

CG 大作业报告

目录

0. 功能展示

1. 点光源光影效果

1.1 阴影：构建深度图（Depth Map）

1.2 光照：布林-冯氏光照模型（Blinn-Phong Lighting）

2. 物理仿真及碰撞检测

2.1 不倒翁摇摆物理仿真

2.2 小球和不倒翁碰撞检测：利用模型包围盒剪枝优化

2.3 小球和不倒翁碰撞物理仿真

2.4 小球间碰撞物理仿真

2.5 考虑重力场、阻力以及静止条件的小球运动

3. 鼠标响应

3.1 获取鼠标世界坐标

3.2 转换为不倒翁的动作

4. 粒子系统火焰

0. 功能展示



房间场景

- 绘制房间
- 实现点光源光照、阴影效果



初始时不倒翁状态

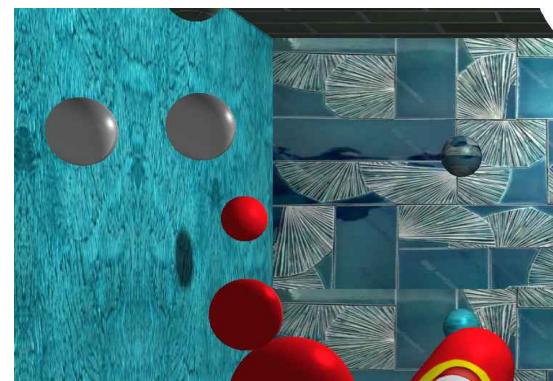


移动、倾斜后的效果对比

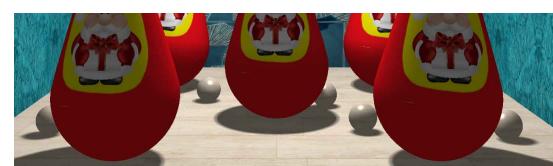
- 鼠标左键拖动不倒翁，若点击区域在重心以下位移，如果在重心以上倾斜晃动



按S键发射30个小球



小球碰到物体会改变外观



最终小球因重力场落在地面上静止不动

- 按键S控制场景中随机出现30个小球向各个方向弹射，初始都是白色小球，碰撞到物体会变外观，做物理仿真，阴影随物体运动发生变化
- 按键F控制场景中出现一个粒子系统实现的火球，火花飞散

1. 点光源光影效果

1.1 阴影：构建深度图（Depth Map）

在传统的阴影映射中，通常使用2D深度纹理来存储从光源视角看到的深度信息。然而，这种方法只能捕捉光源的一个方向，对于点光源或其他需要全方位阴影的情况就不够用了。为了解决这个问题，可以使用深度立方体贴图（Depth Cubemap），它允许我们从光源出发，向六个方向（上、下、左、右、前、后）捕捉深度信息。

- 1. 深度立方图创建：**从点光源的视角生成深度立方图。这需要将场景渲染六次，即每个立方图面一次，为立方体贴图的每个面（共6面）分配一个2D深度纹理。
- 2. 光源空间变换矩阵：**为了将场景的几何体正确地映射到深度立方体贴图中，需要为每个立方体贴图面计算一个光源空间变换矩阵。这个矩阵是由投影矩阵和视图矩阵组合而成的。

$$M_{light} = M_{projection} \times M_{view}$$

其中view矩阵可以进一步展开，

$$M_{light} = M_{projection} \times \text{lookAt}(\text{lightPos}, \text{lightPos} + \text{direction}, \text{up})$$

其中，`Projection` 是投影矩阵，由 `glm::perspective` 生成，`lookAt` 表示视图矩阵，由 `glm::lookAt` 生成，`lightPos` 表示光源位置，`direction` 和 `up` 分别是观察方向和上方向的向量。

- 3. 阴影计算：**在场景的光照过程中使用深度立方图。通过比较片段深度来确定阴影。

$$\text{shadow} = \text{depth} > \text{depthcubemap? } 1.0 : 0.0$$

- 4. 近似过滤（PCF）：**PCF（Percentage-Closer Filtering）通过对多个深度样本取平均来平滑阴影边缘，用于解决全方位阴影映射（Omnidirectional Shadow Mapping）中的分辨率依赖性问题。PCF的基本算法可以用以下数学公式表示：

$$\text{shadow} = \frac{1}{\text{samples}^3} \sum_{x=-\text{offset}}^{\text{offset}} \sum_{y=-\text{offset}}^{\text{offset}} \sum_{z=-\text{offset}}^{\text{offset}} \text{ShadowCompare}(\text{depthMap}, \text{fragPos} + \text{vec3}(x, y, z))$$

其中，`ShadowCompare` 函数用于比较当前深度和深度图中的深度值，`fragToLight` 是片段到光源的向量，`offset` 是采样偏移量，`samples` 是采样点的数量。

根据上述理论，我们可以构建一个深度着色器，包括**Vertex Shader**、**Geometry Shader**、**Fragment Shader**：

- Vertex Shader：将顶点变换到世界空间。
- Geometry Shader：负责将顶点从世界空间变换到6个不同的光源空间，并决定每个原始图元渲染到哪个立方体贴图面。

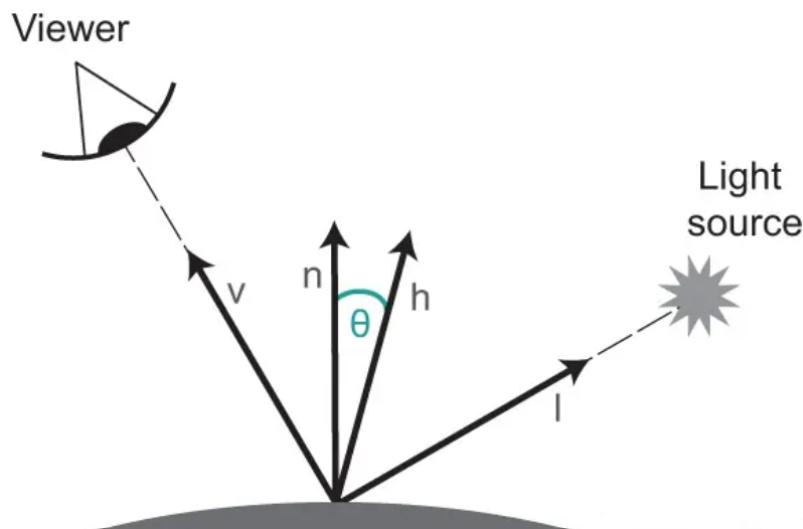
- Fragment Shader：计算每个片段与光源之间的距离，并将其映射到[0,1]范围内，作为深度值。

整个渲染过程分为两个阶段：

1. **生成深度立方体贴图**：首先，将视口设置为深度立方体贴图的尺寸，绑定帧缓冲对象，并清除深度缓冲。然后，配置着色器和矩阵，渲染场景到深度立方体贴图中。
2. **使用深度立方体贴图进行阴影映射**：接着，将视口设置回屏幕尺寸，清除颜色和深度缓冲。配置着色器和矩阵，绑定深度立方体贴图，并渲染场景，此时使用深度立方体贴图来计算阴影。

1.2 光照：布林-冯氏光照模型 (Blinn-Phong Lighting)

布林-冯氏光照模型 (Blinn-Phong Lighting) 是对冯氏光照模型的优化，有差异的地方在于参与点乘运算的元素从视图方向与光源方向替换为视图方向与光源方向的夹角平分线和平面法线向量。



布林-冯氏光照模型它将进入摄像机的光线分为三个部分，分别是环境光(Ambient)、漫反射(Diffuse)和高光反射(Specular)。

- 环境光(Ambient)

【定义】 环境光也称间接光，是光线经过周围环境表面多次反射后形成的，利用它可以描述一块区域的亮度。

【计算】 在光照模型中，通常用一个常量来表示。

$$\vec{I}_{ambient} = k_a \cdot \vec{I}_{light}$$

- 漫反射(Diffuse)

【定义】 当光线照射到一个点时，该光线会被均匀的反射到各个方向，这种反射称为漫反射。

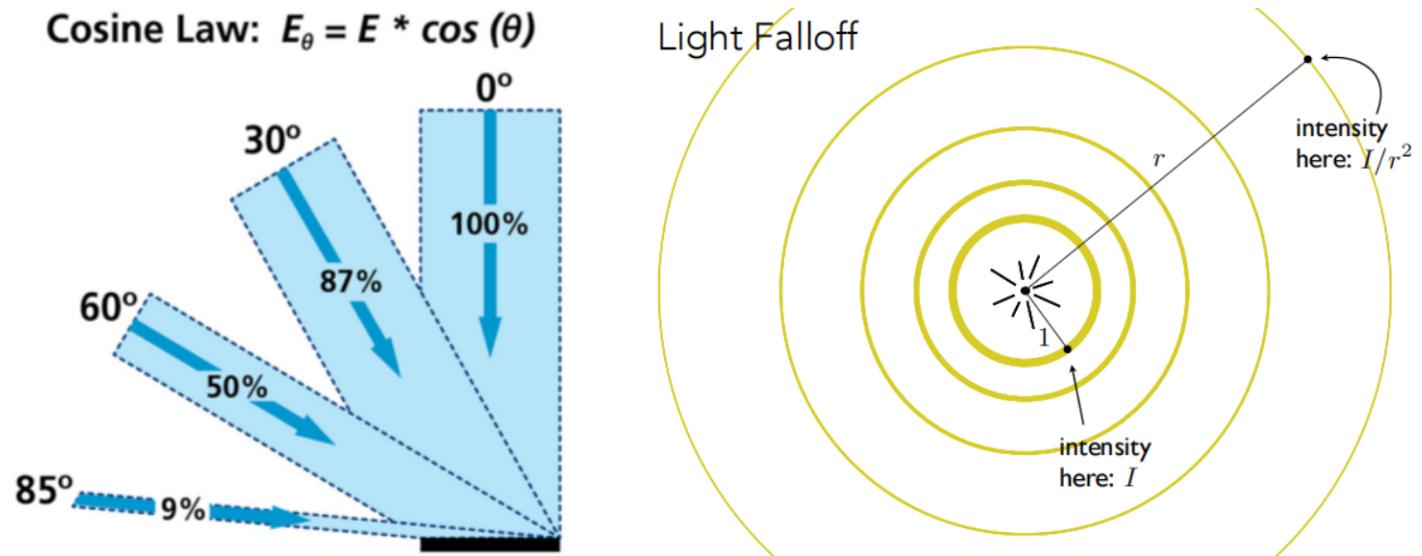
【计算】 影响漫反射光照强度的因素有

- 入射光线与法线的夹角(入射光线角度)：兰伯特余弦定律(Lambert Cosine Law)中指出，漫反射的大小取决于表面法线和光线的夹角，当夹角越大时，漫反射分量越小，当夹角接近90度时，我们认为漫反射几乎为零。

- 入射光线自身的强度：灯光的强度会随着距离的增加而衰减，这是显而易见的。假设我们有一个光源，在距离它单位1的圆上(图中最内圈)每一个点接收到光的强度是 I 。那么根据能量守恒定律，且不考虑衰减，在距离光源 r 位置的圆上每个点接收到光的强度就是 I/r^2 。

$$\vec{I}_{diffuse} = k_d \left(\vec{I}_{light} / r^2 \right) \max \left(0, \vec{n} \cdot \vec{I}_{light} \right)$$

其中 $I_{diffuse}$ 为漫反射光照， k_d 为物体表面的反射系数 (reflection coefficient)， I_{light}/r^2 是当前点接受的光照强度， $\max(0, I_{light} \cdot n)$ 是当前点接受到的能量。

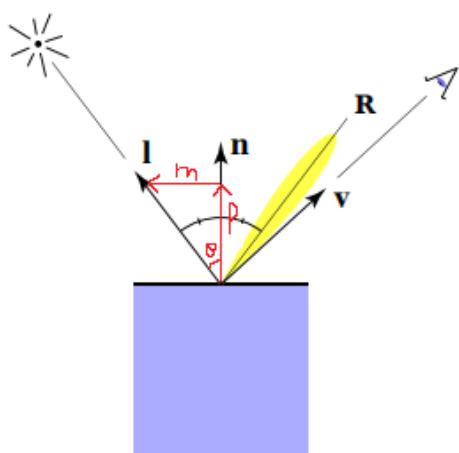


- 镜面反射 (Specular Lighting)

【定义】 反射光导致的局部强光。

【计算】 下图中一根入射光线 l ，照射在光滑的平面上，会沿着 R 方向反射，由于平面并非完全光滑，所以反射光的方向并非只有 R 一个点，而是 R 周边的一小块区域，只要眼睛(摄像机)在 R 附近都可以看得到，越靠近 R 反射光照强度越大。

设 l 与 n 之间的夹角为 θ ，将 m 与 l 的顶部相连，并且 m 垂直 n ，向量 p 的绝对值 $|p|$ 是 l 在 n 上的投影



$$\begin{cases} R = 2m - l \\ m = l - p \\ p = |p| \cdot \frac{n}{|n|} \\ |p| = |l| \cos \theta \\ \cos \theta = \frac{l \cdot n}{|l| \cdot |n|} \end{cases}$$

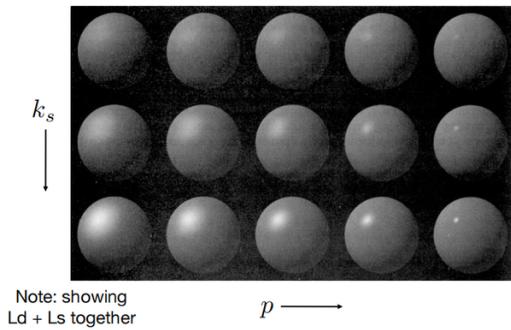
则有

$$p = \frac{l \cdot n}{|n|^2} \cdot n, \quad R = l - 2(l \cdot n) \cdot n$$

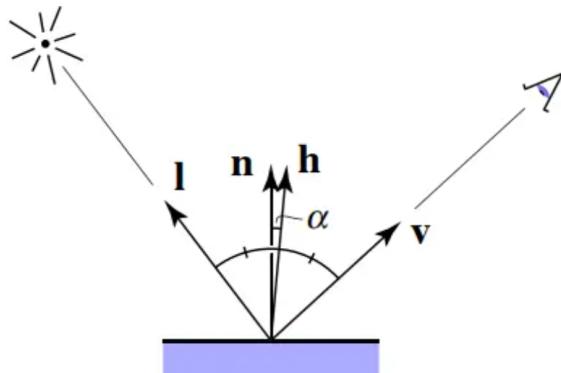
完整的镜面反射公式可以表示为：

$$L_s = k_s \left(I/r^2 \right) \max (0, \cos \alpha)^p = k_s \left(I/r^2 \right) \max (0, v \cdot R)^p$$

其中 k_s 是高光反射系数， I/r^2 是当前点接受的光照强度。 p 次方是为了控制高光反射面积的大小。



和能够看到高光的范围。当 p 的值越大，反射面积(能看到高光的范围)越小。



为了简化计算，通过对向量 l 和向量 v 取平均然后归一化得到一个新的向量 h ，其中 h 被称为半程向量(*bisector*)，使用 h 与法线 n 点乘来计算高光，这样就可以避免计算 R 了。

$$h = \frac{l + v}{|l| + |v|}$$

最终表达为

$$L_s = k_s \left(\frac{I}{r^2}\right) \max(0, \cos \alpha)^p = k_s \left(\frac{I}{r^2}\right) \max(0, n \cdot h)^p$$

2. 物理仿真及碰撞检测

2.1 不倒翁摇摆物理仿真

不倒翁之所以能够左右摇摆后又自动站立，是因为它的质心非常低，通常位于其底部的凸起部分之下。当不倒翁被推动偏离其平衡位置时，由于质心下移，产生了一个恢复力矩，这个恢复力矩会使不倒翁试图恢复到其稳定平衡位置。这个过程中，由于存在摩擦力和空气阻力，不倒翁的摆动会慢慢减小，最终停止并保持直立。

由于这个模型的下半部分可以视作一个球体，我们可以将其视为一个复杂的物理摆。球体的质心位置、质量分布以及其与地面的接触面积都会影响不倒翁的摆动特性。

不倒翁的恢复力矩主要由重力产生，可以使用以下方程来描述：

$$\tau = I\alpha$$

其中 τ 是力矩， I 是不倒翁对于摆动轴线的转动惯量， α 是角加速度。

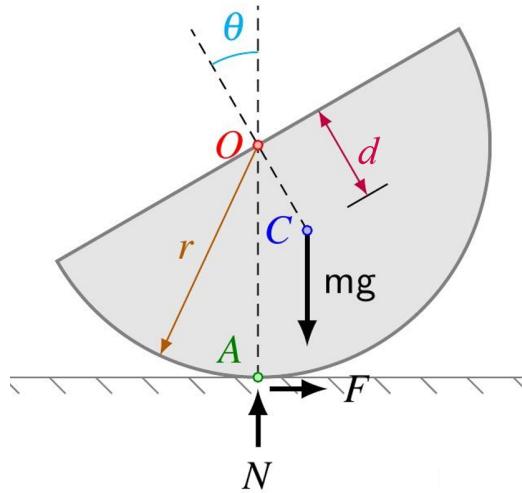
恢复力矩与不倒翁偏离竖直方向的角度成比例，通常可以表达为：

$$\tau = -k\theta$$

其中 k 是恢复力矩系数， θ 是偏离角。

1. 模拟阻尼摆动

当给不倒翁一个初始的角速度之后，我们需要对其进行摆动模拟。考虑到阻尼，不倒翁的运动方程可以表示为：



$$I \frac{d^2\theta}{dt^2} + b \frac{d\theta}{dt} + mgd \sin \theta = 0$$

其中：

- b 是阻尼系数，与角速度的乘积给出了阻尼力矩，
- $\frac{d\theta}{dt}$ 是角速度，
- $\frac{d^2\theta}{dt^2}$ 是角加速度。
- m 是不倒翁的总质量。
- g 是重力加速度（约9.81 m/s²）。
- d 是质量中心到支点的距离。

2. 简化运动方程

$$I \frac{d^2\theta}{dt^2} + b \frac{d\theta}{dt} + k\theta = 0$$

k 是恢复力矩的系数，对于小角度近似 $c \sin(\theta) \approx k\theta$ 。

3. 程序模拟

为了在程序中模拟这种物理行为，我们需要设置一个初始角速度和初始角度，模型的旋转和平移是基于时间变化 (`deltaTime`) 来更新的。

角加速度 $\alpha = \frac{-k \cdot \theta - b \cdot \omega}{I}$ ，根据角加速度更新角速度和角度：

$$\begin{cases} \omega_{new} = \omega + \alpha \Delta t \\ \theta_{new} = \theta + \omega \Delta t \end{cases}$$

2.2 小球和不倒翁碰撞检测：利用模型包围盒剪枝优化

小球与模型碰撞检测的实现方法主要分为两个部分：剪枝优化和碰撞判断。这里对于利用模型包围盒进行剪枝优化进行详细展开。

剪枝是碰撞检测中用于提高效率的技术，它通过减少需要进行详细碰撞检测的对象数量来降低计算负担。在此案例中，使用了轴对齐包围盒（AABB）和包围球来进行剪枝。

AABB包围盒： AABB包围盒的每个面都与坐标轴平行。首先检测小球的AABB是否与模型的AABB重叠，如果不重叠，可以立即排除碰撞的可能性，从而减少后续的计算。其中，在每次对模型坐标变换的时候更新该模型的包围盒。

a. 定义模型变换矩阵：

- 平移矩阵 $\mathbf{M}_{\text{translate}}$
- 缩放矩阵 $\mathbf{M}_{\text{scale}}$
- 旋转矩阵 $\mathbf{M}_{\text{rotate}}$

首先应用平移变换：

$$\mathbf{M}_{\text{translate}} = \begin{bmatrix} 1 & 0 & 0 & offset_x \\ 0 & 1 & 0 & offset_y \\ 0 & 0 & 1 & offset_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

接着应用缩放变换：

$$\mathbf{M}_{\text{scale}} = \begin{bmatrix} scale_x & 0 & 0 & 0 \\ 0 & scale_y & 0 & 0 \\ 0 & 0 & scale_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

最后应用围绕指定轴的旋转变换：

$\mathbf{M}_{\text{rotate}}(\theta) = \text{旋转矩阵，依赖于}\theta\text{和旋转轴}$

点的变换：点 $\mathbf{P} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$ 被变换为 $\mathbf{P}_{\text{transformed}} = \mathbf{M}_{\text{translate}} \times \mathbf{M}_{\text{scale}} \times \mathbf{M}_{\text{rotate}} \times \mathbf{P}$

b. 更新模型包围盒

- 设原始包围盒的最小点为 \mathbf{P}_{\min} 和最大点为 \mathbf{P}_{\max} 。设变换函数为 $\text{transformPoint}(\mathbf{P})$ ，其中 \mathbf{P} 是待变换的点。变换后的点为 $\mathbf{P}'_{\min} = \text{transformPoint}(\mathbf{P}_{\min})$ 和 $\mathbf{P}'_{\max} = \text{transformPoint}(\mathbf{P}_{\max})$ 。
- 对于变换后的包围盒，我们需要找出在每个坐标轴上的最小和最大值。
 - 最小点 \mathbf{P}'_{\min} 和最大点 \mathbf{P}'_{\max} 的坐标分别为：

$$\mathbf{P}'_{\min} = \begin{bmatrix} \min(P'_{\min_x}, P'_{\max_x}) \\ \min(P'_{\min_y}, P'_{\max_y}) \\ \min(P'_{\min_z}, P'_{\max_z}) \end{bmatrix}$$

$$\mathbf{P}'_{\max} = \begin{bmatrix} \max(P'_{\min_x}, P'_{\max_x}) \\ \max(P'_{\min_y}, P'_{\max_y}) \\ \max(P'_{\min_z}, P'_{\max_z}) \end{bmatrix}$$

其中 $P'_{min_x}, P'_{min_y}, P'_{min_z}$ 是变换后的最小点的坐标, $P'_{max_x}, P'_{max_y}, P'_{max_z}$ 是变换后的最大点的坐标。

首先使用AABB包围盒和包围球进行剪枝, 排除那些明显不可能发生碰撞的部分, 然后对剩余的部分进行详细的碰撞检测。

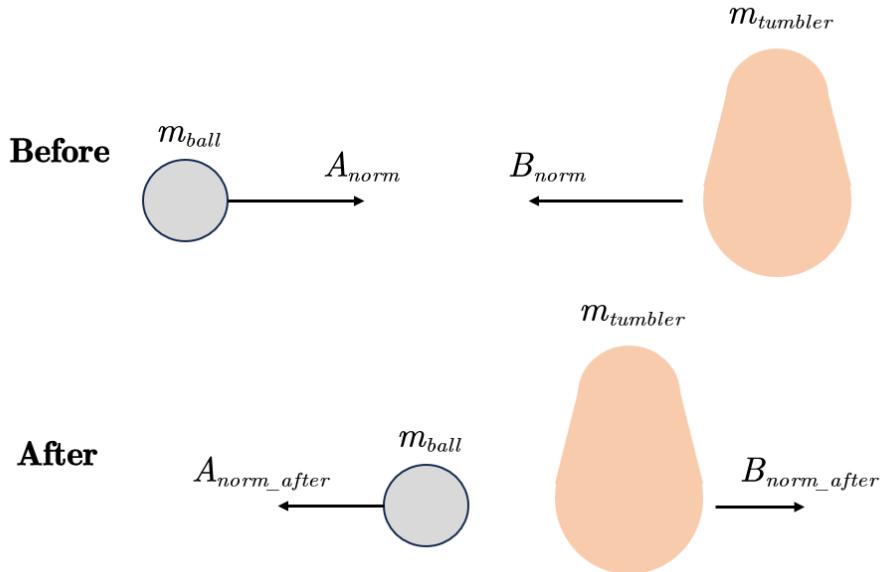
2.3 小球和不倒翁碰撞物理仿真

小球和不倒翁碰撞的物理仿真分为如下三步:

1. 垂直于碰撞表面的部分视作非完全弹性碰撞

假设我们有两个物体 (在本例中是球和不倒翁) 发生碰撞。

A_{norm} 和 B_{norm} 是分别沿法线方向的速度分量, e 是恢复系数, $m_{tumbler}$ 和 m_{ball} 是不倒翁和球的质量。



动量守恒定律表明, 碰撞前后两物体动量的总和保持不变:

$$m_{ball} \cdot A_{norm} + m_{tumbler} \cdot B_{norm} = m_{ball} \cdot A_{norm_after} + m_{tumbler} \cdot B_{norm_after}$$

恢复系数 e 表示碰撞前后相对速度的比例:

$$e = \frac{|B_{norm} - A_{norm}|}{|A_{norm_before} - B_{norm_before}|}$$

得到

$$A_{norm_after} = A_{norm} - \frac{(1+e) \times m_{tumbler} \times (A_{norm} - B_{norm})}{m_{tumbler} + m_{ball}}$$

$$B_{norm_after} = B_{norm} - \frac{(1+e) \times m_{ball} \times (B_{norm} - A_{norm})}{m_{tumbler} + m_{ball}}$$

2. 平行于碰撞表面的部分做摩擦处理

初始平行于平面方向速度分别为 A_{tang} 和 B_{tang} ，并且碰撞是非弹性的。

动量守恒：

$$m_{\text{ball}} \cdot A_{\text{tang}} + m_{\text{tumbler}} \cdot B_{\text{tang}} = m_{\text{ball}} \cdot A_{\text{tang_after}} + m_{\text{tumbler}} \cdot B_{\text{tang_after}}$$

假设碰撞是完全非弹性的，即两物体在接触点处的相对速度为零。因此，

$$A_{\text{tang_after}} = B_{\text{tang_after}}$$

可以得出：

$$A_{\text{tang_after}} = B_{\text{tang_after}} = \frac{m_A \cdot A_{\text{tang}} + m_B \cdot B_{\text{tang}}}{m_A + m_B}$$

3. 结合速度分量：

$$A_{\text{after}} = A_{\text{norm_after}} + A_{\text{tang_after}}$$

$$B_{\text{after}} = B_{\text{norm_after}} + B_{\text{tang_after}}$$

2.4 小球间碰撞物理仿真

1. 垂直于小球碰撞平面的部分视作非完全弹性碰撞

设两个小球的初始速度向量为 \vec{v}_1 和 \vec{v}_2 ，碰撞点的法向量为 \vec{n} ，恢复系数为 e_{ball} 。

首先计算速度在法向量方向的分量。设这些分量分别为 \vec{v}_{1n} 和 \vec{v}_{2n} ：

$$\vec{v}_{1n} = (\vec{v}_1 \cdot \vec{n}) \vec{n}$$

$$\vec{v}_{2n} = (\vec{v}_2 \cdot \vec{n}) \vec{n}$$

对于碰撞垂直于法向量的部分，我们使用完全弹性碰撞模型来计算碰撞后的速度，因为它们在法向量方向的相对速度会反转并乘以恢复系数 e_{ball} 。碰撞后的速度分量为 \vec{v}'_{1n} 和 \vec{v}'_{2n} ：

$$\vec{v}'_{1n} = \vec{v}_{2n} + (1 + e_{\text{ball}}) \cdot \frac{\vec{v}_{1n} - \vec{v}_{2n}}{2}$$

$$\vec{v}'_{2n} = \vec{v}_{1n} + (1 + e_{\text{ball}}) \cdot \frac{\vec{v}_{2n} - \vec{v}_{1n}}{2}$$

2. 因小球相对光滑，平行于碰撞平面的部分视作不变

速度在法向量平面上的分量不会因为碰撞而改变，因为没有力在这个方向上作用于球。因此，我们可以简单地保持这些分量不变：

$$\vec{v}_{1t} = \vec{v}_1 - \vec{v}_{1n}$$

$$\vec{v}_{2t} = \vec{v}_2 - \vec{v}_{2n}$$

3. 碰撞后的总速度

最终，每个球碰撞后的总速度是其在法向量方向和平行于法向量方向速度分量的矢量和：

$$\vec{v}'_1 = \vec{v}'_{1n} + \vec{v}_{1t}$$

$$\vec{v}'_2 = \vec{v}'_{2n} + \vec{v}_{2t}$$

2.5 考虑重力场、阻力、停止条件的小球运动

1. 重力影响

在重力场中，小球受到一个向下的力，即重力。重力可以表达为：

$$\vec{F}_{\text{gravity}} = m \cdot \vec{g}$$

其中 m 是小球的质量（在代码中似乎被假定为单位质量，因此重力等于重力加速度 \vec{g} ）。

在时间间隔 Δt 内，重力对速度的影响可以通过以下方式计算：

$$\vec{v} = \vec{v}_0 + \vec{g} \cdot \Delta t$$

其中 \vec{v}_0 是初始速度， \vec{v} 是更新后的速度。

2. 空气阻力

小球在空气中运动时，会受到空气阻力的影响。阻力通常与速度成正比，可以表示为：

$$\vec{F}_{\text{drag}} = -c_d \cdot |\vec{v}| \cdot \vec{v}$$

其中 c_d 是空气阻力系数， $|\vec{v}|$ 是速度的大小， $\$-\$$ 表示阻力方向与速度方向相反。

考虑到小球的质量为单位质量，阻力导致的加速度为：

$$\vec{a}_{\text{drag}} = \vec{F}_{\text{drag}}$$

因此，在时间间隔 Δt 内，速度的更新可以表示为：

$$\vec{v} = \vec{v} + \vec{a}_{\text{drag}} \cdot \Delta t$$

3. 位置更新

小球的位置根据速度进行更新。位置的更新可以用简单的动力学公式表示：

$$\vec{p} = \vec{p}_0 + \vec{v} \cdot \Delta t$$

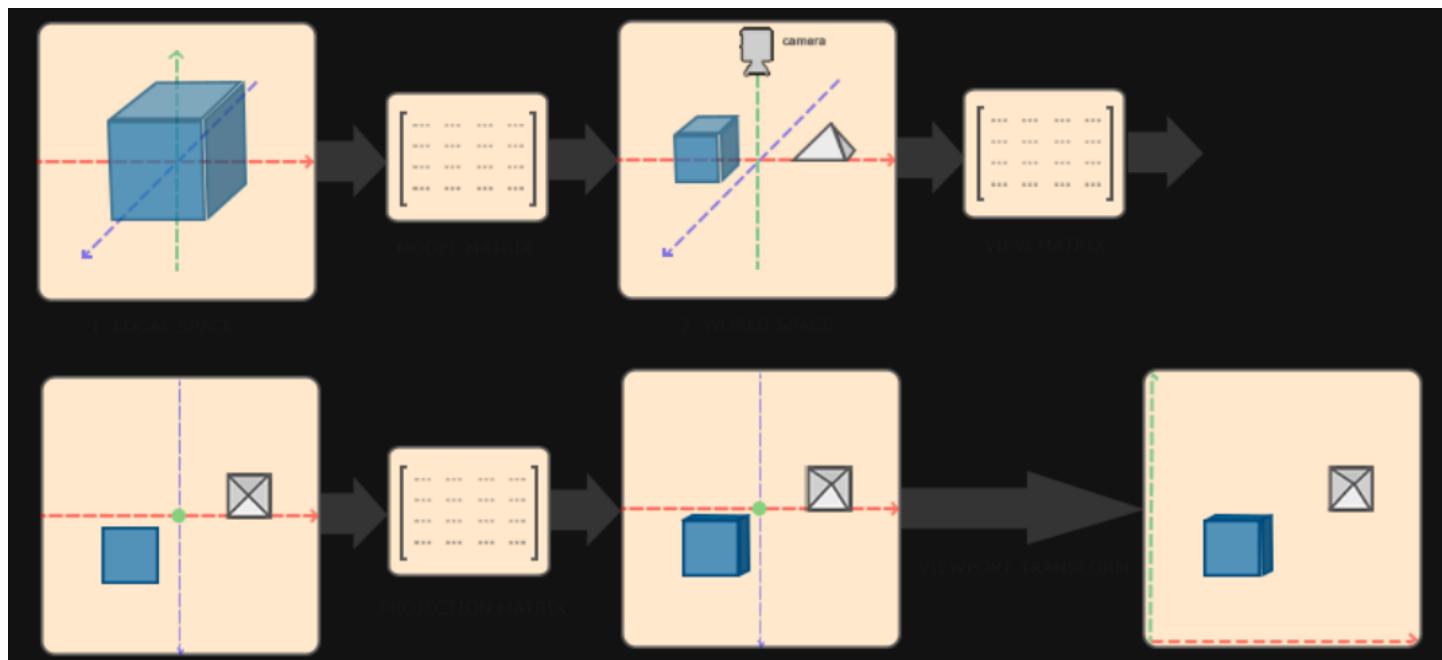
其中 \vec{p}_0 是初始位置， \vec{p} 是更新后的位置。

4. 停止条件

当小球接近地面且速度降至某一阈值以下时，模拟停止。这模拟了实际情况中由于接触地面和其他因素导致的运动停止。

3. 鼠标响应

3.1 获取鼠标的世界坐标



1. 视图和投影矩阵:

- 视图矩阵 View 表示摄像机在世界空间中的位置和方向。
- 投影矩阵 Projection 表示3D场景投影到2D视图平面的方式。

2. 鼠标屏幕坐标:

- 鼠标的屏幕坐标为 (x, y) 。
- 屏幕坐标需要调整以符合OpenGL的坐标系统，特别是Y坐标，因为OpenGL的Y坐标是从底部开始的。转换后的坐标为 $(x, \text{ScreenHeight} - y)$ 。

3. 读取深度缓冲区:

- 在鼠标位置 $(x, \text{ScreenHeight} - y)$ 处读取深度值 z 。

4. 从屏幕空间到世界空间的转换:

- 使用 `gluUnProject` 函数实现转换，其背后的数学原理是逆变换。首先，将屏幕坐标和深度值转换为归一化设备坐标 (NDC)，这个坐标系统中的所有值都在-1到1之间。
- 归一化设备坐标公式为 $\text{NDC} = (2x/\text{ScreenWidth} - 1, 2y/\text{ScreenHeight} - 1, 2z - 1)$
- 然后，使用逆投影矩阵 Projection^{-1} 和逆视图矩阵 View^{-1} 将NDC坐标转换回世界坐标 World。

5. 最终的世界坐标:

世界坐标 World 可以通过以下变换得到：

$$\text{World} = \text{View}^{-1} \cdot \text{Projection}^{-1} \cdot \text{NDC}$$

3.2 转换为不倒翁的动作

平移和旋转的决定

1. 计算位移向量:

- 位移向量 $\vec{d} = \text{newMousePoint} - \text{lastMousePoint}$ 表示鼠标在两帧之间的移动量。

2. 边界条件检查:

- 检查不倒翁移动后是否会超出房间的边界。如果移动后的位置超出边界，函数返回，不执行任何操作。

3. 应用模型变换:

- 应用平移（由 `offset` 控制）、缩放（由 `scale` 控制）和旋转（由 `theta` 和 `direction` 控制）来更新不倒翁的模型矩阵 `model`。

4. 计算变换后的质心:

- 计算不倒翁经过变换后的质心位置 $\text{transformedMassCenter}$

鼠标响应的逻辑

1. 平移逻辑:

- 如果 `newMousePoint.y` 小于 `transformedMassCenter.y`，表明鼠标在质心下方，此时执行平移。
- 平移量由位移向量 \vec{d} 控制，但只考虑 x 和 z 轴的分量（保持不倒翁垂直于地面）。

2. 旋转逻辑:

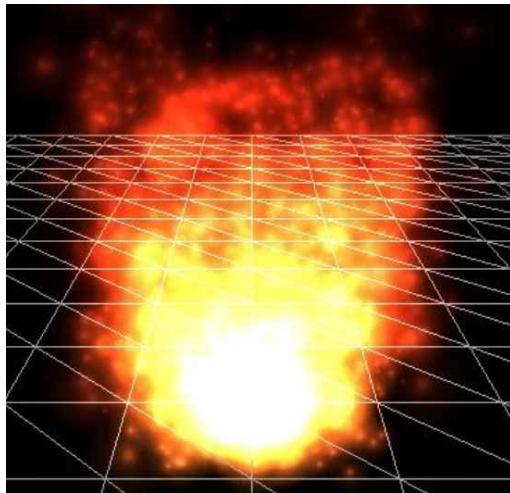
- 如果 `newMousePoint.y` 大于或等于 `transformedMassCenter.y`，执行旋转。
- 旋转方向由鼠标位移向量和垂直于地面的向量的叉积决定： $\vec{direction} = \text{normalize}(\text{cross}(\vec{k}, \vec{d}))$ ，其中 $\vec{k} = (0, 1, 0)$ 。
- 旋转的角度 $\Delta\theta$ 与鼠标位移量和从质心到新鼠标点的“臂长” \vec{arm} 有关：

$$\vec{arm} = \text{newMousePoint} - \text{transformedMassCenter}.$$

$$\Delta\theta = \frac{\|\vec{d}\|}{\|\vec{arm}\|}$$

4. 粒子系统火焰

参考 <https://learnopengl.com/In-Practice/2D-Game/Particles>



opengl教程demo

粒子系统的组成

- 1. 粒子属性：** 每个粒子拥有自己的属性，包括位置、速度、颜色、生命周期。
- 2. 纹理加载：** 粒子系统使用纹理来给粒子赋予特定的视觉效果。
- 3. 粒子的创建和更新：** 系统根据需要生成新粒子，并更新每个粒子的状态。
- 4. 粒子渲染：** 系统渲染存活的粒子到屏幕上。

粒子生成和更新

粒子生成和更新是粒子系统中的核心功能。以下是详细步骤：

- 1. 生成新粒子：** 对于每个新粒子，随机生成其属性（如位置、速度、颜色）。
 - Position
=EmitterState.Position+random_positionPosition
=EmitterState.Position+random_position
 - Velocity
=EmitterState.Velocity+random_velocityVelocity
=EmitterState.Velocity+random_velocity
 - Color
=random_colorColor|
=random_color
 - Life=initial_lifeLife=initial_life

其中，`EmitterState.Position` 和 `EmitterState.Velocity` 是发射器当前的位置和速度，`random_position`、`random_velocity` 和 `random_color` 分别是随机生成的位置、速度和颜色的偏移量。

1. **更新粒子状态**: 每个更新周期, 粒子的位置和颜色根据其速度和生命期进行更新。

- Particle.Position
 - $=\text{Particle.Velocity} \times dt$
 - $=\text{Particle.Velocity} \times dt$
- Particle.Color.alpha
 - $=\text{decay_rate} \times dt$
 - $=\text{decay_rate} \times dt$

其中, `dt` 是从上一次更新以来的时间增量, `decay_rate` 是颜色的衰减率。

渲染

1. **混合模式**: 使用加性混合 (`glBlendFunc(GL_SRC_ALPHA, GL_ONE)`) 为粒子创建“发光”效果。
2. **渲染粒子**: 对于每个活着的粒子, 使用其属性(位置、颜色)在屏幕上渲染。
3. **重置混合模式**: 渲染完成后重置混合模式, 以便系统可以正确地渲染其他非粒子内容。