# 平时作业2报告

1. **概述**

本次作业目标为完成一个obj模型加载器，并实现一些基于模型的模拟操作功能。

1. **工作流程**

本次作业历时14日，其中1~13日用于学习相关OpenGL知识，编写代码，14日用于完成报告。

1. **技术方案**

**①网格类设计**

网格类储存的成员变量包括

* 顶点向量数组（储存顶点坐标、顶点法线向量、顶点颜色、以及材质坐标）
* 顶点索引数组
* 材质数组
* 顶点数组对象（在类实例化的同时会初始化），方便网格的绘制。

为完成作业要求，给网格类添加了几个功能函数：

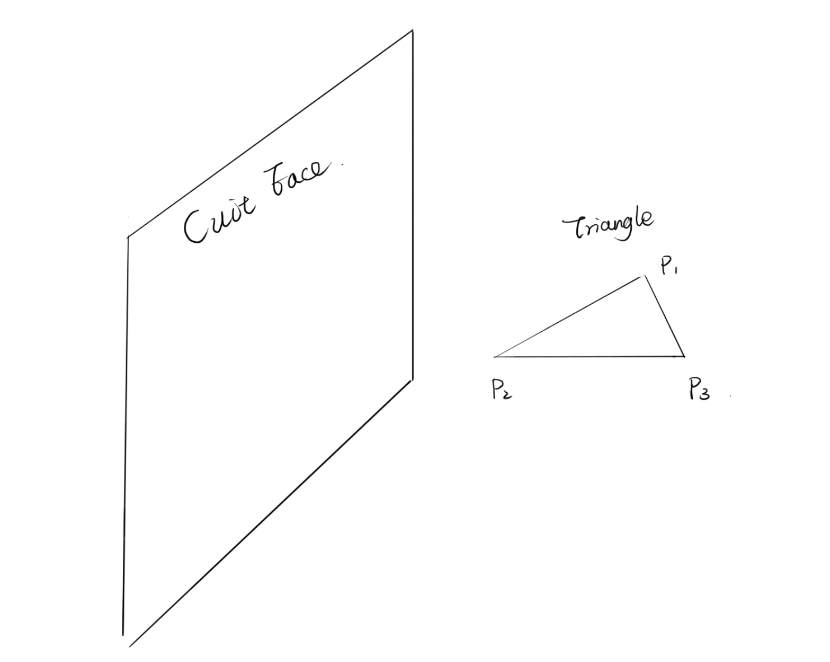
A. 用于获取一个直线和一个平面交点的辅助函数。该函数利用基本的立体几何知识，根据平面方程以及直线方程计算得出交点，该方程原理简单，就不详细介绍。

B. 通过输入一个空间立方体的最小点和最大点，随机返回一个可以切开该立方体的平面方程。该方程实现的基本逻辑是：首先通过立方体的最小点以及最大点，计算出立方体的几何中心以及体对角线矢量，随机生成一个平面法向量方向(a, b, c)，在体对角线上选取一点p，添加约束平面经过点p，解一个一元一次方程即可获得一般平面方程ax+by+cz+1=0可切开立方体，返回该平面方程的法向量即可。

C. 通过输入一个平面和一个网格，将这个网格按照平面切分为左右两部分子网格m1和m2的函数**splitMeshByFace**，该函数在后面的模型破碎和加载当中均有重要作用，因此在此详细讲述。

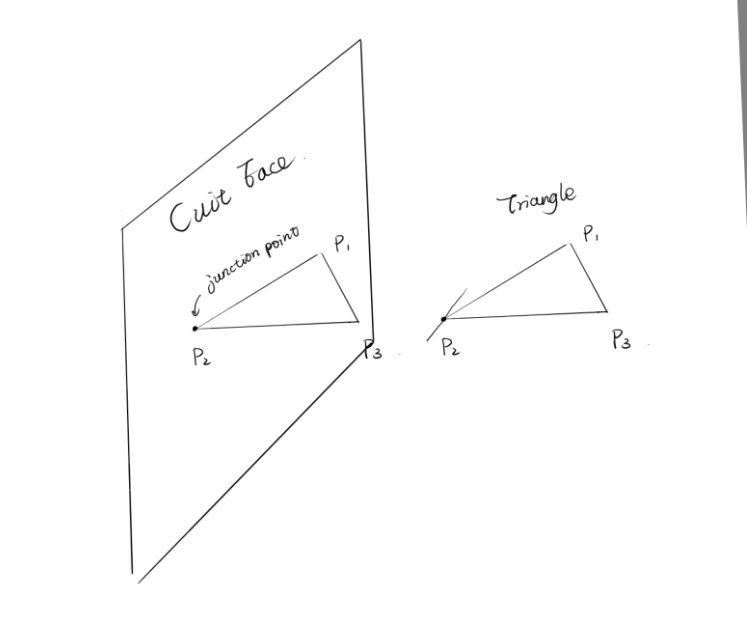
因为我网格类的设计可以保证所有的网格面都会由三角面片组成，将一个大网格切开的过程实质上就是将所有经过该切割面的边分开并重新生成新的边的过程，因此对一个大网格的处理过程被简化为对每个三角面片的处理过程，将一个三角面片的三个顶点代入切割面方程ax+by+cz+1，假设计算得到的结果为r1, r2, r3，根据三个顶点计算得到的结果值，该处理过程可以细分为一下几种情况：

i. 切面不过该三角面



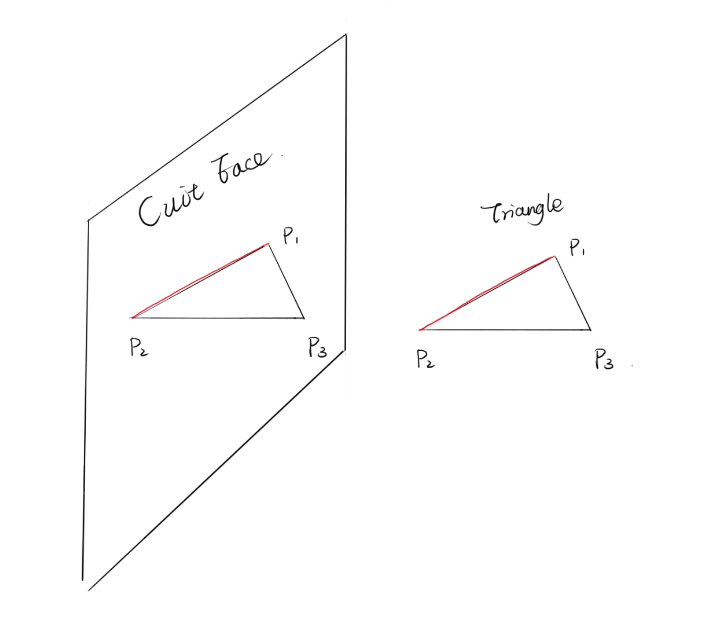
该种情况不需要任何处理，此时r1、r2、r3结果必然同号，只需根据正负号将该三角面放入对应的m1或m2中即可。

ii. 切面经过三角面的一个点，其它两点在切面同侧



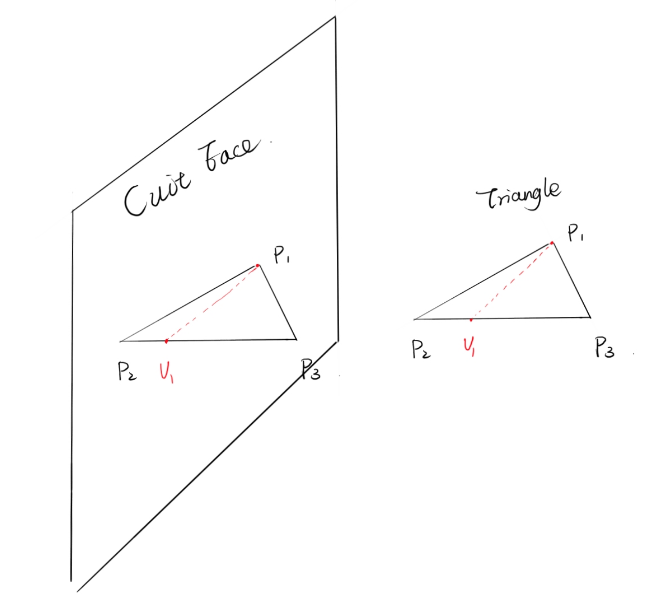
这种情况r1、r2、r3中有一个为0，其余两个同号，只要将2个点按照符号分配到对应的网格中，并将平面上的点加入到两个网格中即可。

iii. 一条边在顶点上，第三点在顶点外



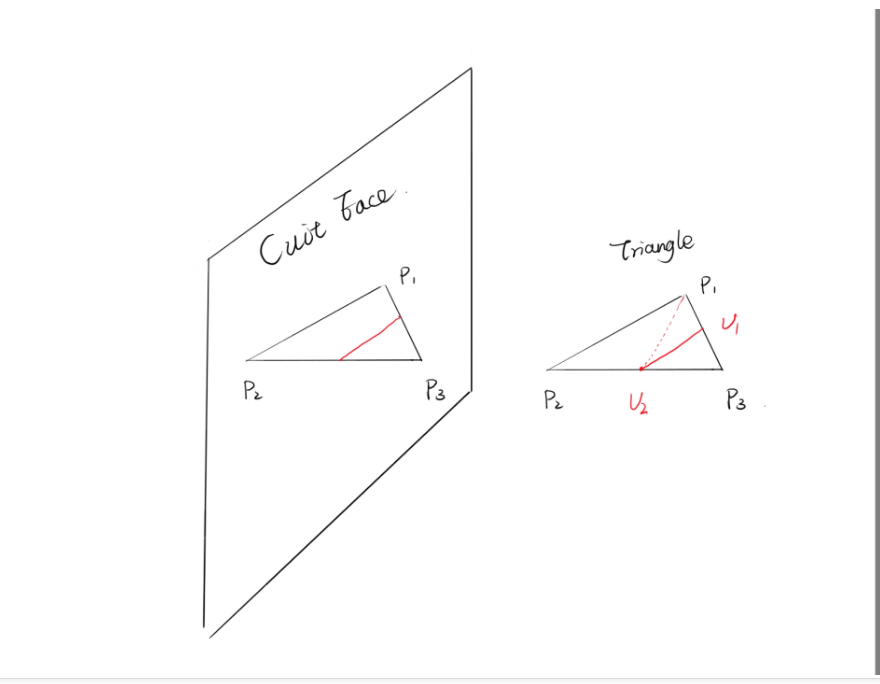
此时将面上顶点加入到两个子网格，将面外顶点按照符号加入对应网格即可。

iv. 三角形的一个顶点在面上，而且有一边和面相交



此时需要断掉p2p3边，对左侧网格，加入三角面p1p2v1，对右侧网格，加入三角面p1v1p3即可。

v. 三角形的两条边都经过切割面



此时r1、r2、r3的取值应该为2负1正或2正1负，假设是图中情况，左侧网格加入三角面p2p1v2、三角面p1v1v2，右侧网格加入三角面v1p3v2即可。

vi. 三角面和切割面重合

此时r1、r2、r3均为0，将该三角面加入到两个子网格中。

这样就处理了所有的情况，也是网格切割函数的主要逻辑部分，其余的一些实现细节就不赘述，但很重要的一点实现细节是，对所有生成的子网格一定保证其顶点的出现顺序的时针方向和原模型的方向一致，这对后面在模型加载中的顶点法线生成十分重要，可以大幅度避免法线方向相反导致错误光照的问题，其余函数如绘制函数之类不详细介绍了。

**②.模型类设计**

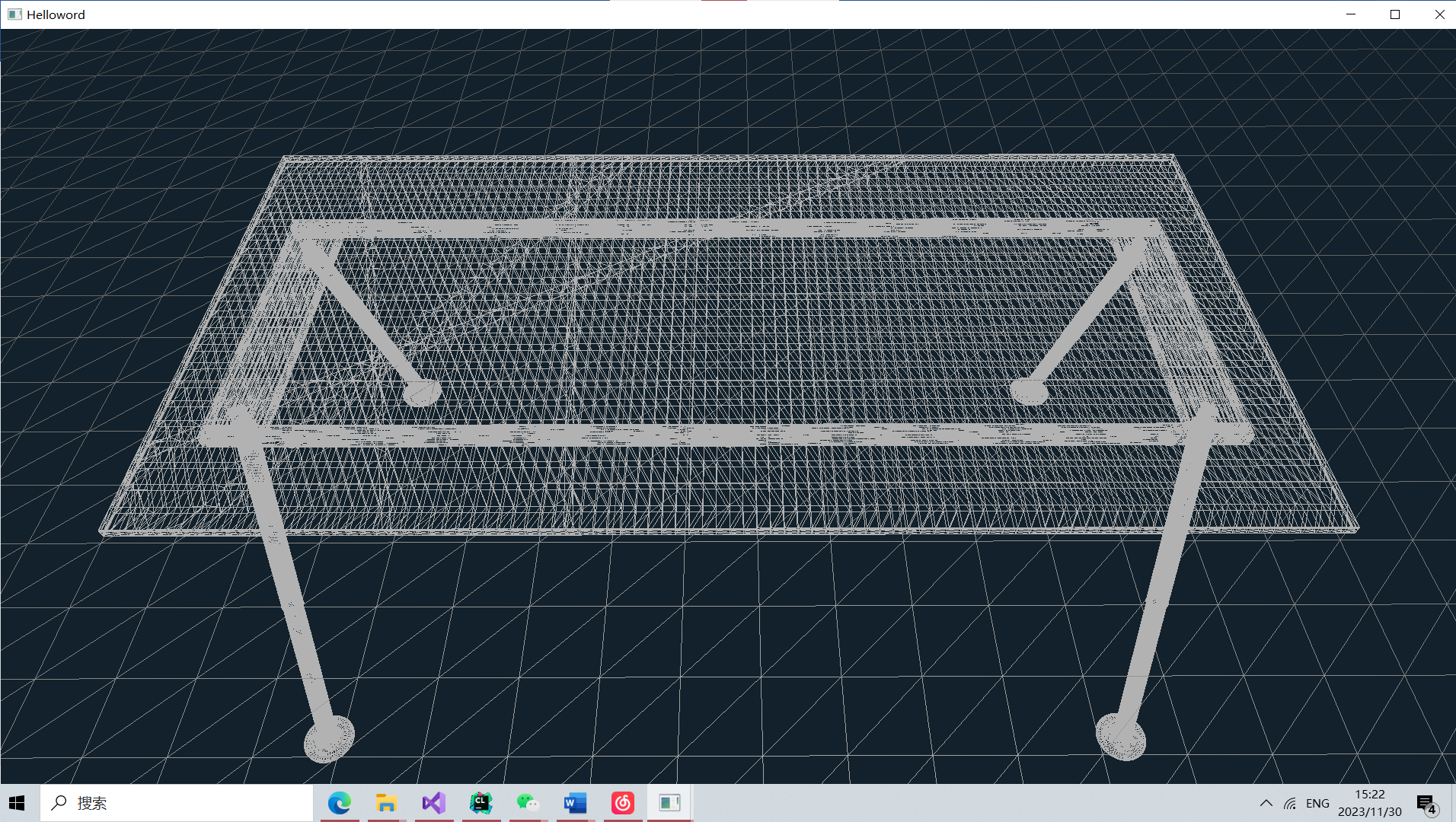
模型类储存的成员变量包括：

* Mesh向量数组meshes，储存当前模型的所有网格
* Mesh向量数组innerMeshes，储存当前模型所有网格沿着法线方向收缩得到的子网格
* Mesh向量数组collideBox，以一个空间立方体的形式模拟碰撞箱（collideBox[0]当中储存的是整体的碰撞箱，之后按顺序储存各个子网格的碰撞箱，之所以有分开储存的子碰撞箱是为了方便分开计算各自的重心坐标和受力）
* 四维矩阵数组trans、rotate，这两个数组的0号位记录整体的位移矩阵和旋转矩阵，之后按顺序记录每个网格的位移矩阵和旋转矩阵，同样是为了方便计算各个子网格的受力。
* 四维矩阵scale，用于保存当前模型的放大或缩小倍数，可以通过模型类的函数**setScale**来更改scale矩阵的值，实现模型的放大缩小。
* 三维向量数组velocity, angularVeocity, acceleration, angularAcceleration, gravityPosition每个数组的0号位用于保存当前整体模型的速度、角速度、加速度、角加速度、重心坐标，之后按照顺序保存各个网格的速度、角速度、加速度、角加速度、重心坐标。
* 其余比如质量、转动惯量、默认重心坐标、（角）速度递减系数等等相关可设置变量

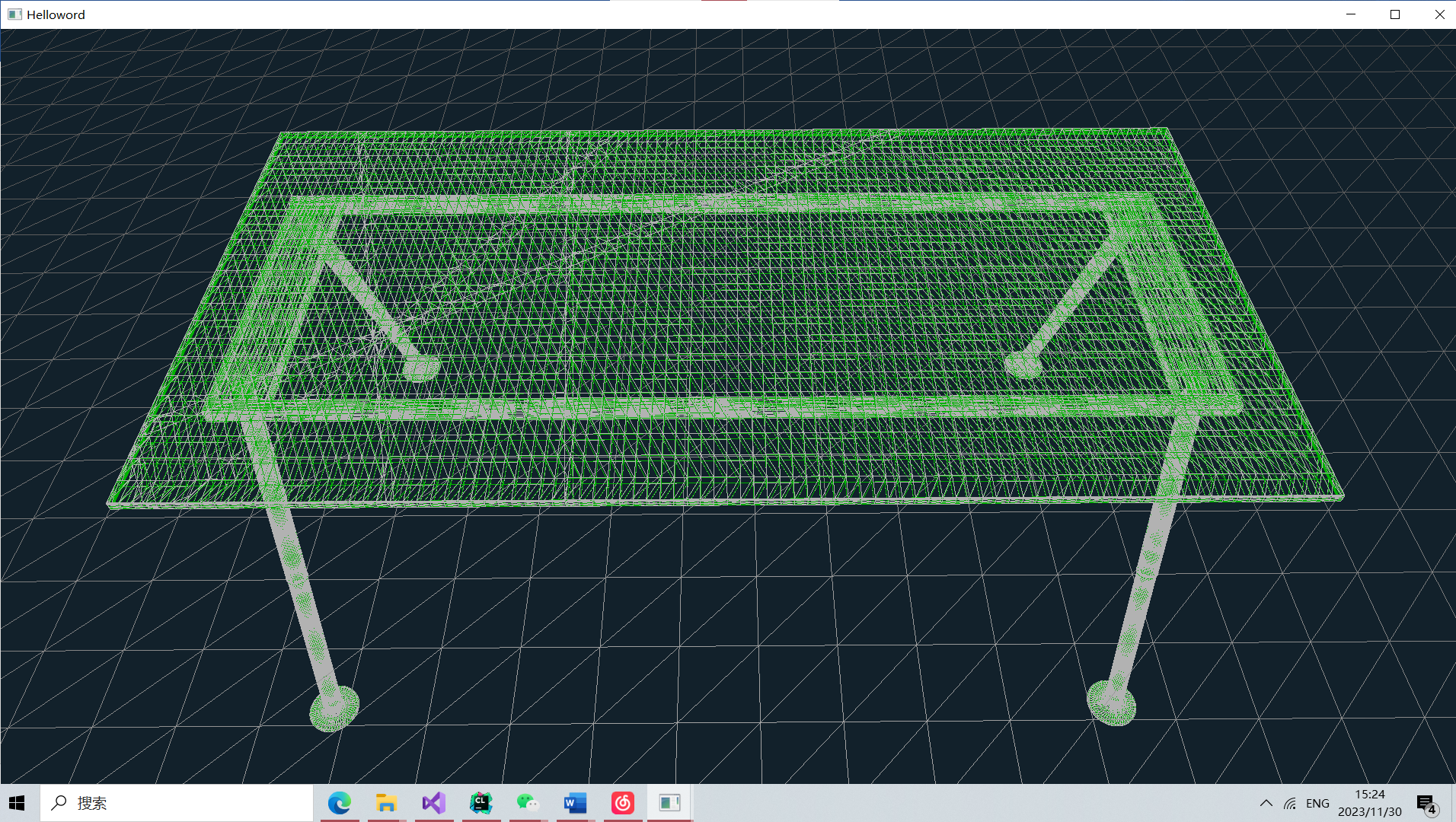
模型类的一些重要函数：

A. **loadModel**，该函数接收一个模型的相对路径和一个bool值，bool值为true代表当前模型可破坏，在加载模型网格的时候会根据整体模型的最小点和最大点对加载出的每个网格进行随机切分（默认设置的是切5~9次，生成2^5~2^9个子网格，但实际上因为再对子网格切割（由于输入的最小点和最大点是相对模型整体而言的，并不是相对于被切割的子网格而言）很可能不会切出新的子网格，所以生成的子网格数量应该会远小于理论值（取决于这个模型网格是否稠密，越稀疏切割出的子网格越少）。该函数调用assimp的内置模型加载方法，对读取得到的aiMesh进行处理转为自己定义的Mesh类，并且完成外部网格切割加载后，根据外部网格的三角面为每个顶点生成自己的法向量（详细方法是将所有经过这个顶点的面的法向量都加到该点的法向量中，最后再进行一次标准化，虽然有些模型自带了法向量，但在这里都必须重新计算，因为被切割生成的新顶点一定没有自带法向量），然后生成各个网格的内网格（只需要将外网格的顶点位置沿着法向量的负方向收缩即可），将内网格的法向量记为外网格法向量的负值，这样内网格也可以产生合理的光照，最后一步是根据保存好的所有网格的最大点和最小点，生成包裹各个网格的碰撞箱并计算他们对应的重心坐标。

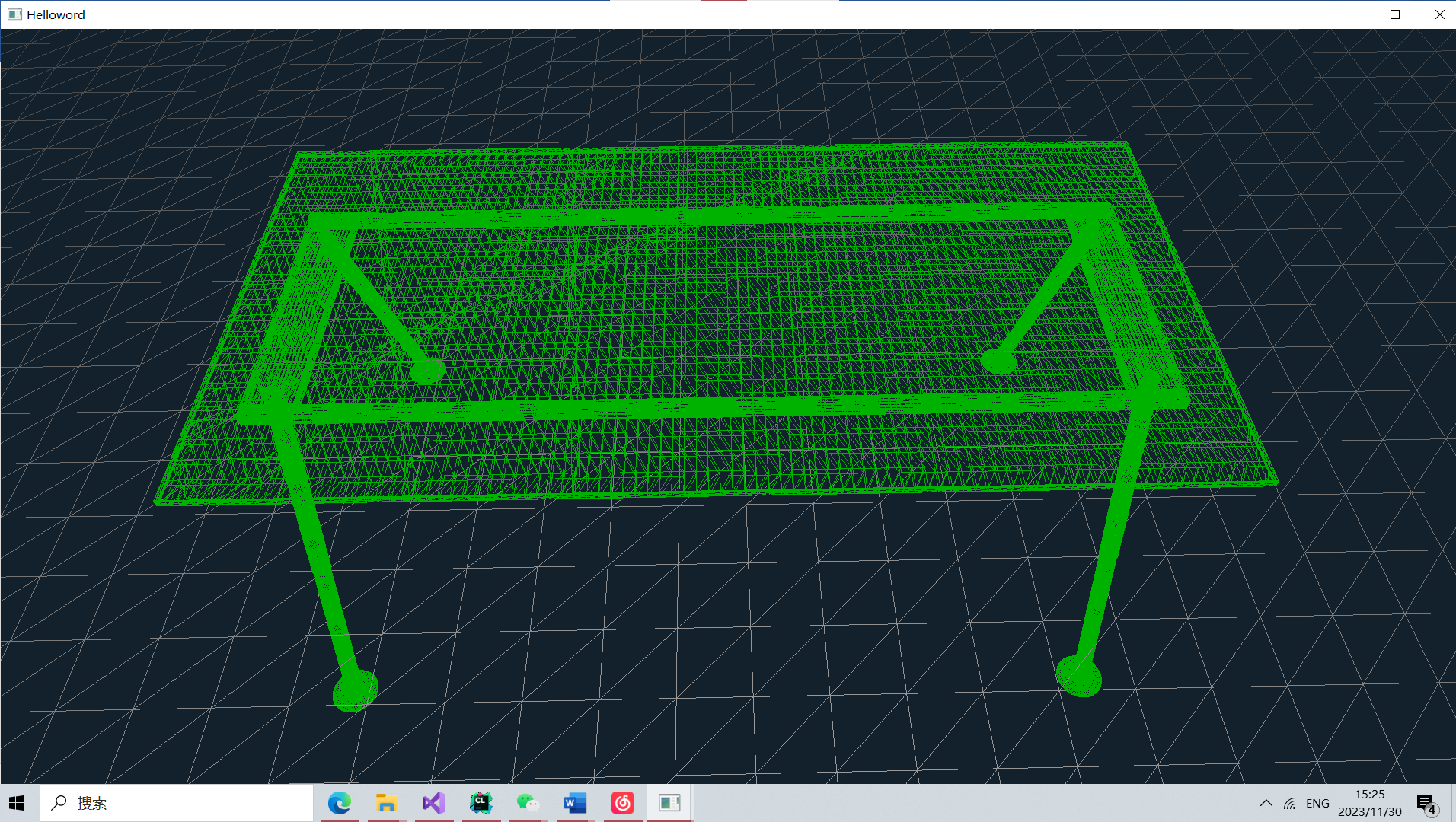
加载好模型后，大致界面如下：



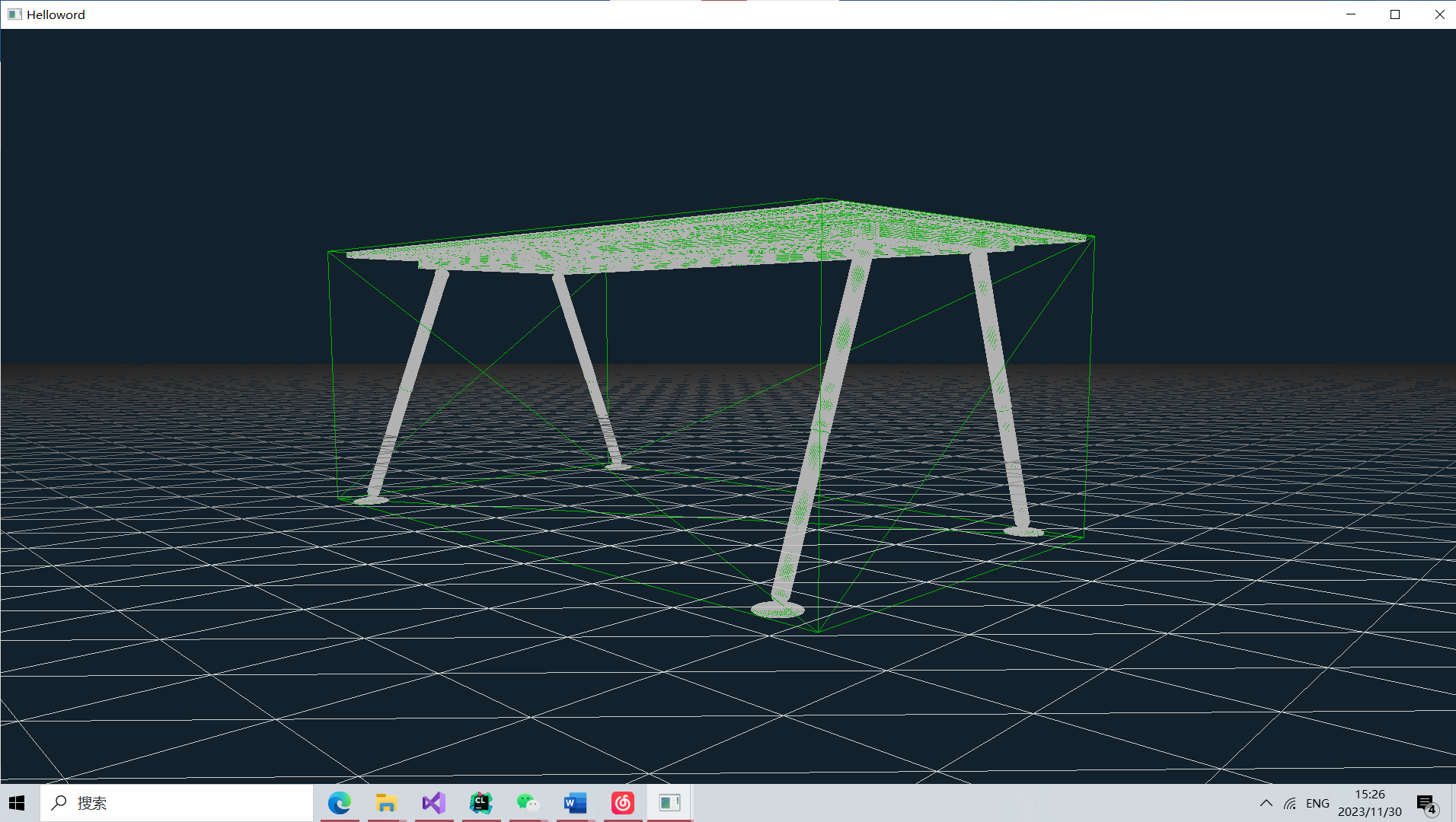
由于应用了切割，桌子模型上方的斜线是切割出来的切面痕迹。加载好模型后，按下i键可以打开内网格（用绿色表示）



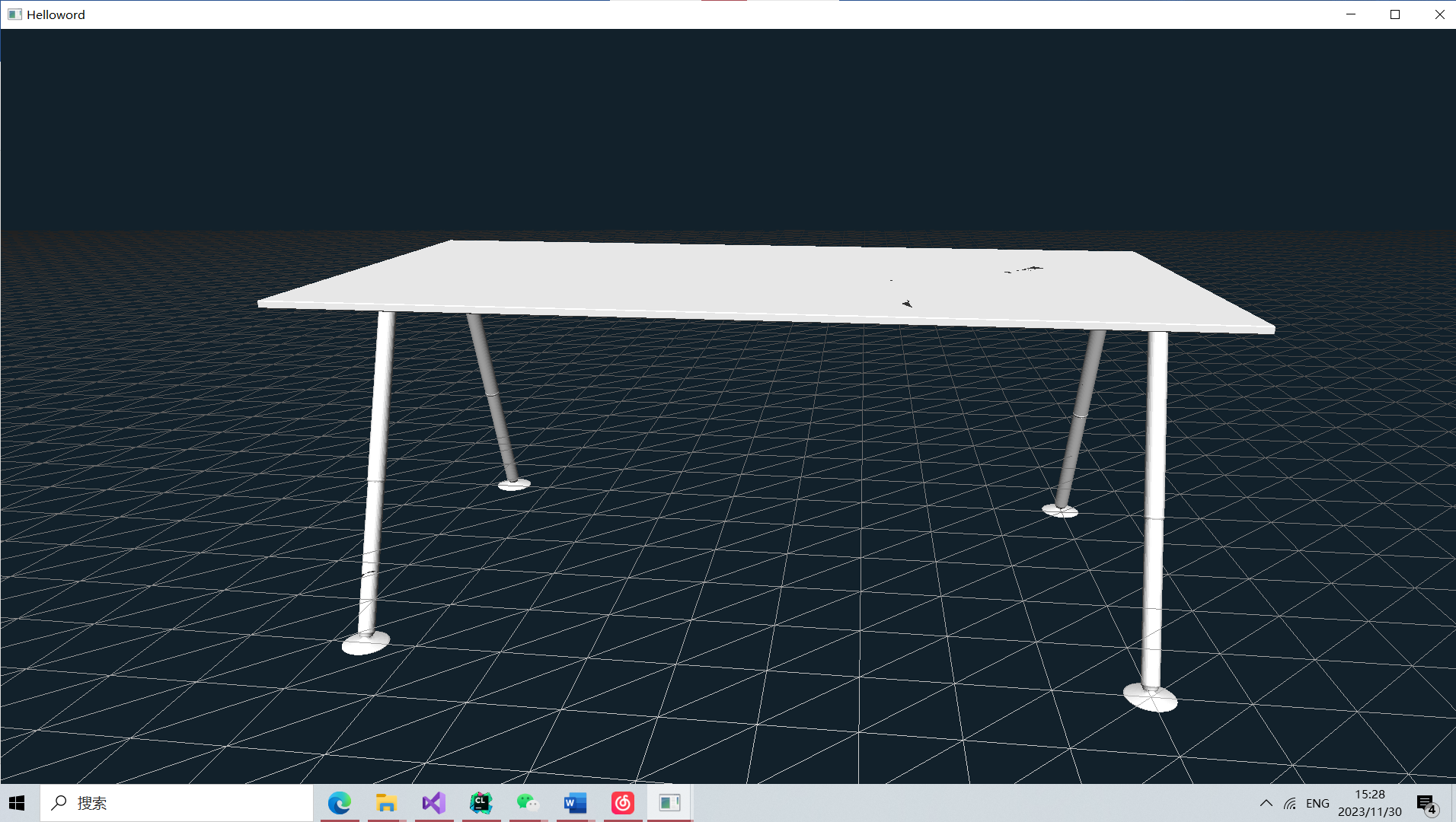
可以按o关掉外网格只显示内网格



按c显示模型的碰撞箱



按l开启全局光照（上方出现的小黑色三角形猜测是原本模型的三角面没有严格遵守单时针方向导致计算出来的法向量方向存在相反的情况，从而得不到正确光照）



B. **useForce**函数，该函数接收三个参数：三维力矢量force（默认为零矢量），作用点坐标positio（默认为0矢量）n，以及受力网格编号meshNum（默认为0，即对整体施加的力），这个函数按照基本的刚体力学规则，根据三维力矢量和作用点坐标计算分别计算出力的切向分力和法向分力，从而得出下一个delta时间（模型内deltaTime的取值，默认为1/60秒，但是不建议更改，可能会带来一些问题）内模型的加速度和角加速度，如果模型的useGravity变量设置为true，即使force的模长度为0也仍然会应用重力（这里直接将加速度的y方向设置为负的gravityCoefficient的值，默认是9.82）。

C. **move**函数，该函数根据当前的velocity和angularVelocity的值，计算下一个delta时间内，模型的位移和转动量，并对trans和rotate矩阵做对应的修改。如果useGround变量设置为true，会在当前网格的重心位置接近defaultGravityPosition的时候调用hitGround函数；如果useGravity变量设置为true，而且重心高于地面时，会同时调用不传变量的useForce函数以获得重力加速度；假如useDecrease变量设置为true，会对速度和角速度应用一个衰减值（衰减值的大小分别由vDecreaseCo和aDecreaseCo控制），以避免出现模型运动一直不停止的情况。该函数不接收参数，但受内部成员变量splited的影响，假如splited为true，代表当前模型已经分裂，会对每个网格分开进行计算，反之则认为模型还是一个整体，按照整体进行计算。

D. **hitGround**函数，该函数是一个半成品，因为它并没有完全模拟一个长方体落地的情况，只是保证了一个长方体的y坐标不会太高或者太低，而是被重心坐标等于原重心坐标而约束。该函数被调用时说明当前网格的重心位置将要或已经低于原本的重心位置，会根据当前模型的状态做出如下的几种判断：1. 模型的y轴坐标距离默认重心位置高于一个deltaTime时间内自由落体所能走过的最大垂直距离，会让它继续下落。2. 模型的y轴坐标低于默认重心位置，但高于地面，而且速度较大，此时会将模型的位置设置为地面位置，并将速度的y方向取反并乘以bounciness变量以获得一个弹性衰减。3. 模型的y轴坐标低于默认重心位置，而且模型速度较小，此时会将模型位置调整到重心位置，将速度的y分量取反再额外乘以一个小于1的值以避免模型在近地面的抖动。

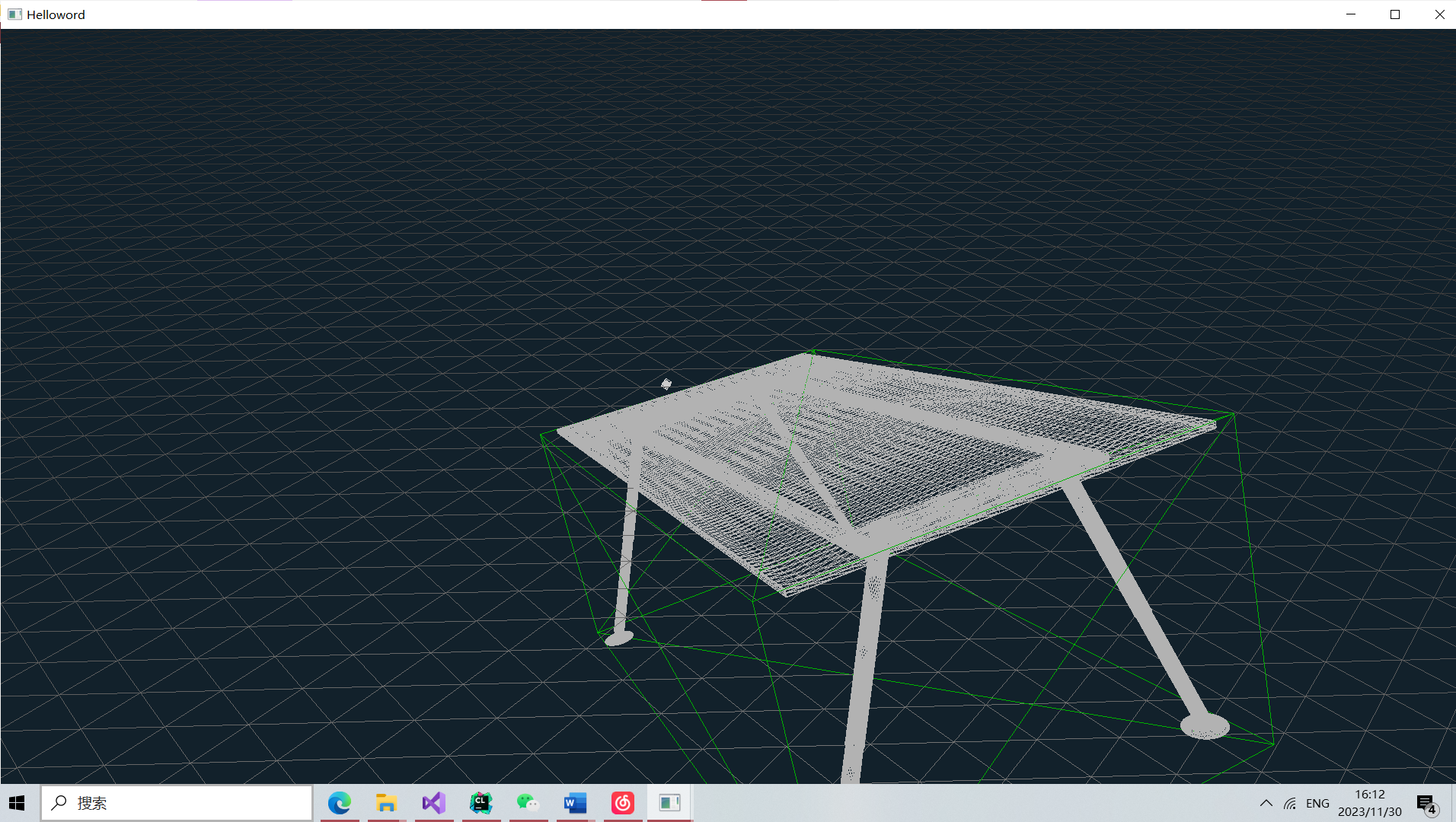
该函数实质上只是模拟了一个质点和地面的碰撞情况，对模型的旋转情况并没有做太多限制，所以只是一个半成品函数。

E. **detectCollision**函数，该函数用于检测两个立方体碰撞箱是否发生碰撞，算法采用的是三维的分离轴定理，详细的实现不过多赘述，具体的分离轴定理内容可以参考以下的文章：[SAT 分离轴算法 · Issue #23 · phenomLi/Blog (github.com)](https://github.com/phenomLi/Blog/issues/23)，如果碰撞检测成功，则根据是否开启模型破碎，分别调用**collisionWithNoShatter**或者**collisionWithShatter**函数

F. **collisionWithNoShatter**函数，该函数是一个静态函数，用于对两个模型整体应用模拟碰撞的效果（也就是说两个模型都会受碰撞的影响，但在程序中我让小球在质量和转动惯量不变的情况下体积缩小进而减小力矩，所以对小球的效果应该不会很明显），不会产生模型破碎。实现流程如下：1. 获取小球的重心位置和模型的重心位置，获取小球

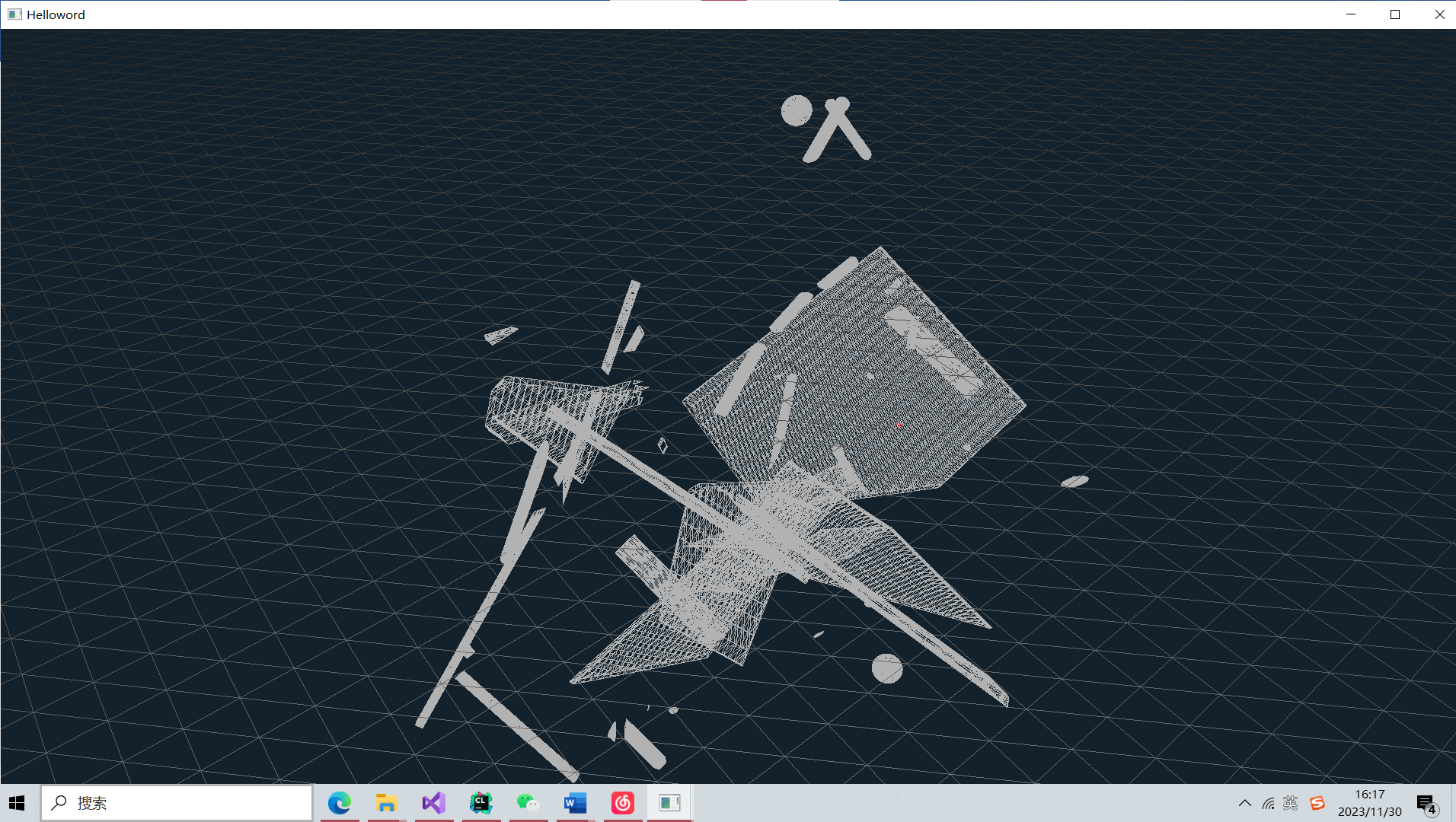
速度矢量和模型速度矢量。2. 获取小球体对角线的半长d1和模型体对角线的半长d2。3. 在小球重心和模型重心连线l的处对模型应用力，在处对小球作用力。4. 因为真正的碰撞模拟受力很复杂，这里我用如下公式做了简化：将被碰撞物受到的碰撞力矢量设定为：

其中f为常数项，获得力和碰撞点后，对两个模型分别调用useForce即可。

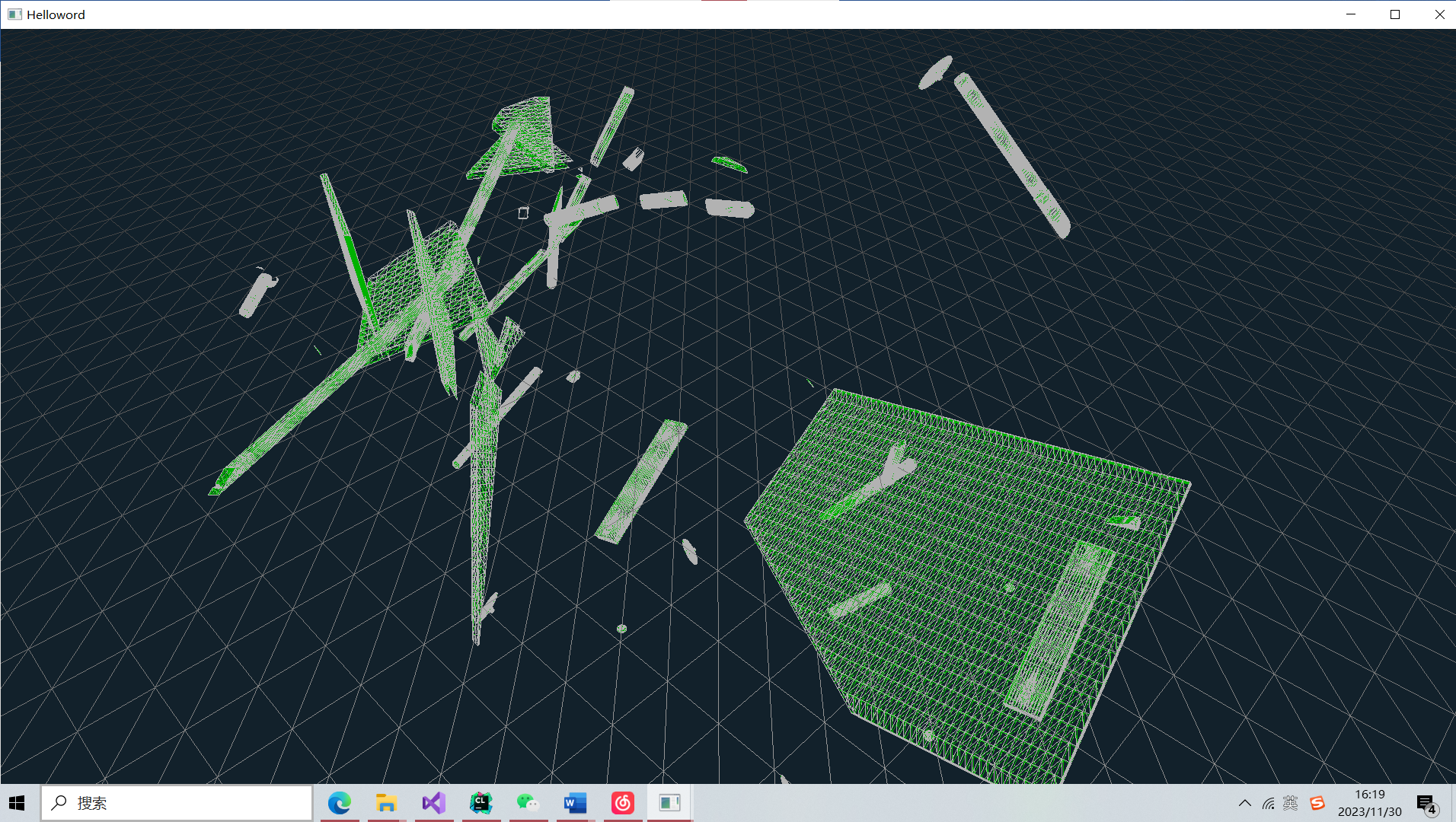


(模型被小球碰撞而发生倾斜，小球发生转动)

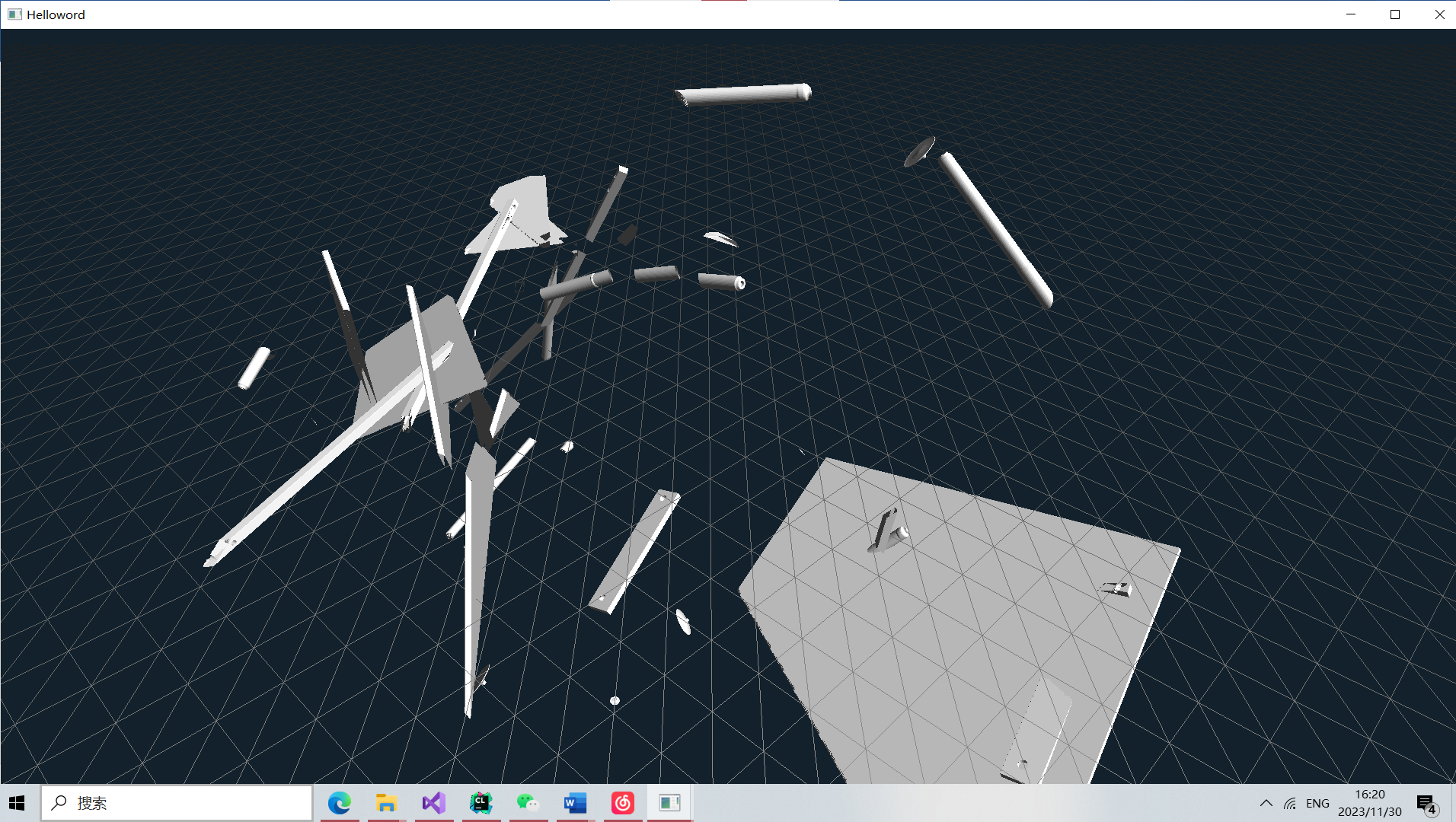
G. **collisionWithShatter**函数，该函数用于模拟小球碰撞后的模型碎裂效果。具体实现方法是对所有的网格分别计算力以及应用受力。受力公式和**collisionWithNoShatter**函数基本相同，但将f设置得稍微大一点，以让模型破碎的效果明显。



（模型因为和小球的碰撞而破碎）



（破碎模型和内表面）



(破碎的模型应用光照)

**③. 其它内容**

其余内容如摄像机、地面网格等和本次作业要求不强相关，因此就不赘述。技术亮点是借鉴了Unity等游戏引擎里的显示方式，做出了相机动态移动速度和网格颜色按照相机位置发生渐变的功能。光照模型采用的是冯氏光照模型，这里因为导入的模型大小不一采用了太阳光（其实可以加一个把模型大小统一化，但是时间原因没来得及做）。具体的实现细节可以参考代码内容。

**4. 小结和建议**

本次作业基本完成了要求的功能，虽然作业过程比较艰辛，踩了很多坑。但还是让我受益良多。实现整体碰撞时，我的模型类没有为所有的网格分别配置速度、加速度、角速度和各种矩阵等变量，而是只作为一个整体来运算，因为需要改动的代码量少，最初的方案是在碰撞时计算碎片，将模型碎片变成子模型来保存下来，但这样试了一下后发现对内存的要求高得不行，于是改为在当前模型中为所有的网格分别配置各种参数的模式，代码重构花了不少时间，但是却获得了很好的效果：要求里的为破碎的模型显示内网格其实我根本就不用花时间完成，因为一切都在模型加载的时候储存在成员变量里了。碰撞也只需要从一个矩阵的运算转化为对多个矩阵的分别运算，大大加快了运行速度，即时地计算碰撞碎裂真是电脑不能承受之重。

但不足的点同样很多，第一就是没有想出比较合适的真正基于模型的所有面计算重心的方式，也没有真正实现一个立方体和地面碰撞所收受到的转向约束，而是通过模型的最大点和最小点取中点来简单地模拟重心，并根据重心位置来约束模型重心的y值，这样就导致一些模型碎片的样子变得很奇怪（很可能它模型非常稀疏，但是最大最小点的位置又很分散），导致破碎后模型碎片的位置非常随机。而且模型在低空时好像还会有些隐藏的bug导致下降速度有问题，虽然很在意但是没时间做了，最近的学习压力有点大（哭死）。

作业要求中的**“断裂切面生成三角形条带”**这个功能原本是尝试了的，但实现方式比较

粗糙，导致最后出现的画面效果不佳，我简单地记录下所有过切面的顶点和新增顶点，并逐一链接形成三角面，这就导致当一个模型不是单纯的凸模型时，如果切面过了凹面，很可能凹面上会出现切面的粘连线（有点像3D打印的粘连），导致视觉效果很糟糕，如果要实现较好的效果可能还要涉及一些切面网格凹凸判断的问题，时间太少坑太大，所以我把这一块删了没做了。其余的功能应该是正常完成了。

**5. 程序使用说明**

点击程序源代码目录下的start.exe程序即可开启程序

（开始之前请将模型obj文件放入model文件夹中，命名为model.obj，推荐使用我已经放置在model文件夹中的提供的模型中的桌子模型，不然一些大小不合适的模型用程序内默认的参数表现得可能很奇怪，比如因为模型比发射的小球还小而显得自身质量密度和转动惯量都很大，导致力矩作用不明显。 如果坚持用其它的模型，最好调一下源代码当中的质量、转动惯量和模型的scale矩阵，并重新编译一下）

**W/A/S/D键 控制摄像机移动**

**Q/E键 控制摄像机画面上升和下降**

**鼠标左键按住并拖动 转动摄像机画面**

**F键 摄像机归位（回到原点）**

**I键 打开/关闭内网格**

**O键 打开/关闭外网格**

**L键 打开全局光照**

**B键 发射/回收小球**

**G键 切换碰撞模式，默认为不会造成模型碎裂的碰撞（小球白色），可切换为会造成模型碎裂的碰撞（小球红色）**

**M键 模型归位（回到原点），被破坏的模型恢复原状**