

# 可视计算与交互概论 Lab2 报告

邱荻 2000012852

2022 年 11 月 30 日

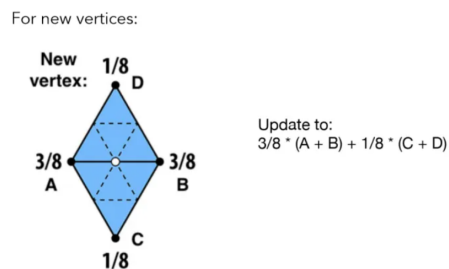
## 1 介绍

这次 Lab 中，我实现了可视计算与交互概论课程几何处理部分介绍的几种重要的算法或思想，包括几何表示中的 Loop Mesh Subdivision 算法，几何表示中的 Spring-Mass Mesh Parameterization 算法，几何处理中的 Mesh Simplification 算法，几何处理中的 Mesh Smoothing 算法，几何重建中的 Marching Cubes 算法。

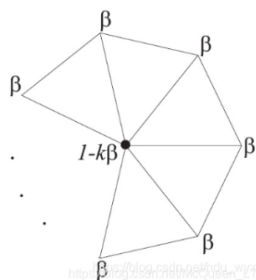
## 2 实现思路 and 结果

## 2.1 Loop Mesh Subdivision

对新生成的点:



移动原来就有的点：



$$O' = (1 - n\beta) \cdot O + \beta \sum_{i=0}^{n-1} V_i$$

where  $\beta = \frac{1}{n} \left[ \frac{5}{8} - \left( \frac{3}{8} + \frac{1}{4} \cos \frac{2\pi}{n} \right)^2 \right]$

先新生成点，并且记录这条边和新生成点的对应关系，以便后面连接点形成面。然后对每个旧点找出它的 neighbor，按照公式移动旧点。最后按面遍历，连接点形成新的面（维护 indices，每 3 个 indices 对应一个面），每个面变成四个面。

结果如下：

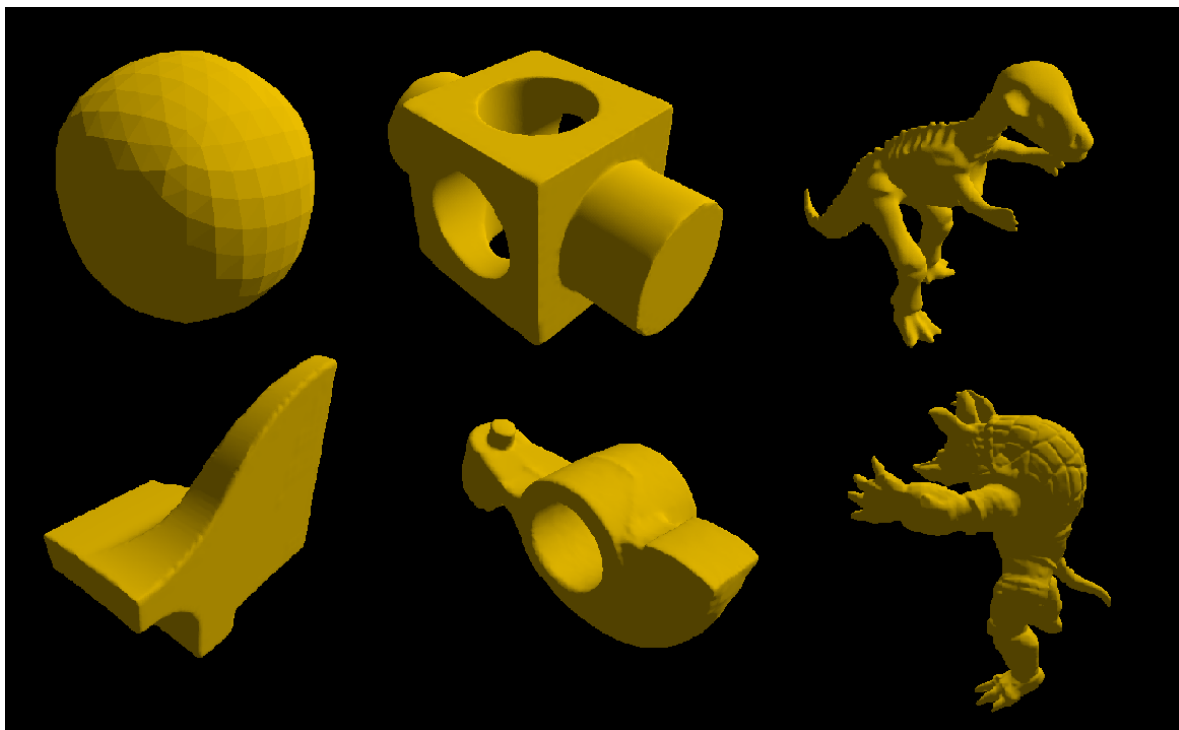


图 1: Loop Mesh Subvision

## 2.2 Spring-Mass Mesh Parameterization

基于弹簧质点的三角网格参数化算法。该方法将三角网格模型中的每一条边都看作为一个弹簧，先将三角网格的边界映射到预先定义好的多边形上，然后通过最小化整个三角网格模型的弹性势能来参数化内部空间点。

1. 检查边界点，用 `v.IsSide()` 找出第一个，然后用 `GetSideNeighbor()` 顺着找一圈的边界点，要顺着找，因为是边界点是顺着映射到平面的边界。
2. 初始化边界点上的 UV 坐标，我这里选用圆边界。

设网格曲面  $M$  总共有  $n$  个顶点，其中有  $k$  个边界顶点  $v_i (i = 1, \dots, k)$ ，找出网格的边界顶点。网格曲面边界顶点  $v_i$  的弦长参数化为：

$$t_i = \frac{\sum_{j=1}^{i-1} |v_{j+1} - v_j|}{\sum_{j=1}^{n-1} |v_{j+1} - v_j|} \quad (1)$$

单位圆域，则边界起始点和边界点  $v_i$  之间的圆弧所对应的圆心角为  $2t_i\pi$ ，则边界顶点参数化的结果为：

$$(x_i, y_i) = (\cos 2t_i\pi, \sin 2t_i\pi) \quad (3)$$

简单起见使用  $w_{ij} = 1/n_{ij}$  的平均权重，且要注意本题的  $uv$  要在  $[0,1] \times [0,1]$ ，所以用  $0.5 + 0.5 \cdot \cos(2 \cdot \pi \cdot t \cdot i)$ 。

3. 迭代求解中间点上的 UV 坐标，使用入  $ij=1/n_i$  的平均权重，

$$\mathbf{t}_i - \sum_{j \in N_i} \lambda_{ij} \mathbf{t}_j = 0$$

最终结果如下：

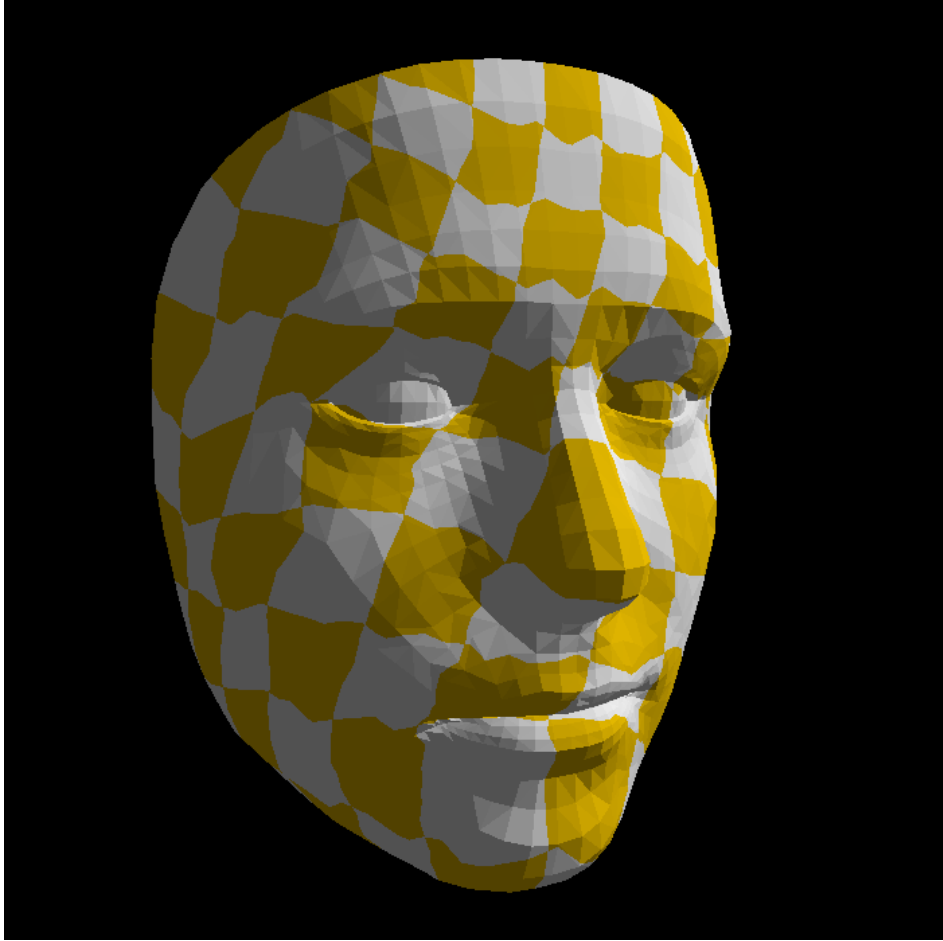


图 2: Spring-Mass Mesh Parameterization

## 2.3 Mesh Simplification

这个函数是 lab2 最难的一个，涉及到的细枝末节的操作比较多，bugde 起来也更难，因为要 de 后面的模型的 bug 得先等几分钟等第一个球跑完（雾）。

思路如下：

1. 为每个初始顶点计算二次代价矩阵  $Q_i$ 。定义  $plane(v_i)$  表示  $v_i$  对应的那些原始三角面，则

$$K_p = pp^T$$

$$Q_i = \sum_{p \in plane(v_i)} K_p$$

按面来遍历，方便得到点与面对应。a,b,c,d 的计算：

$$\vec{n} = \vec{p_1 p_2} \times \vec{p_1 p_3} = \begin{vmatrix} i & j & k \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = ai + bj + ck = (a, b, c)$$

$$a = (y_2 - y_1) * (z_3 - z_1) - (y_3 - y_1) * (z_2 - z_1)$$

$$b = (z_2 - z_1) * (x_3 - x_1) - (z_3 - z_1) * (x_2 - x_1)$$

$$c = (x_2 - x_1) * (y_3 - y_1) - (x_3 - x_1) * (y_2 - y_1)$$

2. 寻找合法点对的规则:

1.  $(v_1, v_2)$  是一条边
2.  $\|v_1 - v_2\| < t$ ,  $t$  由用户设置

寻找时计算出最优搜索点和规则, 把合法点对按照 cost 大小插入 set。其中最优搜索点的计算方法如下:

$$\vec{v} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

4. 迭代地从顶点对中找出代价最小的那一对进行顶点合并, 直到剩余顶点的数量小于所需的比例。用一个 set 来记录有边的点对, 这样在合并的时候产生新点, 就可以把原来的点连的边去掉, 加上新点应该连的边。最后维护面的时候, 扫描原来的面上的点, 如果该点已经被合并, 就用新点来当这个面的点。点的替代关系用并查集来维护, 新点是旧点的父节点。

并且在对 vector 的全局变量操作前要 clear, 不然换下一个模型的时候变量会直接在之前的 vector 后面加, 会炸。

(等了很久终于跑出来的) 最终结果如下:

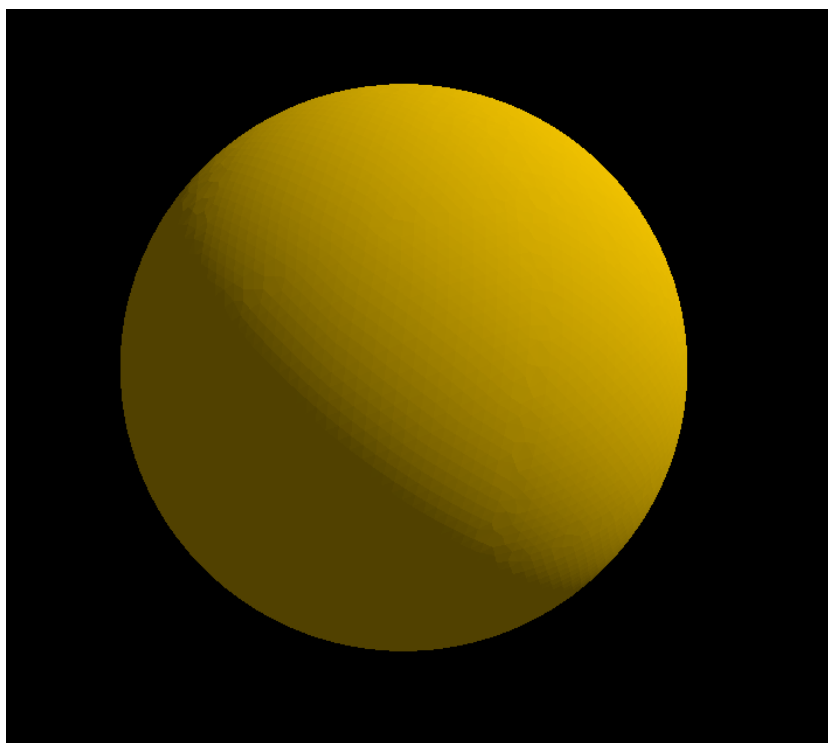


图 3: Simplify=2



图 4: Simplify=4

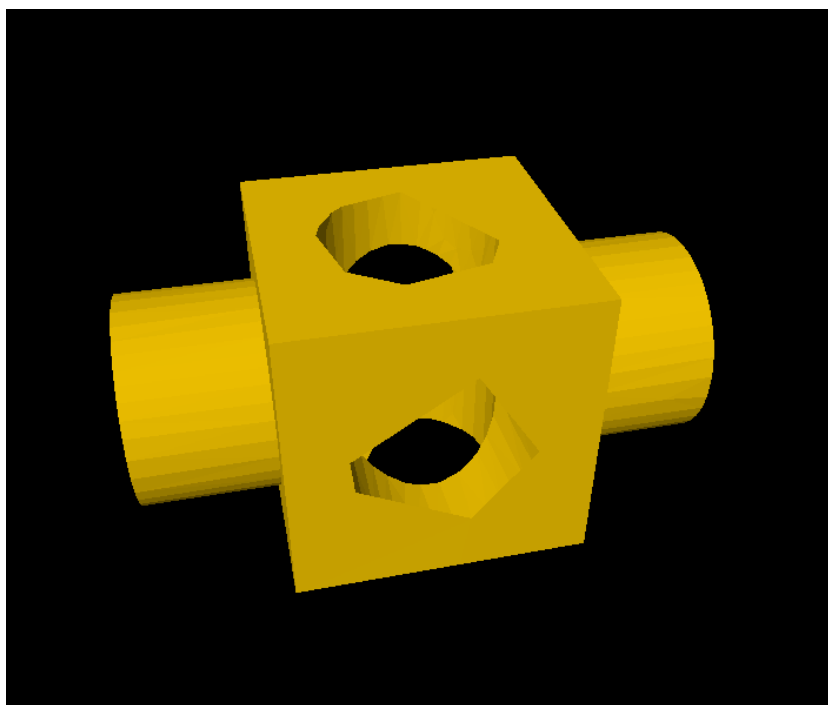


图 5: Simplify=4

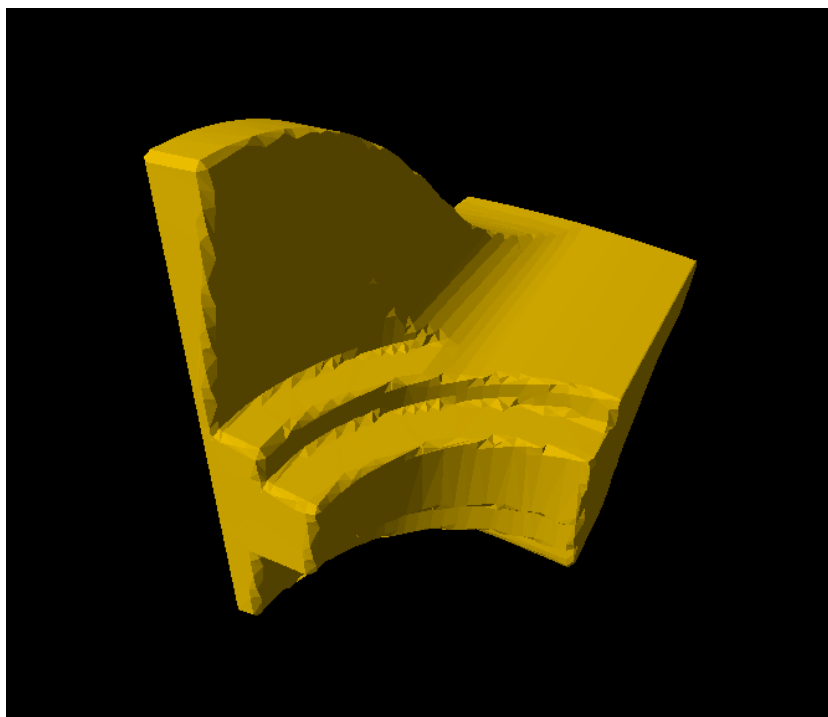


图 6: Simplify=4



图 7: Simplify=2

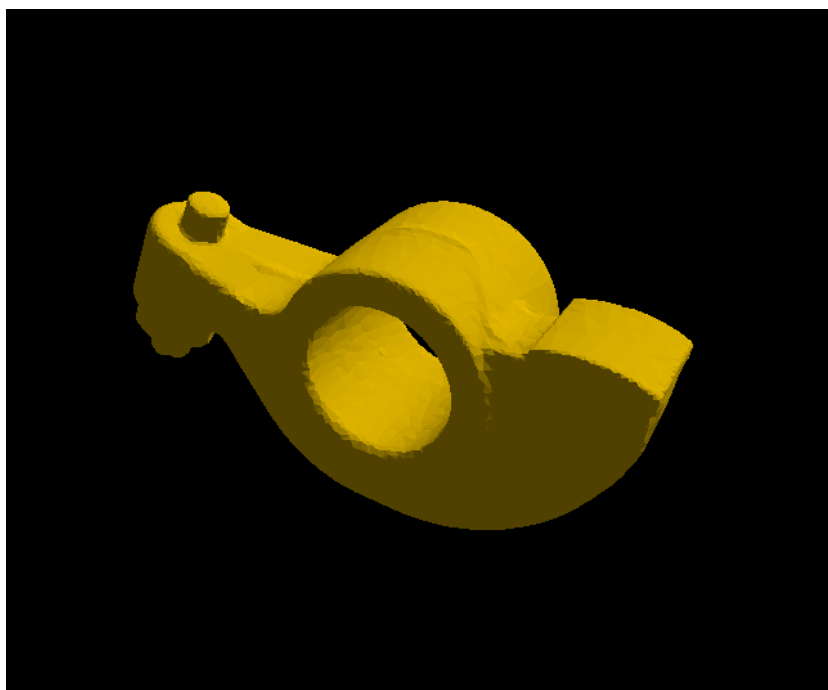


图 8: Simplify=2

## 2.4 Mesh Smoothing

大体流程按照 tutorials 给的来。

1. 对每个顶点  $v_i$  , 计算邻居位置的加权平均

$$v_i^* = \frac{\sum_{j \in N(i)} w_{ij} v_j}{\sum_{j \in N(i)} w_{ij}}$$

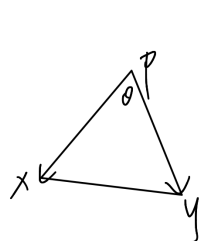
2. 其中使用 Uniform Laplacian 时  $w_{ij} = 1$  ; 使用 Cotangent Laplacian 时  $w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$

3. 更新顶点:  $v_i = (1 - \lambda)v_i + \lambda v_i^*$

4. 重复1-3步, 直到迭代次数达到上限

Uniform Laplacian 很好实现, 直接按给的流程码就行。Cotangent Laplacian 略难, 因为我是扫描的点, 所以在 Cotangent Laplacian 要取得  $w_{ij}$  时得到这两个点连起来形成的边的 OppositeVertex 和 PairOppositeVertex() 有一定的困难, 所以我先建立了一个由两个点得到一条边的索引。这样可以很方便得到需要的四个点。还要注意不是任何两个点之间都有顺着的边, 可能是反向的边。

余切值计算:



$$\cos \theta = \frac{\vec{PX} \cdot \vec{PY}}{|\vec{PX}| \cdot |\vec{PY}|}$$

$$\cot \theta = \frac{\cos \theta}{\sin \theta} = \frac{\cos \theta}{\sqrt{1 - \cos^2 \theta}}$$

当角度很小的时候,  $\cot$  很大, 有可能超过浮点数的范围, 然后就会形成窟窿。所以在  $\cot$  很大时我们可以人为加以限制, 这里取  $\max(\min(w, 900.0), 0.0)$ , 不能取太大了, 取太大了的话迭代多了参数大了也有洞。

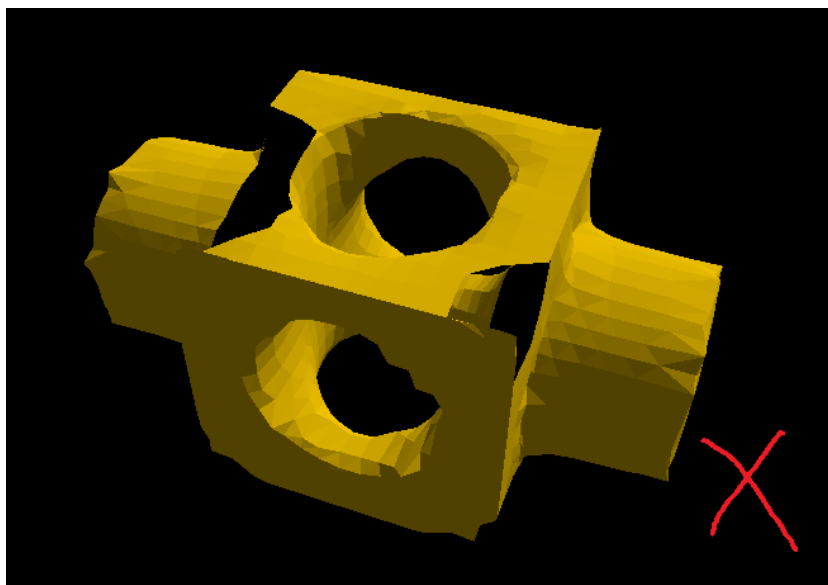


图 9: 不限制参数大小时有漏洞

最终结果如下:



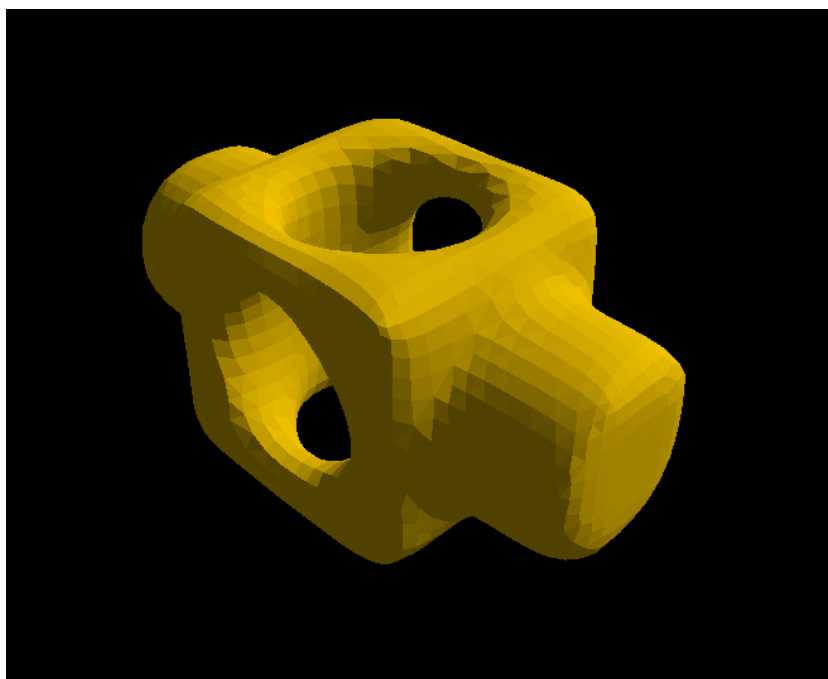


图 10: Uniform Weight Iteratoin=5 Smoothness=0.9



图 11: Uniform Weight Iteratoin=8 Smoothness=0.7

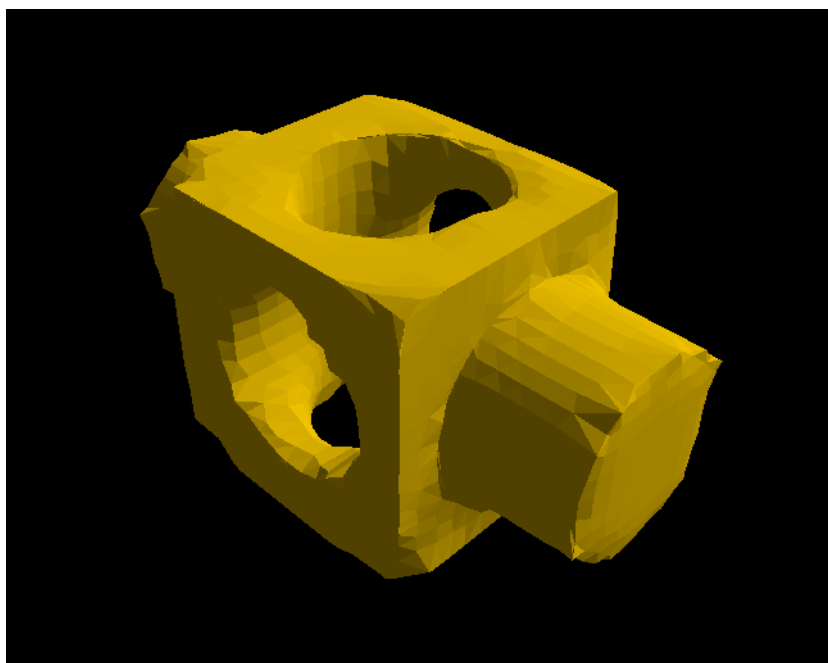


图 12: Cotangent Weight Iteratoin=5 Smoothness=0.9



图 13: Cotangent Weight Iteratoin=10 Smoothness=1.0

## 2.5 Marching Cubes

关键是找出零势坐标然后连接。流程如下：

1. 遍历每个 cube，检查每个顶点的势能值。
2. 查 cEdgeStateTable 表得到每条边上是否有点。

3. 扫描 12 条边，插值得到交点。
4. 查 cEdgeOrdsTable 表得到面的连接。

其中由于一条边属于两个 cube，所以有可能同一个点会在两个 cube 中同时生成，于是我用 `map<Vec3, int>` point to index 记录点与索引值的对应，用来检查这个点是否已经生成过了。并且由于 `glm::vec3` 没有重载 `<`，也没有定义哈希表，不能用于 `map` 或 `unordered map`，所以我自己定义了一个 `Vec3` 用于记录坐标，并且重载了小于号，这样就可以在 `map` 里使用。

最终结果如下：

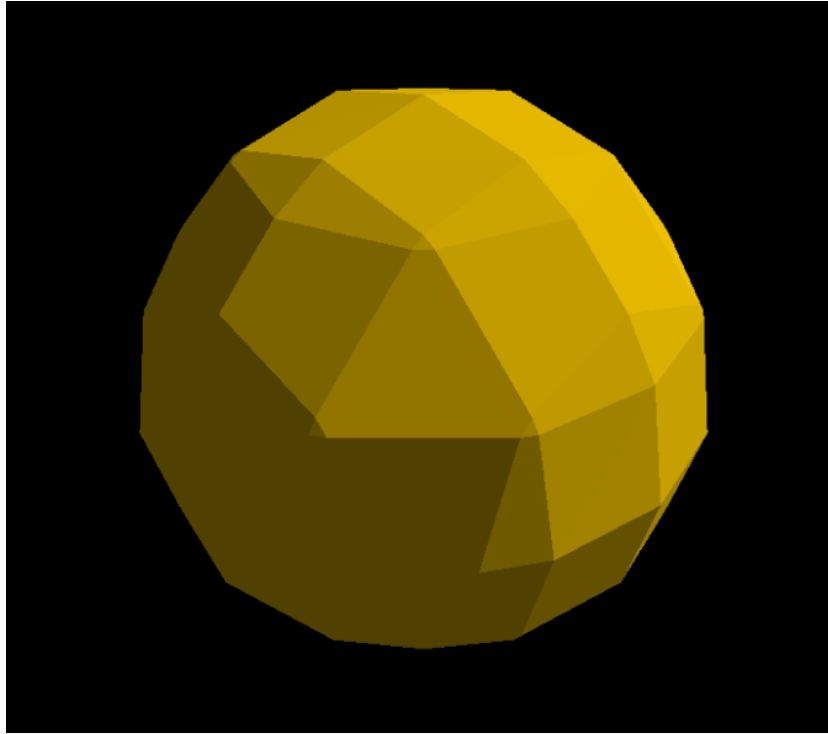


图 14: Resolution=10

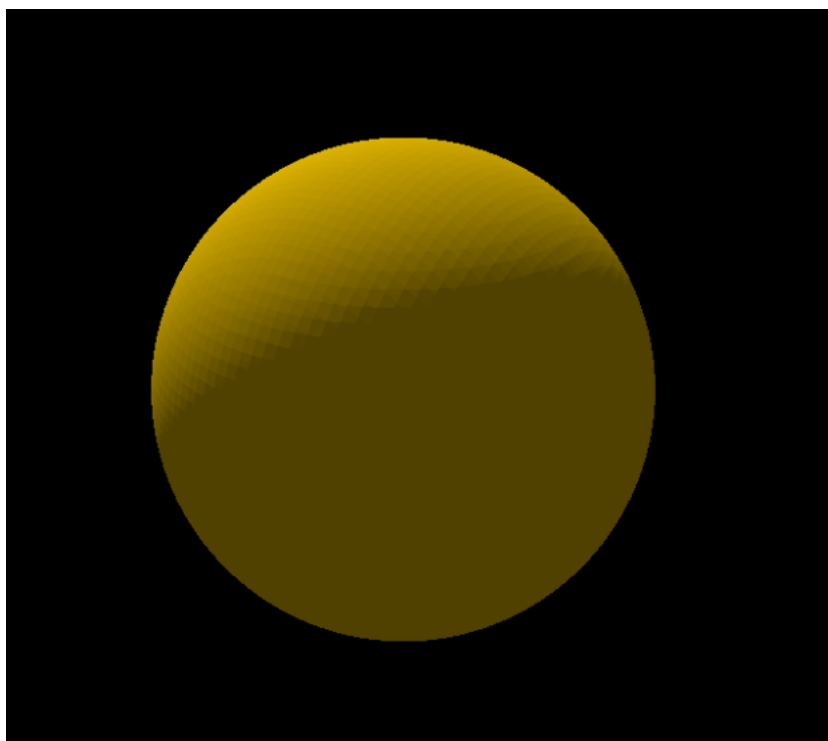


图 15: Resolution=90

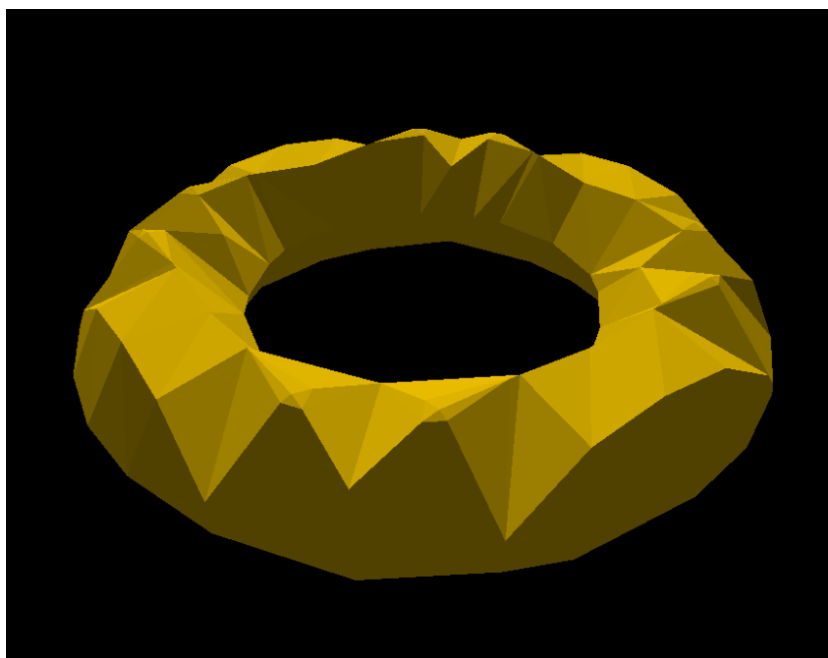


图 16: Resolution=10

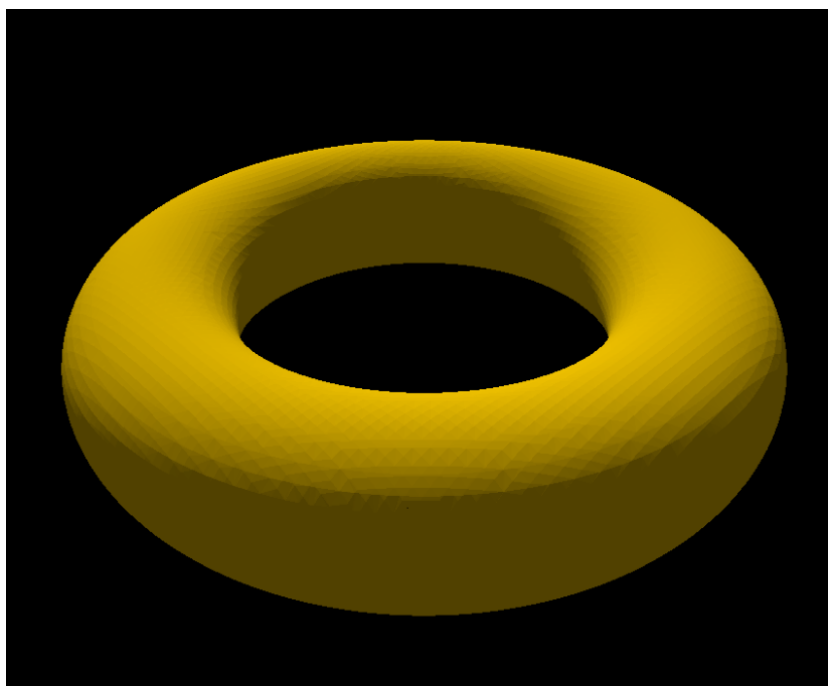


图 17: Resolution=90

### 3 总结

这个 lab 比上一个困难，我觉得知识点和 debug 的难度都高于上一个 lab，而且我有好几个函数里包含的 bug 是因为整型相除转化为浮点数的时候类型转换出了问题，事后发现觉得自己很蠢，说明细节很重要。通过这次 lab，我对这几种算法的理解更深了，不仅仅停留在纸上谈兵的理论，也进行了实践，并且在实践中修正或者加深了自己对它们的理解，能力得到了提升，同时也对框架中给出的数据结构更加熟悉了，知道了图形学有关的数据结构该怎么设计，收获很大。