, 0, 0, 1, 0);
}

//鼠标交互事件
//鼠标点击改变视点方向
void mouse(int button, int state, int x, int y)
{
    switch (button)
    {
    case GLUT_LEFT_BUTTON:
        if (state == GLUT_DOWN)
        {
            eye_x = eye_x * cosf(-theta) + eye_z * sinf(-theta);
            eye_z = eye_z * cosf(-theta) - eye_x * sinf(-theta);
            v.dx = eye_x;
            v.dy = 1;
            v.dz = eye_z;
            reshape(nearplane_width, nearplane_height);
            glutPostRedisplay();
        }
        break;
    case GLUT_RIGHT_BUTTON:
        if (state == GLUT_DOWN)
        {
            eye_x = eye_x * cosf(theta) + eye_z * sinf(theta);
            eye_z = eye_z * cosf(theta) - eye_x * sinf(theta);
            v.dx = eye_x;
            v.dy = 1;
            v.dz = eye_z;
            reshape(nearplane_width, nearplane_height);
            glutPostRedisplay();
        }
        break;
    default:
        break;
    }
}

//主调函数
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(nearplane_width, nearplane_height);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("背面剔除算法");

    init();

    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMainLoop();
    return 0;
}
```

My_3DVector.h

```
#pragma once

//三维齐次坐标下的点
struct my_homogeneous_point
{
        float x;
        float y;
        float z = 0;
        float ratio;
};

//定义向量
class my_3Dvector
{
public:
        float dx = 0;
        float dy = 0;
        float dz = 0;
        float len;
public:
        my_3Dvector() {}

        my_3Dvector(float x, float y, float z)
        {
                dx = x;
                dy = y;
                dz = z;

                len = sqrtf(powf(dx, 2) + powf(dy, 2) + powf(dz, 2));
        }

        //start点指向end点的向量
        my_3Dvector(my_homogeneous_point start, my_homogeneous_point end)
        {
                dx = end.x - start.x;
                dy = end.y - start.y;
                dz = end.z - start.z;
                len = sqrtf(powf(dx, 2) + powf(dy, 2) + powf(dz, 2));
        }

        //叉乘  this X input_vector
        my_3Dvector cross_multiply(my_3Dvector input_vector)
```

```
    {
        float new_dx = dy * input_vector.dz - dz * input_vector.dy;
        float new_dy = dz * input_vector.dx - dx * input_vector.dz;
        float new_dz = dx * input_vector.dy - dy * input_vector.dx;
        return   my_3Dvector(new_dx, new_dy, new_dz);
    }

    //点乘  this * input_vector
    float dot_multiply(my_3Dvector input_vector)
    {
        return dx * input_vector.dx + dy * input_vector.dy + dz * input_vector.dz;
    }
};
```