计算机图形学作业 2 报告

徐琢雄 521021910560

1. 操作方法

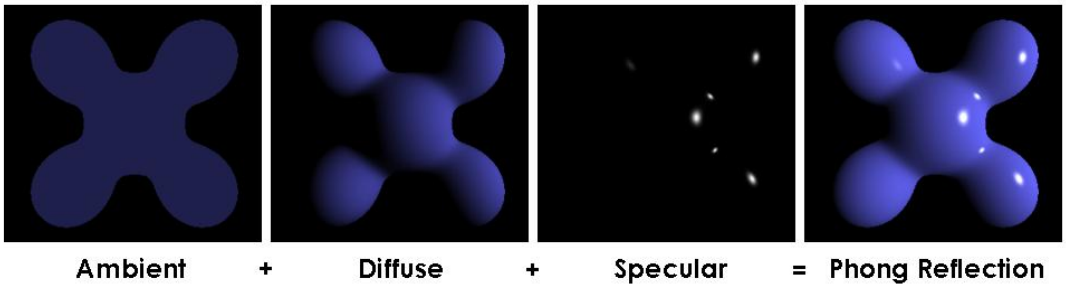
1.程序运行后，用 WSAD 键可以前后左右移动，移动鼠标可以转动视角。
2.程序运行后，即出现一个小球向模型运动，撞到模型后，终端输出撞击点坐标，模型出现裂缝，但不会立即破碎，此时可以移动查看裂缝细节。
3.模型被撞击后，按下空格键，模型破碎，碎片向四周飞溅，再次按下空格键可以暂停，此时可以移动查看断面细节。
4.模型被撞击前就按下空格键，模型被撞击后会立即破碎，省去出现裂缝阶段。
5.源代码在 src/homework/homework2 文件夹中，用 cmakelist 即可编译。

2. 实现方案

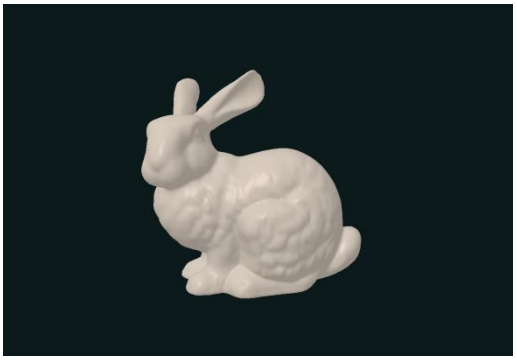
这个项目中，我依次实现了模型的导入和渲染、内表面生成、碰撞检测、破碎效果，其中最大的工作量在于破碎部分，考虑了物理仿真和物品破碎的实例，达到了比较接近真实的效果。

2.1.1. 模型导入&渲染

我使用 assimp 库来导入 obj 文件，导入的文件存储在 aiScene 结构中，其中存储了所有顶点，法向量和面元。我实现了一个面向对象的 model 类，把 aiScene 中的所有数据导入了 model 类，用于更方便地管理各个顶点和面元。
model 类中最主要的结构是 meshes，这是一个以类 mesh 为元素的 vector，分别存储了外表面，内表面，条带，小球，每个 mesh 都对应着一个 VAO，当数据需要改变时，先改变 mesh 中存储的数据，改变完成后调用 mesh 的 SendPosition 方法，这个函数会把数据通过 GlBufferData 发送到对应的 VBO，因为 VBO 数量很少，而且每次只在所有 mesh 改动结束后一起发送，这样就减少了 CPU 和 GPU 的通信，提高了性能。
渲染过程，我采用了 Phong 光照模型，物体的最终颜色来自于三个分量：环境光，漫反射，镜面反射的叠加：



我编写了自定义的 vertex shader 和 fragment shader，通过 uniform 传入模型矩阵，视口矩阵和投影矩阵，还传入了光源位置、颜色等信息。在 fragment shader 中，分别计算环境光，漫反射光和镜面反射光：环境光为常量；漫反射光根据法向量和光照角度确定；镜面反射根据光源位置，相机位置，法向量共同确定。随后，这三种光的分量按照各自系数加到一起，构成了最终的颜色。
最终渲染的模型如下图。

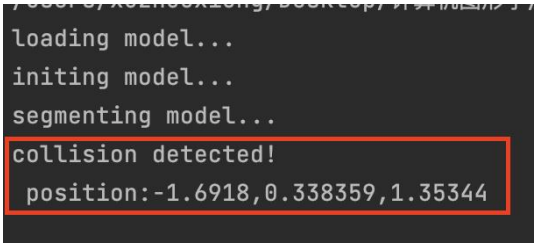


2.1.2. 内表面生成&导出

导入模型后，内表面的生成分为两步，一是将所有顶点按照法向量负方向内缩，二是将法向量反向，将模型的所有顶点这样操作后，再使用 assimp 导出为内表面模型即可。

2.1.3. 小球运动&碰撞检测

小球运动的实现，确定一个方向向量，然后在每一帧让 delta time 乘方向向量，然后加到小球的所有坐标上，再把数据发到 VBO，就实现了小球的运动。
对于小球的碰撞检测，由于需要碰撞检测的物体少，计算量不大，我采用的方法是在载入模型时就计算所有三角面元的中心位置，存放在 model.face_centers 容器中，每一帧都遍历这个容器，计算它到小球中心的位置的距离，若距离小于半径，就代表着碰撞，之所以可以用中心位置来检测碰撞是因为模型三角面元数量巨大，三角面元的中心分布很密集。检测到碰撞时，这个三角面元中心位置就是近似碰撞位置。在终端输出其坐标。

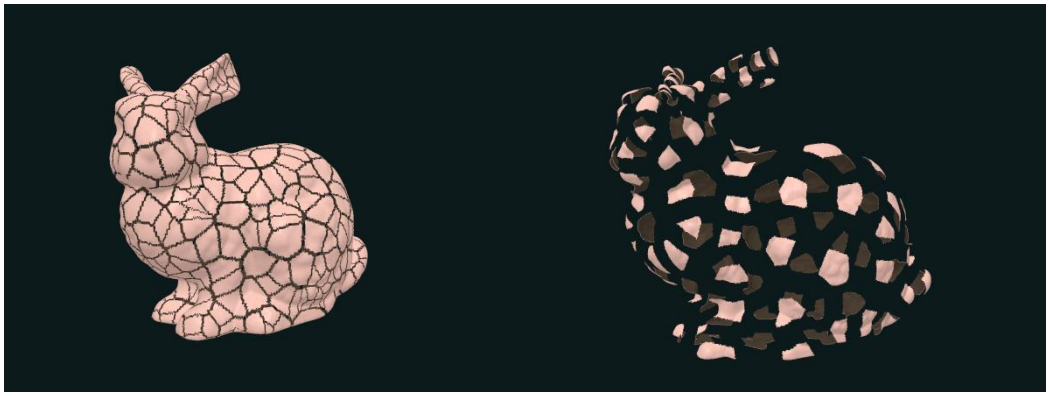


2.1.4. 破碎

实现破碎效果，我的实现路线是预先分割模型为多个相连的碎片，生成断面的条带，再根据撞击方向和速度以及重力加速度，更新碎片的位置，达到碎片飞行的效果。

2.1.4.1. 模型分割

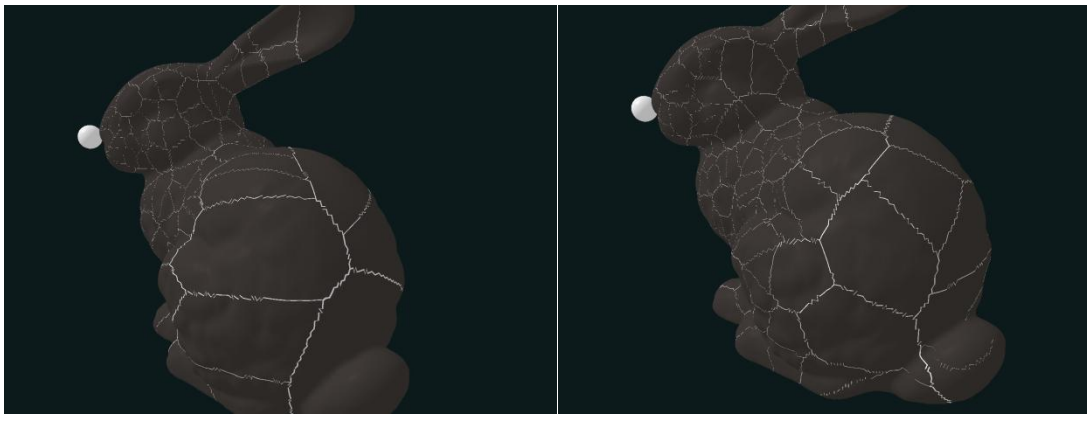
我使用 k-means 聚类实现模型的分割。用于聚类的数据点是每个面元的中心点，初始化的点是随机抽取的中心点，每次迭代过程中，中心点朝着总距离最小的方向移动。使用 kmeans 聚类后，模型被划分为了大小相仿的碎片：



然而，这样的碎片对比真实的撞击还有所欠缺，实际的撞击产生的碎片，距离撞击点越近，碎片越小：

为了实现近似真实的撞击效果，我不再采用随机初始化的中心点，而是按照高斯分布的概率采样三角面元中心点，距离撞击中心越近的点被选中的概率越高。具体是实现上，我先进行随机抽样，对于抽样得到的点，先按照高斯分布生成随机的距离，然后与抽样得到的点到碰撞中心的距离比较，如果小于这个随机距离，就选入聚类中心点，这样能保证最终离碰撞点越近，碎片分布越密集。

测试结果如下所示，为清晰显示裂缝，将模型调成深色：



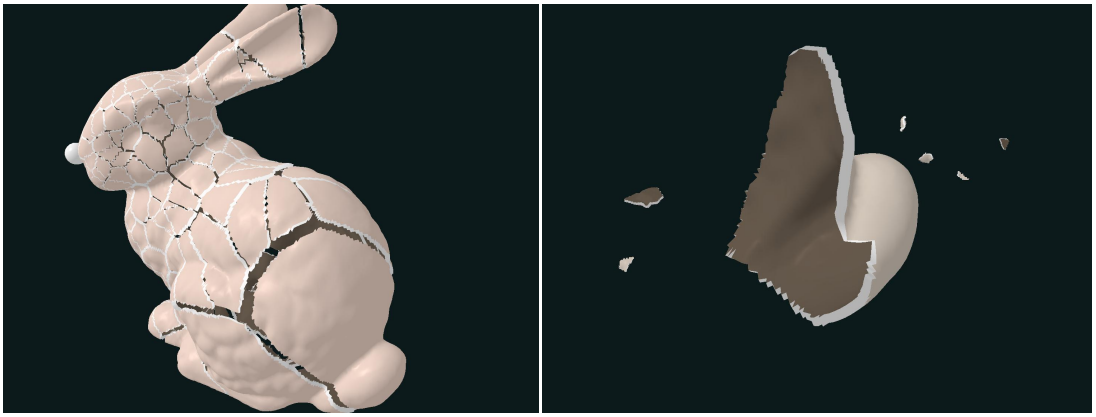
从结果可以看到，碰撞点附近碎片较小，而碰撞点较远的位置碎片较大，更接近于真实环境。

2.1.4.2. 碎片边缘检测&条带生成

k-means 聚类得到了碎片，我将其保存在 part_face_idx 中，其中每个元素代表了一个碎片，每个元素都是一个 vector，里面存储着组成碎片的面元的 index。
生成边缘条带，我的算法是：

1. 遍历所有碎片
2. 对于每一块碎片，建立一个边集合
3. 对于每一块碎片，遍历其所有三角面元，查找三角面元的边是否在边集中，如果存在，就在边集中删掉这条边，如果不存在，就加入这条边，最终边集中剩下的边就是碎片的边界。

这个算法基于的思想是：如果遍历所有三角面元的所有边，那么碎片内部的边会被遍历两次，而碎片边缘的边只会被遍历一次，那么对于第二次访问到的边将其删除，最终剩下的就是边缘。
对于剩下的边缘，我按照对应的内外表面的顶点顺序，存入顶点数组，统一放在一个 mesh 中，同时还维护了一个 vector 存储每个碎片的边缘有多少面片。最终结果如下所示：



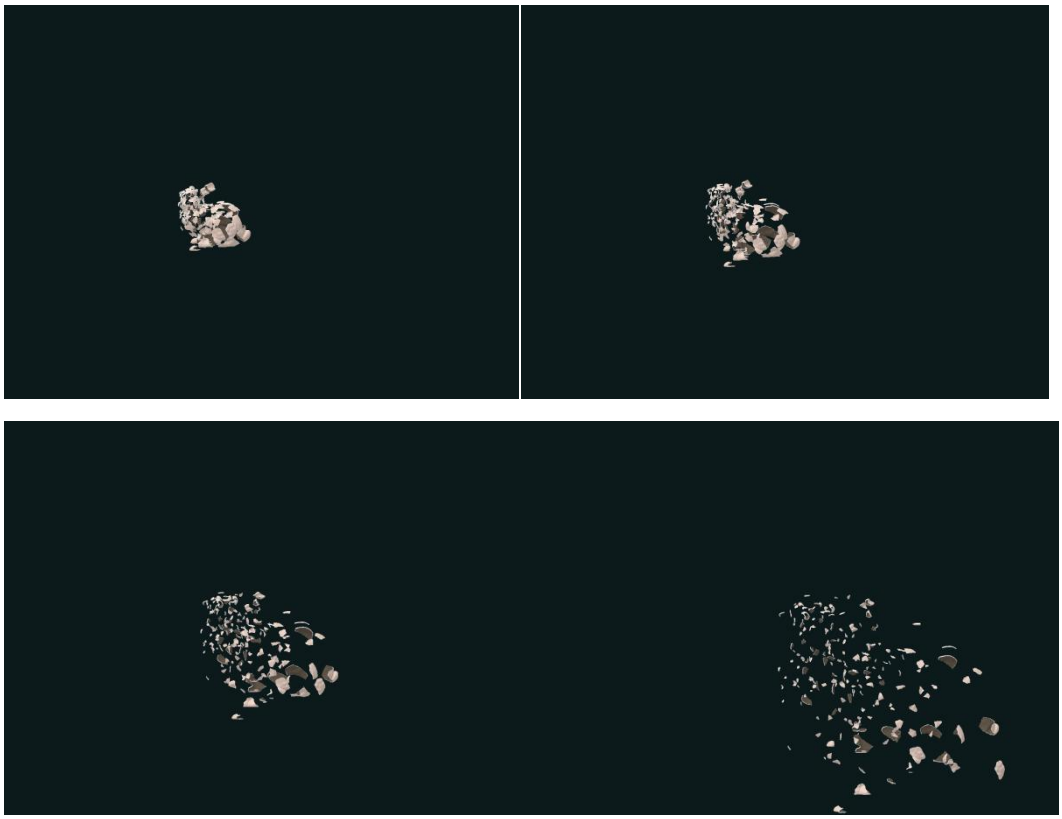
可以看到，碎片的边界都能够正确显示。

2.1.4.3. 碎片飞行

碎片飞行的实现，我采用了齐次坐标和坐标变换矩阵，坐标的变换由多个分量组成：

1. 碎片从模型中心点向四面八方飞溅，这个方向从整个模型中心点指向每个碎片中心点
2. 为了模拟碎片之间相互碰撞，飞溅方向还加上了一个随机向量，测试中，没有这个随机向量，破碎太有规律了，就会显得十分呆板。
3. 小球给了碎片动能，碎片根据动量交换，具有一个小球飞行方向的速度
4. 重力让碎片下坠，碎片有重力方向的速度

将这四个方向的速度变化 delta_time 就是各个碎片的位置变化，然后依次连乘位置变换矩阵，再乘以对应碎片的所有顶点坐标，就得到了更新后的坐标。大概效果如下所示：



碎片表现出了比较真实的溅射和飞行效果。