# 计算机图形学大作业报告

徐琢雄 521021910560

## 1. 操作说明

构建：在文件夹 HW3 中，用 CmakeList.txt 构建项目，编译 target homework3，可执行文件在 bin 目录下。

这个项目可以使用的按键有：W、A、S、D、F、G、I、左 SHIFT、鼠标移动、鼠标左键。

程序运行场景的初始状态为一个房间，有五个不倒翁，30 个小球在随机位置，随机方向运动。

具体操作方式为：

1.**方向和视角**：鼠标移动可以转动视角，WASD 方向键可以进行当前视角下的前后左右移动。

2.**火球**：按下 F 键出现火球以及碰撞的粒子系统，再次按下火球消失，不参与碰撞。

3.**暂停**：按住左 SHIFT 键，所有运动暂停，此时仍然可以用 WASD 和鼠标在场景中移动，观看某个瞬间的物品；松开左 shift 运动继续。

4.**鼠标拖动不倒翁**：按下 G 键，视角锁定，屏幕中出现白色的鼠标（鼠标也可能在屏幕外面，移动鼠标让其回到屏幕上）。鼠标悬浮在不倒翁上按住左键，不倒翁大小变大表示被选中，此时可以移动鼠标（保持左键按下），即可拖动和晃动不倒翁。

5.重新初始化：当经过一段时间，小球和不倒翁都静止后，可以按 I 键重新初始化小球的位置和速度。

## 2. 系统设计

### 2.1. 光照设计

#### 2.1.1.光照模型

本项目的光照模型采用了 Blinn-Phong 光照模型。Blinn-Phong 光照模型是一种基于经验的光照模型，用于渲染计算机图形中的表面光照。该模型基于 Phong 光照模型进行改进，以提供更加合理的高光反射效果。

1. 环境光照（Ambient Lighting）

Blinn-Phong 模型考虑了环境光对表面的影响，通过一个常数项 Ka 表示表面对环境光的反射率。环境光照表达式为：

$$I_{ambient} = I_a \cdot K_a$$

其中，$I_a$ 是环境光强度，$K_a$ 是环境光反射系数。

2. 漫反射光照（Diffuse Lighting）：

模型考虑了光源方向和表面法线之间的夹角，通过漫反射系数 Kd 来表示表面对漫反射光的反射率。漫反射光照表达式为：

$$I_{diffuse} = I_d \cdot K_d \cdot \max(0, \vec{L} \cdot \vec{N})$$

其中，$I_d$ 是光源强度，向量 L 是光源方向，向量 N 是表面法线。

3. 镜面反射光照（Specular Lighting）：

Blinn-Phong 引入了一个半程向量（Half Vector），表示光源方向和视线方向的平均方向。通过镜面反射系数 Ks 表示表面对镜面反射光的反射率。镜面反射光照表达式为：

$$I_{specular} = I_s \cdot K_s \cdot (\vec{H} \cdot \vec{N})^{n_s}$$

其中，$I_s$ 是光源强度，H 是半程向量，$n_s$ 是反射高光度。

4. 合并光照成分:

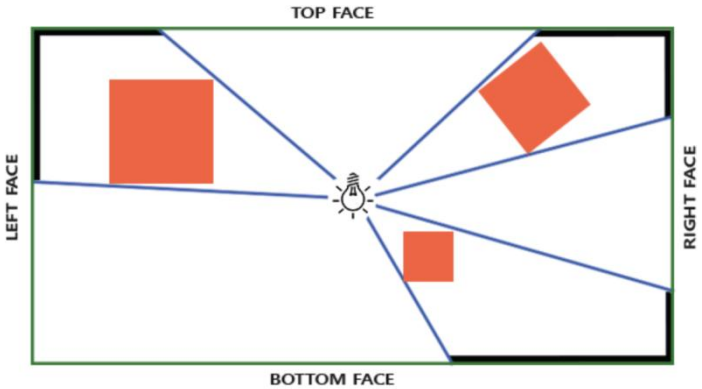
最终的光照效果由环境光照、漫反射光照和镜面反射光照三者叠加而成：

$$I_{final} = I_{ambient} + I_{diffuse} + I_{specular}$$

Blinn-Phong 光照模型在实际渲染中广泛应用，通过调整反射系数和高光参数，可以获得各种表面材质的逼真光照效果。

### 2.1.2. 阴影模型

项目的阴影模型采用点光源产生的阴影效果，光源位置在房间顶部正中，被光源照射到的位置有高光、漫反射、环境光，未被光源照射的位置只有环境光，阴影模型在实现中，先渲染 6 次，得到 6 个面的深度贴图；在实际渲染中，根据深度贴图的数值判断对应的面是否受到光照，再采用对应的光照模型。



## 2.2. 物理设计

### 2.2.1.物体的理想化模型

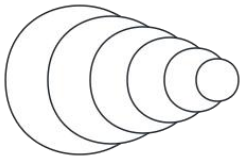
在物理系统的设计中，我对于这个场景做了如下的理想化近似：

1.所有物体都是刚体，不存在非弹性形变

2.所有物体都只收到重力和弹力的作用，在物体碰撞时，遵循动量守恒定律；当刚体受力时，遵循刚体力学

3.小球近似于质量均匀分布的小质量刚性球

4.不倒翁近似为多个小球的嵌套。不倒翁近似为 6 刚性小球叠合的大质量组合刚体，不倒翁是轴对称的，而且底面是一个半球，不倒翁的重心位于底部球体中心的下方，所以在重力作用下不倒；在碰撞时，不倒翁的碰撞点来自于不倒翁上被撞到的小球。不倒翁剖面示意图如下：



5.火球近似于一个速度不会改变的刚性球体，质量和小球一样。

### 2.2.2.力学设计和运动学设计

本项目中，小球由于其对称性，受到的力总是指向其球心，所以小球和其余物体的碰撞都按照动量守恒计算，不涉及力学计算；小球之间碰撞采用完全弹性碰撞，用动量守恒得出碰撞后速度（最终速度还要乘上衰减率）：

$$v_{1f} = \frac{(m_1 - m_2) \cdot v_{1i} + 2 \cdot m_2 \cdot v_{2i}}{m_1 + m_2}$$

$$v_{2f} = \frac{(m_2 - m_1) \cdot v_{2i} + 2 \cdot m_1 \cdot v_{1i}}{m_1 + m_2}$$

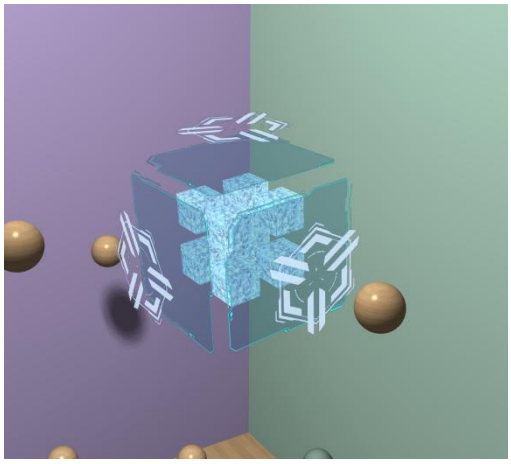
不倒翁的情况比较复杂，因为等效于一个质量非均匀分布的刚体，所以受力时，先计算对不倒翁的力矩矢量（以地面接触点为转轴）。然后，对于其姿态变化，使用刚体力学求出其角加速度，乘两帧间隔 DeltaTime 求得其角速度，然后得到角度变化量；对于其位置变化，本项目假设小球和地面之间无滚动，所以由角速度在水平方向的分量可以求得不倒翁的速度，进而求得不倒翁在地面的位移。

## 2.3. 特效设计

对于火球效果，我没有采用传统的 Sprite 粒子的火焰效果，而是设计了新的尾迹效果，整体美术风格为赛博朋克式的科幻效果，主要表现在粒子构成的空间吸收和坍缩特效。

粒子系统的特效主要有四大部分：

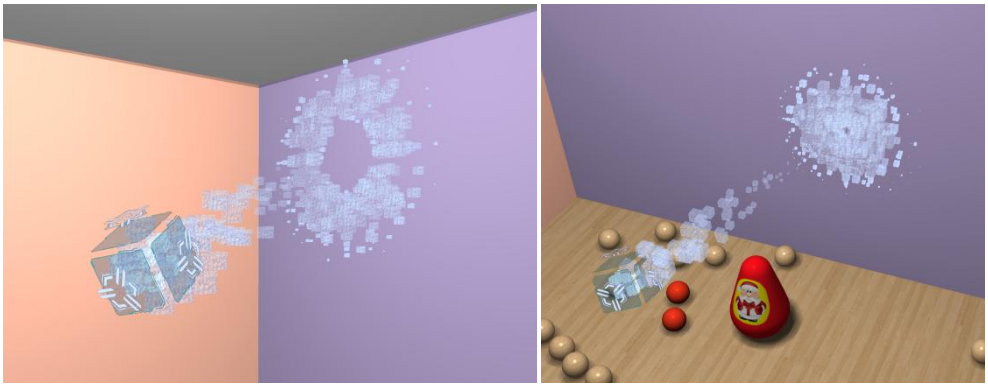
1.火球：火球采用三层结构，最内层是八个晶体核心，外面两层是有呼吸效果的能量罩。



单独火球效果

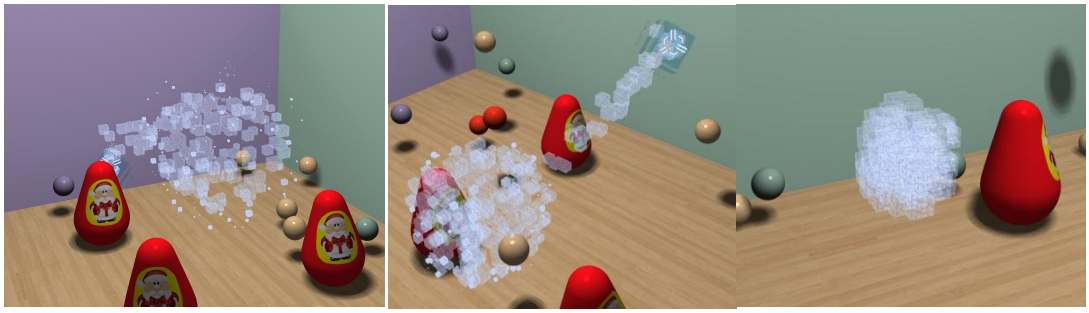
2.火球尾迹:火球尾迹采用多个贴图的立方体块，在火球的行进路线上随机生成，并且在生命周期内大小缩小，透明度降低，表示火球的能量使空间出现空洞的效果

3.火球和墙面碰撞：火球和墙面碰撞采用碰撞点开始不断生成方块向四周蔓延，每个方块在生命周期内先缓慢变大，然后急剧缩小，表现空间的扩散效果



火球，尾迹与碰撞墙面特效

4.火球和物体碰撞：火球与物体碰撞时，在其周围一个不断缩小的范围内随机生成方块并且向中心运动和缩小，表现空间坍缩效果，当方块收缩到最小后还有一个能量爆发效果。



小球被撞击后的不同阶段

## 2.4. 反走样

本项目采用 MSAA 多重采样技术实现抗锯齿效果，大致原理是：

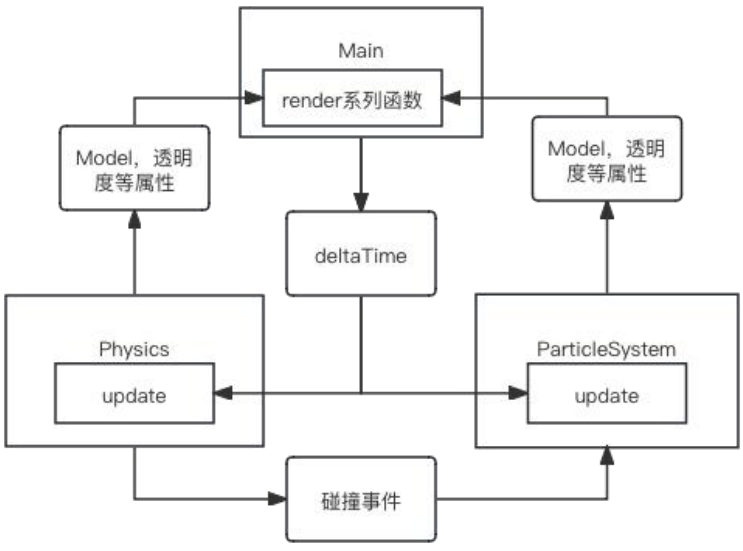
1.在渲染管线的光栅化阶段，对每个像素进行多次采样，每个采样点有自己的颜色和深度值。
2.对于每个像素，判断它是否被多个图元覆盖，如果是，就计算每个图元对应的采样点的覆盖比例，称为覆盖因子。
3.根据覆盖因子，对每个采样点的颜色和深度值进行加权平均，得到最终的像素颜色和深度值。
4.将最终的像素颜色和深度值写入颜色缓冲和深度缓冲，用于后续的渲染过程。

# 3. 实现方案

## 3.1. 总体结构

项目的总体架构是一个 Main 函数负责渲染和处理输入；Physics 类负责物理仿真实现；ParticleSystem 类负责粒子系统；Model 类负责渲染 obj 模型。

物理系统和粒子系统不负责渲染，它们的职责是提供不同类型物体的 model 矩阵和位置，速度，透明度等参数，供 Main 函数实现渲染；Main 函数在每个渲染周期会用时间小量 delta time 给物理和粒子系统更新状态，然后在渲染时，不同的渲染函数从物理和粒子系统中获取需要的参数，进行渲染；另外，物理系统也通过回调函数向粒子系统发送碰撞事件。总体大致架构如下：



## 3.2. 主程序 Main

### 3.2.1.光照与阴影实现

本项目的点阴影由深度贴图实现，具体流程如下：

1.创建深度立方体贴图：使用一个帧缓冲对象，将一个立方体贴图纹理附加到它的深度附件上，用来存储点光源周围的深度值。

2.渲染场景到深度立方体贴图：使用一个几何着色器，将场景中的每个三角形投影到六个不同的光空间变换矩阵中，分别对应立方体贴图的六个面。在片段着色器中，计算片段到光源的距离，并写入到深度值中。

3.使用深度立方体贴图渲染阴影：使用一个光照着色器，根据片段的位置和光源的位置，计算一个方向向量，用来从深度立方体贴图中采样最近的深度值。然后将当前的深度值和采样的深度值进行比较，判断片段是否在阴影中。最后，根据阴影的结果，调整光照的漫反射和镜面反射分量。

### 3.2.2.Shaders

本项目使用了四个可编程管线的着色器，分别用于不同目的：

1.light：绘制点光源和粒子，为了性能，颜色只取环境光，可以绘制半透明物体

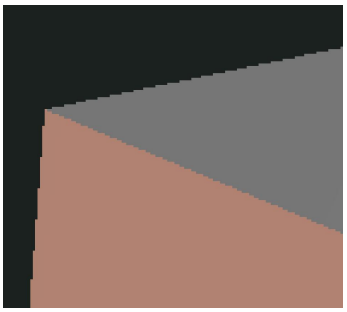
2.point_shadow_depth：绘制深度贴图，存储在缓冲区，供真正的渲染步骤使用

3.point_shadow：渲染带阴影的物体，光照采用 Blinn-Phong 光照模型

4.cursor：渲染鼠标，只渲染在屏幕空间，不做坐标变换

### 3.2.3.MSAA

在渲染初始化阶段设置采样数并且启动多重采样，对比效果如下：



未使用 MSAA

MSAA

### 3.2.4.场景渲染

Main 函数的场景渲染分为初始化和渲染循环两个阶段。

在初始化阶段，Main 函数主要加载了 Glad 和初始化窗口，绑定鼠标，初始化物理和粒子系统，加载纹理等操作，以加快渲染阶段的效率。

渲染阶段，在每个渲染循环中，Main 函数的主要流程如下：

1.Main 函数计算和上一帧的时间间隔，并且处理输入事件，将事件间隔发送给物理和粒子系统，更新其状态。

2.开始渲染，Main 函数依次做的工作是：

1）创建深度立方体映射变换矩阵、

2）渲染场景到深度贴图、

3）正常渲染带阴影的场景、

4）渲染不参与阴影计算的物体（光源，火球，粒子，鼠标）

3.交换帧缓存

这几个流程的顺序是固定的，因为先获得深度缓冲才能得到阴影的分布；另一方面，半透明的粒子要等到不透明的物体（房间、小球、不倒翁）渲染完成后才能渲染，否则会占据深度缓冲而“遮蔽”后面的物体。

## 3.3. 物理系统 Physics

### 3.3.1.物体状态参数

小球的主要参数和介绍如下：

exist：表示小球是否存在

idx：表示小球的编号，其中 idx=0 的小球就是火球，这样的设计是因为火球的运动学计算和小球答题类似，所以只要从编号区分开火球和小球即可。

radius，position，velocity：小球的半径，位置和速度

color：小球的颜色类别，这是为了实现小球撞击物体时变成物体的颜色或者材质

不倒翁的主要状态参数和介绍如下：

radius：半径，特指底面半径

m_bias：重心偏移底面球体的距离

position，velocity，angular_velocity，rotate_axis，I：小球的位置，速度，角速度，旋转轴，转动惯量

model：不倒翁的姿态矩阵，特指不倒翁的旋转导致的姿态变化（即不倒翁在模型空间的姿态），将旋转和位移分离，能够大大减少计算量，最后将位置，姿态，大小三个分量组合就能得到不倒翁的模型矩阵，从而将不倒翁从局部空间变换到世界空间。

### 3.3.2.力学和运动学实现

力学和运动学实现在 update 函数中，因为力是靠角速度最终影响不倒翁的姿态和位置的，所以所有的力的作用都只要改变角速度这一个变量就可以，减少了代码冗余和不必要的耦合。

在每一帧的状态更新中有如下几个步骤：

1. 计算不倒翁的摆动：通过刚体力学公式计算重力对不倒翁角加速度的影响，然后更新角速度

2. 计算小球的运动和墙面碰撞：遍历小球，给速度的竖直分量加上因为重力产生的加速（火球不受重力影响），然后给位置变量加上速度导致的偏移；综合位置和速度判断小球是否和墙面产生碰撞，为了防止重复计算，只有小球和墙面接触且将要靠近时才判定碰撞；如果碰撞，设置小球的颜色。

3. 计算小球之间的碰撞：遍历小球，每个循环再遍历之后的小球，判断是否发生碰撞（通过位置距离是否大于半径和，同样要求接触以及互相靠近）；如果碰撞，通过动量守恒公式更新小球的速度（如果是火球，就向粒子系统发送事件）

4. 计算小球和不倒翁的碰撞：遍历不倒翁，每个循环体遍历小球，碰撞的判断方法是：先判断小球和不倒翁的距离是否小于某个值（这个判断将过滤掉大部分小球），对于足够近的小球，将不倒翁分解成 6 个球，依次判断 6 个球是否和小球接触，如果接触，就可以由对应的球和小球的位置得到碰撞的力的方向和作用点，然后通过力矩算出角加速度，更新不倒翁的角速度

## 3.4. 粒子系统 ParticleSystem

### 3.4.1.粒子状态参数和粒子系统组成部分

粒子的主要状态参数如下：
life：粒子的生命值
position，velocity：粒子的位置和速度
scale，k_scale：粒子的尺寸和尺寸常量
alpha，k_alpha：粒子的透明度和透明度常量
type：粒子的类型，不同粒子的尺寸和透明度由不同的更新函数决定

ballHit 类和 wallHit：负责处理一次和物体或者墙面撞击的事件，具体过程为每一帧会在撞击点周围按照一些规则添加数个粒子并且设置粒子属性
ParticleSystem 是整个粒子系统的实现类，主要有如下属性：
ballHit 和 wallHit 的队列：每当发生一次撞击，就在对应的撞击类的队列中压入一个实例
粒子的数组：存放粒子的数组，为了防止初始化和消除引发的性能损失，数组长度固定，有一个 head 和 tail 表示数组中活跃粒子的头和尾（因为这个例子系统中，粒子的生命是定值，所以 head 和 tail 之间的粒子一定都是活跃的）

### 3.4.2.粒子的生成和状态更新

生成：在粒子的生成过程中，先查看当前数组是否已经满了，如果满了就返回-1，否则将粒子数组的 tail 处的粒子激活（设置生命值），然后设置粒子的类别，返回粒子的索引，由调用者来给例子赋予各项参数。

更新：更新过程中，将头和尾之间的粒子遍历，减少生命值，按照不同类别的粒子更新其尺寸和透明度。

对于 wallHit 和 ballHit 类，在每一帧都会在撞击点周围生成粒子，同时减少生命值，当生命值小于 0 就把队列的头部出队。

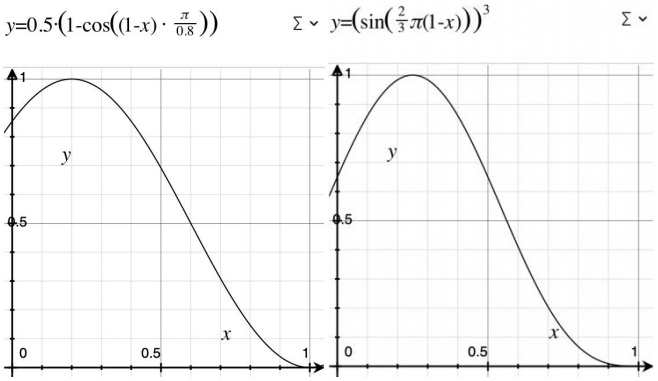
把粒子用数组存放，撞击事件用队列存放，主要的考虑是例子数量多，生成和销毁性能开销较大；撞击事件的个数远小于粒子数，用队列实现方便。

在设计上，对于物理系统，只要在撞击的那一帧调用一次粒子系统，添加撞击事件即可，而不用存储一个撞击事件的状态，每一帧都去调用一次粒子系统的函数，这样大大减少了维护的状态个数和调用次数；同样的，对于粒子系统的撞击事件类，只要每一帧生成一定数量的粒子即可，粒子的状态由整个类的 update 函数统一维护，同样减少了调用开销。

### 3.4.3.不同类型的粒子

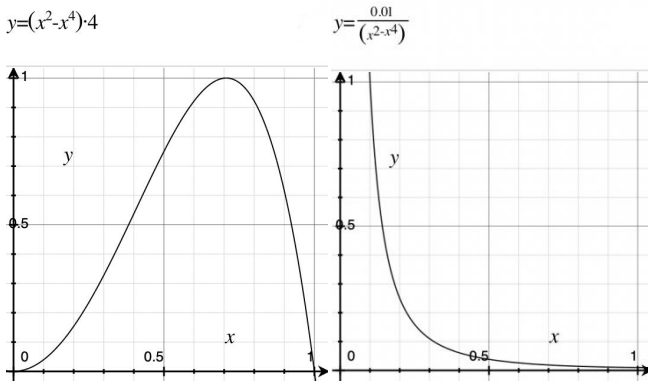
这里展示项目中用到的不同种类的粒子尺寸和透明度随时间的变化函数，横坐标是时间，从创建时刻到生命值归零，都已经归一化到[0,1]范围，尺寸和透明度也已归一化到[0,1]范围，实际还要乘 k_scale 和 k_alpha

1. 尾迹粒子：



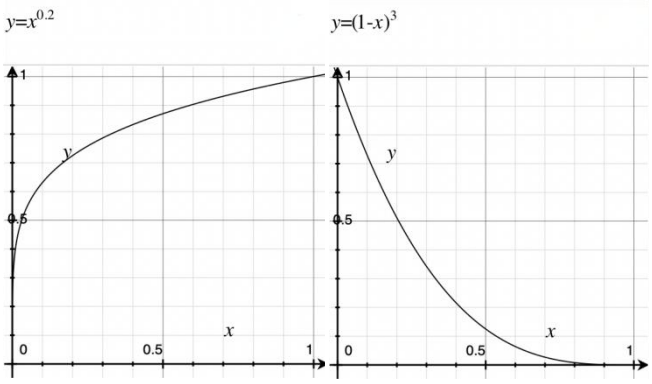

尺寸　　透明度

2. 碰撞粒子：




尺寸　　透明度

3. 爆发粒子：




尺寸　　透明度

### 3.4.4.不同类型的特效实现

火球尾迹：每间隔随机帧就在火球的位置生成粒子，同时粒子随时间渐渐变小，变透明，整体上就实现了尾迹效果；

与墙面碰撞特效：每一帧都在碰撞点周围（对应墙的平面内）固定半径随机生成数个粒子，粒子随时间先变大后变小，而且透明度与大小成反比；生成点距离碰撞点半径逐渐增大，这样就实现了粒子不断生成蔓延的效果；

与物体碰撞特效：每一帧都在撞击点周围固定半径生成粒子，粒子都向中间运动，并且保证在生命值归零时恰好运动到中间，当碰撞事件的生命值归零时，碰撞中心点出现两个不断变大变淡的粒子，模拟能量爆发的场景。