L系统真实感树木——计算机图形学静态真实感实物项目报告

1. 目的

使用所学的图形学知识画出具有真实感的静态树木

1. 工具

使用L-系统实现对真实树木的分形描述，使用opengl和c++将L-系统描述转化为图形

1. 方法
   1. L-系统
      1. 简介

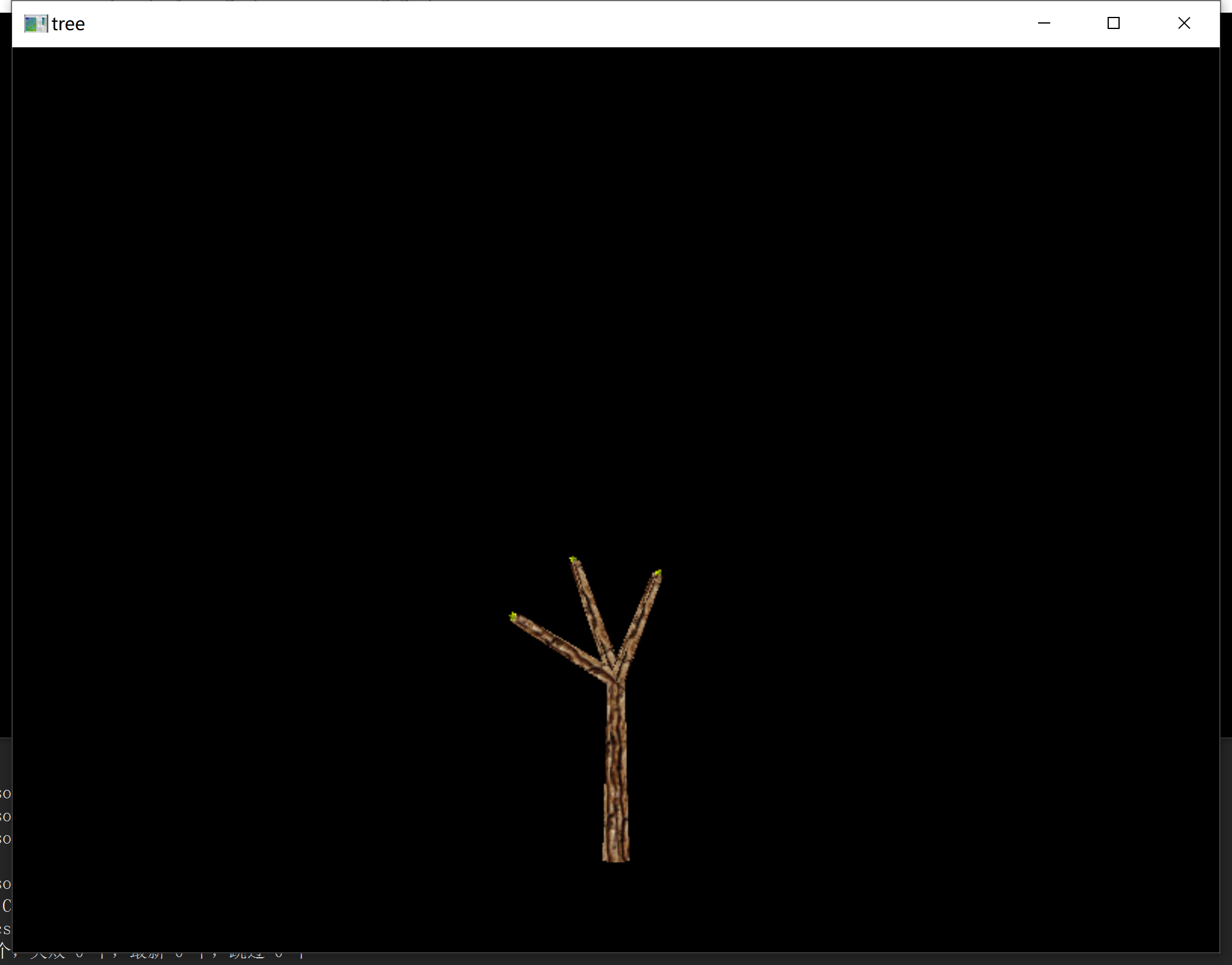
L系统是Lindenmayer系统的简称，是由荷兰Utrecht大学生物学家和植物学家Lindenmayer于1968年提出的有关生长发展中的细胞交互作用的数学模型，被广泛应用于植物生长过程的研究。L系统是一系列不同形式的正规语法规则，可以用于植物生长过程的建模。也可用于生成自相似的分形。此项目主要应用了L系统可以生成自相似的分形的特点，结合树木结构的分形特性生成具有真实感的静态树木。

* + 1. 语法
       1. 语法简介

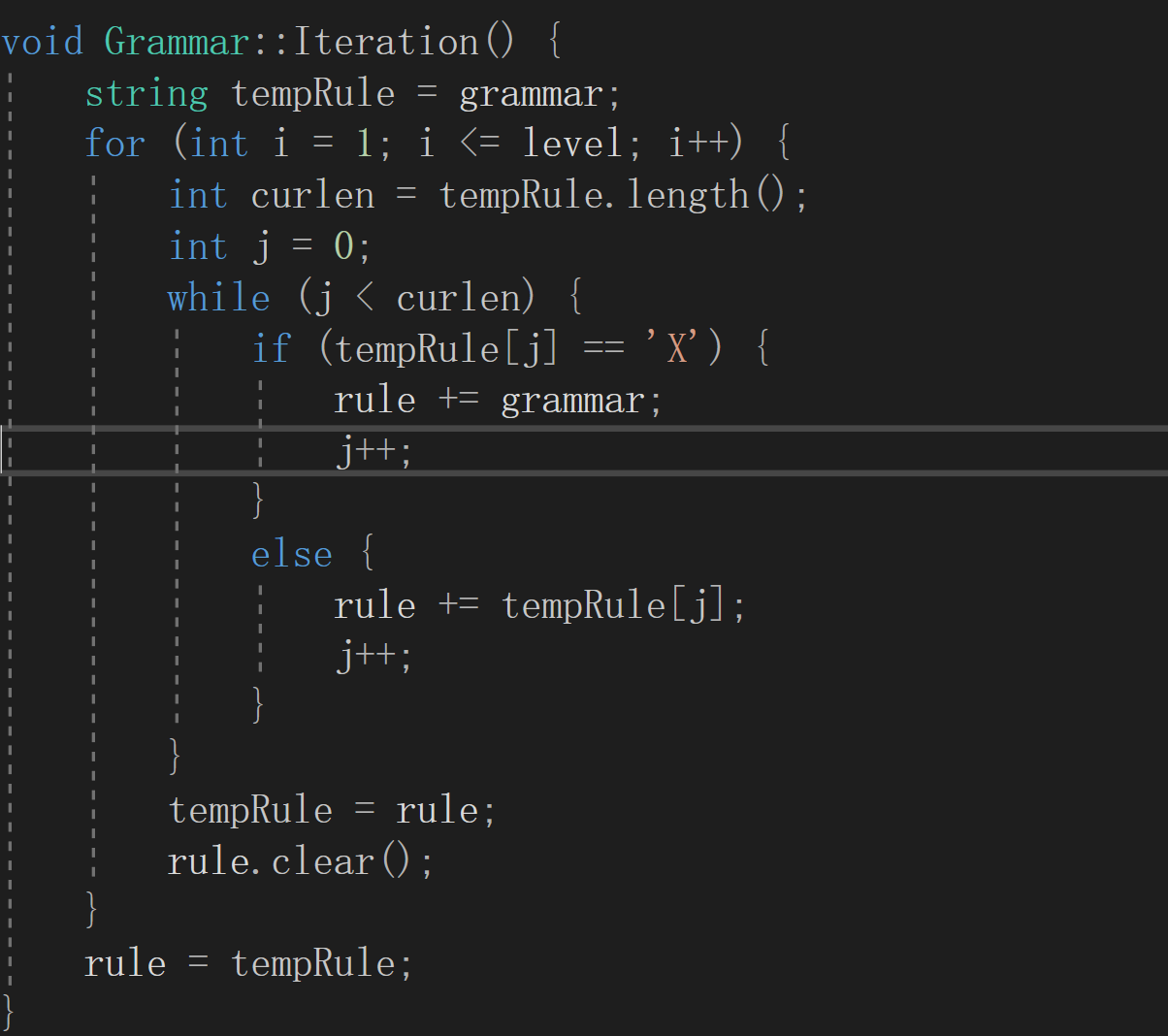
这里用到的语法为FA[\*+X][-/&X][/%X]B，其中非终结符为{X}，其余为终结符，语法每个符号的含义为：

* F：树干
* A：半斤收缩表示子树开始
* +（-）：绕x轴逆时针（顺时针）旋转25度
* \*（/）：绕y轴逆时针（顺时针）旋转25度
* &（%）：绕z轴逆时针（顺时针）旋转25度
* B：收缩半径表示子树结束
* [（]）：标志子树生成正式开始（结束）
  + - 1. 语法迭代过程

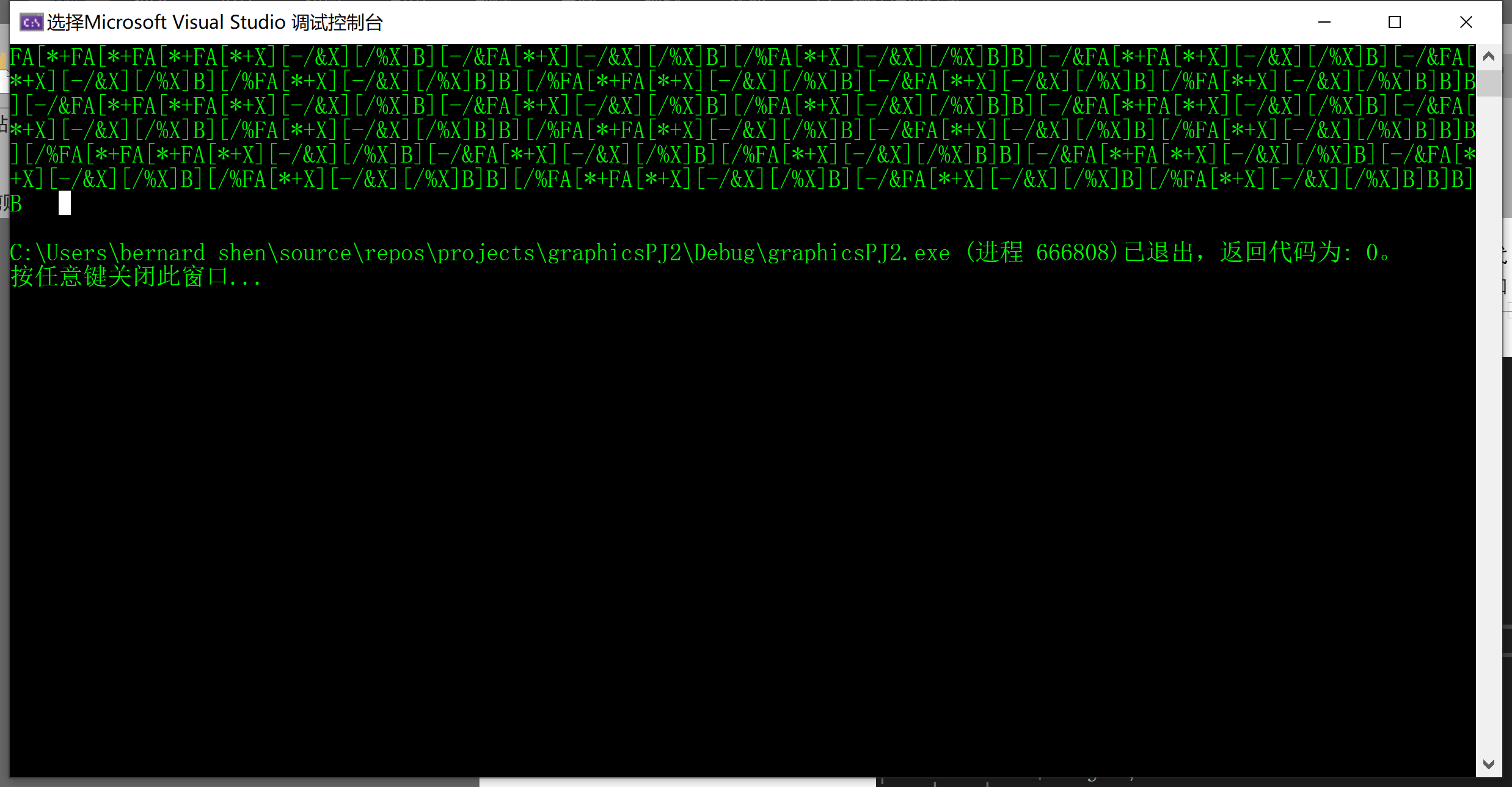
上述语法仅为L系统第0次迭代的结果，如果仅仅按照上述语法生成树木则效果如下：



可见仅为几个简单的线条，因此需要对上述文法进行多次迭代，迭代算法大致思想为遇到其中的非终结符X就展开为整个语法规则，代码如下：



其中level参数为迭代的层数，经过三层迭代后最终的语法规则如下：

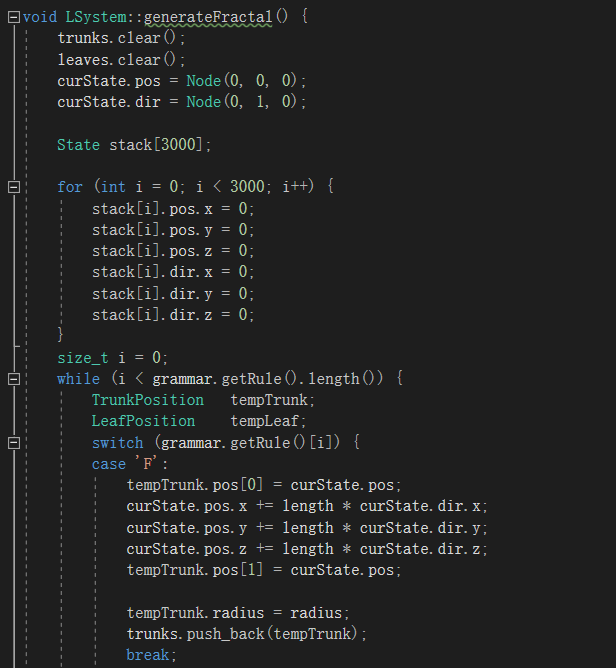


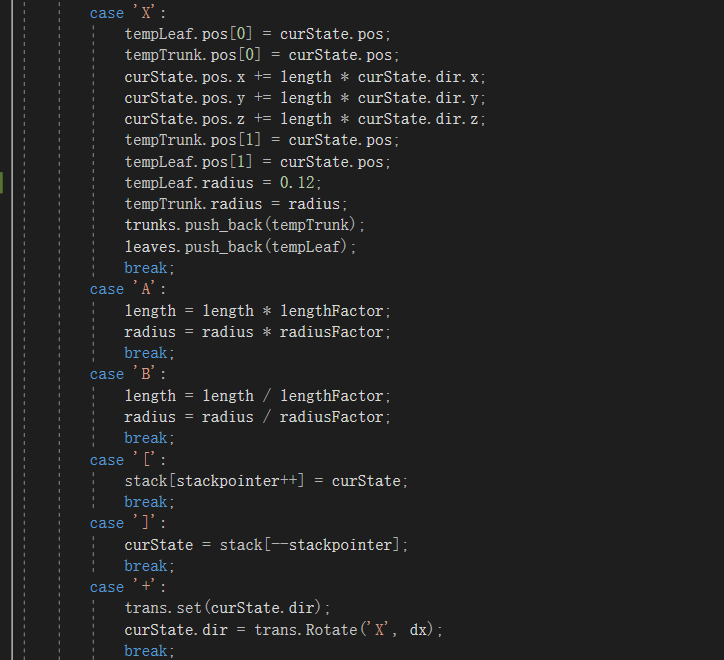
* 1. 生成算法
     1. 基本思路

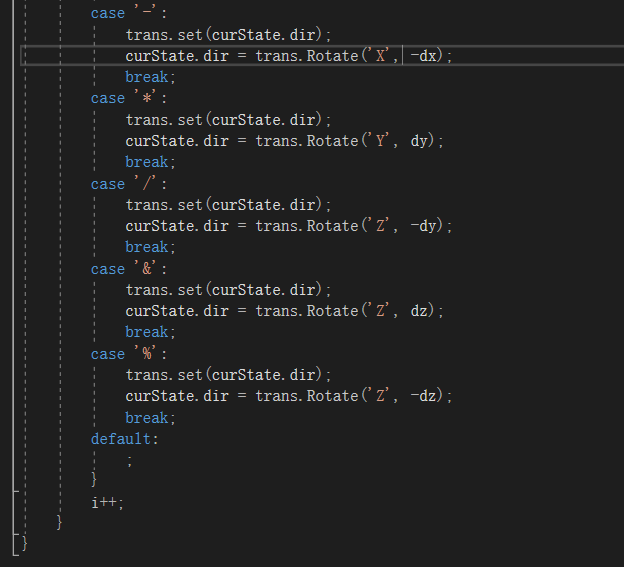
算法的基本思路是根据之前迭代得到的规则，逐一进行扫描字符，对不同字符进行特殊处理，主要任务为生成树干，由于树干可以由圆柱形表示，因此需要指导圆柱体开始的坐标以及圆柱体的延申方向，将圆柱体的开始坐标以及延申方向作为当前状态：

* F：画树干：生成一个圆柱体
* A：子树开始：对圆柱体的初始半斤乘以收缩因子收缩半径，得到略细的树干
* [：子树正式开始：将当前的绘画状态入栈以便于子树生成结束后恢复
* ]：子树生成完成：将战中的状态恢复
* +（-）：子树绕x轴逆时针（顺时针）旋转：将当前生成状态的方向绕x轴逆时针（顺时针）旋转25度
* \*（/）：子树绕y轴逆时针（顺时针）旋转：将当前生成状态的方向绕y轴逆时针（顺时针）旋转25度
* &（%）：子树绕z轴逆时针（顺时针）旋转：将当前生成状态的方向绕z轴逆时针（顺时针）旋转25度
* B：子树：将圆柱体半径除以收缩因子进行扩大半径得到略粗的圆柱体树干
* X：树叶：生成一个正方体作为树叶的初始形状之后通过加载贴图使其具有树叶的纹理，最后将超出纹理的部分变为透明

树生成的算法如下：







至此生成了整棵树的结构，包括树干和树叶。其中值得注意的是对于树叶的处理，树叶的处理不同于枝干使用了固定大小的生成半径，这么做主要是为了使树叶的大小不受其所在枝干的粗细所影响。

此函数推出后生成两个列表一个为树枝列表，包含每根树枝的起始坐标和方向向量。另一个为树叶列表，包含每片树叶的起始坐标和方向向量，之后只要对这两个列表中的元素进行逐一绘制就可以得到整颗分形树。

* + 1. 算法流程

这部分主要介绍算法整体的执行流程，包括主函数、绘制枝干、绘制树叶。

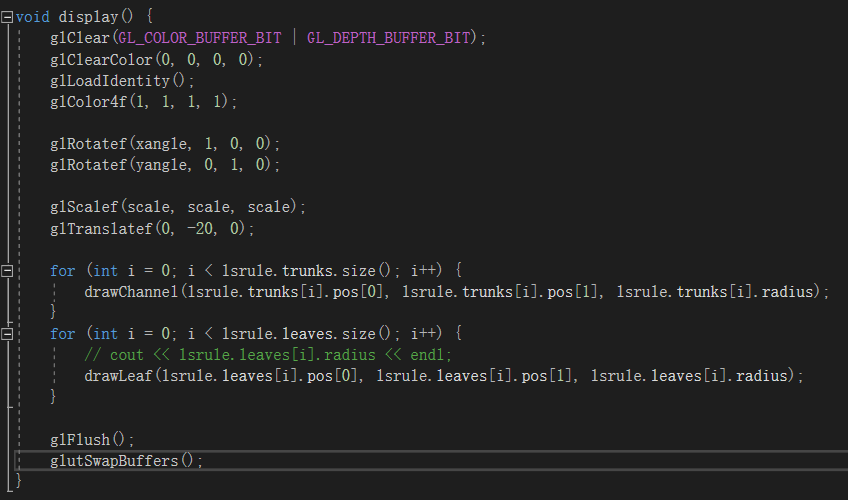
* + - 1. 主函数



这里主要使用glut库，其中glutReshapeFunc负责调用改变窗口大小的事件处理函数reshape；glutDisplayFunc负责调用绘制图形的函数display；glutMousFunc负责调用鼠标事件处理函数mouse；glutMotionFunc负责调用动作处理函数motion；glutKeyboardFunc负责调用键盘事件处理函数keyboard；init函数主要负责对窗口进行初始化、调色等一系列操作。

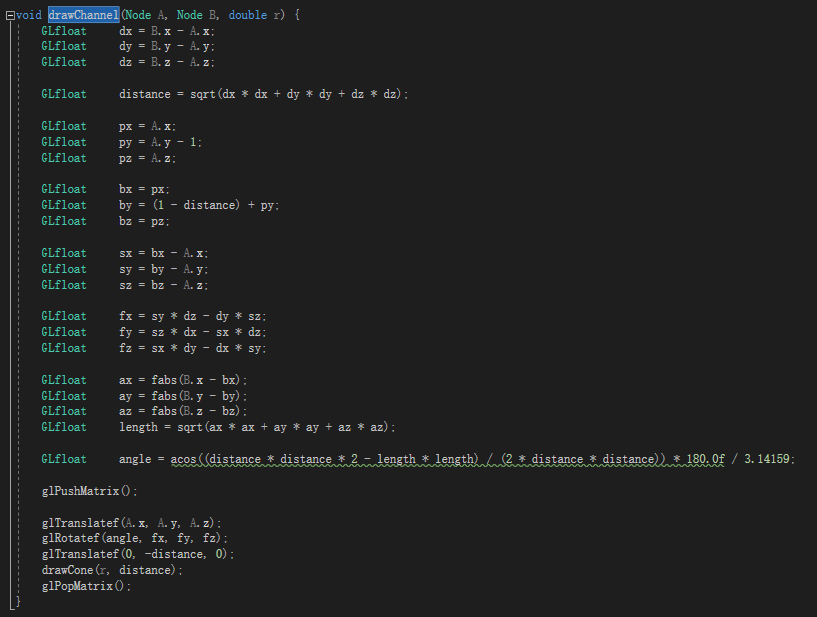
* + - 1. 图形绘制

图形绘制主要由display函数负责进行：

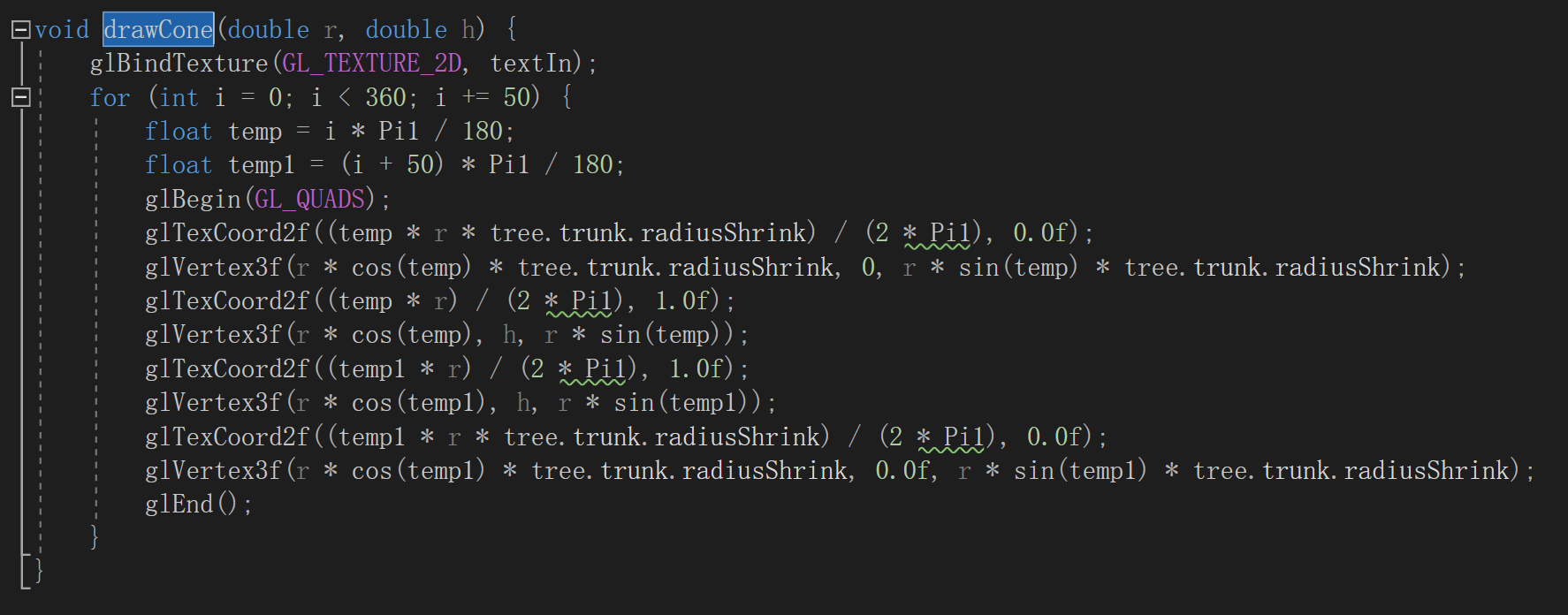


Display对之前由generateFractal函数生成的树结构进行逐一处理，对于树枝列表中的每个元素，调用drawChannel进行树枝绘制，对树叶列表中的每个元素调用drawLeaf进行逐一绘制。下面将逐一介绍绘制树干和树叶的方法。

* + - * 1. 枝干绘制

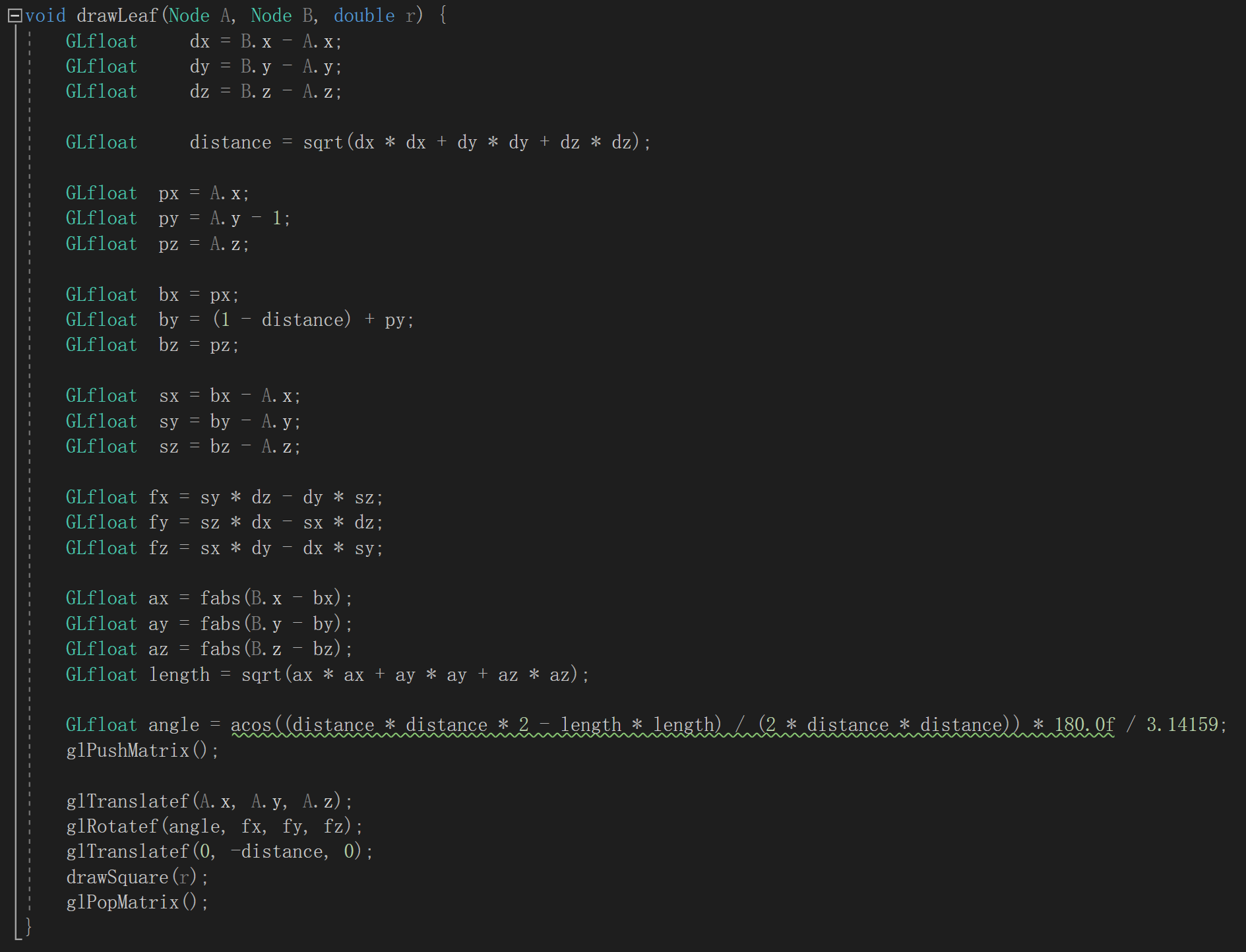


此函数先计算出圆柱的起点与方向，然后通过glTranslatef和glRotatef使当前绘制起点到圆柱的七点，然后调用drawCone绘制圆柱：

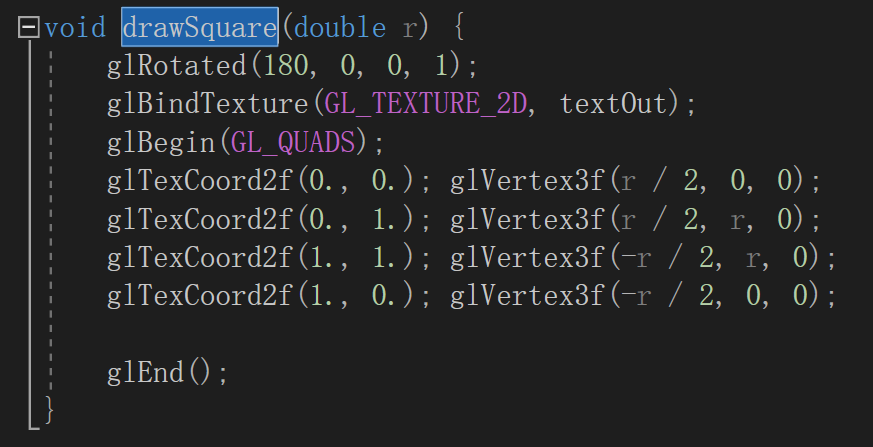


此函数用棱柱逼近圆柱，每次函数画棱柱的一个表面。

* + - * 1. 树叶绘制



此函数先计算树叶的起始坐标以及方向和大小等参数，然后利用glTranslatef和glRotatef将起始位置移动到相应位置，最后调用drawSquare画出方块：



此函数根据半径计算出四个顶点并画出方形。

* + 1. 增加真实感
       1. 生成更真实的树干

要生成更真实的树干主要可以从两个方向入手，首先是树干的纹理，其次是树干的粗细，下面我将介绍这两方面的优化。

* + - * 1. 树干纹理

树干纹理主要使用opengl的纹理功能，使用了网上找到的树干纹理：



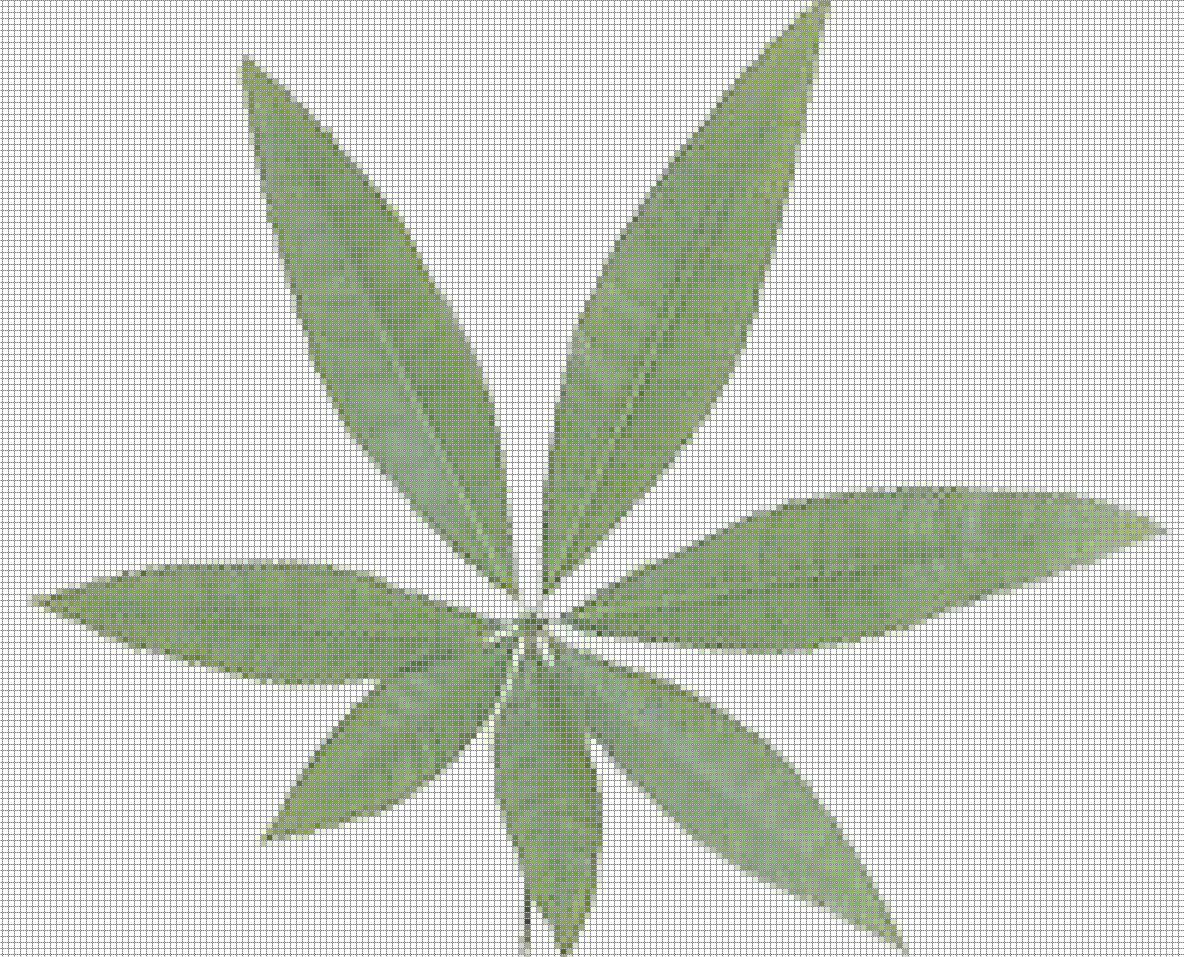
在前面的绘制树干的过程中使用glBindTexture绑定树干的纹理，在绘制断点之前使用glTexCoord2f指定纹理的坐标。由此得到带有纹理的多棱柱。

* + - * 1. 树干粗细

这方面主要考虑到真实世界中的树干往往不是等宽的圆柱，往往呈现上细下粗的形状，因此为了使树干更具真实性，我对树干棱柱的绘制过程使用了半径衰减的方式，使用半径衰减因子，因此最后生成的棱柱的每个面实际为等腰梯形而非长方形。

* + - 1. 生成更真实的树叶
         1. 树叶纹理

树叶方面给主要考虑使用具有真实感的树叶纹理渲染到正方形上：



除此之外需要将不含树叶纹理部分的正方形颜色变为透明，基于以上两点生成的树叶更具真实感。

* + - 1. 子树组合

子树组合主要满足子树小于父树，加上之前对于树干的半径收缩因子，将子树的树干初始半径设置为父树末端的半径（见前面generateFractal函数），因此剋将子树与父树无无缝衔接。

* + - 1. 初始语法更新

初始的语法在树的生成上会产生树只向一个方向延申的效果，效果图如下：



上述的树可以理解为右边为物理意义上的南面，因此枝叶由于向光性向南方向延申，但是这不是很符合一般意义上的树木，这主要是因为初始语法上存在的问题，因此需要对迭代前的初始语法进行修改。

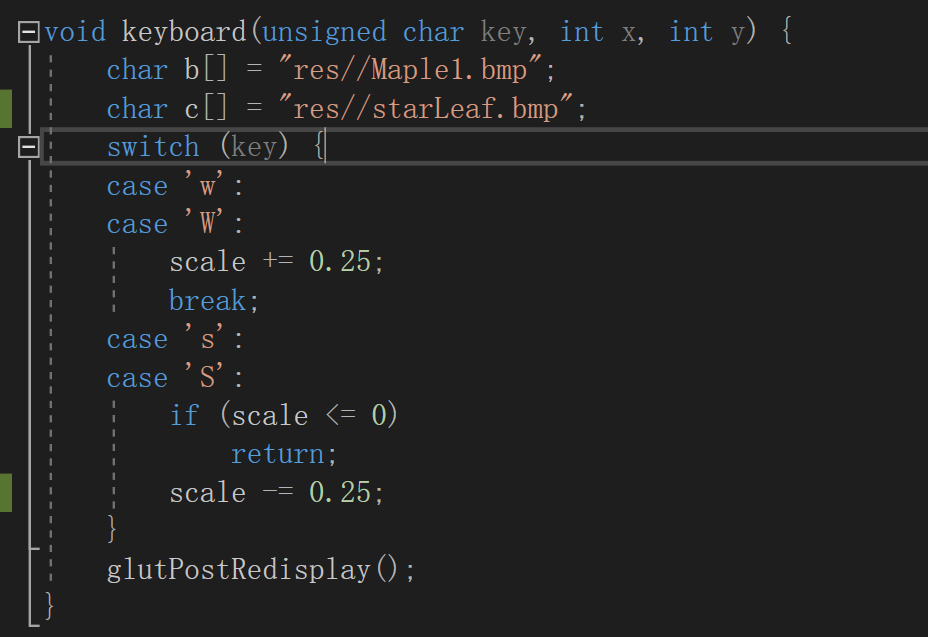
初始语法为：FA[\*+X][-/&X][/%X]B，初始的方向向量为(0,1,0)为y轴正方向，可以得到每个子树的方向向量为：

由此可见第一棵和最后一棵子树在y方向上的的旋转都是白费的，基于以上观察对旋转稍作调整：FA[\*+&X][-/&X][/%X]B使三棵子树的方向产生一点偏差由此产生更具真实感的树：

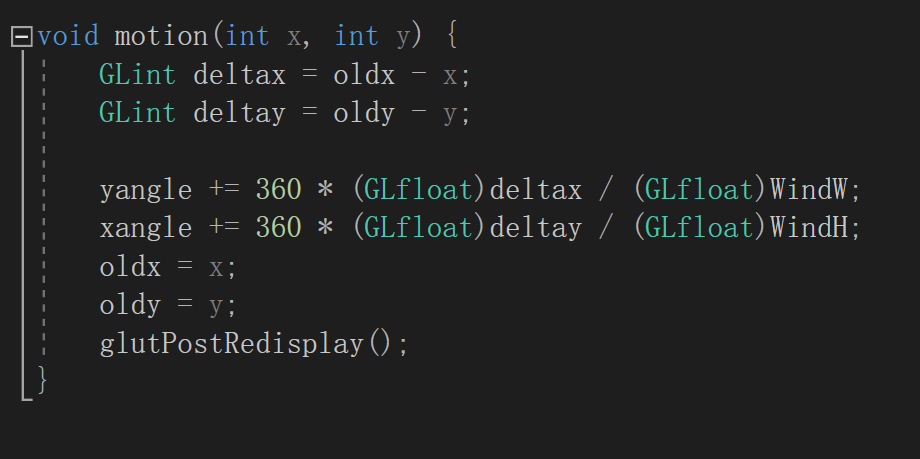


* + 1. 操作响应

除了在生成树之外还对，我还对操作进行了一定的响应，对于鼠标拖动操作我进行了画面的旋转，使用w和d分别控制放大、缩小，代码如下：



此函数响应键盘的操作，对界面进行放大、缩小。



此函数负责响应鼠标的拖拽旋转功能。

1. 效果展示

最终效果如下：



使用调整过后的文法迭代三次得到的树结构。

1. 项目感想

通过此次项目，我学习到了图形学库opengl的基本使用方式，了解了分形的概念以及L系统，跟深入的理解了课上学到的图形学知识。