“魁地奇桌球”第二版设计报告

沈斯杰　5130379036

2015. 12. 13

一、 项目简介

(一) 第二版要求

此次项目的具体要求如下：

在第一次作业的基础加入每个魁地奇阵地里的旗帜，旗帜必须是通过自己建模实现，需要有贴图，并在最后加入旗帜飘动的动画。具体内容包括：

（1）用各种曲线函数设计并构建旗帜模型（35 分）；

（2）在旗帜模型上加入自己独特的贴图（20 分）；

（3）在旗帜模型中加入旗帜飘扬的相关动画（30 分）；

（4）详细的设计报告以及标准格式的提交文件（15 分）；

以上要求全部实现。

(二) 游戏截图

(三) 游戏基础设置

1. 游戏的按键设置如下：（红色部分为与第一版不同）

按键	功能
A/D	转动杆的方向

鼠标左键	推杆
鼠标右键	为杆蓄力
J/L	左右移动球台
U/O	上下移动球台
I/M	前后移动球台
鼠标中键	复原最初位置和视角
↑/↓/←/→	转移视角

2. 球台上有 15 个球，其中球号为 5/6/7/13/14/15 为移动的“游走球”，黑 8 为“飞贼”。初始状态下，“鬼球”会在桌面平面内随机移动，“飞贼”会在桌球及其上空随机移动，经过一段时间后停在桌面上休息，如此循环。
3. 双方阵营有旗帜，以及球台上的标题旗帜。

二、 本次主要工作介绍

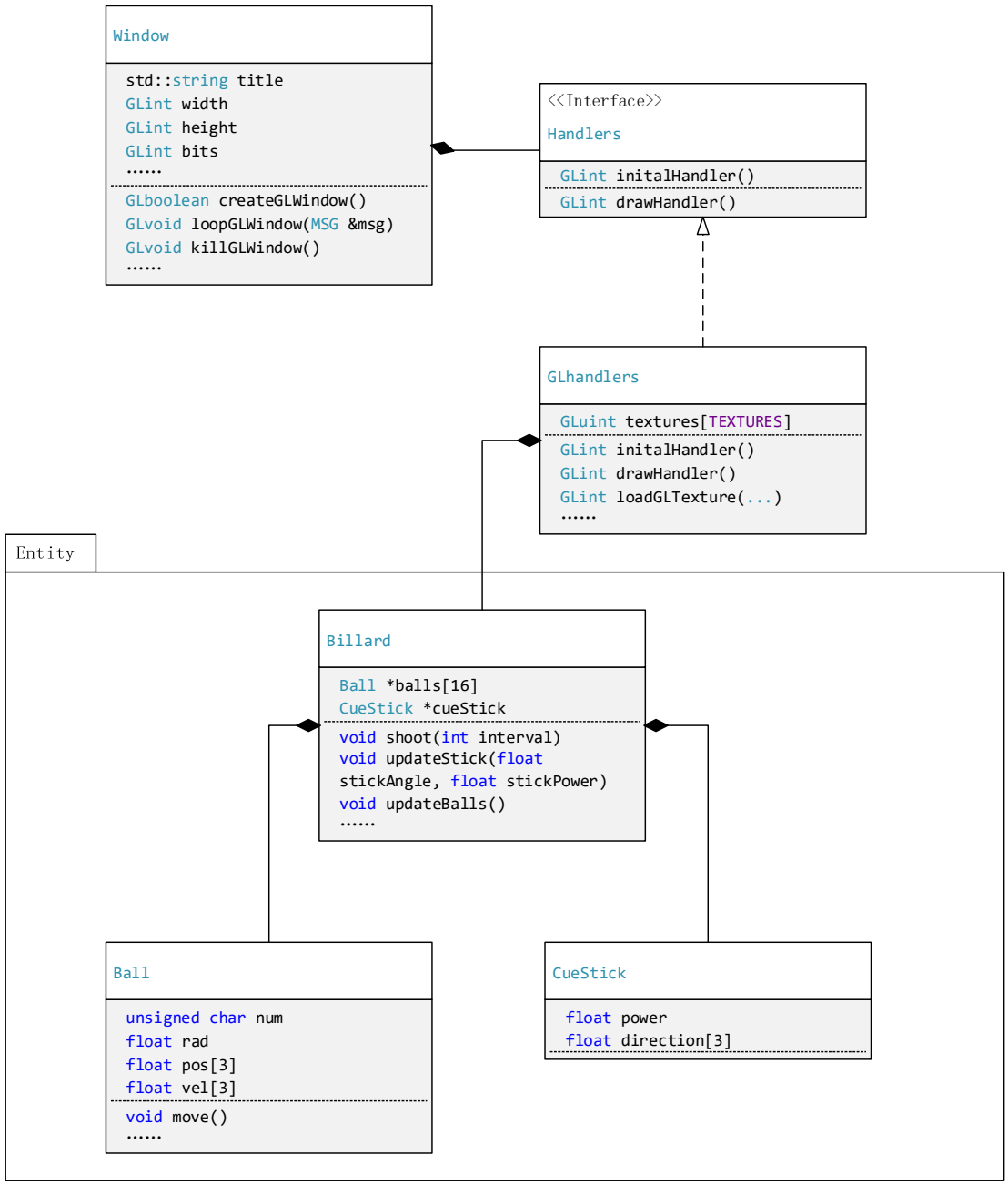
(一) 程序结构优化

在第一版的程序中，主要的架构如下：

头文件	.cpp 文件	职能
background.h	background.cpp	绘制（底层）背景
draw.h	draw.cpp	绘制模型(包括光照等)
	texture.cpp	绘制纹理
entity.h	entity.cpp	球和球杆的实体类
	billard.cpp	游戏逻辑
	main.cpp	初始化和入口函数

进行调整的有：

1. 将原本 main.cpp 中的 initGL()函数移入 draw.h 中作为接口，因为这一部分在逻辑上适合 openGL 的绘制联系在一起的。
2. 本来 billard.cpp 中，对于游戏的逻辑耦合度太高，将球的移动直接在这个文件中实现，现在在实现游戏逻辑中，按照摩擦、力度中改变球的速度，然后调用球这个类中的 move()方法，这样更符合实际情况，耦合度也会降低。
3. 将程序改成面向对象的设计结构，使逻辑更加清晰。大致包图如下（还有一些小类没有画出）。
4. 对球加入部分碰撞效果。



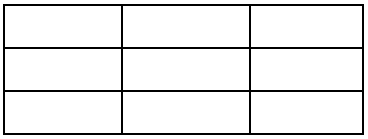
(二) 旗帜的设计

1. 双方阵地的旗帜选用贝塞尔曲面进行设计。关键代码如下：

```
/* For Animation */
for (int i = 0; i < UNUM; i++) {
    for (int j = 0; j < VNUM; j++) {
        ctrlpoints[i][j][0] =
            sin(2 * PI*(-2.5 * i / 14.0f + t / float(T))) - sin(2 * PI* t / float(T));
        ctrlpoints[i][j][0] *= 0.2;
    }
}
t = (t + 1) % T;
glMap2f(GL_MAP2_VERTEX_3,       // Bezier
    0.0, 1.0, 3*VNUM, UNUM,
    0.0, 1.0, 3, VNUM,
    &ctrlpoints[0][0][0]);
glMap2f(GL_MAP2_TEXTURE_COORD_2,    // Texture
        0.0, 1.0, 2, 2,
        0.0, 1.0, 4, 2,
        &texpts[0][0][0]);
glMapGrid2f(60, 0.0, 1.0, 20, 0.0, 1.0);
```

一开始的两次循环是对于控制顶点进行刷新，这是在 loop 的函数里面，每一次都会进行计算，使用的是波动函数的样式。计算好控制顶点之后，就可以进行顶点的计算和纹理的映射了。其中，控制顶点的个数由 UNUM 和 VNUM 表示u,v两个方向的控制顶点数。

2. 在球台上的横幅是用多个四边形面片进行构造的。



比如，如上图的大四边形中，横竖各 4 个顶点，可以用构成 3*3 个小四边形进行组成，横幅中使用的就是这样的思想。

首先，在初始化函数中，我们先将这些顶点的位置计算出来。

```
// initial banner control points
for (int i = 0; i < BANNER_UNUM; i++) {
    for (int j = 0; j < BANNER_VNUM; j++) {
        bannerPoints[i][j][0] = (i * BANNER_LENGTH / BANNER_UNUM) - BANNER_LENGTH / 2;
        bannerPoints[i][j][1] = (j * BANNER_WIDTH / BANNER_VNUM) - BANNER_WIDTH / 2;
        bannerPoints[i][j][2] = 0.5 * sin(((i * 8.0f / 360.0f)*PI*2));
    }
}
```

然后，我们在渲染的时候将这些顶点间的小四边形一一渲染出来，并且贴上相应位置的纹理。详细代码可以见 draw.c:renderBanner()中。

这样构造出的旗帜，它的动画实现比较有意思。类似于物理中的机械波，每一个点的位置都是前一个点在上一时刻的位置。

```
/* Animation */
static int t = 0;
if (t == 10)        // 10 is a period
{
    for (int j = 0; j < BANNER_VNUM; j++) {
        float hold = bannerPoints[BANNER_UNUM - 1][j][2];
        for (int i = BANNER_UNUM - 1; i > 0; i--) {
            bannerPoints[i][j][2] = bannerPoints[i - 1][j][2];
        }
        bannerPoints[0][j][2] = hold;
    }
    t = 0;
}
t++;
```

3. 两类曲面构造的差别：

(1) 对于贝塞尔曲面，只要指出控制顶点的位置之后，既可以构造出曲面。省去了曲面顶点的手动计算，而且纹理一次贴成。其实，对于动画的实现也可以用第二种方法，但是我想尝试一下波动函数的效果，所以用了

两种方法。

(2) 对于小四边形面片组成的曲面，优点是实现起来比较直观，可以直接构造曲面，而不需要通过控制顶点，缺点就是纹理映射有点麻烦，需要更具不同位置进行纹理映射。这在代码中也有所体现。

三、 后续工作设想

(一) 缺陷修复

这次第二版后，有一个比较严重的 bug 就是，每次开局会出现很奇怪的现象，比如球的位置不正确，准备修复。

(二) 场景的构建

目前只有一个桌面，下次要加入整个场景的渲染。

(三) 操作方式的改变

现在的操作方式还是和普通桌球类似，下次准备加入更多的特效，而且游戏方式中可能也用不到杆了，争取做出更多酷炫的效果。

(四) 更多优化

要在最后提交的时候提交一份让自己满意的程序。