计算机图形学 - 魁地奇桌球

杨铭
5130379022

2016 年 1 月 12 日

目 录

1 简述

1.1 开发环境

操作系统 windows7 旗舰版

开发软件 Visual Studio 2015 community

模型制作 3Ds MAX 2013

图形库 OpenGL4.4

CPU Intel Xeon E3-1231v3 @3.4GHz

使用了 opengl 的核心库和相关辅助库 gl、glu、glut、glew、glaux，使用 3ds Max 制作场景和地形模型

1.2 完成功能

在第二次迭代的基础上，完成了以下功能：

- 读取 obj 模型文件并进行场景和地形的模型制作与渲染
- 完善光照效果，追加一个紧跟母球的聚光灯
- 实现简单版本的 3D 物理引擎
- 使用 perlin 噪声为小球生成随机纹理
- 天空盒
- 完善视角控制
- 粒子特效

2 设计说明

2.1 程序结构

主程序为 QuidditchApp.exe，跟 res 文件夹放在统计目录下可以直接运行。

程序结构 程序包含：

- Club
- Flag
- Hill

- Object
- Orb
- Particle、ParticleSystem
- Skybox
- Table
- Terrain

等图形类，以及

- Controller
- Camera
- Texture
- Physical
- Point3d、Vector3D

等辅助类，其中 **Controller**负责交互控制、**Camera**负责视角控制、**Texture**负责保存 perlin 噪声生成的纹理、**Physical**是物理引擎、**Point3D**和**Vector3D**为三维点和向量辅助类

2.2 文件路径说明

”res” 文件夹必须放在主程序同级目录下，否则无法正常加载.obj 文件和纹理

3 实现过程

3.1 读取 obj 模型文件和绘制

.obj 是一种通用的 3D 模型文件格式。由 Alias|Wavefront 公司为 3D 建模和动画软件”Advanced Visualizer” 开发的一种标准，适合用于 3D 软件模型之间的互导。.obj 文件一般分为 4 个部分，分别记录定点坐标、法向量坐标、纹理映射坐标和面片连接方式本次迭代实现了对 obj 的读取和绘制，并且使用了 VAO（Vertex Array Object）的加速绘制方法，使得对于 40000 个点，上万个面片的绘制也能做到快速。但是较多个模型的场景会导致.obj 文件的变大，在读文件时速度较慢。

```
for (std::list<SubMesh*>::iterator i = children.begin();
    i != children.end(); i++)
{
  Material mt = material_map[(*i)->mtName];
  SetMaterial(GL_FRONT_AND_BACK, GL_AMBIENT, mt.ka);
  SetMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE, mt.kd);
  SetMaterial(GL_FRONT_AND_BACK, GL_SPECULAR, mt.ks);
  glBindTexture(GL_TEXTURE_2D, mt.kd_texid);
  glVertexPointer(3, GL_FLOAT, 0, (*i)->ptBuffer);
  glTexCoordPointer(2, GL_FLOAT, 0, (*i)->uvBuffer);
  glNormalPointer(GL_FLOAT, 0, (*i)->nmBuffer);
  glDrawArrays(GL_TRIANGLES, 0, (*i)->vNum);
}
```

我自己绘制了地形和城墙，坦克模型从网上下载，纯属观赏（使用了一个立方体碰撞包围盒覆盖它）

3.2 光照效果

聚光灯 对 OpenGL 里的光照设置 GL_SPOT_DIRECTION，这个光源会被认为是聚光灯。通过对它位置、聚光程度和收敛角的设置，可以达到追踪母球的效果

```
glEnable(GL_LIGHT1);
glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 24.0);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light1_spotdir);
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 1.0);
```

上面的代码设置了一个初始位置在母球上方的光源，并指明它的方向垂直指向地面，也就是母球。然后设置了它的收敛角度和汇聚强度

需要注意的是光照的位置和方向会受到视角矩阵的影响，需要每一帧重新设置它们

```
void setlights(void)
{
        Point3D mPos = crashManager->getMPos();
        light1_position[0] = mPos.x;
        light1_position[1] = mPos.y;
        glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
        glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
        glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, light1_spotdir);
}
```

最后添加了交互控制，按'b' 键可以开关聚光灯

3.3 实现简单版本的 3D 物理引擎

由于地形是读取的 obj 文件，为了方便代码的编写和计算，我在设计地形时做了简化，使得地形被剖分的点的坐标均为整数，并且所有坐标上的点都均匀分布。

```
#
# object Plane002
#

v  0.0000 0.0000 10.0000
v  1.0000 0.0122 10.0000
v  1.0000 0.0639 9.0000
v  0.0000 0.0122 9.0000
v  2.0000 0.0565 10.0000
v  2.0000 0.1461 9.0000
v  3.0000 0.1236 10.0000
v  3.0000 0.2486 9.0000
v  4.0000 0.2041 10.0000
v  4.0000 0.3613 9.0000
v  5.0000 0.2892 10.0000
v  5.0000 0.4749 9.0000
v  6.0000 0.3709 10.0000
v  6.0000 0.5810 9.0000
```

Figure 1: 地形 obj 格式

这样我们就可以构建一个地形的高度表，对应 **Hill**类中的 heightDiagram。一个高度表是一个二维数组，它的两个维度分别为地形横向的点数和纵向的点数，而模型保证了这些点均匀分布，就可以简单的根据点的坐标映射到高度表的坐标中，从而根据点的 x、y 轴坐标直接索引到点所在的高度

```
float Hill::getHeight(float x, float y)
{
        if (heightDiagram!=NULL)
        {
                // round to nearest integer
                int px = int(x);
                int py = int(y);
                float frcx = x - px;
                float frcy = y - py;
                if (frcx >= 0.5)
                        px += 1;
                if (frcy >= 0.5)
                        py += 1;
                py += 10;
                return heightDiagram[py][px];
        }
        else
        {
                return 0.0f;
        }
}
```

这样我们可以实现让小球永远贴着地表做三维运动。在每一帧里判断有没有小球落在了“高低”的范围内，如果有，则根据它的坐标，找到离小球最近的 3 个“高地”的点，接着通过线性插值的方式计算出小球的高度。

当然这种实现并不是真实的物理模型，只是对真实世界的一种简化。

3.4 使用 perlin 噪声为小球生成随机纹理

Perlin 噪声 使得随机变得“有序”的一种算法或者说思想。perlin 噪声使用插值和多频率叠加等方式使得随机生成的点被连接成为光环的曲线，然后通过曲线中所有的点映射到任意大的范围内生成数据。

我使用了一个二维的噪声生成器，然后通过 cos 插值和 3 个频率的叠加生成了最终的噪声数据。并把这个数据映射到 0-255 的范围内作为灰度值来使用

```
Texture* genTexture(int seed)
{
  Texture *tex = new Texture();
  tex->sizeX = TEX_X;
  tex->sizeY = TEX_Y;
  tex->data = new unsigned char[TEX_X*TEX_Y * 3];
  for (int i = 0; i < TEX_Y; i++)
  {
        for (int j = 0; j < TEX_X; j++)
        {
          unsigned char value = LEVEL*perlinNoise(float(j+seed)
                                / 64.0f, float(i+seed) / 64.0f);
          tex->data[(j + i*TEX_X) * 3]     = value;
          tex->data[(j + i*TEX_X) * 3 + 1] = value;
          tex->data[(j + i*TEX_X) * 3 + 2] = value;
        }
  }
  return tex;
}
```

3.5 天空盒

使用 Terrgen 工具生成一张全景图，然后使用 CubeTheShpere 工具将其分解为 6 张图片，分别对应天空盒的前后左右上下六个方向。然后在场景中绘制一个大的立方体，并按照正确的顺序将上面生成的图片贴在对应的立方体面上即可完成

```
void Skybox::render()
```

```
{
  glPushMatrix();

  glTranslatef(x, y, z);
  glRotatef(90, 1.0, 0, 0);
  glEnable(GL_TEXTURE_2D);
  glDisable(GL_LIGHTING);

  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
  glBindTexture(GL_TEXTURE_2D, DOWN_TEX);

  glBegin(GL_QUADS);
        glTexCoord2f(1.0f, 1.0f);
    glVertex3f(-EMAX,EMAX,-EMAX);
        glTexCoord2f(0.0f, 1.0f);
    glVertex3f(-EMAX,EMAX,EMAX);
        glTexCoord2f(1.0f, 0.0f);
    glVertex3f(EMAX,EMAX,EMAX);
        glTexCoord2f(0.0f, 0.0f);
    glVertex3f(EMAX,EMAX,-EMAX);
  glEnd();
  ...
```

3.6 完善视角控制

增加了横向和远近方向上的控制。使得它不再像以前那样蛋疼了（虽然依旧比较蛋疼）

3.7 粒子特效

原理 粒子特效，即使用大量小的物体做随机运动来模拟自然界中粒子的效果。进一步可以控制粒子运动的趋势，改变粒子的颜色，实现粒子的碰撞，增加粒子运动时的动态模糊等增加真实感和画面的绚丽程度

为了实现粒子效果，我定了 **Particle** 类，作为一个单独的粒子，再定义一个控制粒子的类 **ParticleSystem**，作为所有粒子系统的基类。独特的粒子继承

自 **ParticleSystem**，但是对其一些属性或者绘制方式做了修改

```
class Particle
{
public:
  Particle() {};
  Vector3D position;
  Vector3D velocity;
  Vector3D acceleration;
  Color color;
  float age;
  float life;
  float size;
  bool active;
};
```

Spark Spark 为球与球碰撞产生的火花粒子特效，继承自 **ParticleSystem**，它使用了一张半透明的贴图，并且使用彩虹表中的随机颜色对纹理进行混合模式的上色。对粒子应用较快的四散的初速度和恒定的重力加速度，模拟爆炸火花的效果。

```
void Spark::render()
{
  if (isActive())
  {
        glEnable(GL_TEXTURE_2D);
        glDisable(GL_LIGHTING);
        glDisable(GL_DEPTH_TEST);
        glBindTexture(GL_TEXTURE_2D, TEX_PARTICLE);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        for (vector<Particle*>::iterator iter = particles.begin();
         iter != particles.end(); iter++)
        {
          if ((*iter)->active)
```

```
16              {
17                      float x = (*iter)->position.x;
18                      float y = (*iter)->position.y;
19                      float z = (*iter)->position.z;
20                      Color color = (*iter)->color;
21                      float alpha = 1.0 - (*iter)->age / (*iter)->life;
22                      glColor4f(color.r, color.g, color.b, alpha);
23                      float sz = 0.5;
24                      glPushMatrix();
25                      glBegin(GL_TRIANGLE_STRIP);
26                      glTexCoord2d(1, 1);
27          glVertex3f(x + sz / 2, y + sz / 2, z);
28                      glTexCoord2d(0, 1);
29          glVertex3f(x - sz / 2, y + sz / 2, z);
30                      glTexCoord2d(1, 0);
31          glVertex3f(x + sz / 2, y - sz / 2, z);
32                      glTexCoord2d(0, 0);
33          glVertex3f(x - sz / 2, y - sz / 2, z);
34                      glEnd();
35                      glPopMatrix();
36              }
37          }
38          glEnable(GL_LIGHTING);
39          glDisable(GL_TEXTURE_2D);
40          glEnable(GL_DEPTH_TEST);
41          glDisable(GL_BLEND);
42      }
43  }
```

4 效果展示

由于模型文件较大 (8.89M)，导致每次开始执行要浪费大量时间在读文件上，而且多个面片在对光照的映射处理导致帧数也有一定的降低

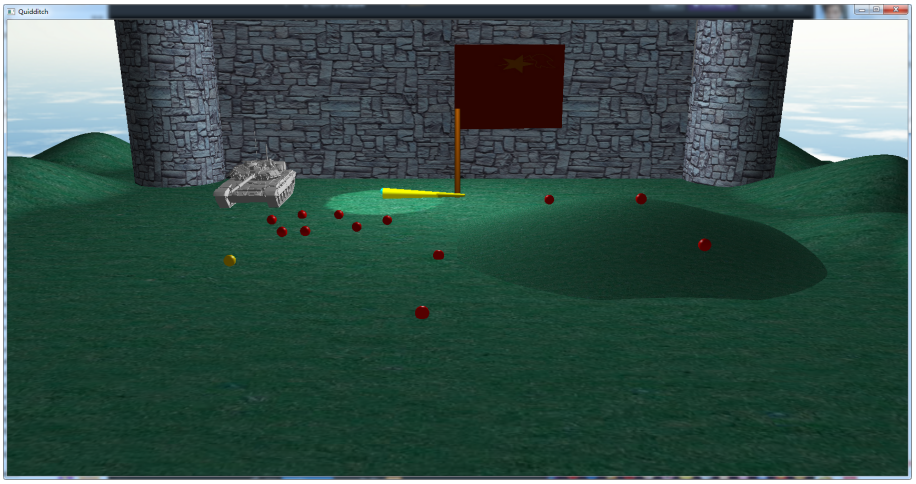

Figure 2: 结果图