**魁地奇桌球小游戏设计报告**

作者：曹立

学号：5130379057

版本号：v2.0

**目录**

1. 目录…………………………………………………… 1
2. 项目简介……………………………………………… 2
3. 游戏构成元素介绍…………………………………… 3
4. 游戏规则和操作说明………………………………… 5
5. 程序设计与实现……………………………………… 6
6. 游戏运行截图………………………………………… 9

**项目简介**

本项目是课程《计算机图形学》的大作业，使用C++语言编写，

OpenGL图形库开发。开发这个小游戏的主要原因是学习对于OpenGL图形库的使用以及进行对《计算机图形学》这门课程的实践应用。

在本小游戏中，玩家可以在3D场景中打桌球，只是规则与传统意义上的打桌球又有所不同。因为是在游戏中，所以我们可以做到许多现实中做不到的事情。本游戏添加了一些现实中桌球没有的因素，使桌球更富有趣味。然而，桌球本身又无法完全脱离现实。我们希望现实中的一些物理要素（比如说运动的特性）可以尽可能地呈现在程序中，使得程序看起来更加逼真。

**游戏构成元素介绍**

|  |  |  |
| --- | --- | --- |
| 元素名称 | 截图 | 介绍 |
| 桌球台 | table.png | 游戏将在桌球台上进行，所有的球（除了金色飞贼小球偶尔离开桌面外）都不会离开桌面。桌面的绘制通过OpenGL绘制多边形和纹理映射完成。 |
| 主球 | mainBall.png | 就和普通的桌球一样，只能用桌球杆来击打该主球。通过主球把其他球打入球洞获得分数。 |
| 鬼球 | ghostBall_natureTex.png | 分数最低的一种球，容易击中，没有什么特别的属性。 |
| 游走球 | wanderBall.png | 每隔一段时间，游走球会被赋予一定的速度。游走球比起鬼球更难击中，因而也有更高的分数。 |
| 金色飞贼小球 | goldenBall.png | 金色飞贼小球会在桌面上方随意飞行，每隔一段时间会降落到桌面上。此时它可以被主球撞击。金色飞贼小球极难击中，因此分数最高。 |
| 桌球杆 | cue.png | 用以击打主球的桌球杆。前端会有虚线指向主球。玩家可以任意调整桌球杆的方向。 |
| 旗帜 | flag.png | 旗帜。只是作为摆设，没有实际作用。会不断随风飘荡。为了实践Beizer曲线而画的。 |

**游戏规则和操作说明**

游戏规则：

1.游戏中的桌面上初始时共有四种球：白色母球1颗、鬼球6颗、游走球6颗以及金色飞贼小球1颗；

2.玩家可以操控球杆，从不同的方向击打白色母球。白色母球被击中后会以某种速度沿球杆方向移动；

3.玩家通过打白色母球碰撞其他球体，令其他球体落入桌面的孔洞中，获得相应的分数（鬼球10分，游走球20分，金色飞贼小球100分）；

4.玩家只有在白色母球静止时才能使用球杆击打它；

5.若玩家将白色母球打入孔洞中，则获得的分数清零，白色母球将重新出现在桌面中央；

6.游戏以局为单位，一局时间为一分钟，在一局中获得分数越高成绩越好；

7.若在一局时间结束前将所有球打完，所有球将重新出现在桌面上，游戏继续进行直到一局结束；

操作说明：

1.W/S/A/D键分别对应围绕中心向上/向下/向左/向右调整视角；

2.Q/E键为球杆分别对应围绕主球逆时针、顺时针旋转；

3.空格键为用球杆击打主球；

4.O键打开灯光，P键关闭灯光；

**程序设计与实现**

逻辑视图：

**逻辑层**

**交互层**

用户输入

状态更新模块

物体模块

控制模块

显示模块

输入模块

输入模块：使用了DirectX的接口从键盘获取用户输入；

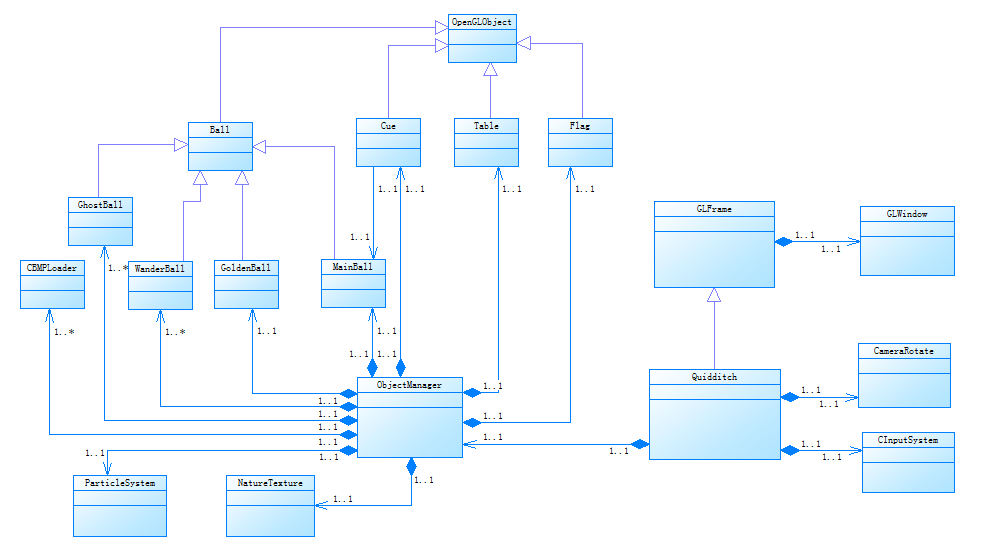
控制模块：负责创建物体并控制状态更新模块；

状态更新模块：负责时时更新物体的状态；

物体模块：为一些类，存储物体的属性，比如大小、位置、速度等；

显示模块：调用OpenGL的函数，根据物体的属性渲染图案；

类图：



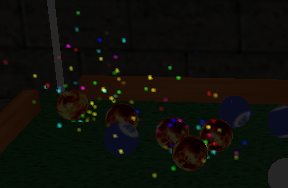
OpenGL技术：

1.纹理创建与映射。本程序中的物体的外观大部分使用了纹理贴图来进行绘制。使用的贴图清单如下：

|  |  |  |
| --- | --- | --- |
| wood.bmp | green_area.bmp | sjtu.bmp |
| wander_ball.bmp | golden_ball.bmp | cue.bmp |
| Wall.bmp | flare.bmp |  |

2.Beizer曲线构造曲面。本程序中飘动的旗帜即由Beizer曲线构造。调整观察角度，从上观察，旗帜面呈现为正弦函数。将旗帜切为许多的小方格，初始化它们位置的代码如下：

|  |
| --- |
| /// Initialize points  **for** **(**int x **=** 0**;** x **<** 45**;** **++**x**)**  **{**  **for** **(**int y **=** 0**;** y **<** 45**;** **++**y**)**  **{**  m\_Points**[**x**][**y**][**0**]** **=** float**(**x **/** FLAG\_PARAMETER **-** 4.5f**);**  m\_Points**[**x**][**y**][**1**]** **=** float**(**y **/** FLAG\_PARAMETER**);**  m\_Points**[**x**][**y**][**2**]** **=** float**(**sin**((((**x **/** FLAG\_PARAMETER**)** **\*** 80.0f**)** **/** 360.0f**)** **\*** PI **\*** 2.0f**));**  **}**  **}** |

3.粒子系统。当两小球碰撞时会产生向四周蹦出粒子的效果，如下：

接下来简要讲解粒子系统的实现。粒子的结构体代码如下：

|  |  |
| --- | --- |
| struct Particle  **{**  Vector3 position**;** // Position of the particle  Vector3 velocity**;** // Velocity of the particle  Vector3 acceleration**;** // Acceleration of the particle  float lifetime**;** // Life time of the particle  float dec**;** // Vanishing speed of the particle  float size**;** // Size of the particle  float color**[**3**];** // Color of the particle  **};**  每当两个小球发生碰撞，ParticleSystem类将会产生一个粒子爆炸(ParticleExplosion)的对象，这个对象中包含一个Particle的数组，并有自己的生命周期。当生命周期结束时，这个对象也会消亡，并从ParticleSystem对象中删除。  ParticleExplosion中初始化粒子的实现如下，这样子的初始化使得粒子均匀地向四周蹦出并且逐渐下落并消逝：   |  | | --- | | void ParticleExplosion**::**Init**(**int num**)**  **{**  m\_Num **=** num**;**  m\_List **=** **new** Particle**[**m\_Num**];**  /// Initialize each single particle  **for** **(**int i **=** 0**;** i **<** m\_Num**;** i**++)**  **{**  float x**,** y**,** z**,** vx**,** vy**,** vz**;** // Position and velocity  x **=** 0.0005f **\*** **(**rand**()** **%** 9**);**  y **=** 0.0005f **\*** **(**rand**()** **%** 9**);**  z **=** 0.0005f **\*** **(**rand**()** **%** 9**);**  float v **=** 0.00004f**;**  float alpha **=** **(**rand**()** **%** 360**)** **\*** PI **/** 180.0f**;**  float beta **=** **(**rand**()** **%** 90**)** **\*** PI **/** 180.0f**;**  int random **=** rand**()** **%** 2000**;**  vy **=** v **\*** sin**(**beta**)** **\*** random**;**  vx **=** v **\*** cos**(**beta**)** **\*** cos**(**alpha**)** **\*** random**;**  vz **=** v **\*** cos**(**beta**)** **\*** sin**(**alpha**)** **\*** random**;**  m\_List**[**i**].**position **=** Vector3**(**x**,** y**,** z**);**  m\_List**[**i**].**velocity **=** Vector3**(**vx**,** vy**,** vz**);**  m\_List**[**i**].**acceleration **=** Vector3**(**0.0**,** **-**0.0008f**,** 0.0**);**  m\_List**[**i**].**lifetime **=** 100.0f**;**  m\_List**[**i**].**size **=** 0.07f**;**  m\_List**[**i**].**dec **=** 0.05f **\*** **(**rand**()** **%** 50**);**  /// Set color  int index **=** rand**()** **%** 7**;**  m\_List**[**i**].**color**[**0**]** **=** colors**[**index**][**0**];**  m\_List**[**i**].**color**[**1**]** **=** colors**[**index**][**1**];**  m\_List**[**i**].**color**[**2**]** **=** colors**[**index**][**2**];**  **}**  **}** | |

4.光照。仅仅使用了简单的光照模型，使用了一个光源，包含环境光、漫射光、镜面光三种成分，打开、关闭光照后的效果对比如下：

|  |  |
| --- | --- |
| light2.png | light1.png |

光照代码如下：

|  |  |
| --- | --- |
| |  | | --- | | void Light**::**Init**()**  **{**  glEnable**(**GL\_LIGHTING**);**  /// Set default light  glLightfv**(**GL\_LIGHT0**,** GL\_AMBIENT**,** ambientLight**);**  glLightfv**(**GL\_LIGHT0**,** GL\_DIFFUSE**,** diffuseLight**);**  glLightfv**(**GL\_LIGHT0**,** GL\_SPECULAR**,** specularLight**);**  glLightfv**(**GL\_LIGHT0**,** GL\_POSITION**,** lightPosition**);**  glEnable**(**GL\_LIGHT0**);**    glMaterialfv**(**GL\_FRONT**,** GL\_SPECULAR**,** specularRef**);**  glMateriali**(**GL\_FRONT**,** GL\_SHININESS**,** 64**);**  **}** | |

5.Perlin噪声函数制作的自然纹理附着到鬼球上，单独显示的效果如下：

|  |
| --- |
| perlinNoise.png |

接下来简要介绍用Perlin噪声函数制作该纹理的过程，该纹理由类NatureTexture的一个对象生成，与之有关的NatureTexture的变量和方法如下：

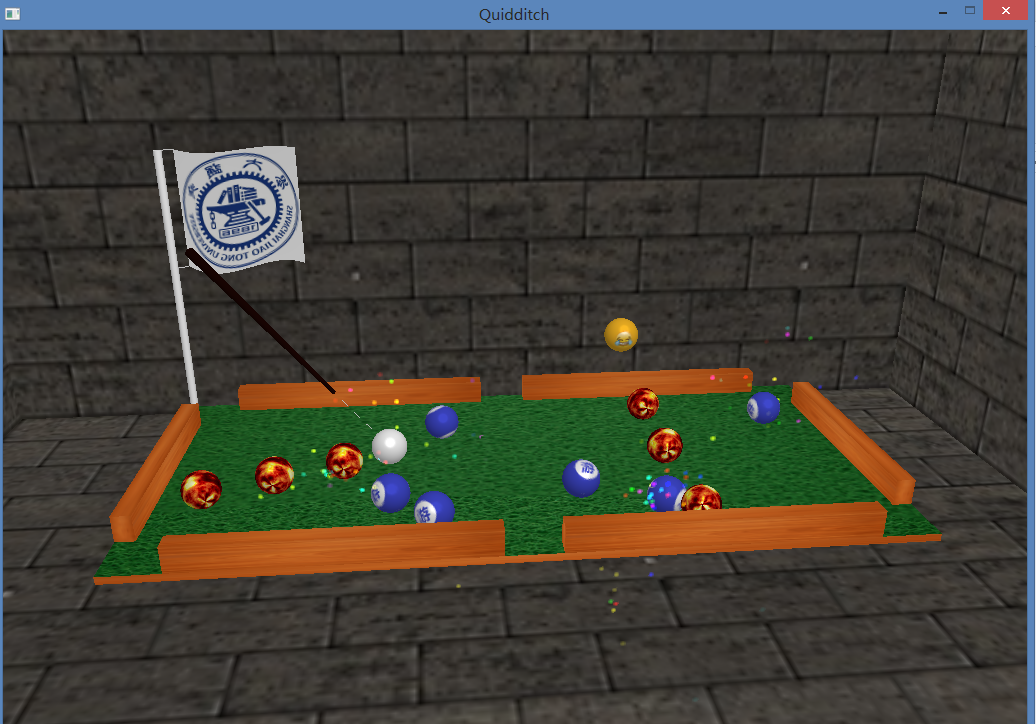
|  |
| --- |
| private**:**  /// Members  int imageWidth**;** // Width of the image(default 256)  int imageHeight**;** // Height of the image(default 256)  unsigned char **\*** image**;** // Data of the image  /// Methods of generating a texture by perlin noise  void GenerateNoise**();**  float **\*\*** GenerateWhiteNoise**();**  float **\*\*** GenerateSmoothNoise**(**float **\*\*** baseNoise**,** int octaveCount**);**  float **\*\*** GeneratePerlinNoise**(**float **\*\*** baseNoise**,** int octave**);**  float **\*\*** MapGradient**(**float **\*\*** perlinNoise**);**  float Interpolate**(**float x0**,** float x1**,** float alpha**);** |

最顶层的函数为GenerateNoise，其实现如下：

|  |
| --- |
| float **\*\*** whiteNoise **=** GenerateWhiteNoise**();**  float **\*\*** perlinNoise **=** GeneratePerlinNoise**(**whiteNoise**,** 6**);**  float **\*\*** colorData **=** MapGradient**(**perlinNoise**);**  int index **=** 0**;**  **for** **(**int row **=** 0**;** row **<** 256**;** row**++)** **{**  **for** **(**int col **=** 0**;** col **<** 256**;** col**++)** **{**  index **=** row **\*** 256 **+** col**;**  **((**int **\*)**image**)[**index**]** **=** **(**int**)**colorData**[**row**][**col**];**  **}**  **}** |

过程并不复杂，先用随机数产生白噪声。然后产生PerlinNoise。在产生PerlinNoise的过程中多次调用GenerateSmoothNoise根据octave产生几组平滑噪声，最后将几组噪声叠加。生成噪声之后再将噪声映射到RGB值，并传入OpenGL纹理需要的数据中就大功告成。

**游戏运行截图**

****