可视计算与交互概论 Lab2 报告

王蕙钰 2100018733

1. 介绍

本次 Lab 要求实现几何处理部分介绍的几种算法，包括 Loop Mesh Subdivision，Spring-Mass Mesh Parameterization，Mesh Simplification，Mesh Smoothing 和 Marching Cubes 算法。

2. 实现思路和结果

Case 1: Loop Mesh Subdivision

每次迭代都按照 1）将原有顶点添加到新 Mesh，并根据公式一重新计算它们的位置， 2）添加新的顶点，根据公式二求得它的位置 3）维护 indices，生成新的面 进行。

2. 更新原有顶点：对于每个原有顶点 $\mathbf{v}$，根据以下公式更新后的位置 $\mathbf{v}'$

$$\mathbf{v}' = (1 - n * u)\mathbf{v} + \sum_{i=1}^{n} u\mathbf{v}_i$$

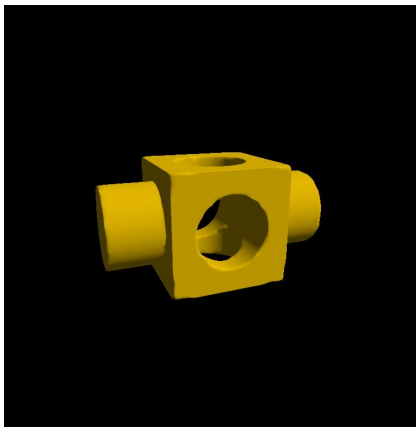
公式一

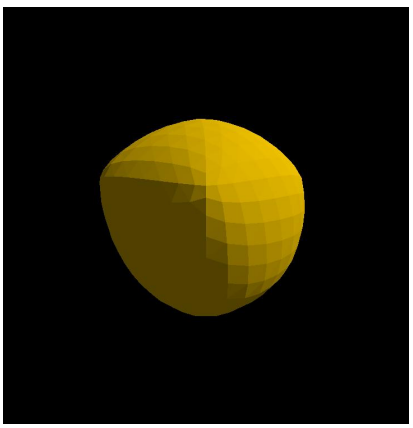
1. 计算新顶点：
   - 对于每一条边，如果这条边被两个三角形面所包含，如8.11左图所示，则由这条边的两个端点 $\mathbf{v}_0, \mathbf{v}_2$ 和“跨过”这条边的两个顶点 $\mathbf{v}_1, \mathbf{v}_3$ 加权平均计算出新顶点 ep，具体的计算公式为
   $$ep = \frac{3}{8}(\mathbf{v}_0 + \mathbf{v}_2) + \frac{1}{8}(\mathbf{v}_1 + \mathbf{v}_3)$$
   - 如果这条边只被一个三角形面所包含（即为边界上的边），则直接取这条边的中点作为新顶点.

公式二

添加点部分按照公式计算即可，连点成面部分特别注意点的顺序要保持一致。

效果：





左图：block（iterations=3）

右图：cube（iterations=3）

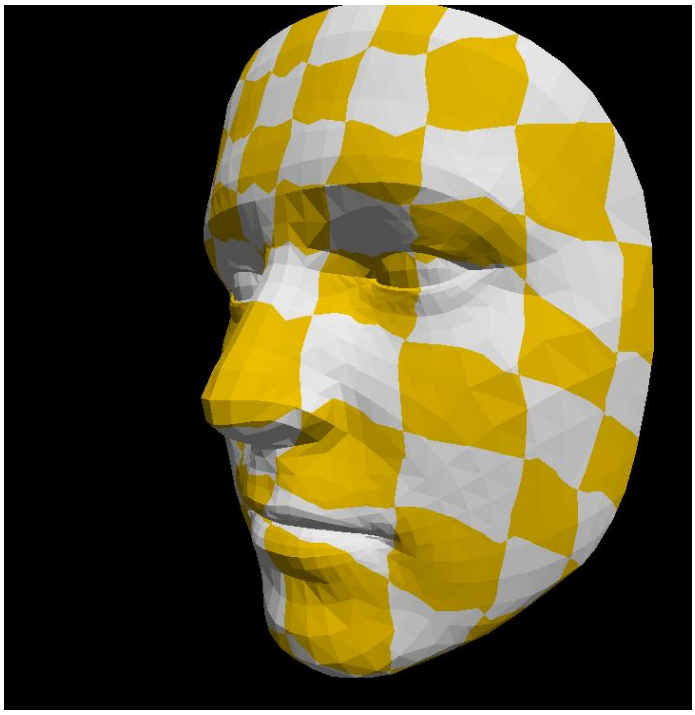
Case 2: Spring-Mass Mesh Parameterization

1. 首先初始化边界点上的 uv 坐标。先找到一个边界点，接着以此为起点通过

v->BoundaryNeighbors()绕一周找到所有边界点。初始化我选择的是以（0.5，0.5）为圆心的圆，满足 uv 在 0 到 1 之间。

2. 通过 Gauss-Seidel 迭代法求解中间的点的 uv。

效果：



face（iterations=1000）

Case 3: Mesh Simplification

1. UpdateQ 用于计算每个面的 Kp 矩阵

计算公式如下：ax+by+cz+d = 0 (a^2 + b^2 + c^2 = 1)

$$\mathbf{K_p} = \mathbf{pp}^\mathsf{T} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

2. Makepair 用于计算每条边的最优坍塌点和代价

最优坍塌点的计算方法如下：

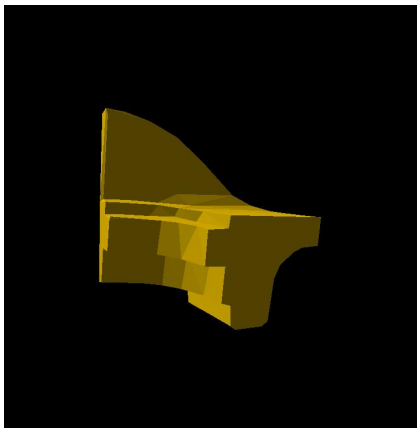
$$\bar{\mathbf{v}} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{23} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

特别注意要单独讨论矩阵不可逆的情况。
代价 cost=v^T Q v

3. 初始化设置：每个点的 Q 矩阵为相邻面的 Kp 矩阵之和

每次迭代中，删除代价最小的边，并更新 v1 坐标和 v1 变动带来的面的 Kp，有关点的 Q，和有关 pairs。

效果：



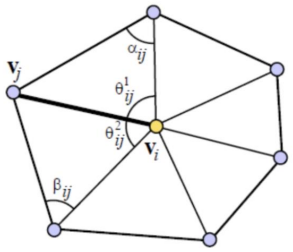

左图：fandisk（simplify=4）
右图：dinosaur（simplify=2）

Case 4: Mesh Smoothing
根据教程：

1. 对每个顶点 `vi`，计算相邻顶点位置的加权平均：

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

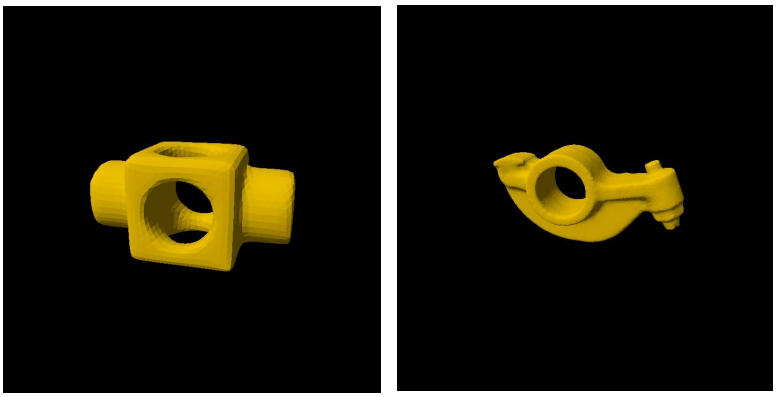
2. 其中使用 Uniform Laplacian 时相邻顶点权重均为 1，使用 Cotangent Laplacian 时采用余切权重 `wij = cot αij + cot βij`，如图：



3. 更新顶点：`vi = (1 − λ) * vi + λ * vi'`；

其中为了实现 Cotangent Laplacian 需要得到四个顶点以计算余切值，因此建立了一个索引，可以以 Vi，Vj 两点得到之间的边从而得到四个点的坐标。特别注意 w 太大时有洞，需要限制大小。

效果：



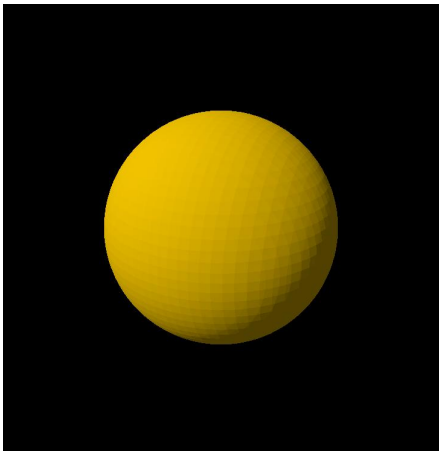
左图：block（iteration = 5，smoothness = 0.5, cotangent）
右图：rocker（iteration = 5，smoothness = 0.5, uniform）

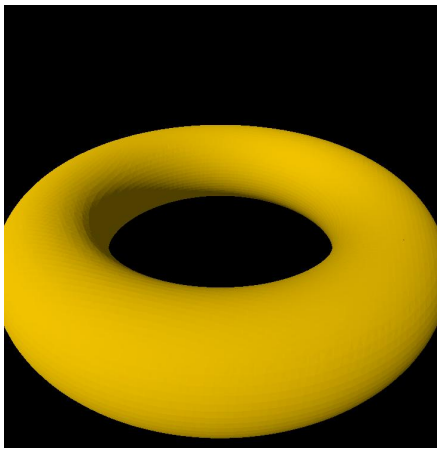
Case 5: Marching Cubes

1. 遍历每个 cube，从 v0 开始逐个顶点判断正负，用一个一个 8 位的二进制数记录顶点情况，再通过查表生成一个 12 位的二进制数表示这个 cube12 条边上是否有 mesh 顶点。
2. 对于该 cube，遍历每一条边，如果有 mesh 顶点则线性差值求坐标并添加至 output。

特别需要注意的是由于一条边可能为多个 cube 共有，为了防止重复计算需要建立一个索引，方便查找现有的 mesh 顶点中是否已经包含了这个点。为了制作 map 我又自定义了一个 struct，并重载了<运算符。

效果：





左图：sphere（resolution = 55）
右图：torus（resolution = 100）