**CG2023大作业实验报告**

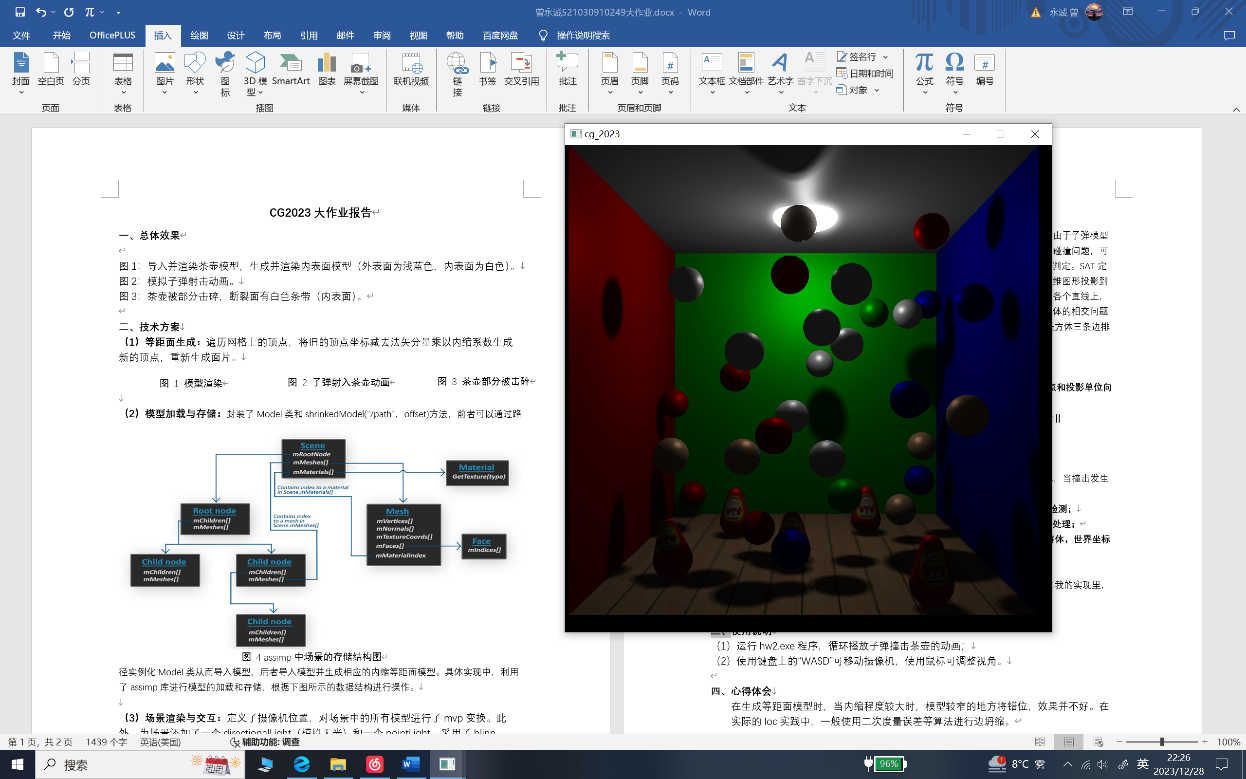
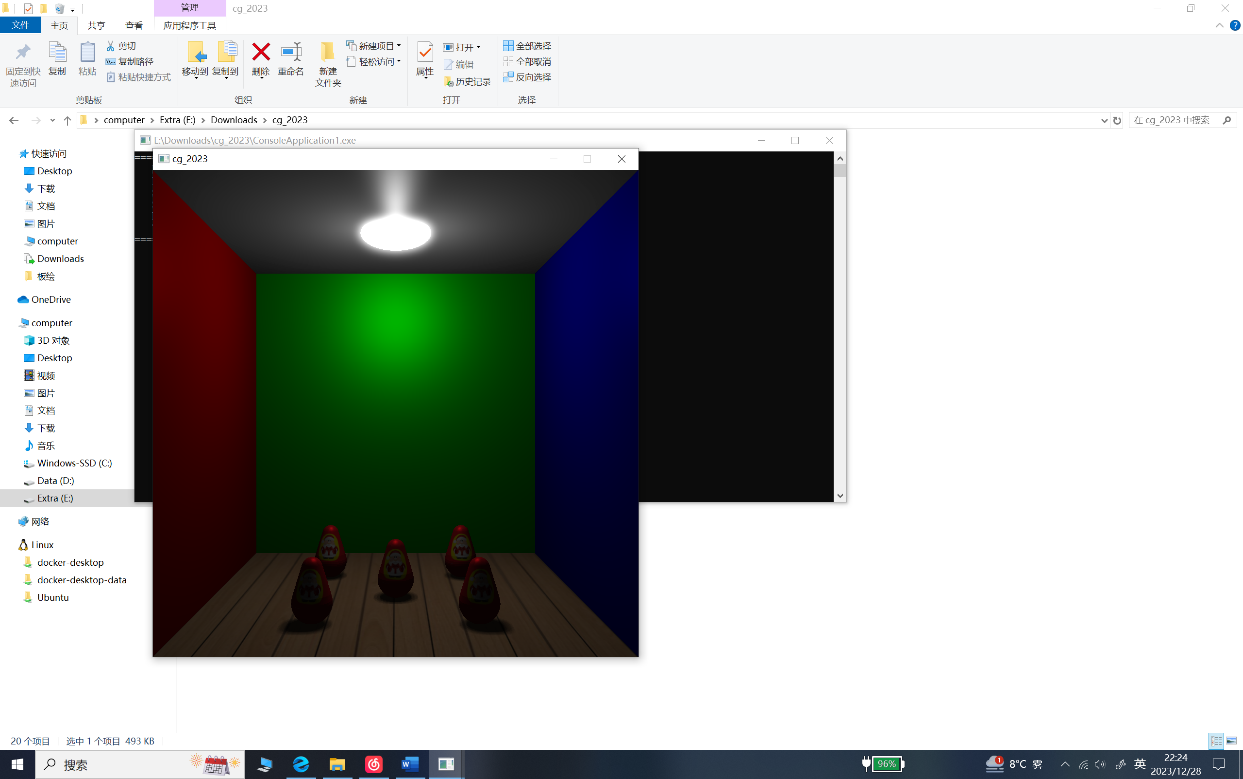
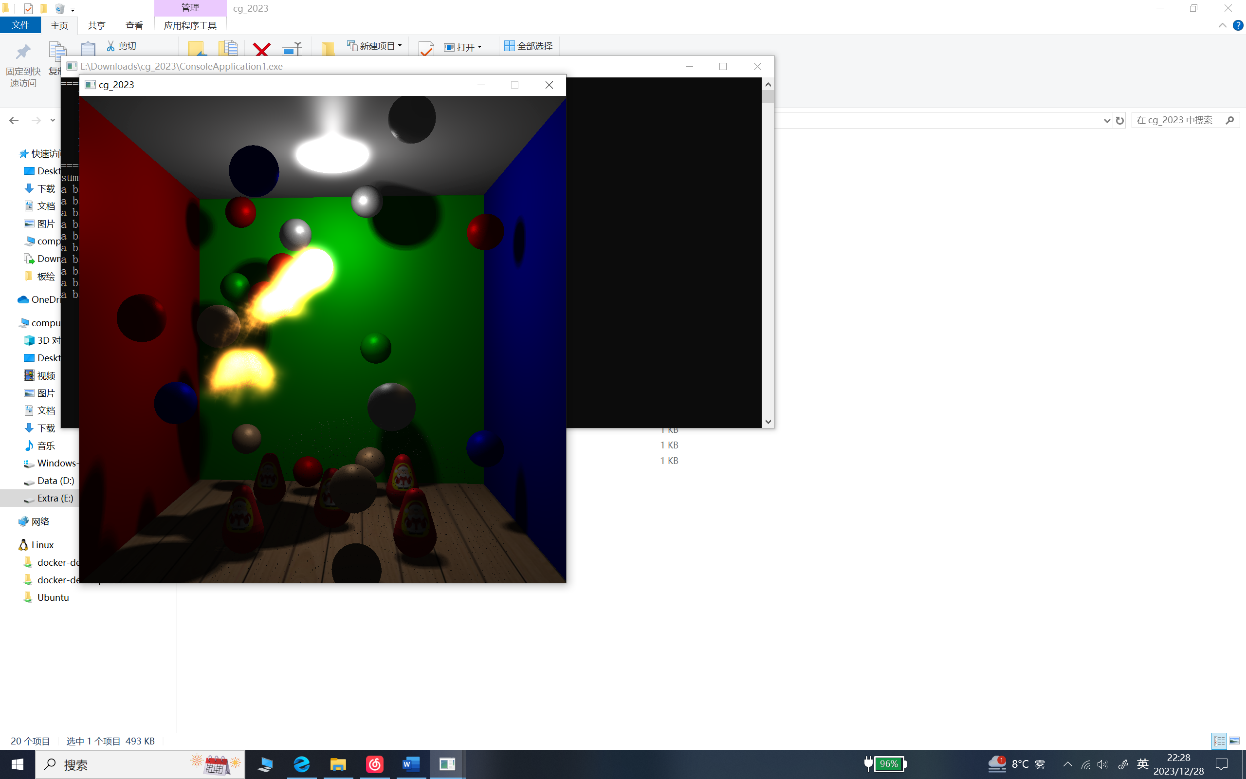
1. **总体效果**

图3火球（右）和被燃烧成灰烬的小球（左）

图1场景加载，光照渲染和阴影映射展示

图2小球在场景中碰撞

**二、技术方案**

**（1）场景构建：**如图所示，墙面、天花板和地板由glDrawArrays（GL\_TRIANGLE,…）方法直接绘制，不倒翁模型由assimp库直接导入并绘制，场景经由MVP变换显示到视图中。  
  
**（2）不倒翁交互：**把鼠标的光标放在不倒翁重心下，保持左键可拖动不倒翁，放在不倒翁重心上，则可推倒不倒翁，使其摇晃。在实现鼠标与不倒翁的交互时，采用了鼠标“拾取”技术，即通过屏幕坐标逆推出世界坐标。具体实现如下：  
 （a）通过鼠标在屏幕上的位置读取深度值z1，得到屏幕坐标（x1, y1, z1）；

（b）把坐标除以窗口宽高进行归一化得到（x2, y2, z2）；

（c）把坐标映射到（-1，1）标准空间得到（x3, y3, z3）；

（d）根据z值，近平面near\_plane和远平面far\_plane算出齐次坐标w，将坐标的各个分量再乘以w得到经MVP变换后的坐标（x4, y4, z4, w）；

（e）依次乘以投影矩阵Projection的逆和视图变换矩阵View的逆得到鼠标位置对应的世界坐标(x5, y5, z5)；

（f）得到世界坐标后，将其与不倒翁的aabb盒进行相交判定，然后取其y值与重心坐标y值进行比较，判断动作是拖动还是推倒。

**（3）不倒翁物理仿真：**在把鼠标放在不倒翁重心上方时，根据帧与帧之间的dx和dy晃动不倒翁。dx决定了不倒翁沿着视角cameraRight法矢摇晃的幅度，dy决定了沿着视角cameraFront法矢摇晃的幅度。用简谐振荡实现不倒翁的摇晃动画仿真，不倒翁的倾角计算公式如下：

其中为最大振幅，由鼠标坐标的偏移量（或碰撞小球的速度）决定，为衰减项，表征不倒翁的倾角因为摩擦力和空气阻力而逐渐减小的规律，项表征不倒翁摇晃的周期性。此外，如果不倒翁已经处于运动状态TumblerState::Moving时，又遭到鼠标的拖动或小球的撞击，则会进行简谐震荡的合成，即对原有的振动方向和新的振动方向进行矢量合成，振幅更新为（根据能量守恒）。  
  
**（4）小球物理仿真：**

（I）生成：按下“B”键，场景将随机出现30个小球，生成小球时，将空间均分为125份，通过c++algorithm库中的shuffle方法打乱容器元素，取前30个序号，即随机的30个位置。

（II）运动：小球在房间中做匀速运动，根据时间帧和速度矢量对其位置进行更新；  
（III）球与球之间的碰撞检测：在双重循环中，对每两个小球进行碰撞检测，小球发生碰撞当且仅当 | && |) ，前者限制了距离关系，后者限制了速度关系，以免小球在碰撞发生后的第二帧被重复检测出碰撞。碰撞发生后，在小球质量相等且均匀的前提下，发生对心碰撞，速度更新公式如下：||)

（IV) 球与墙壁之间的碰撞检测：单独对球和六个方向的墙面进行碰撞检测，以法矢为（1,0,0）的墙面为例，发生碰撞当且仅当

&& < 0

碰撞发生后，假设小球不损失能量，速度更新为glm::reflect(,)

（V）球与不倒翁之间的碰撞检测：将不倒翁视作aabb包围盒，问题抽象为球体与立方体的求交问题，先分别求出xyz三个维度下球与包围盒的最小距离：  
 = max(box.min.x , min(，box.max.x)) ……

取距离最小的维度a，然后计算，与不倒翁发生碰撞当且仅当：

&&

前者描述了限制关系，后者限制了速度关系。碰撞发生后，假设小球不发生能量损失，速度更新为

glm::reflect(,)，不倒翁获得一个方向为，振幅与小球速度成正比的振动分量。

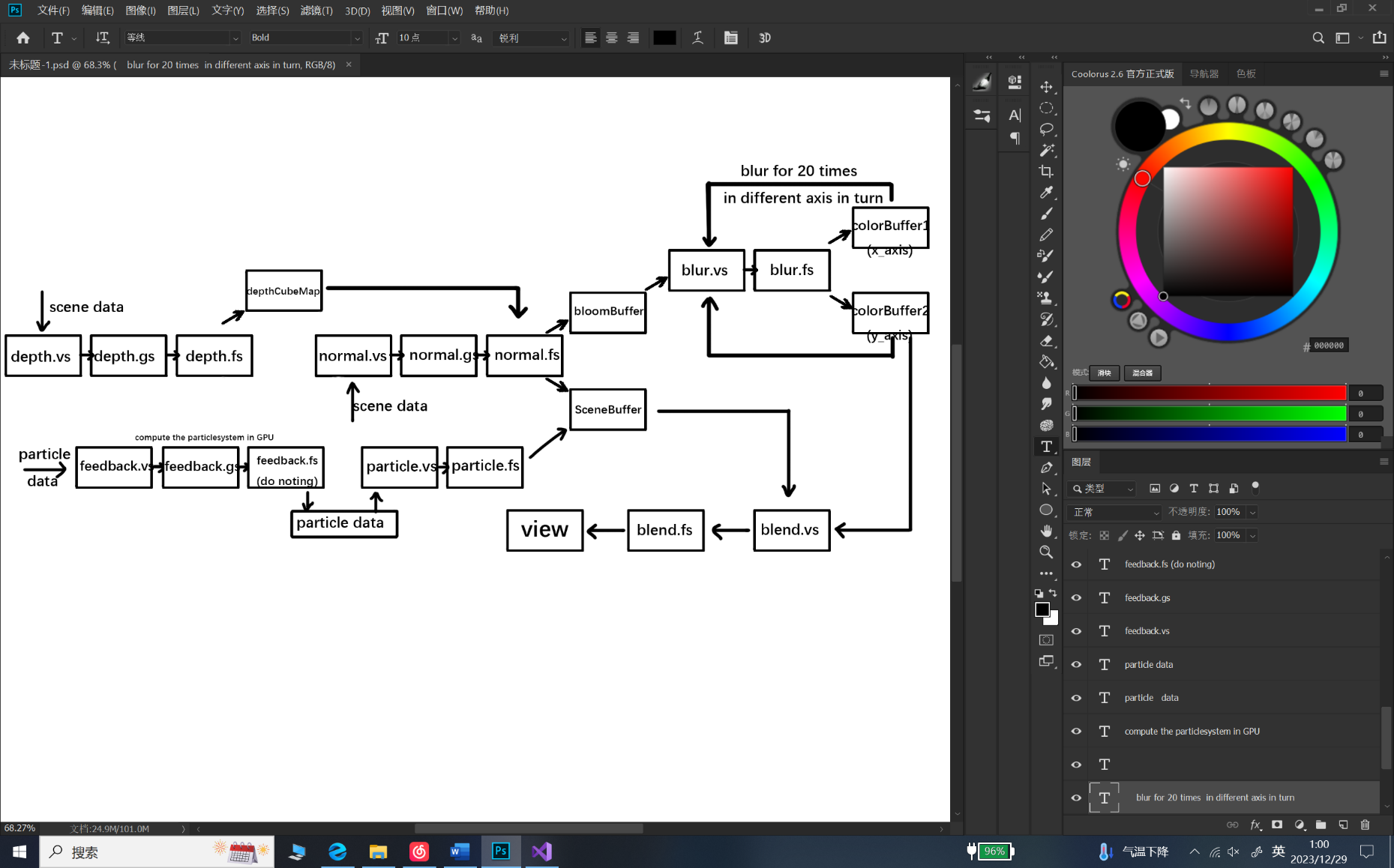
**（5）场景渲染：**如图1，场景实现了blinn-phong光照渲染，shadow mapping阴影映射和bloom泛光效果。

图4渲染管线框架

如上图所示，渲染流程如下：  
（a）以灯为视角渲染场景，经过**depthShader**生成立方体深度贴图；  
（b）在**normalShader**中，利用（a）生成的深度贴图渲染带阴影效果的场景，同时输出fragColor到sceneBuffer，brightColor（亮度大于1的值）到bloomBuffer；   
（c）在**feedbackShader**中的几何着色器进行粒子系统的计算与更新；  
（d）在**particleShader**中进行粒子系统的渲染，写入sceneBuffer中；  
（e）在**blurShader**中，对得到的bloomBuffer进行高斯模糊，交替进行横向和纵向的模糊，重复20次，最后输出；  
（f）将输出的buffer与之前的sceneBuffer一同作为贴图，通过**blendShader**管线渲染视口长方形，在片段着色器中进行贴图的混合，得到最终效果；  
（g）其他实现细节：在进行阴影映射时，为了模拟阴影模糊的效果，将对深度进行不同方向的多次采样，采样的范围与camera与fragPos的距离有关，距离越大，采样范围越大，阴影越加模糊；为了防止出现条纹状阴影，在对片段的深度与贴图中的最小深度进行对比时，应当减去一个合适的偏移量bias。

**（6）火球和灰烬特效（粒子系统）：**（a）粒子系统ParticleSystem设计：本项目中，为了构建火球和灰烬特效，我实现了一个粒子系统。粒子类Particle拥有七个属性，类型type（LAUNCHER表示发射器，ROCKET表示自由粒子），age为粒子的剩余寿命，lifetime为粒子的寿命，alpha值为透明度，size为粒子的大小，position和velocity分别为粒子的位置与速度。粒子系统类ParticleSystem主要有两个方法，构造函数以及renderSystem方法，renderSystem方法又将调用renderParticle和updateSystem两个子方法，依次进行粒子系统的渲染和更新计算。

（b）Transform Feedback技术：由于单个粒子系统的粒子数量多达上万，在CPU上进行计算显然不够合理，为此，我利用了opengl在3.0版本后加入的特性transform feedback，该特性允许我们在GPU中分配一块transform feedback buffer，可以接收顶点着色器（或者几何着色器，如果有）的运算结果，并且这个buffer可以被CPU相对高效地读取。根据这个特性，我把粒子的更新运算放到了feedbackShader中的几何着色器。具体操作就是先把要更新的粒子作为VAO输入顶点着色器，顶点着色器简单地传递数据，在几何着色器中设置一些必要的uniform参数，并且进行最主要的更新计算，然后CPU再读取这个buffer，从而得到更新后的粒子属性。以上流程用到了两个transform feedback buffer，运算和读取是在这两个buffer中交替进行的。

（c）火焰物理仿真：对于一个模拟火焰的粒子系统，在初始化时，首先会生成一定数量的launcher粒子，它们只负责发射粒子，本身并不移动。为了模拟火焰效果，发射器粒子的分布遵循中间密，两边疏的高斯分布，具体实现中用随机数分布来模拟高斯分布。然后，在feedbackShader的几何着色器中，根据粒子的类型进行计算，如果是发射器，则更新它的age，当寿命结束时，重置年龄并发射rocket粒子；如果是rocket粒子，则更新它的age，alpha，size，velocity和position，同样的，size和alpha值要遵循开始和结尾小，中间大的高斯分布。在进行位置更新时，CPU将提前传入一个加速度a和离心率eccentricity，a用来模拟重力加速度（主要是为了灰烬的特效模拟），rocket粒子的初始速度将加上eccentricity \* vec3（position.x, 0, position.z）分量，从而使粒子呈现一定的散开趋势。在进行场景渲染时，开启blend，并以点精灵和billboard的方式对粒子进行渲染。此外，系统中粒子默认速度朝向y轴正向，在实现火球的特效仿真时，可直接通过rotationMatrix调整粒子发射角度。  
  
（d）灰烬特效仿真：当火球与小球或者不倒翁发生碰撞时，一方面，生成粒子系统实例ashes，传入合适的参数（比如一个较大的离心率），然后把系统的朝向调整为y轴负向，模拟灰烬特效；另一方面，小球的破碎和自由落体动画，与作业2类似，只需在几何着色器normal.gs中更改小球的三角形面片位置即可。具体更新公式如下：

\* t + 0.5 \* G \* t \* t \* vec3(0, -1, 0)

其中n为法矢，G为重力加速度，把运动分解为沿着法矢的匀速运动和自由落体即可。

**三、使用说明**（1）WASD键控制摄像机前后左右移动，左shift下降，space上升；  
（2）长按鼠标左键，同时把光标放置在不倒翁重心下或上，微微移动鼠标，可使其移动或者摇摆；

（3）按下B键，场景中随机生成30个速度方向随机的小球；

（4）按下F键，召唤一个朝着鼠标方向飞行的火球，可多次召唤。

1. **心得体会**

通过大作业的实践，体会到了渲染管线的设计流程，增进了对图形编程的理解。在实践中，发现当粒子数量达到六位数时，或者火球数量太多（光源过多）时，开始出现卡顿，经过查阅资料，得出结论：如果要进行更复杂的渲染，如PBR以及多光源的阴影映射，可以考虑延迟渲染技术。