# 计算机图形学大作业报告

## 1. 概述

本次作业实现了一个对不倒翁以及小球之间互相碰撞进行的物理模拟OpenGL程序，其中涉及到的重要知识点有：平面网格创建、模型运动、射线检测、坐标系转换、物理模拟、光影效果、粒子系统。

本次作业耗时大约15日，其中1~6日对平时作业2的代码进行了一些重构，修正了部分不完全正确的函数实现，并添加了一些新的功能；7~12日完成物理模拟部分，13~15日完成粒子系统以及光照。

## 2. 代码结构以及类实现简单介绍

本次作业中使用到的重要类（按照使用的先后排序）包括：

**I. Ground类**：提供快速生成一个给定长、宽以及网格大小的平面，其中每个网格填充自定义的纹理图片。

**II. Model类**：提供从文件导入模型或复制其它模型的构造函数，记录模型的（角）速度，（角）加速度，三维尺寸，模型碰撞箱，保存模型使用的粒子效果列表、模型的model矩阵，提供模型的受力函数、碰撞函数、地面碰撞函数、射线检测函数。

**III. Camera类**：保存摄像机的proj和view矩阵，摄像机位置以及朝向。

**IV. Particle**和**ParticleGenerator**类：Particle类表示一个粒子，储存这个粒子的速度、方向以及生命周期、是否可再生成，ParticleGenerator类则是粒子的发射器，含有储存所有粒子的vector，以及粒子的基本类型。

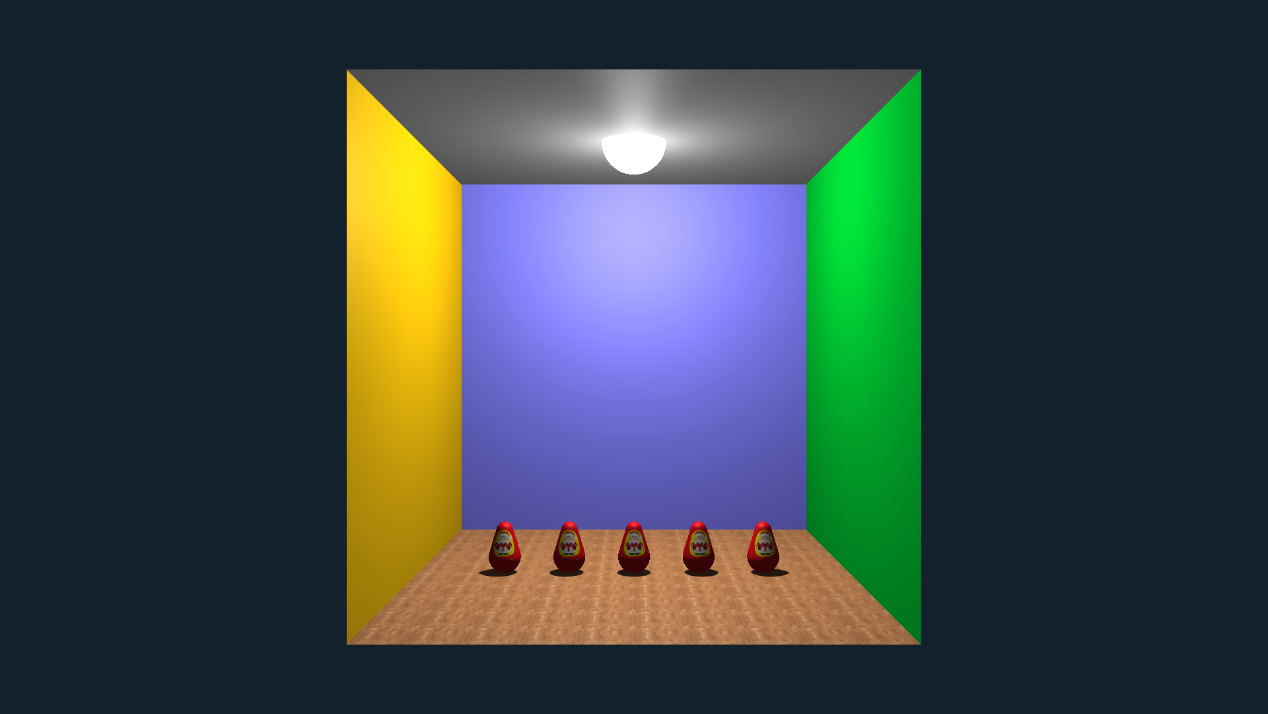
## 3. 重要功能的分析以及实现过程

**I. 基本场景搭建**

本次作业要求的基本场景为：三个墙面、一个天花板和一个木纹地面，天花板正中有一个半球形的灯。

显然，墙面、天花板和地面都可以简化为平面网格，只需要往网格内部添加对应纹理即可。而这个建立平面网格的过程非常简单，在此不赘述，重点是为了接收光照，对每个点都需要记录对应的法向量，在进行旋转的时候要注意将法向量转向正确方向以便正常受光。详细的实现可以见Ground类的代码。其中除了地板材质是网络资源外，其余的墙壁材质是用windows自带的画图系统填充得到的。所有材质均为jpg格式。

至于自放光的半球形的灯，我下载了一个半球形的模型，将其导入，并单独编写了自发光材质的着色器文件，将所有点的输出像素都绑定为固定色（这里是白色），这样半球形并不会受到实际的光照（实际光照来源为一个放置在半球形中央位置的点光源）影响，表现得像是其本身在发光。

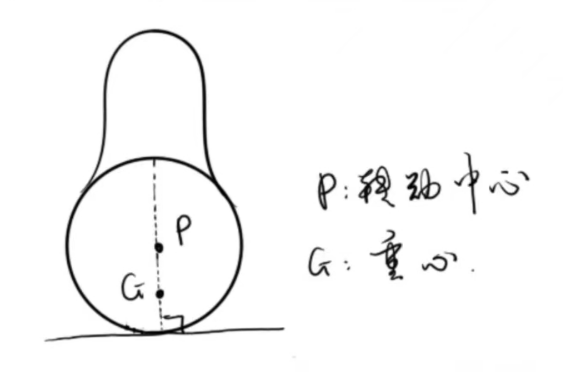


搭建的场景效果，加入阴影

**II. 不倒翁受力以及约束物理分析**

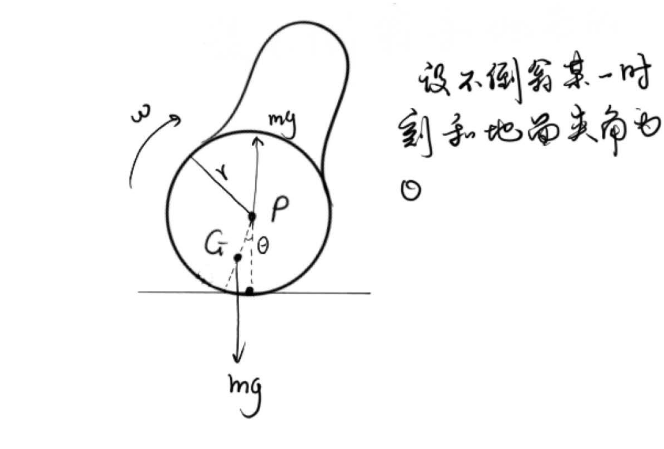
a. 不倒翁在地面上：

一个刚体的不倒翁应该具有一个转动中心、一个转动半径和一个重心，而且保证重心位置应该在转动中心之下：



如图，不倒翁在静止状态下其重心、转动重心连线应当与地面垂直，此时不倒翁受力平衡

假设不倒翁因为受外力而发生倾斜，而且在某一时刻，具有顺时针的角速度w，且此时不倒翁重心和转动重心连线于地面法线夹角为θ：



此时不倒翁在重心处受重力mg，在地面支点处受支撑力mg，两力方向相反，大小相同，且支撑力始终过转动重心。

由于重心与转动中心连线于地面法线夹角为θ，假设重心与转动中心连线长为d，根据几何关系不难推算出，此时不倒翁所受力矩（在图中指向屏幕外）为：

假设不倒翁转动惯量为inertia，每一帧间经过的时间为deltaTime（本程序中为1/60s），在一个deltaTime的时间内，不倒翁的角速度改变量为：

于是，假如不倒翁失去平衡且不受外力，在一个deltaTime的过程中，需要对不倒翁进行如下的操作：

1. 改变不倒翁的旋转矩阵，以经过转动中心的旋转轴绕角速度方向旋转 \* deltaTime旋转角
2. 考虑不倒翁因为平衡旋转而带来的在水平方向的位移，将其位移矩阵移动r \* w \* deltaTime
3. 将下一帧的角速度改变为
4. 为了模拟摩擦力对不倒翁速度的影响，对角速度应用速度衰减（乘以一个衰减系数）

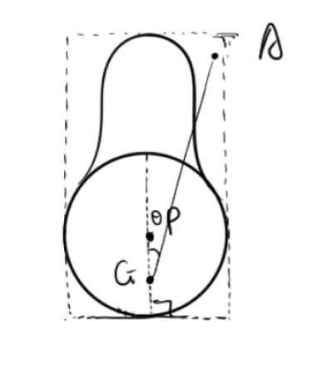
稍微对参数进行调整，就可以得到不倒翁在地面摇晃的近似物理模拟。

b. 不倒翁被提在空中：

本程序中可以提起不倒翁的位置需要在不倒翁重心上方，如果点击重心下方区域，获得的反馈是给不倒翁一个向屏幕内的力（只带来位移，不带来旋转）。

假设鼠标点击位置为如图所示的A点（不倒翁的OBB碰撞箱用虚线画出），根据平衡状

态，不难得出结论：不倒翁最后平衡的位置必须保证鼠标提起点和重心的连线于地面垂直。



此时不倒翁只在重心处受垂直向下的重力mg，而转动中心为A点

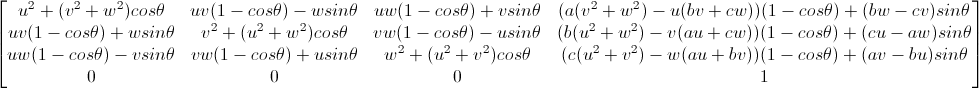
设提起点到重心的距离为d，不难看出，不倒翁受力矩大小为：

在一个deltaTime时间内，不倒翁的角速度改变量为：

在一个deltaTime时间内，不倒翁的以提起点A为旋转中心的旋转矩阵需要再转动角度：

这里牵涉到另外一个重要的问题：因为模型类的旋转矩阵默认是绕着自己的几何中心进行旋转，而此时点击的位置很可能（几乎是百分之百）不在物体的几何中心。此时，还需要一个算法来实现模型绕任意一个点进行旋转——而且这个旋转中心（考虑到鼠标的移动）应该是可变的。

这里又牵涉到线性代数的内容，而绕某一点旋转的算法实现原理非常复杂，具体推导可以参考以下的文章：[绕某一点旋转矩阵，分析，\_绕任意点旋转矩阵-CSDN博客](https://blog.csdn.net/u013407012/article/details/103721902)。我仅仅是挪用这个算法最后计算出来的旋转矩阵：



有了绕任意点旋转算法之后，对于不倒翁被提起需要进行的动作就变得明晰了： 在被提起的瞬间，将不倒翁的旋转中心放置到提起位置，然后应用之前所述的物理力即可。但还有一个难题：如何获取提起点？这个问题会在接下来的射线检测部分得到解决。

**III. 射线检测**

射线检测的部分，首先关心的是：如何在鼠标点击的时候，得到正确的世界坐标系下的射线？显然，射线的原点不需要过多思考，将其设定为当前照相机的世界坐标似乎是一个合乎逻辑的选择，所以接下来需要解决的是：知道射线顶点，如何正确求取射线的方向矢量？

当使用鼠标在画面上点击的时候，通过OpenGL提供的函数可以很轻松地获取当前鼠标点击的窗口坐标，假设这里获取的窗口坐标为，接下来将这个窗口坐标转化为归一化设备坐标(NDC)，画面左上角为(-1, 1)，右下角为(1, -1)。

摄像机获取一个点的世界坐标并投放到屏幕上，需要经过左乘投影矩阵和视图矩阵， 将整个视图投影到oxy平面上，而想要获取一个点击位置到摄像机的射线方向，其实就是它的逆过程，将归一化设备坐标转化为裁剪空间的坐标，然后左乘视图矩阵以及投影矩阵的逆矩阵，就可以得到实际的射线方向向量了。

现在已经获取了射线原点以及射线的向量方向，关于如何判断一个射线和模型的碰撞箱是否相交，如果相交，求取射线和碰撞箱的交点。这个判断的基本步骤如下：

1. 首先判断射线原点是否在模型的碰撞箱之内，如果在，则直接返回射线原点坐标为交点。

判断一个点是否在一个空间长方体内比较困难，但是判断一个点是否在一个长宽高和xyz轴平行的长方体内却很容易，不难推断，一个在这种长方体内的点的坐标x、y、z和这种长方体的最小点min和最大点max只需满足以下关系：

*min.x <= x <= max.x AND min.y <= y <= max.y AND min.z <= z <= max.z*

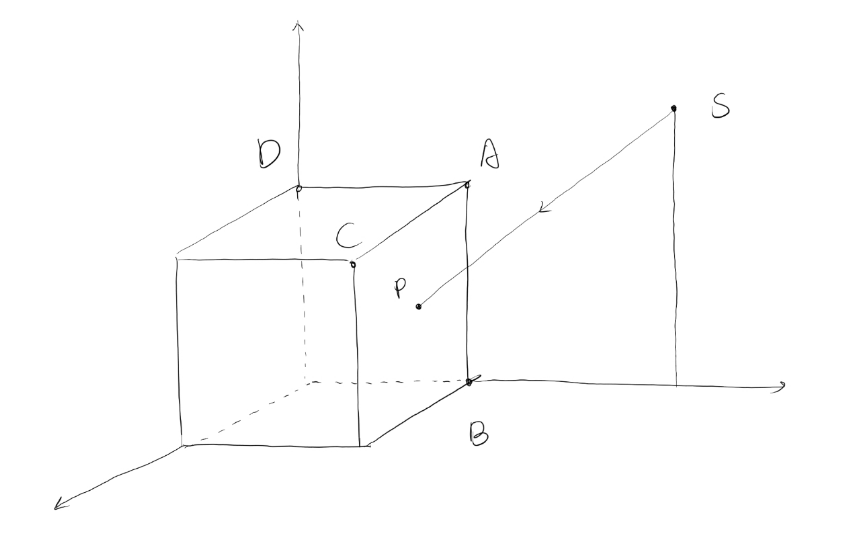
但因为模型可能会出现旋转和移动，即使一开始加载模型时，碰撞箱的各边和一条坐标轴平行，出现旋转和移动之后，边和轴便不平行了。而此时模型上的一点P的位置应该表示为：

只需利用基本的线性代数知识做一点变换，我们便不需要使用这个经过旋转和位移的复杂立方体，相反，可以对射线的原点S乘以矩阵的逆得到S’：

点S在经过位移和旋转后的复杂立方体内与S’在未经过位移和旋转的初始立方体内是等价的，由此可以快速判断出射线原点是否在模型碰撞箱之内。

2. 其次，因为很可能射线不会经过一个模型，首先计算出模型中心点到射线的垂直距离，如果垂直距离大于模型碰撞箱的体对角线半长，很明显该射线不会经过这个碰撞箱，直接退出，否则，进行下一步。这一步骤只需要基本的立体几何点乘操作即可计算出，所以不赘述。

3. 最后，由于此时保证射线原点并不在模型的碰撞箱中，而且射线和以模型中心为球心，碰撞箱体半径为球径的球有交点，可以认为此时射线和模型碰撞箱的几何关系很可能呈现类似以下的情况：



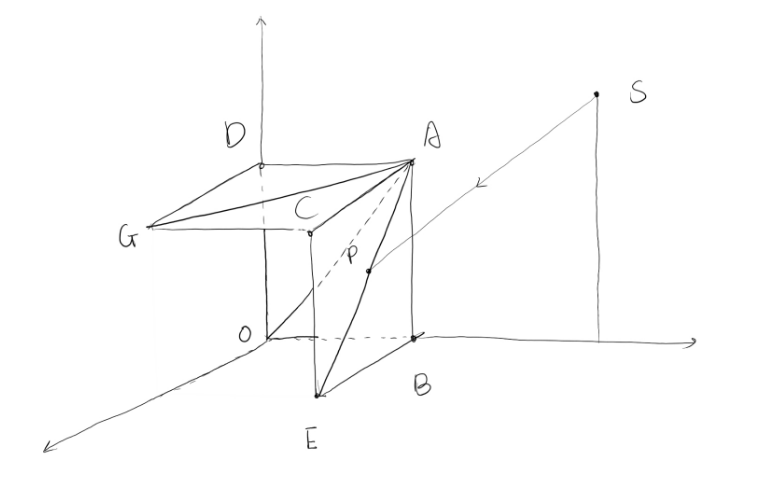
本图中，射线原点为S，和碰撞箱交点在ABC面上，S在ADB面上

当然，很可能射线的方向不幸和图中情况相反，但这种情况也不妨碍下面的判断。此时，在碰撞箱的8个顶点当中，必然可以找出一个距射线原点S距离最短的顶点（图中为点A，如果有最短距离一致的顶点，出于算法的一般性考虑可以任意选择其中之一，如果出于优化角度考虑，可以根据几何关系直接排除掉一些不需要判断的面）。沿着包含该点的边做延伸，可以发现三个距离射线原点S距离最短的矩形面（图中为矩形ACD、ABC、ABD，另一点未写出）。

和人眼最多只能看见长方体的三个面这一客观事实类似，从一点出发的射线如果和碰撞箱有交点，必然会首先击打在这上述三个面的其中之一，而不可能首先和距离较远的另外三个面相交，此时又简化了问题，我们只需要判断该射线是否与这三个矩形面相交即可。

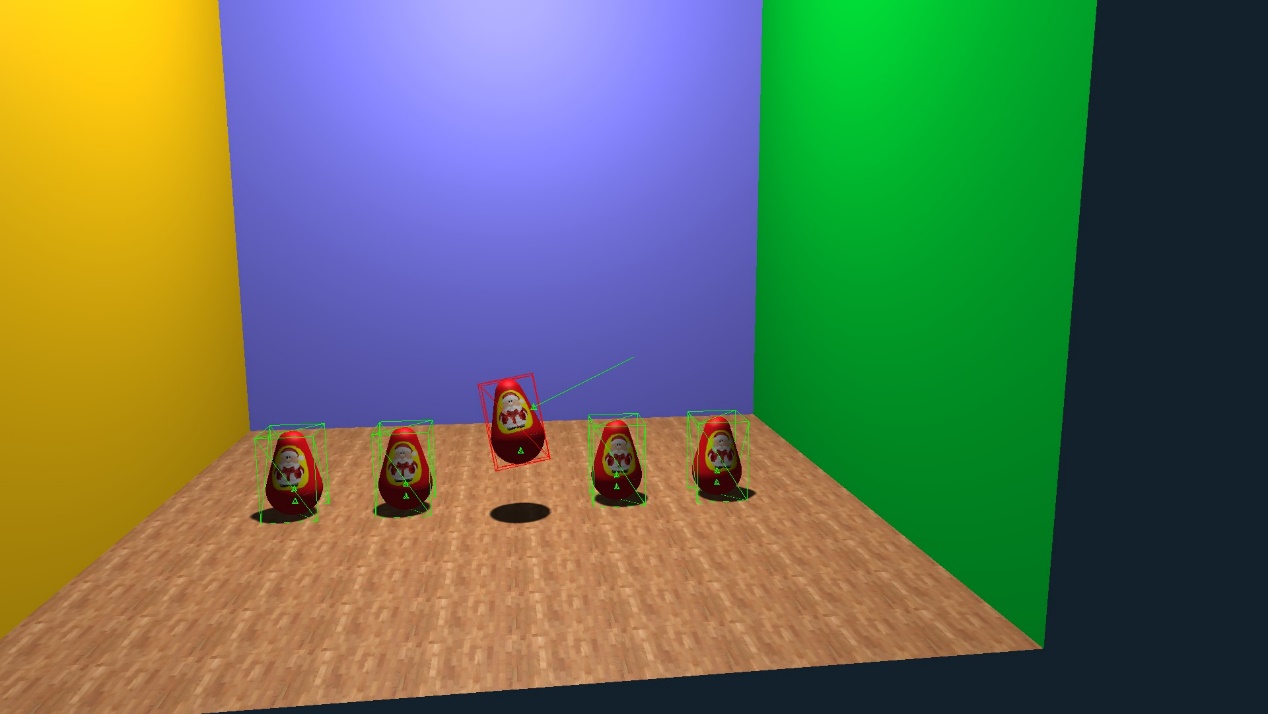
继续分析，显然任意四边形面可以分解为两个三角面，而关于空间射线与三角形面的交点判断，早有定论：[**Moller-Trumbore算法**](https://zhuanlan.zhihu.com/p/468132444)。该算法详情请参考文章，这里不赘述。

经过上述的分析，原本的射线与长方体相交问题边简化为以下的射线与六个三角形判断交点的问题：



图中例子，只需判断射线和AGD,ACG,AEC,ABE,ABO,AOD三角形的相交情况即可

这样一来，射线和碰撞箱求交点的问题就迎刃而解了。



模型碰撞箱受点击呈现红色，此时旋转中心被放置到了射线和碰撞箱的相交点，重心有向着旋转中心下方运动的趋势

**IV. 墙壁碰撞约束**

本程序中，model类中包括了maxLimit和minLimit两个三元组，分别储存模型运动可以达到的最小点和最大点，由最小点和最大点构成的边和坐标轴平行的长方体就是模型的运动范围。

因为模型不仅仅会因为彼此碰撞而发生反弹，程序还需要考虑模型和墙壁、天花板、地面的碰撞，判断是否发生碰撞很容易：只需要模型的位置P在某一个方向上满足：

就会发生碰撞。当然，模型在多个方向上同时满足这个条件也是可能的。该程序中，对天花板、四面墙壁的碰撞被视为完全弹性碰撞（不损失速度），只需要将速度在对应方向上的分量置反，将物体进行一个反向的微小位移即可，但对于地面的碰撞，因为物体受到重力影响，而我又希望最终物体可以静止在地面上而不发生抖动，所以需要进行一些特殊的处理。这里我的处理步骤如下：

1. 对于速度较快的下落运动，当做完全弹性碰撞（对应分量乘以模型的碰撞系数）处理。

2. 对于速度较慢的下落运动，在弹性碰撞的基础上，增加对动能的损耗。

3. 在物体的高度较低（一般在几十厘米左右）时，停止重力的作用。

这样既可以实现物体的下落动能损失，而且可以防止物体在近地面的抖动，获得了比较良好的视觉效果。

**V. 模型材质变化**

这个功能在本程序中的实现并不复杂，因为所有的物体只有一种材质，只要在小球和不倒翁或者墙壁碰撞的时候切换为对应的材质即可。具体内容在此不赘述，可参考model类中的hitWall函数和changeTexture函数。

**VI. 光照以及阴影实现**

光照和阴影实现可以参考LearnOpenGL教程的“[高级光照](https://learnopengl-cn.github.io/05%20Advanced%20Lighting/01%20Advanced%20Lighting/)”部分，我只做了一点光照参数上的修改，让其获得一个比较好的视觉效果，具体内容不赘述，实现可以查看default.frag着色器文件。

**VII. 粒子系统实现**

本程序总共写了三种粒子：拖尾(TRAILING)、扩散(SPREAD)、以及上升(EVAPORATE)粒子，其中拖尾粒子是可再生粒子，而后两种粒子的生命周期只有一次。每种粒子都定义了一个初始的颜色值baseColor，随着粒子的生命流逝，这些粒子以自定义的颜色改变量发生颜色变化，比如火球的拖尾颜色从最亮的灰白色逐渐过渡到暗红色，小球燃烧从一开始的亮橙色快速过渡到黑色，营造出一种燃烧的感觉。

在本程序中，每个粒子实际上都是一个所有面颜色相同的小等四面体，当然，因为计算粒子是一个开销非常大的过程，所以本程序中粒子的数量比较少（最多的拖尾效果也只用了3000个粒子），但如果粒子的密度大，还是可以获得不错的视觉效果。

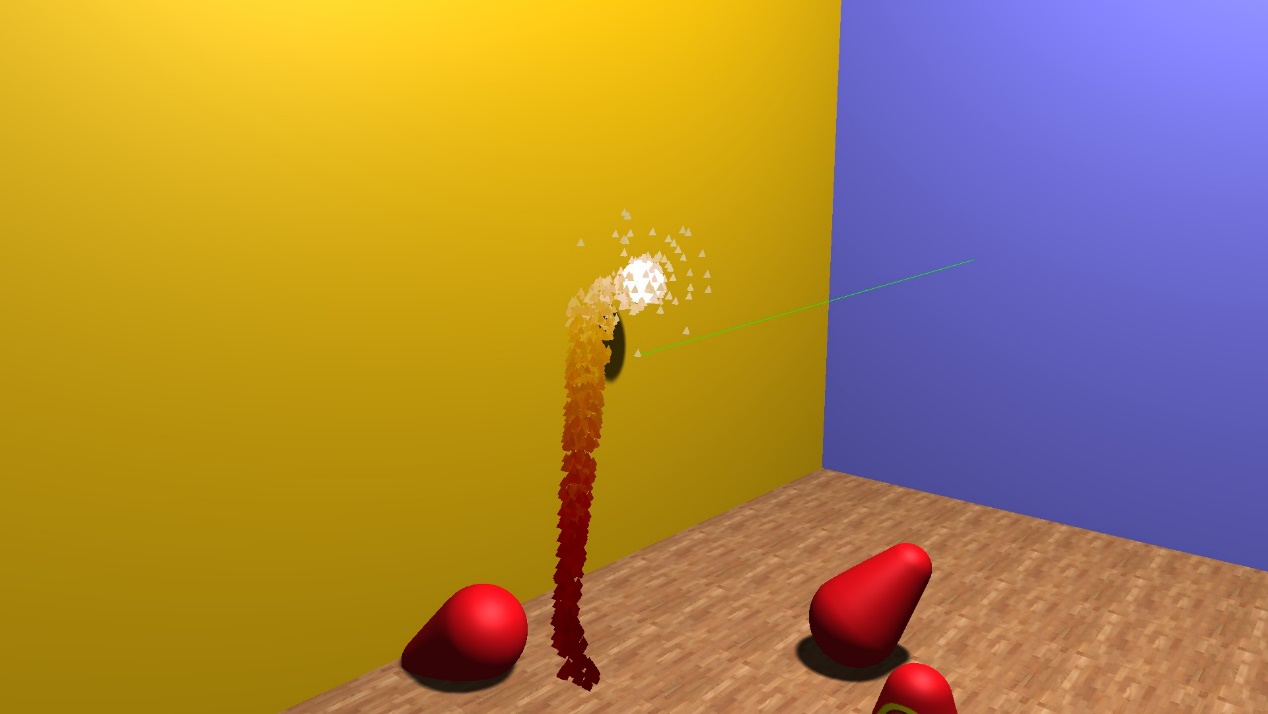
“火球燃烧”效果的实现方式是：将火球本身视为一个自发光的白色球体，从火球的表面产生一些亮白色的粒子，粒子的运动方向的反向射线方向会汇聚于一点（也就造成拖尾前面粗后面细），在运动过程中粒子颜色不断改变。

“碰撞产生火花”效果的实现方式是：在检测到碰撞的时候就播放一遍SPREAD粒子的效果即可。

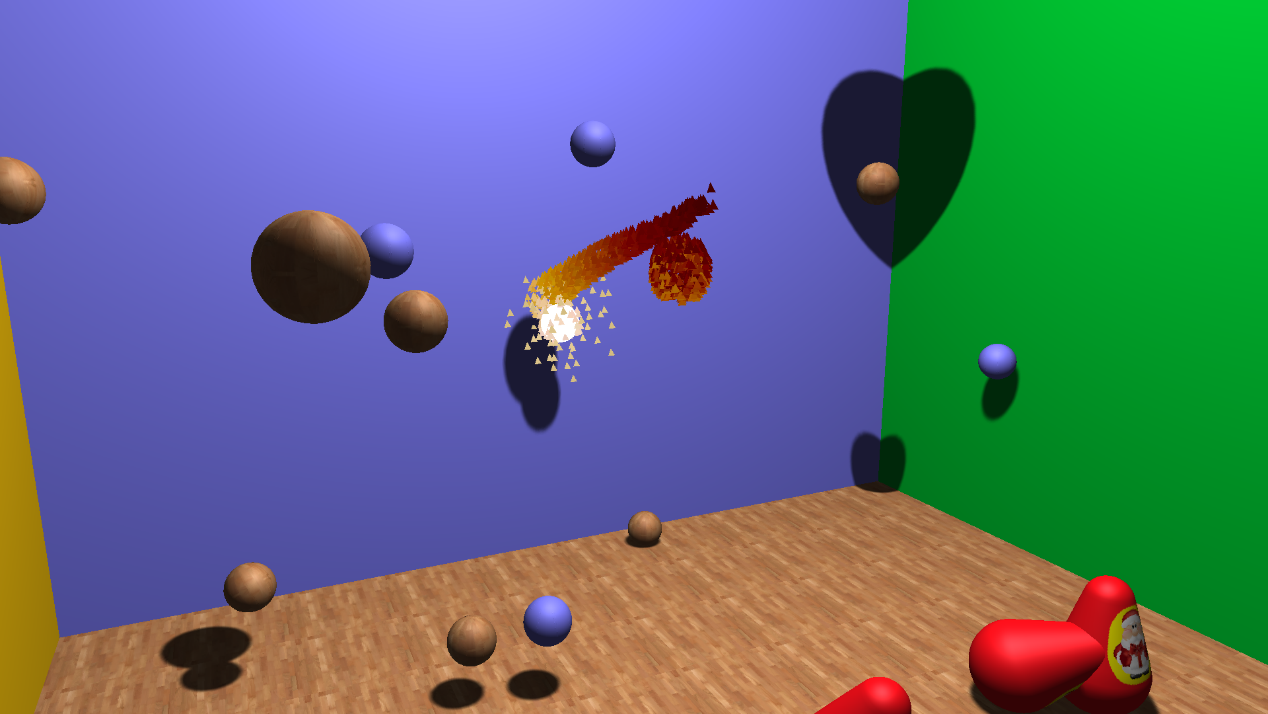
“小球燃烧并消失”效果的实现方式是：在小球被碰撞的瞬间，将小球切换为不可视状态，同时在小球的表面一次性产生所有的EVAPOTATE粒子，这些粒子一开始是亮白色的，随着时间的流逝逐渐变为暗红色，最后上升消失，模拟小球燃烧成灰烬并消散的效果。



不倒翁、小球和火球



火球和墙壁碰撞，转向的同时出现一些粒子散射



火球和被碰撞后正在燃烧的小球

## 4. 总结

本次大作业耗时较长，但要求任务基本完成，而且从中获益良多，了解了很多图形学的基本原理和应用。

## 5. 程序操作说明

点击src目录下的start.exe开启程序

WASD：控制摄像机移动

QE：控制摄像机垂直上升或下降

C：显示碰撞箱以及模型重心、旋转中心

鼠标左键点击画面并拖动：转动摄像机

鼠标左键点击不倒翁重心上方位置：拖动不倒翁

鼠标左键点击不倒翁重心下方位置：给不倒翁施加视线方向的力

B：在画面中加入多个小球，再按移除小球

F：在画面中加入火球，再按移除火球

R：重置摄像机位置

M：重置模型位置

P：暂停所有运动（包括粒子），再按恢复，暂停时不可拖动不倒翁，但可移动画面

ESC：退出程序