计算机图形学大作业设计报告

学号：515030910035　　姓名：吕嘉伟

总体概述

本次大作业使用OpenGL实现了一个基于粒子系统的“私人小天地”，有花草树木、假山喷泉、房屋栅栏，也有各种小动物，结合粒子系统实现各种效果。

编程环境

- 操作系统：windows10
- IDE：Visual Studio 2015
- 使用的库：OpenGL,GLUT,GLEW,GLAUX,windows API,eigen

实现的内容

1.天空盒和地面的建模
2.场景光照，分为点光源和聚光灯形式
3.场景建模，包括树木花草、房屋、栅栏、喷泉、篝火等
4.摄像机的实现，包括镜头旋转，水平和竖直移动
5.obj文件及其对应纹理mtl的读取与显示
6.粒子系统，包括多种发射模式和运动形式，包含部分物理仿真
7.鼠标键盘的交互事件
8.文字提示信息的绘制

操作说明

- 视角水平竖直移动：WASD
- 视角旋转：按住鼠标左/右键并移动鼠标
- 镭射粒子方向控制：IJKL
- BGM音乐开关：M
- 场景灯光切换：Z
- 粒子运动状态切换：X
- 粒子纹理切换：C
- 打开菜单选择显示的粒子：鼠标中键

整体架构

文件名	描述
camera	摄像机类，控制视角，可进行水平、竖直移动以及视角的旋转
utility	工具类，包含常用工具函数，例如读取纹理文件
sky	天空盒类，使用六面天空盒将整个场景包围起来
ground	地面类，使用纹理贴图的方式绘制不同的地面
wall	围墙类，使用纹理贴图的方式绘制包围“小天地”的围墙
fence	栅栏类，通过建模单个长方体，绘制整个场景中的所有栅栏，区分地区
tree	树木和花类，通过多边形建模，根据参数绘制树木和花
particle	粒子和粒子系统类，包含粒子类，即单个粒子的属性；粒子系统类，即发射器
objLoader	obj类，用于读取obj文件及其对应的mtl纹理，并绘制
drawScene	用于绘制部分场景，如房屋、喷泉、篝火等

具体实现

Camera类

基本原理

主要应用了UVN相机的原理并自己加以改进。

相机注视的向量为N，相机的上方向向量为V，相机的右方向向量为U。当要改变相机位置和朝向的时候，只需要将UVN矩阵和相应的变换矩阵相乘即可。换句话说，我们定义了一个视景矩阵，把世界坐标系转换成UVN坐标系，然后由UVN坐标系来解释世界坐标系中物体的坐标。

平移

- 水平移动：我的做法是将相机位置所在的点沿着N或U向量（即相机注视方向或右方向）在XZ平面上的投影向量移动相应的距离，以保证Y轴的高度不变。
- 竖直移动：比较简单，将相机位置所在点沿着Y轴方向向上或向下移动相应距离。

旋转

- 基础旋转：主要由roll、yaw、pitch三种基本旋转组成
    - roll：绕N轴旋转,U'=Ucos - Vsin, V'=Usin + Vcos
    - yaw：绕V轴旋转,N'=Ncos - Usin, U'=Nsin + Ucos
    - pitch：绕U轴旋转,V'=Vcos - Nsin,N'=Vsin + Ncos
- 进阶旋转：通过roll、yaw、pitch的组合实现沿*屏幕*水平竖直旋转，以及相机自身旋转
    - rotateX：通过鼠标交互传进的参数，进行简单变换后调用yaw()
    - rotateY：通过鼠标交互传进的参数，进行简单变换后调用pitch()
    - rotateRoll：通过鼠标交互传进的参数，进行简单变换后调用roll()

utility

utility中有两个函数，`GLint LoadTexture(const char*)`和`bool BuildTexture(char *szPathName, GLuint &texid, BYTE r, BYTE g, BYTE b)`，前者是我自己一开始实现的，可以实现读取BMP位图并绑定到纹理句柄。但在实际使用的过程中，读取有些BMP图时会出现有黑色背景的情况，一直无法解决，后来找到了别人实现的第二种方法，可以将指定的rgb颜色设为透明。

读取**BMP**位图**`GLint LoadTexture(const char*)`**

自己实现的函数很容易理解，首先创建一个GL_TEXTURE_2D的句柄并绑定，然后读取文件，为了方便，我调用了Glaux库的函数`auxDIBImageLoad()`，它将图片文件读入并返回一个类型未`_AUX_RGBImageRec*`的指针，包含了图片的信息，例如sizeX，sizeY等。我使用了`gluBuild2DMipmaps()`来创建纹理贴图，原因是使用`glTexImage2D()`时所采用的位图文件分辨率必须为：2^n*2^n，而`gluBuild2DMipmaps()`支持任意分辨率位图文件。

之后使用`glTexParmeteri()`函数来确定如何把纹理象素映射成像素。

- GL_REPEAT：重复边界纹理
- GL_TEXTURE_MAG_FILTER: 放大过滤
- GL_LINEAR: 线性过滤, 使用距离当前渲染像素中心最近的4个纹素加权平均值.
- GL_TEXTURE_MIN_FILTER: 缩小过滤
- GL_LINEAR_MIPMAP_NEAREST:使用GL_NEAREST对最接近当前多边形的解析度的两个层级贴图进行采样,然后用这两个值进行线性插值.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_LINEAR);
```

Sky类

实现的方式比较简单，先读取6张天空盒图片作为纹理，分别是前、后、左、右、上、下，然后再指定的位置绑定纹理使用GL_QUADS画矩形，包围成一个长方体盒子，注意纹理映射时的坐标。

Ground类

原本想实现的是有高低差的地形terrain，后来因时间不够暂时放弃了这一想法，准备在后续继续开发过程中实现。现实现的是简单使用纹理贴图进行绘制的平面地板。首先读入四张纹理图，然后以(0,0,0)为中心将整个场景分为4块，每块由多小块构成，主要是考虑到光照的原因，由多个小块构成的情况下，在*光照下*表现的更为细腻。

```
--------------------
|        |        |       整个场景分为4个大块，为200x200大小
|   1    |    2   |       每个大块分为16个小块，为50x50大小
|        |        |
-------------------|
|        |        |
|   3    |    4   |
|        |        |
--------------------
```

Wall类

实现方法和前两者大同小异，读入纹理图后进行贴图绘制，要注意的点是各个面法向量的方向设置，不然会出现光照异常的情况。具体布局如下。

```
------------------------
|                      |
|     1          2     |     1、2、3大正方形是60x60大小，
|                      |     4小正方形是40x40大小
|                      |
|                  ----|
|                  |
|     3       4    |
|           |------
-------------
```

Fence类

栅栏的实现方法是先建模一个简单的“单体栅栏”，然后再场景中每隔一定距离画一个”单体栅栏“，实现对场景中区域的分割。

单体栅栏建模

单体栅栏的建模分为两种，分别是宽沿着X方向和宽沿着Z方向，两者的外形没有区别。

绘制一个长方体，长宽高分别为1.0f,0.2f,3.0f，使用的是画6个面的方法，面上使用纹理贴图，并注意面的法向量设置，防止光照计算出现问题。

Tree类

树木的实现方法和Fence类似，建模“单个树”，然后再场景中按需求（例如树的高度、树冠宽度、树冠层次数）进行绘制。

单个树建模

一颗树主要用到以下属性：

- height：树的高度，包含树干+树冠
- width：树冠的最大宽度
- levels：树冠的层次数
- textures：纹理

树的建模中使用到了glut库中的`gluNewQuadric()`绘制二次曲面，本次主要使用的是`gluCylinder()`用来画圆柱和圆锥面。

首先画树干，下底面半径为0.5f，上底面半径为0.4f，高度为height*0.4的圆柱面，使用树干的纹理贴图。然后坐标位置上移0.3height，开始绘制树冠。第一个圆锥底面半径为width，高度为height*0.6，之后根据levels的层次数，上移一小段距离后画第二层的圆锥，此时半径和高度均除以二。代码示例如下：

```
double curHeight = size.x() * 0.6, curWidth = size.y();
    for (int i = 0; i < size.z(); i++) {
        gluCylinder(quadric, curWidth, 0.0f, curHeight, 32, 32);
        glTranslatef(0.0, 0.0, curHeight * 0.7);
```

```
        curHeight = curHeight / 2;
        curWidth = curWidth / 2;
    }
```

Flower类

花朵的实现分为两个类，AFlower类对单个花朵进行建模，Flower类绘制一片花朵。

单个花朵建模

花的建模较之树木相比更为复杂，用到的参数如下：

- petal_color：花瓣颜色
- stalk_color：花茎颜色
- petals：花瓣数量
- petal_radius：花瓣半径
- growth：花瓣大小
- stalk_size：花茎长度
- stalk_angle：花茎的旋转角度
- x_axis_stalk：旋转轴向量
- y_axis_stalk：旋转轴向量
- z_axis_stalk：旋转轴向量

在初始化的时候，以上参数的数值均为一定范围内随机生成。

首先画花茎，根据旋转轴旋转对应角度，使用`gluCylinder()`绘制圆柱形，坐标轴上移。

在花茎中间画花叶，使用GL_QUAD_STRIP微分法画四边形带的方法绘制sin/cos函数弧形叶片，部分代码示例如下：

```
radian_angle = 0;
y1 = 0;
glBegin(GL_QUAD_STRIP);
for (angle = 0; angle <= 180; y1 -= 0.08) {
      x1 = petal_radius*sinf(radian_angle);
      z1 = petal_radius*cosf(radian_angle) - petal_radius;

      glVertex3f(-x1, y1, z1);    /* 左右对称 */
      glVertex3f(x1, y1, z1);

      if (angle > 159)
        angle += 10;
      else
         angle += 20;
      radian_angle = (3.1416 / 180)*angle;
}
glEnd();
```

之后绘制花蕊，使用`gluCylinder()`方法。

最后绘制花瓣，方法和画花叶一样，用微分法画四边形带绘制弧形的一片花瓣，重复多次绘制整个花瓣。

粒子系统

粒子系统分为两个类，Particle类和ParticleSystem类，前者是单个粒子，后者是整个粒子系统。

Particle类

Particle类主要有以下属性：

- velo：粒子速度
- acc：粒子加速度
- color：粒子的颜色
- position：粒子的位置
- size：粒子的大小
- angle：粒子的旋转角度
- lifetime：粒子的生命值
- dec：粒子生命衰减速率
- texture：粒子的纹理
- motion_mode：粒子的运动方式
- has_tex：是否含纹理
- is_forever：是否不消失

单个粒子的绘制：

绘制粒子时，我采用的是绘制一个四边形面片的方法，同时为了看上去较为立体，使用了选择角度的方法，面片在XYZ轴方向上随机旋转一个角度，在粒子数量较多时不会出现每个粒子都看上去一样的情况。

若粒子含有纹理，则使用`draw_with_tex`函数绘制，将坐标轴移到粒子position所在位置，调用`glRotatef`旋转角度，`glScalef`控制大小。之后为了效果，启用颜色混合，使粒子重合的地方展现半透明的效果。根据设定的color和texture绘制四边形面片，完成单个粒子的绘制。

单个粒子状态的更新：

首先更新粒子的位置，即position向量与velo向量相加，然后更新粒子的速度，即velo向量与acc向量相加。

之后判断motion_mode，粒子的运动模式，这是为了使粒子的运动更为多样化，如果不设置这个参数，则粒子按照设定的初始速度和加速度运动。*以下的运动模式均是针对大作业进行的特别设置，不具有普适性*。

- Y方向正弦运动：当Y方向速度大于0.1f或小于-0.1f时，加速度反向。用于实现水面波动。
- X方向正弦运动：当X方向速度大于0.1f或小于-0.1f时，加速度反向。用于模拟螺旋发射。
- 碰撞检测1：当粒子位置小于一个值的时候，Y方向速度反向，大小衰减。用于水落到水面产生水花。
- 碰撞检测2：Y方向检测如上。XZ方向也有碰撞检测，当碰到竖直面时，法向速度反向，大小衰减。用于粒子碰撞墙壁。

最后更新粒子生命，当前生命减去衰减速率。

ParticleSystem类

粒子系统类实际上是一个发射器，主要包含了粒子数组，粒子的数目，以及发射器的两个关键坐标。

在使用时，首先进行初始化，`ParticleSystem(int init_num, float init_x1, float init_y1, float init_z1, float init_x2, float init_y2, float init_z2)`，主要设定了系统内粒子的数目以及发射器的两个关键坐标。

下面详细说一下发射器坐标的用法：

- 点发射：发射器的两个坐标设为相同（即XYZ均相同），则所有粒子均从该点处发射出。
- 线发射：发射器的两个坐标的连线保证平行于任意坐标轴（即XYZ有两个相同），则所有粒子均从该线段处发射出。
- 面发射：发射器的两个坐标作为对角线的矩形面平行于任意坐标轴平面（即XYZ有一个相同），则所有粒子均从该矩形面发射出。
- 体发射：发射器的两个坐标作为长方体体对角线的两个端点（XYZ均不相同），则所有粒子从该长方体内发射出。

初始化完成后，对粒子数组进行设置，`void init_system(Particle* (init)(), bool(*judge)(Particle*))`，我采用的方法是传入一个Particle类函数指针对粒子进行初始化，这样做的好处是方便在main文件中对单个粒子进行配置，可随时修改；同时传入的另一个函数指针是一个判断函数，在这次大作业中用于判断粒子是否“出界”，若“出界”则删除该粒子重新生成。

在绘制时，调用`void ParticleSystem::update()`函数进行绘制并更新粒子状态，绘制和更新状态调用单个粒子的函数方法，更新后检查粒子状态是否已“死亡”或“出界”，若是则删除该粒子重新生成。

ObjLoader类

ObjLoader类在第一次小作业的基础上进一步的改进，当然中间参考了网上很多相关教程，主要完成的工作是解析obj文件同时对对应的mtl纹理文件进行解析。本次实现的ObjLoader还存在着很大缺陷，下文会提到这一点。

obj文件解析

obj文件主要由以下属性构成：

- 顶点数据
  - v：几何体顶点
  - vt：贴图坐标
  - vn：顶点法线
- 元素
  - f：面
- 成组
  - g：组名称
  - o：对象名称
- 渲染属性
  - mtllib：材质库
  - usemtl：材质名称

根据上面的属性，设计专门的数据结构用于存储，如`struct UV{double u,v;}`表示贴图顶点，`struct Triangle{ int Vertex[3]; int Normal[3]; int UV_Pnt[3]; ObjMaterial *mat; }`表示三角形面片等。读取文件时一行一行解析，存入相应的数据结构。

mtl文件解析

当obj文件中出现mtllib时，意味着要到mtl文件中查找相应的材质库，这时需要解析mtl文件。mtl文件主要由以下属性构成：

- newmtl：新材质
- Ka：环境反射
- Kd：漫反射
- Ks：镜面反射

- Tr：滤光透射
- d：渐隐指数
- Ns：反射指数
- Ni：折射值
- illum：光照模型
- map_xx：为xx参数指定纹理文件

同样，我们用一个类ObjMaterial来保存mtl文件中的信息，一行一行解析保存。

渲染

渲染时，遍历我们保存的所有Triangle三角形面片结构，设置相应的材质，进行绘制。

缺陷

对obj文件中face进行解析时，只考虑到了三角形面片的情况，但实际上face可以是N边形面片，一开始未考虑那么多。所以在实际应用时，若出现face是多边形的情况，会出现模型表面缺失面片的现象，这一点在后续还需要改进。

drawScene

drawScene文件主要完成一些简单物体的建模，包括：

- 篝火：使用画圆柱体的方法画一段木头，然后绕一个中心点画一圈，形成一个篝火，火焰由粒子系统展现
- 房屋：使用前后左右四面矩形面搭建房屋的墙壁，屋顶使用两个倾斜矩形面和两个三角形面实现
- 喷泉：使用搭建长方体的方法，用四个长方体搭建一个水池，中间使用类似方法
- 水池：与喷泉类似

具体场景（main文件）解析

基础场景

- 天空盒
- 地面
- 围墙
- 栅栏
- 光照

前面四种上面已经说过，现在说一下光照，我实现了两种光照效果，LIGHT0是聚光灯效果，LIGHT1是点光源效果，两种光源可以互相切换。点光源设置在场景的正上方(0,30,0)的位置，环境光参数均为0.3，漫反射和镜面反射参数均为1.0。聚光灯位置和点光源位置相同，光线方向竖直向下，最大散布角是45°。使用聚光灯效果时可以明显观察到物体光线和阴影的效果，使用点光源效果时面向光源的面是亮的，背向的则是暗的，整个场景较亮。

模型

- 房屋
- 篝火
- 喷泉
- 水池
- 树木
- 花朵

- 皮卡丘
- 恐龙
- 假山

前面六种上面已经提过，后面三种使用了ObjLoader导入obj文件和其纹理，其中恐龙和皮卡丘的效果较好，假山的效果较差。研究obj文件后发现假山使用的不是三角形面片而是多边形面片，这正是上面提到的缺陷所在，需要改进。

粒子系统

- 篝火火焰：体发射，初速度加速度一定范围随机
- 雪花：面发射，初速度一定范围随机，Y方向加速度固定，ZX方向加速度随机
- 瀑布：线发射，初速度加速度固定
- 喷泉：体发射，初速度一定范围随机，加速度固定
- 镭射火焰：线发射，初速度可交互控制，YZ方向加速度固定，X方向加速度在正负区间随机
- 模拟太阳：体发射，类似篝火火焰
- 特殊展示用系统：点发射，参数交互控制

开发过程中遇到的难点

在开发过程中也遇到了许多问题，下面按时间顺序说说遇到的难点。

设计Camera的过程中，视角的旋转问题是比较难的地方，我参考了网上的博客并自己加以修改完成了整个摄像机的实现。

在基本场景搭建的时候，比如天空盒和地面，主要的难点在于读取纹理文件，以及绘制时纹理坐标绑定的问题。对于栅栏，以及房屋、喷泉之类自己建模的物体，难点就在于坐标的计算比较繁琐。

对于粒子系统部分，关键在于参数的配置，如何让粒子系统看上去比较真实，以及多种运动模式的实现。

对于obj模型的读取，一直是很难的一部分，参考了很多内容，比如obj和mtl文件格式解析，obj文件读入等。

对于整体来说，还有光照和法线的问题也是遇到过问题的地方，在启用光照的情况下，物体材质和颜色应该如何设置，还有多种颜色混合的时候混合的参数设置问题。

后续改进工作

1. obj文件解析中多边形面片处理问题
2. 生成高低起伏地形的问题
3. 水面波模拟问题，比如利用法线贴图或gerstner曲线模拟