

第三组实验报告——构建L-system分形树

第三组实验报告——构建L-system分形树

第三组小组成员分工

L-system分形树基本原理

- 1、L-system文法
- 2、绕x, y, z轴的旋转矩阵

构建L-system分形树实验过程

- 1、文法生成器
- 2、生成的文法进行解释
- 3、绘制树干、树叶
- 4、纹理贴图
- 5、增加局部光照效果：启用initParameter()中的如下代码）
- 6、天空盒背景
- 7、交互摄像机
- 8、四季变换

最终实现效果

总结&&个人心得

第三组小组成员分工

成员	学号	贡献
伍斌	17343124	收集资料，实现天空盒、地面、项目整体调参、编写报告与PPT，春夏秋冬变换，交互摄像机、传统光照
郎森淋	18342043	收集资料，实现项目雏形、构建L-system分形树本体
聂硕琳	18342078	收集资料，上台汇报
郭锦	18342021	收集资料

L-system分形树基本原理

(参考博客：https://blog.csdn.net/qq_31615919/article/details/78976063)

1、L-system文法

在L-系统中，产生式是并行地、同时替换所给字符串中的所有字符。它的规则：经过语法规则迭代重写得到的仅仅是一串字符串，还需要对这些字符进行解析。Prusinkiewicz提出解析L-系统的海龟解释方法。海龟解释的基本思想如下：平面上海龟的当前状态由一个三原数组(x,y,a)来定义，其中笛卡儿坐标(x,y)表示海龟的位置，角度 a 为海龟的前进方向，可以解释为海龟面对的方向。给出步长 d 和角度的增量 $sita$ ，海龟根据以下符号所代表的命令做出反应。

F：海龟向前移动一个步长 d ，海龟的位置变为 (x_1, y_1, a) ，其中， $x_1 = x + d\cos(a)$ ， $y_1 = y + d\sin(a)$ ，在点 (x,y) 和 (x_1,y_1) 之间画一条线段。

f：向前移动一个步长 d ，不画线段。

+：向左转角度 $sita$ ，海龟的下一个状态为 $(x_1, y_1, a+sita)$ ，角的正方向为逆时针方向。

-：向右转角度 $sita$ ，海龟的下一个状态为 $(x_1, y_1, a-sita)$ 。

[：将海龟的当前状态压入堆栈。

]：从堆栈中弹出当前海龟的状态。

扩展到三维空间，海龟在空间的当前位置是由三个向量 $[x,y,z]$ 表示的，他们分别表示海龟前进的方向、向左的方向和向上的方向，三个方向向量是相互垂直的单位向量。

2、绕x, y, z轴的旋转矩阵

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix}$$

$$R_y(\alpha) = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

http://blog.csdn.net/qq_31615919

构建L-system分形树实验过程

1、文法生成器

```
#include <iostream>

using namespace std;
#include "Grammar.h"
int main()
{
    Grammar grammar;
    grammar.Iteration();
    cout << grammar.getResult() << endl;
    return 0;
}
```

```
class Grammar {
public:
    string w;
    Grammar();
    ~Grammar();
    void Iteration();
    string getResult();
private:
    string result;
    vector<string> generations;
    string search(char c);
    int level;
};

Grammar::Grammar() {
    w = "F[*+L][-/!L][/@L]";
    level = 6;
    //F是树干
    //+ - 绕x轴转动dx -dx
    /* / 绕y轴转动dy -dy
    //! @ 绕z轴转动dz -dz
    generations.push_back("F[+@L][/*!L][!-L]");
    generations.push_back("F[@L][*-L][@/-L]");
    generations.push_back("F[-L][+@L][-!L]");
    generations.push_back("F[*+L][-*L][!/L]");
}
}
```

结果：

2、生成的文法进行解释

```
class LSystem {
public:
    vector<Trunk> trunks;
    vector<Leaf> leafs;
    LSystem();
    ~LSystem();
    void generateFractal();
    double getRadiusFactor() { return radiusFactor; }
    double getLengthFactor() { return lengthFactor; }
    double getLeafRadius() { return leafRadius; }

//private:
    int level;
    double dx, dy, dz;
    double leafRadius;
    double leafLength;
    double length, lengthFactor;
    double radius, radiusFactor;
    State curState;
    Grammar grammar;
    int numTrunks, numLeafs;
    string result;
};
```

其中的generateFractal():

```

void LSystem::generateFractal() {
    trunks.clear();
    leafs.clear();
    curState.pos = glm::vec3(0, 0, 0);
    curState.dir = glm::vec3(0, 1, 0);
    curState.length = length;
    curState.level = 1;
    curState.radius = radius;
    //状态栈
    stack<State> stacks;
}

for (int i = 0; i < result.size(); i++) {
    char c = result[i];
    Trunk tmp;
    //树干
    if (c == 'F') {
        tmp.start = curState.pos;
        curState.pos += curState.dir * (float)curState.length;
        tmp.end = curState.pos;
        tmp.radius = curState.radius;
        tmp.level = curState.level;
        trunks.push_back(tmp);
        continue;
    }
}

```

```

//绕X轴转dx角度
else if (c == '+') {
    curState.dir = MyRotate::RotateX(curState.dir, dx);
    continue;
}
//绕X轴转-dx角度
else if (c == '-') {
    curState.dir = MyRotate::RotateX(curState.dir, -dx);
    continue;
}
//绕X轴转dy角度
else if (c == '*') {
    curState.dir = MyRotate::RotateY(curState.dir, dy);
    continue;
}
//绕X轴转-dy角度
else if (c == '/') {
    curState.dir = MyRotate::RotateY(curState.dir, -dy);
    continue;
}
//绕X轴转dz角度
else if (c == '!') {
    curState.dir = MyRotate::RotateZ(curState.dir, dz);
    continue;
}
//绕X轴转-dz角度
else if (c == '@') {
    curState.dir = MyRotate::RotateZ(curState.dir, -dz);
    continue;
}
}

```

```

//将当前的状态压入栈中保存
//将当前的长度和体积按比例减小
else if (c == '[') {
    stacks.push(curState);
    curState.length *= lengthFactor;
    curState.radius *= radiusFactor;
    continue;
}
//出栈
else if (c == ']') {
    curState = stacks.top();
    stacks.pop();
    continue;
}
//叶子节点
else if (c == 'L') {
    Trunk t = trunks[trunks.size() - 1];
    Leaf l;
    l.dir = t.end - t.start;
    l.dir = glm::normalize(l.dir);
    l.pos = t.end;
    l.end.x = l.pos.x + l.dir.x * leafLength;
    l.end.y = l.pos.y + l.dir.y * leafLength;
    l.end.z = l.pos.z + l.dir.z * leafLength;
    leafs.push_back(l);
}
else {
    //DO NOTHING
}

```

语法解释：

F	树干
+	绕X轴旋转dx角度
-	绕X轴旋转-dx角度
*	绕Y轴旋转dy角度
/	绕Y轴旋转-dy角度
!	绕Z轴旋转dz角度
@	绕Z轴旋转-dz角度
[进栈 (进入枝干，并更新状态)
]	出栈 (返回到上一级枝干的状态)

rotate旋转实现：

```

namespace MyRotate {
    static glm::vec3 RotateX(glm::vec3 target, float angle) {
        angle = glm::radians(angle);
        glm::mat3 matrix(0.0f);
        matrix[0][0] = 1.0f;
        matrix[1][1] = cos(angle);
        matrix[2][2] = 1.0f;
        matrix[1][2] = sin(angle);
        matrix[2][1] = -sin(angle);
        return matrix * target;
    }
}
```

```

        matrix[1][2] = sin(angle);
        matrix[2][1] = -sin(angle);
        matrix[2][2] = cos(angle);
        glm::vec3 ret = matrix * target;
        return ret;
    }

    static glm::vec3 RotateY(glm::vec3 target, float angle) {
        angle = glm::radians(angle);
        glm::mat3 matrix(0.0f);
        matrix[0][0] = cos(angle);
        matrix[0][2] = -sin(angle);
        matrix[1][1] = 1.0f;
        matrix[2][0] = sin(angle);
        matrix[2][2] = cos(angle);
        glm::vec3 ret = matrix * target;
        return ret;
    }

    static glm::vec3 RotateZ(glm::vec3 target, float angle) {
        angle = glm::radians(angle);
        glm::mat3 matrix(0.0f);
        matrix[0][0] = cos(angle);
        matrix[0][1] = sin(angle);
        matrix[1][0] = -sin(angle);
        matrix[1][1] = cos(angle);
        matrix[2][2] = 1.0f;
        glm::vec3 ret = matrix * target;
        return ret;
    }
}

```

3、绘制树干、树叶

这里卡的蛮久的，因为对openGL的不熟悉，发现很多初始的配置如果没有写全，在我创建的Window上是看不到一棵树，在这里卡了比较长的时间（哎~）

代码：

```

void drawtree(GLboolean shadowRender)
{
    GLboolean textureOn = glIsEnabled(GL_TEXTURE_2D);

    if (shadowRender)
        glDisable(GL_TEXTURE_2D);

    /*绘制树 */
    //	glColor4f(0.0, 0.0, 0.0, 0.5);
    glTranslatef(0, -20, 0);
    glScalef(scale, scale, scale);
    Draw_Cube(-40.0, 40.0, -2.0, 0.0, -40.0, 40.0); //画地面
    glEnable(GL_TEXTURE_2D);
    for (int i = 0; i < myLSystem.trunks.size(); i++) {
        //cout << lsrule.trunks[i].radius << endl;

```

```
    DrawChannel(myLSystem.trunks[i].start, myLSystem.trunks[i].end,
myLSystem.trunks[i].radius);
}
LoadGLTextures(b, &texout, 0, p, q);
for (int i = 0; i < myLSystem.leafs.size(); i++) {//画树叶
//cout << lsrule.trunks[i].radius << endl;
DrawLeaf(myLSystem.leafs[i].pos, myLSystem.leafs[i].end,
myLSystem.getLeafRadius());
}

if (shadowRender && textureOn)
glEnable(GL_TEXTURE_2D);
}
```

别的代码和文档中的相同

效果：迭代5、6次





4、纹理贴图

从网上找到了BMPLoader实现函数



5、增加局部光照效果：启用initParameter()中的如下代码）

```
27  /*
28   glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
29   256, 256, 0, GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);
30
31   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
32   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
33   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
34   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
35
36   //设置材料的属性
37   GLfloat materialSpecular[] = { 1.0f, 0.9f, 0.9f, 1.0f }; //材质的镜面反射颜色
38   GLfloat materialDiffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f }; //散射白色光
39   GLfloat materialShininess[] = { 10.0f }; //镜面反射指数
40   glMaterialfv(GL_FRONT, GL_SPECULAR, materialSpecular);
41   glMaterialfv(GL_FRONT, GL_DIFFUSE, materialDiffuse);
42   glMaterialfv(GL_FRONT, GL_SHININESS, materialShininess);
43
44   //设置光源的属性
45   GLfloat lightAmbient[4] = { 1.0, 1.0, 1.0, 1.0 };
46   GLfloat lightDiffuse[4] = { 0.9, 0.9, 0.9, 1.0 };
47   GLfloat lightSpecular[4] = { 0.9, 0.9, 0.9, 0.0 };
48   GLfloat lightPosition[4] = { 50.0, 40.0, 30.0, 0.0 };
49   glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
50   glLightfv(GL_LIGHT0, GL_DIFFUSE); //指定光源0的漫射光参数
51   glLightfv(GL_LIGHT0, GL_SPECULAR); //指定光源0的镜面光参数
52   glLightfv(GL_LIGHT0, GL_POSITION, lightPosition); //指定光源0的位置
53
54   //启用光源
55   glEnable(GL_LIGHT0);
56   glEnable(GL_LIGHTING);
57 */
```

6、天空盒背景

(参考博客：[OpenGL--天空盒疯狂的程序员-CSDN博客天空盒](#))

我们所实现的天空盒实际上就是一个覆盖场景四周的长方体，但它的各个面上贴有表示天空的纹理图片，即四周的4面纹理的边与顶面纹理的边相连，同时四面纹理前后相连，纹理大小要是2的N次方(32,64,128,...)，我们使用的贴图如下：



六个面分别为top、left、front、right、back、bottom。

构建的天空盒类：

```
/** 天空盒类 */
class CSkyBox
{
public:
    /** 构造函数 */
    CSkyBox();
    ~CSkyBox();

    /** 初始化 */
    bool Init();

    /** 渲染天空 */
    void CreateSkyBox(float x, float y, float z,
                      float width, float height,
                      float length);

private:
    CBMPLoader m_texture[6];    /*< 天空盒纹理 */
};
```

初始化天空盒

```
bool CSkyBox::Init()
{
```

```

char filename[128];                                /**< 用来保存文件名
*/
const char* bmpName[] = { "back", "front", "bottom", "top", "right", "left" };

for (int i = 0; i < 6; i++)
{
    sprintf(filename, "res/%s", bmpName[i]);
    strcat(filename, ".bmp");
    //cout << filename << endl;
    if (!m_texture[i].LoadBitmap(filename))           /**< 载入位图
文件 */
    {
        cout << "装载位图文件失败!" << endl;
        exit(0);
    }
    glGenTextures(1, &m_texture[i].ID);
    glBindTexture(GL_TEXTURE_2D, m_texture[i].ID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
/** 创建纹理 */
    gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, m_texture[i].imageWidth,
m_texture[i].imageHeight, GL_RGB, GL_UNSIGNED_BYTE,
m_texture[i].image);
}
return true;
}

```

构造天空盒

```

void CSkyBox::CreateSkyBox(float x, float y, float z, float box_width, float
box_height, float box_length)
{
/** 获得场景中光照状态 */
GLboolean lp;
glGetBooleanv(GL_LIGHTING, &lp);

/** 计算天空盒长 宽 高 */
float width = MAP * box_width / 8;
float height = MAP * box_height / 8;
float length = MAP * box_length / 8;

/** 计算天空盒中心位置 */
x = x + MAP / 8 - width / 2;
y = y + MAP / 24 - height / 2;
z = z + MAP / 8 - length / 2;

glDisable(GL_LIGHTING);                         /**< 关闭光照 */

/** 开始绘制 */
glPushMatrix();
//glTranslatef(-x, -y, -z);
glTranslatef(x, y, z);

```

```
/** 绘制背面 */
glBindTexture(GL_TEXTURE_2D, m_texture[0].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
glTexCoord2f(1.0f, 0.0f); glVertex3f(x + width, y, z);
glTexCoord2f(1.0f, 1.0f); glVertex3f(x + width, y + height, z);
glTexCoord2f(0.0f, 1.0f); glVertex3f(x, y + height, z);
glTexCoord2f(0.0f, 0.0f); glVertex3f(x, y, z);

glEnd();

/** 绘制前面 */
glBindTexture(GL_TEXTURE_2D, m_texture[1].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
glTexCoord2f(1.0f, 0.0f); glVertex3f(x, y, z + length);
glTexCoord2f(1.0f, 1.0f); glVertex3f(x, y + height, z + length);
glTexCoord2f(0.0f, 1.0f); glVertex3f(x + width, y + height, z + length);
glTexCoord2f(0.0f, 0.0f); glVertex3f(x + width, y, z + length);

glEnd();

/** 绘制底面 */
glBindTexture(GL_TEXTURE_2D, m_texture[2].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
glTexCoord2f(1.0f, 0.0f); glVertex3f(x, y, z);
glTexCoord2f(1.0f, 1.0f); glVertex3f(x, y, z + length);
glTexCoord2f(0.0f, 1.0f); glVertex3f(x + width, y, z + length);
glTexCoord2f(0.0f, 0.0f); glVertex3f(x + width, y, z);

glEnd();

/** 绘制顶面 */
glBindTexture(GL_TEXTURE_2D, m_texture[3].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
glTexCoord2f(0.0f, 1.0f); glVertex3f(x + width, y + height, z);
glTexCoord2f(0.0f, 0.0f); glVertex3f(x + width, y + height, z + length);
glTexCoord2f(1.0f, 0.0f); glVertex3f(x, y + height, z + length);
glTexCoord2f(1.0f, 1.0f); glVertex3f(x, y + height, z);

glEnd();

/** 绘制左面 */
glBindTexture(GL_TEXTURE_2D, m_texture[4].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
```

```

glTexCoord2f(1.0f, 1.0f); glVertex3f(x, y + height, z);
glTexCoord2f(0.0f, 1.0f); glVertex3f(x, y + height, z + length);
glTexCoord2f(0.0f, 0.0f); glVertex3f(x, y, z + length);
glTexCoord2f(1.0f, 0.0f); glVertex3f(x, y, z);

glEnd();

/** 绘制右面 */
glBindTexture(GL_TEXTURE_2D, m_texture[5].ID);

glBegin(GL_QUADS);

/** 指定纹理坐标和顶点坐标 */
glTexCoord2f(0.0f, 0.0f); glVertex3f(x + width, y, z);
glTexCoord2f(1.0f, 0.0f); glVertex3f(x + width, y, z + length);
glTexCoord2f(1.0f, 1.0f); glVertex3f(x + width, y + height, z + length);
glTexCoord2f(0.0f, 1.0f); glVertex3f(x + width, y + height, z);

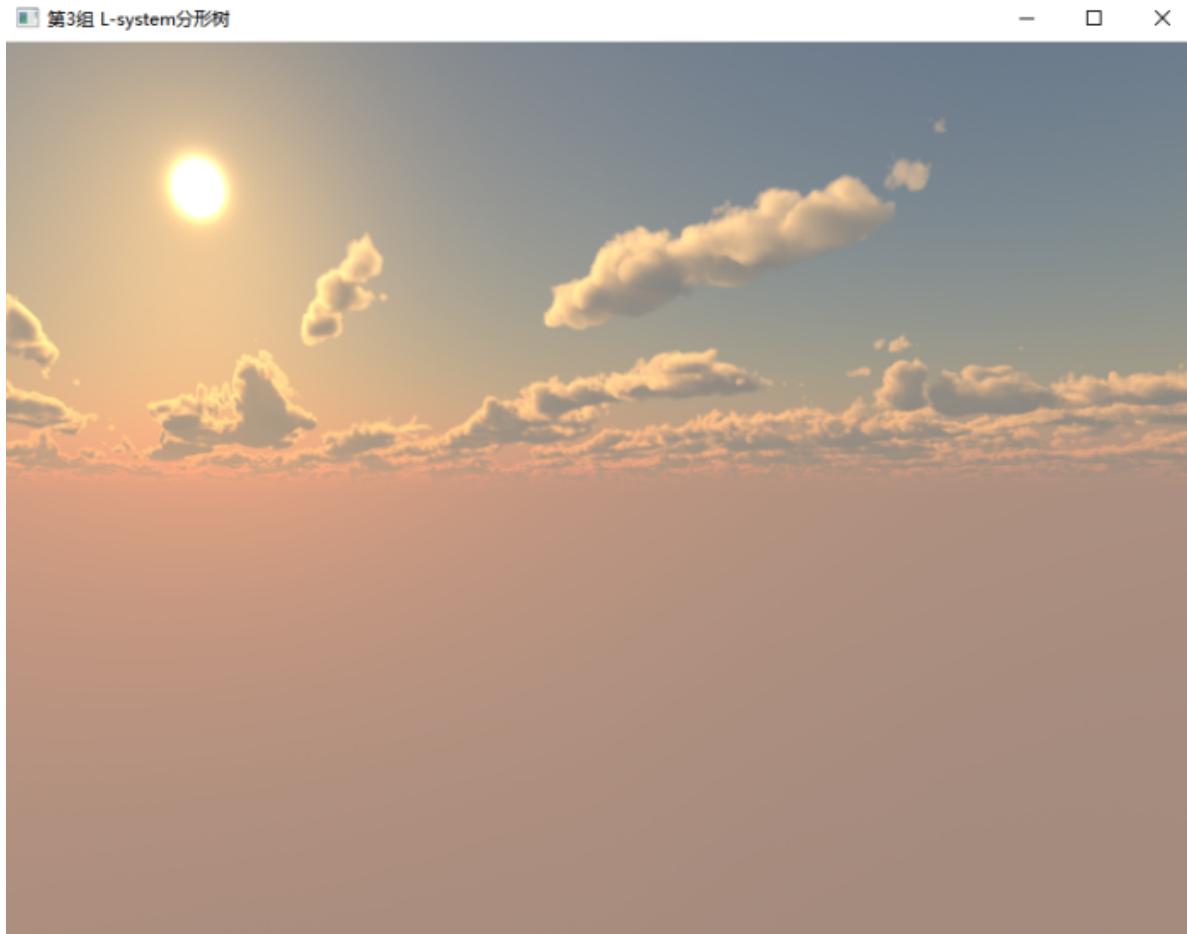
glEnd();

glPopMatrix();                         /** 绘制结束 */

if (!p)                                /** 恢复光照状态 */
    glEnable(GL_LIGHTING);
}

```

天空盒实现效果：



额外构建一个扁平立方体并贴图作为地面：

```

void Draw_Cube(GLfloat x1, GLfloat x2, GLfloat y1, GLfloat y2, GLfloat z1,
GLfloat z2)

```

```

//由立方体组成，可知六个参数即可构造八个顶点
int i, j;
GLfloat vertex[8][3] = {
    //八个顶点，从v1-v8
    x1,y1,z2, //0
    x2,y1,z2, //1
    x1,y2,z2, //2
    x2,y2,z2, //3
    x1,y1,z1, //4
    x2,y1,z1, //5
    x1,y2,z1, //6
    x2,y2,z1, //7
};

GLint surface[6][4] = {
    //v1对应0,以此类推
    4,5,7,6, //back
    0,1,3,2, //front
    2,3,7,6, //top
    0,1,5,4, //bottom
    0,2,6,4, //left
    1,3,7,5 //right
};

CBMPLoader texture1;
char g[128] = "./res/gras.bmp";
texture1.LoadBitmap(g);

//cout << texture1.imageHeight << " " << texture1.imageWidth << endl;
BYTE* ttexture = (BYTE*)malloc(256 * 256 * 3);
for (int i = 0; i < 256; i++) {
    for (int j = 0; j < 256; j++) {
        if (depthImage[i][j] < 0.99) {
            //cout << texture1.image[(255 - i) * 256 * 3 + j * 3] << endl;
            ttexture[(255 - i) * 256 * 3 + j * 3] = 0;
            ttexture[(255 - i) * 256 * 3 + j * 3+1] = 0;
            ttexture[(255 - i) * 256 * 3 + j * 3+2] = 0;
        }
        else {
            if (texture1.image) {
                ttexture[(255 - i) * 256 * 3 + j * 3] = texture1.image[(255 - i) * 256 * 3 + j * 3];
                ttexture[(255 - i) * 256 * 3 + j * 3 + 1] =
                texture1.image[(255 - i) * 256 * 3 + j * 3 + 1];
                ttexture[(255 - i) * 256 * 3 + j * 3 + 2] =
                texture1.image[(255 - i) * 256 * 3 + j * 3 + 2];
            }
        }
    }
}
glGenTextures(1, &texture1.ID);
 glBindTexture(GL_TEXTURE_2D, texture1.ID);
//设置纹理过滤
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
//加载纹理
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, texture1.imageWidth,
    texture1.imageHeight, GL_RGB, GL_UNSIGNED_BYTE,

```

```

    ttexture);
CBMPLoader texture2;
texture2.LoadBitmap(g);

glGenTextures(1, &texture2.ID);
glBindTexture(GL_TEXTURE_2D, texture2.ID);
//设置纹理过滤
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

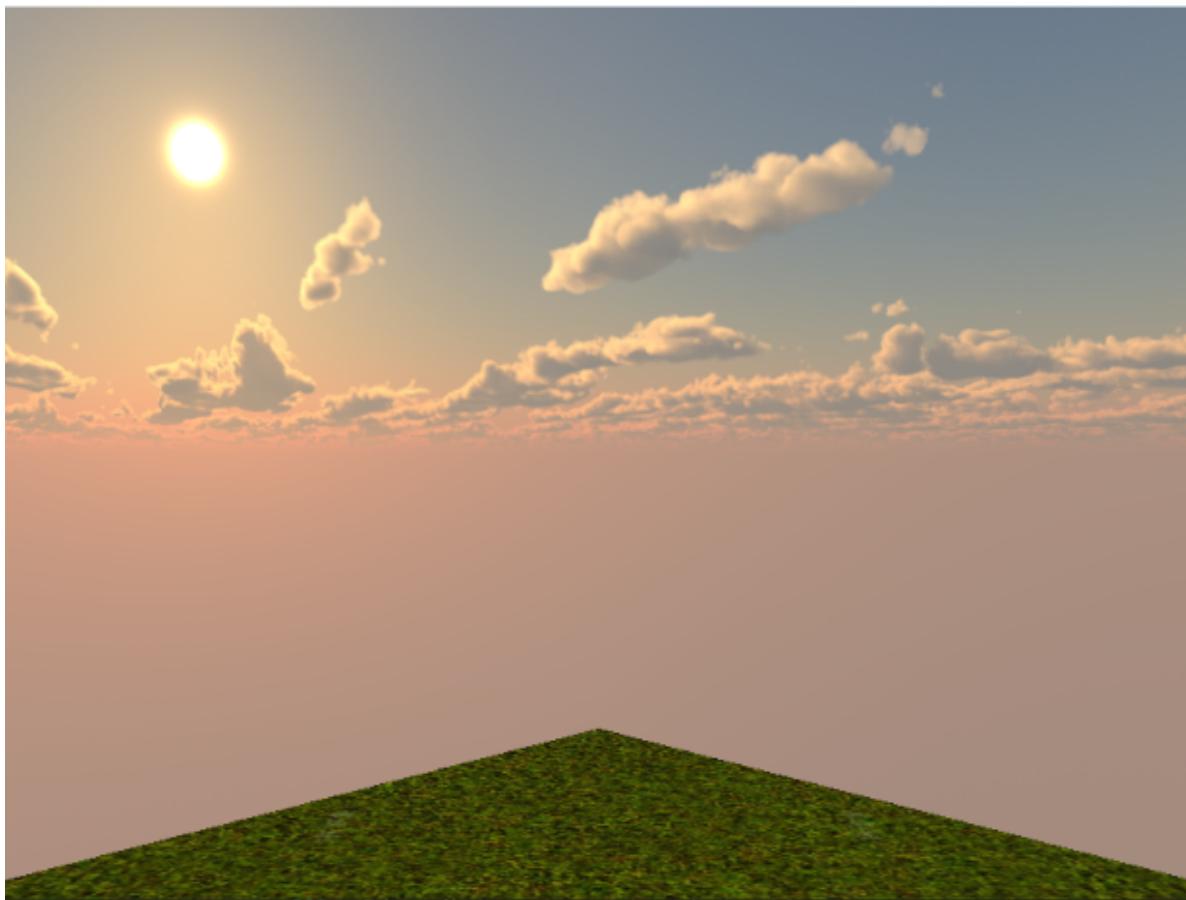
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, texture2.imageWidth,
    texture2.imageHeight, GL_RGB, GL_UNSIGNED_BYTE,
    texture2.image);
//将每个立方体分成六个面绘制
for (i = 0; i < 6; i++)
{
    if (i == 2) {
        glBindTexture(GL_TEXTURE_2D, texture1.ID);
        //glBindTexture(GL_TEXTURE_2D, shadowMapTexture);
        glBegin(GL_POLYGON);

        glTexCoord2f(0.0, 0.0); glVertex3fv(vertex[surface[i][0]]);
        glTexCoord2f(0.0, 1.0); glVertex3fv(vertex[surface[i][1]]);
        glTexCoord2f(1.0, 1.0); glVertex3fv(vertex[surface[i][2]]);
        glTexCoord2f(1.0, 0.0); glVertex3fv(vertex[surface[i][3]]);
    }
    else {
        glBindTexture(GL_TEXTURE_2D, texture2.ID);
        glBegin(GL_POLYGON); //指定绘制方式, 填充多边形 GL_POLYGON 不填充多边形
GL_LINE_LOOP

        glTexCoord2f(0.0, 0.0); glVertex3fv(vertex[surface[i][0]]);
        glTexCoord2f(0.0, 1.0); glVertex3fv(vertex[surface[i][1]]);
        glTexCoord2f(1.0, 1.0); glVertex3fv(vertex[surface[i][2]]);
        glTexCoord2f(1.0, 0.0); glVertex3fv(vertex[surface[i][3]]);
    }
    //glDrawPixels(256, 256, GL_LUMINANCE, GL_FLOAT, depthImage);
    Vector3 p1 = Vector3(vertex[surface[i][1]][0] - vertex[surface[i][0]][0],
    vertex[surface[i][1]][1] - vertex[surface[i][0]][1], vertex[surface[i][1]][2] -
    vertex[surface[i][0]][2]);
    Vector3 p2 = Vector3(vertex[surface[i][3]][0] - vertex[surface[i][0]][0],
    vertex[surface[i][3]][1] - vertex[surface[i][0]][1], vertex[surface[i][3]][2] -
    vertex[surface[i][0]][2]);
    Vector3 n=p1.crossProduct(p2).normalize();
    if (i % 2 == 1)
        n = n * (-1.0f);
    //cout <<i<<" "<<n.x <<" "<<n.y <<" " " <<n.z << endl;
    glNormal3f(n.x, n.y, n.z);
    glEnd();
}
}

```

实现效果:



加上树后：



7、交互摄像机

```
void Camera::yawCamera(float speed) {
    Vector3 eye = m_Position - m_View;
    float rad = speed * 3.14159 / 180.0f; //求旋转角度的弧度
    Vector3 translate[3];
    translate[0] = Vector3(cosf(rad), 0, sinf(rad));
    translate[1] = Vector3(0, 1, 0);
    translate[2] = Vector3(-sinf(rad), 0, cosf(rad));
    Vector3 res;
    res.x = eye.x * translate[0].x + eye.y * translate[1].x + eye.z * translate[2].x;
    res.y = eye.x * translate[0].y + eye.y * translate[1].y + eye.z * translate[2].y;
    res.z = eye.x * translate[0].z + eye.y * translate[1].z + eye.z * translate[2].z;
    m_Position = m_View + res;
}

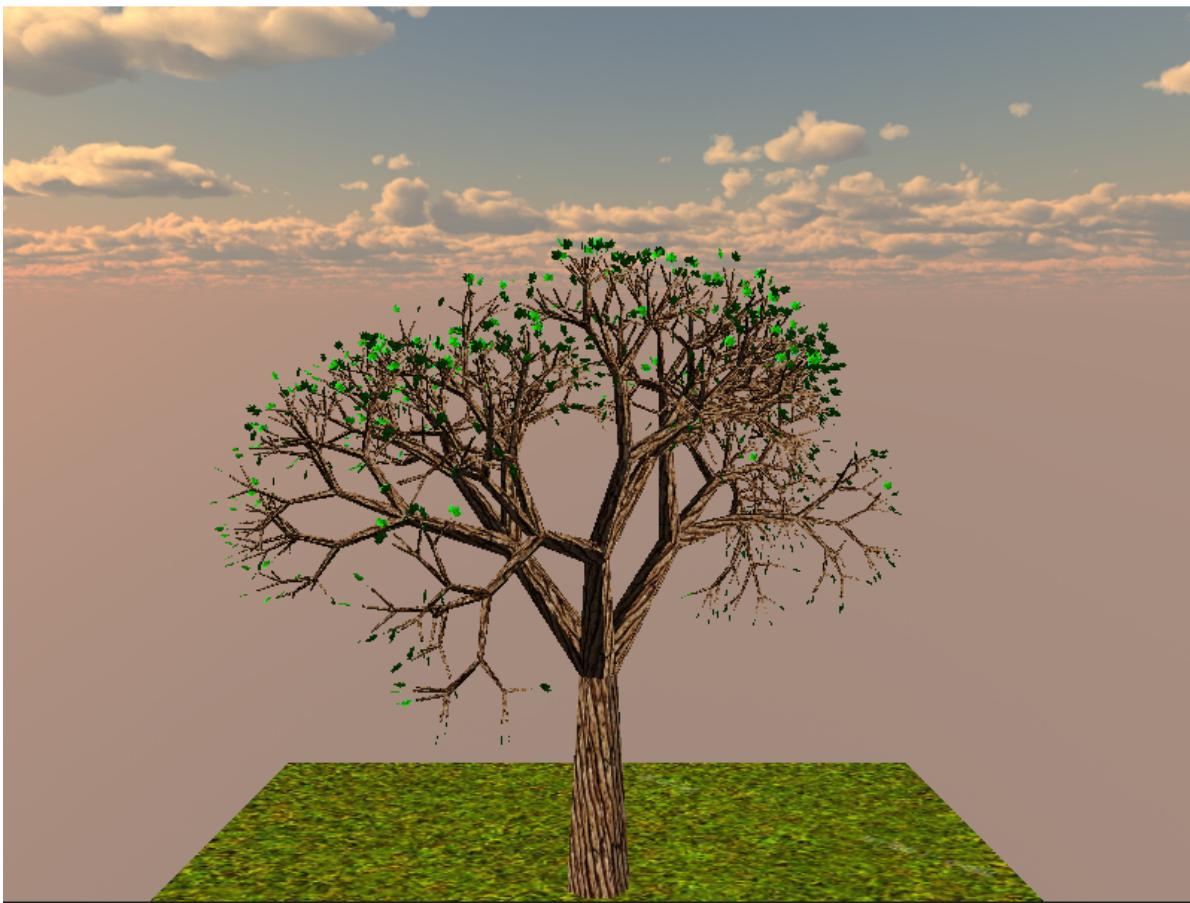
void Camera::rotateCamera(float speed) {
    Vector3 eye = m_Position - m_View;
    float rad = speed * 3.14159 / 180.0f; //求旋转角度的弧度
    Vector3 n = eye.crossProduct(m_UpVector).normalize();
    Vector3 translate[3];
    translate[0] = Vector3(n.x * n.x * (1 - cosf(rad)) + cosf(rad), n.x * n.y + (1 - cosf(rad)) * n.z + sinf(rad), n.x * n.z + (1 - cosf(rad)) * -n.y + sinf(rad));
    translate[1] = Vector3(n.x * n.y * (1 - cosf(rad)) - n.z * sinf(rad), n.y * n.y * (1 - cosf(rad)) + cosf(rad), n.y * n.z * (1 - cosf(rad)) + n.x * sinf(rad));
    translate[2] = Vector3(n.x * n.z * (1 - cosf(rad)) + n.y * sinf(rad), n.y * n.z * (1 - cosf(rad)) - n.x * sinf(rad), n.z * n.z * (1 - cosf(rad)) + cosf(rad));
    Vector3 res;
    res.x = eye.x * translate[0].x + eye.y * translate[1].x + eye.z * translate[2].x;
    res.y = eye.x * translate[0].y + eye.y * translate[1].y + eye.z * translate[2].y;
    res.z = eye.x * translate[0].z + eye.y * translate[1].z + eye.z * translate[2].z;
    if (res.crossProduct(m_UpVector).dotProduct(n) < 0) {
        m_UpVector = m_UpVector * (-1.0f);
    }
    m_Position = m_View + res;
}
```

用键盘输入的方式获取交互过程：

```
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case '1':
            myCamera.setSpeed(1.0f);
            break;
        case '2':
            myCamera.setSpeed(10.0f);
            break;
        case 'W':
            myCamera.rotateCamera(myCamera.getSpeed());
            break;
        case 'S':
            myCamera.rotateCamera(-myCamera.getSpeed());
            break;
        case 'A':
            myCamera.yawCamera(myCamera.getSpeed());
            break;
        case 'D':
            myCamera.yawCamera(-myCamera.getSpeed());
            break;
        case 'Q':
            myCamera.moveCamera(myCamera.getSpeed());
            break;
        case 'Z':
            myCamera.moveCamera(-myCamera.getSpeed());
            break;
    }
}
```

实现效果：

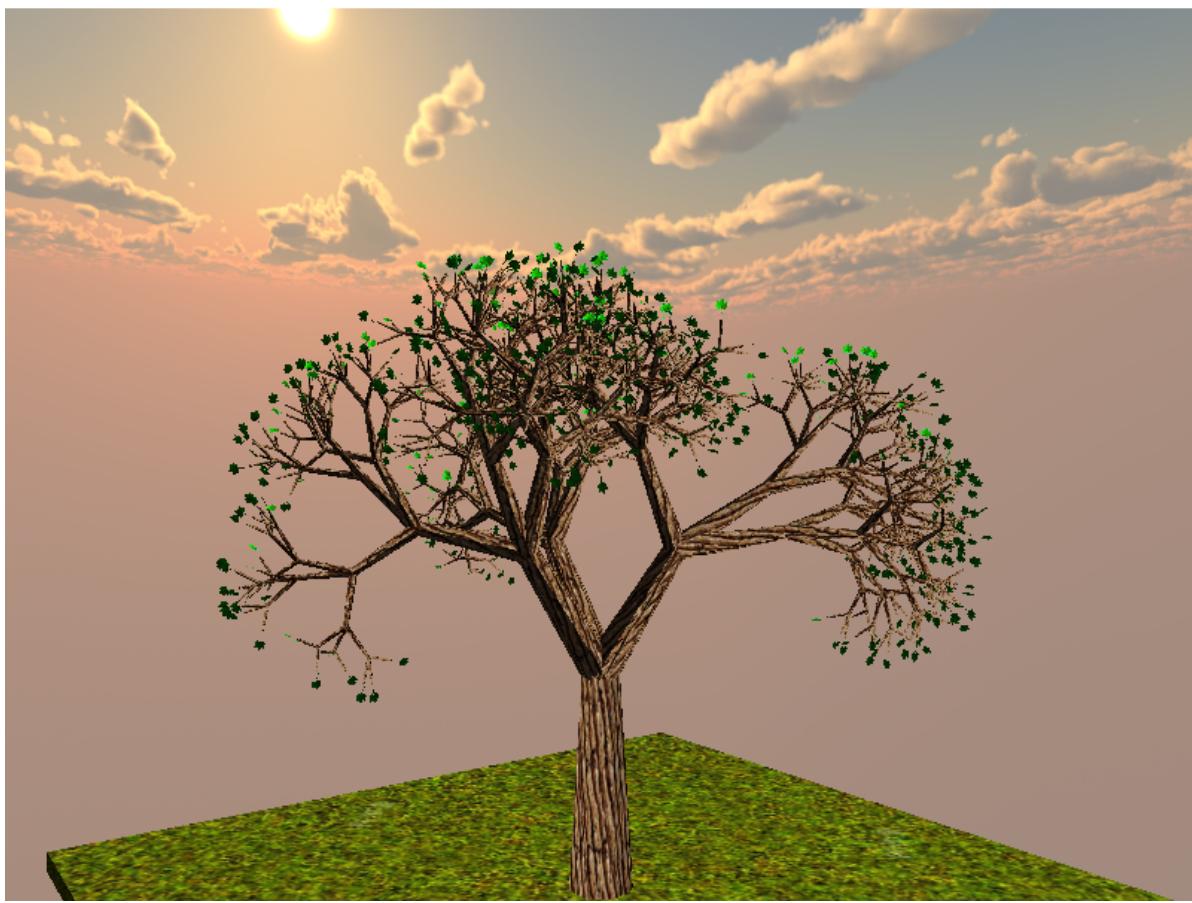
初始状态：



左旋：



右旋：



俯视：



仰视：



缩小:



缩小:



8、四季变换

思路：通过改变颜色的值和叶子的半径以及透明度实现。

```
//实现树叶春夏生长变换
case '7':
    myLSystem.leafRadius *= 0.97;
    if (myLSystem.leafRadius < 0.1)
        myLSystem.leafRadius = 0.1;
    break;
case '8':
    myLSystem.leafRadius /= 0.97;
    if (myLSystem.leafRadius > 5.0)
        myLSystem.leafRadius = 5.0;
    break;
//春夏秋冬四季变换
case '0':
    myLSystem.leafRadius /= 0.97;
    if (myLSystem.leafRadius > 5.0) {
        myLSystem.leafRadius = 5.0;
        p += 4;
        if (p > 255) {
            p = 255;
            q -= 5;
            if (q < 0)
                q = 0;
        }
    }
```

```
    }  
    break;
```

实现效果

春:



夏:



秋:

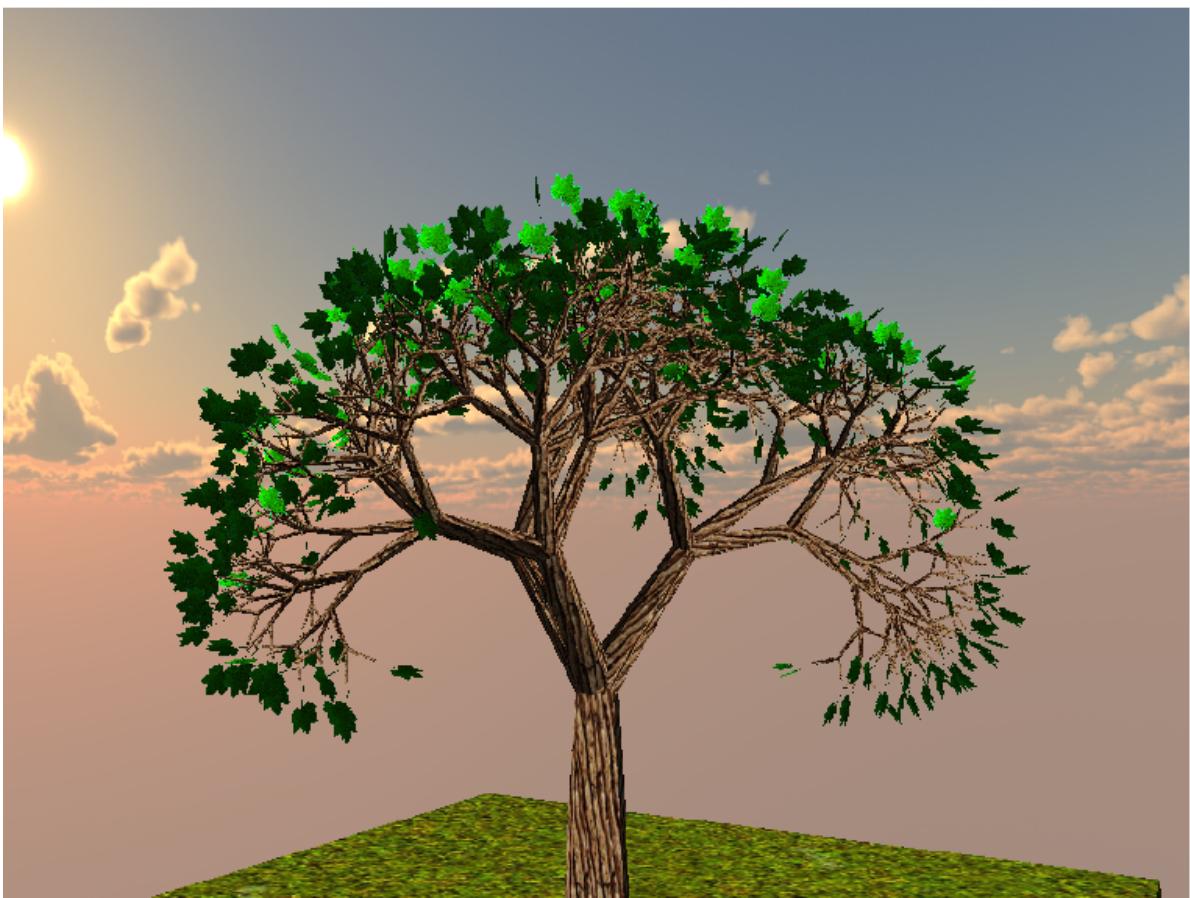


冬:



最终实现效果

迭代6层



迭代8层



迭代7层及其余交互效果参考/demo中第三组——构建l-system分形树.mp4

总结&&个人心得

伍斌：

本次项目由小组分工合作完成，除去由于时间原因未能实现阴影外，项目实现得还算顺利。我主要负责实现天空盒和地面、春夏秋冬变换，交互摄像机、传统光照、项目整合调参封装打包、实验文档编写、PPT制作等工作，最大的实现难度除去天空盒实现外，在于调整参数使得树、地面、光照和天空盒成为一个比较优美的整体，在此过程中我对opengl及相关的库文件有了更深入的，可以说痛彻心扉的理解。我从这次项目中收获良多，从初期查阅资料、设计项目结构到参与每一步的实现和最终打包，尽管存在与小组成员之间相当多的沟通障碍，比如找不到人、大家时间都不够等，但是最终还是熬了好几天把项目实现了。这对我的合作能力以及编码能力是一次极大的提升。

郎森淋：

作为中山大学计算机学科中一门专业选修课程，《计算机图形学》能够说是对我们之前所学的一些知识进行了总结，如离散数学、数字图像处理等。更重要的是，经过这门课程，我们学生从单纯理论上的学习一步步地过渡到了画面渲染的实践。在高老师丰富的授课资料以及展示精彩视频的这种生动的授课方式，让原本死巴巴的知识点变得有吸引力。在助教精心准备的Assignment中，让我对作业的信心满满，少了那种毫无头绪的慌张。私下里也会问助教一些比较呆的问题，助教也是很耐心的帮助我，真的很谢谢！最后是我们小组的宝们，虽然每个人的分工不同，但大家齐心协力真的很帅！

聂硕琳：

本次项目主要是实现L-system分形树。我通过阅读相关资料了解到：L-system被认为是植物生长的数学理论，常用于构建树的3d模型，可用于虚拟室外景观，是一种非常实用的技术，于我个人而言是我较为喜欢的研究方向。因为本次项目涉及到的内容广泛且整体工程代码量较为庞大，各个组员对工作量进行了详细的分工，但在衔接工作时还是需要磨合的时间。在与同组同学讨论中，我从他们的思路中获得了崭新的启发，也解决了自己代码存在的问题，收获了许多。我个人负责的是实现树的春夏秋冬变化的部分，最终完成了通过改变叶片参数改变最终效果的工作。同时，在实验中我也对3d图像渲染的各方面有了跟深入的了解。

郭锦：

这次大实验是对这一个学期知识的回顾与应用。通过这次实验，我对光照模型的理解更加熟悉，对冯氏光照模型的环境光、漫反射光、镜面反射光熟记于心；我也对 OpenGL 中的摄像机观察空间更加深入，更加深刻地理解了相机的朝向、位置、观察点；同时我学习到了 L-system 分形图形文法的相关知识。这次实验同样是一次团队项目，小组中合理分工，团结合作，共同学习，一起进步，在我们的共同努力下，设计出了满意的 3D 场景。我们积累了宝贵的经验，留下了美好的回忆。
