homework 9

基于二次误差的边坍塌算法

这个算法来自于1997年Michael Garland的文章：Surface simplification using quadric error metrics。虽然年代也比较久了，但是是最经典的简化算法之一，也是我个人使用最多的简化算法。

二次误差的边坍塌算法背后的思想实际上是很简单的。整个算法不断迭代的是两个步骤：

1. 寻找适合的边（误差尽量小）

2. 对该边进行坍塌，文章中称为contraction操作

但是一般来说，去掉一些边可能会引起三角面片的flip，因此改进的版本一般会检查是否会发生flip。

二次误差的定义

二次误差定义是非常巧妙的。对于这个误差定义，要满足的一个基本条件就是：对于原始的模型来说，误差为0，一个很合理的定义就是点到平面的距离。在三角网格中，每个顶点实际上是多个平面的交点，假设顶点v，而它对应的某个平面为$\mathbf{p}$，其中：$\mathbf{p}=\{a,b,c,d\}$，满足：

$$\mathbf{v}^\top\mathbf{p}=\{a,b,c\}\cdot v+d=0$$

而一个点到该平面的距离的平方，可以写成：

$$\begin{aligned}|d|^2&=(\mathbf{v}^\top\mathbf{p})(\mathbf{v}^\top\mathbf{p})\\&=\mathbf{v}^\top\mathbf{p}\mathbf{p}^\top\mathbf{v}\end{aligned}$$

因此每一个平面可以使用矩阵$\mathbf{p}\mathbf{p}^T$来计算误差，这样另外一个好处就是它是可以直接相加的：

$$\begin{aligned}\Delta(\mathbf{v})&=\sum_{\mathbf{p}\in\text{ planes }(\mathbf{v})}\left(\mathbf{v}^\top\mathbf{p}\right)\left(\mathbf{p}^\top\mathbf{v}\right)\\&=\sum_{\mathbf{p}\in\text{ (planes }(\mathbf{v})}\mathbf{v}^\top\left(\mathbf{p}\mathbf{p}^\top\right)\mathbf{v}\\&=\mathbf{v}^\top\left(\sum_{\mathbf{p}\in\text{planes }(\mathbf{v})}\mathbf{K}_\text{p}\right)\mathbf{v}\end{aligned}$$

我们成这个相加之后得到的矩阵为每个顶点的Q矩阵。

边坍塌

到目前位置我们定义的是顶点的Q矩阵，可以轻易计算出新的顶点的误差。在本篇论文提出的方法中，简化具体操作是边坍塌。当一个边的两个顶点收缩到一个顶点时，那么这个新的顶点应该会有两个误差，也就用原来两个顶点的Q矩阵求，当然，这两个Q矩阵也是直接可以相加来求的，也就是每条边的Q矩阵为：$\mathbf{Q}=Q_1+Q_2$.

那么如何求得新的顶点？也就是要最小化误差：

$$\mathbf{v}=\arg\min_{\mathbf{v}}\mathbf{v}^\top\mathbf{Q}\mathbf{v}$$

由于误差是二次误差，所以这个问题非常简单，也就是求解下面的方程：

$$\mathbf{Q}\mathbf{v}=0$$

当然直接求解是会出问题的，因为得到$\mathbf{v}$是一个0向量，而它实际上是个齐次坐标，齐次坐标最后一维如果为0是没有意义的。既然我们只希望优化的是其中的x，y，z部分。如果将上式展开：

$$\begin{aligned}\mathbf{v}^\top \mathbf{Q}\mathbf{v} = &q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 \\ &+ 2q_{23}yz + 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44}\end{aligned}$$

对各个分量分别求导使得为0，等价为求解：

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

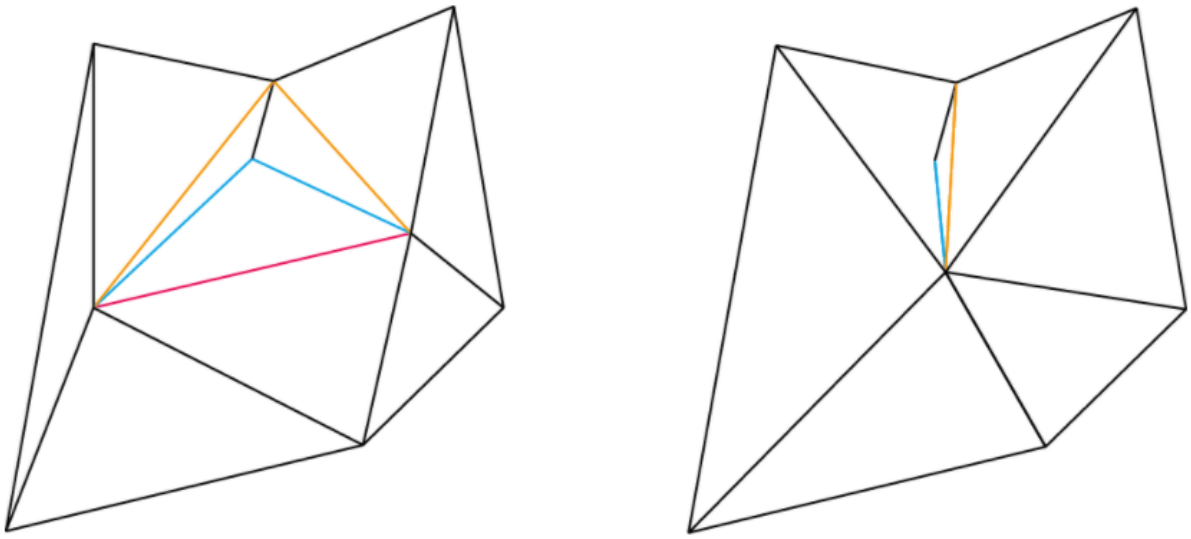
这样我们优化的就只是顶点坐标了。

如果上述方程可解，那么可以直接得到新的坐标值，如果不可解，可以选择两个端点或者中点。

判断Flip

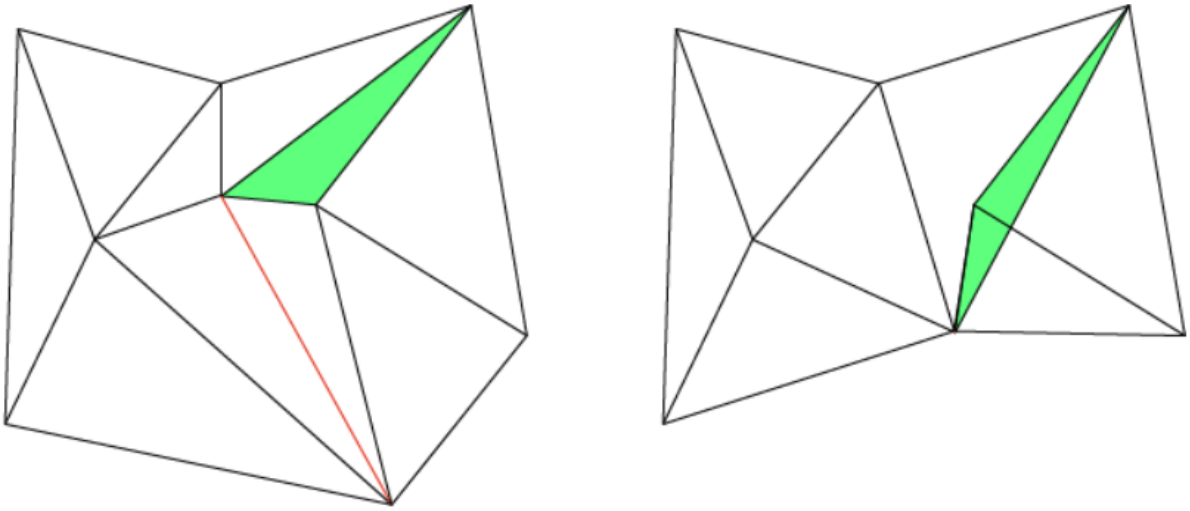
在实现过程中，发现会有两种flip出现：

- 一种是网格的边坍塌会造成网格的连接关系发生变化，或者说拓扑结构发生了变化，如下图：



对于这种flip的判断，主要是判断即将收缩的边的两个顶点点延申出来的边相交了几次。上图中，这两个顶点延申的边一共有三次相交。一般来说相交次数为2次或者1次（当这个边为border时候）。通过这样的检查，可以排除掉这样的flip。

- 对于第二种，主要是网格的面的朝向（法向量的角度）发生了比较大的翻转，如下图：



这样的反转检查起来很简单，就是判断连接的三角形坍塌前与坍塌后的法向对比，如果法向量的角度大于某个值，就舍弃这条边的收缩。

记录回退

对回退操作的记录，一个可行的做法是维护一个操作栈（Stack）。栈的特性是先进后出，后进先出，这样保证了回退的时候首先退回的是最新的操作。另外，边坍塌的对偶操作，就是点的分裂成边（Split）。在回退时，只要记录了原先两个点的位置，分裂时候只需找到顶点，并且维护面以及边的关系即可。但是这里需要注意的一点，为了维护回退这个功能，我们不能将之前的点完全删除。比如发生了3次边坍塌操作分别是：$[v1, v2] \rightarrow v2; [v3, v4] \rightarrow v3; [v1, v3] \rightarrow v1;$

经过这几次坍塌之后，目前网格离$v1$，$v2$，$v3$，$v4$只剩下$v1$。如果我们将网格重新compact之后，任何$v1$，$v2$，$v3$的信息都删去了，那么我们即使可以从$v1$回退到边$[v1, v3']$，此时的$v3'$是一个新分配的顶点，接下来回退$[v3] \rightarrow [v3, v4]$时候，我们是无法通过存在栈中的$v3$来找到$v3'$的。因此为了回退，我们需要轻量维护之前已删去顶点的一些信息，比如他们的位置，或者ID应当保留。当进行回退时候，可以再次找到需要分裂的顶点。

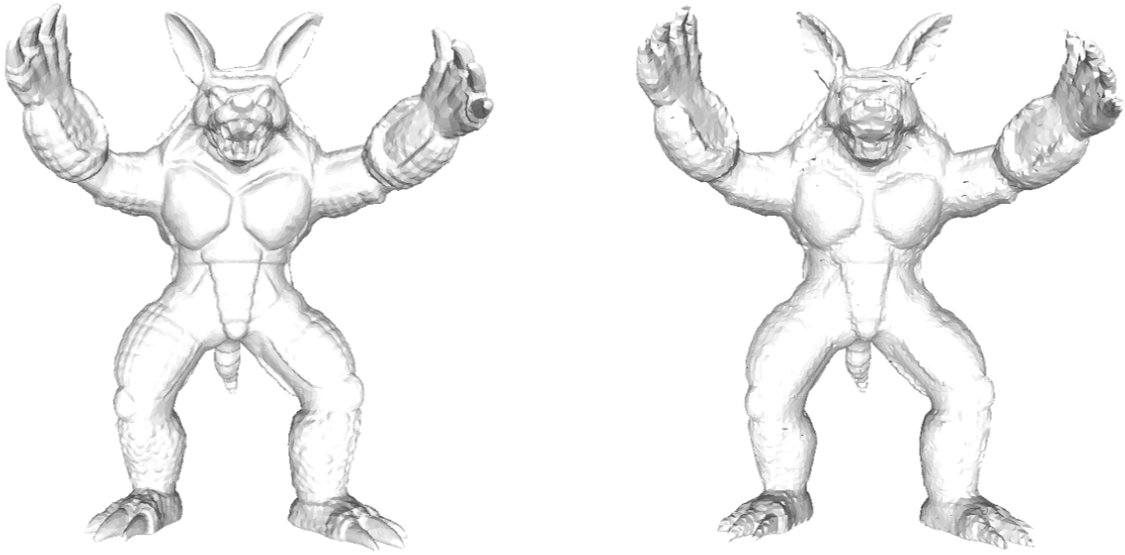
具体实现

本次作业在之前写的HalfEdge基础上实现了基于二次方误差的边坍塌算法。需要注意的点有下面几个：

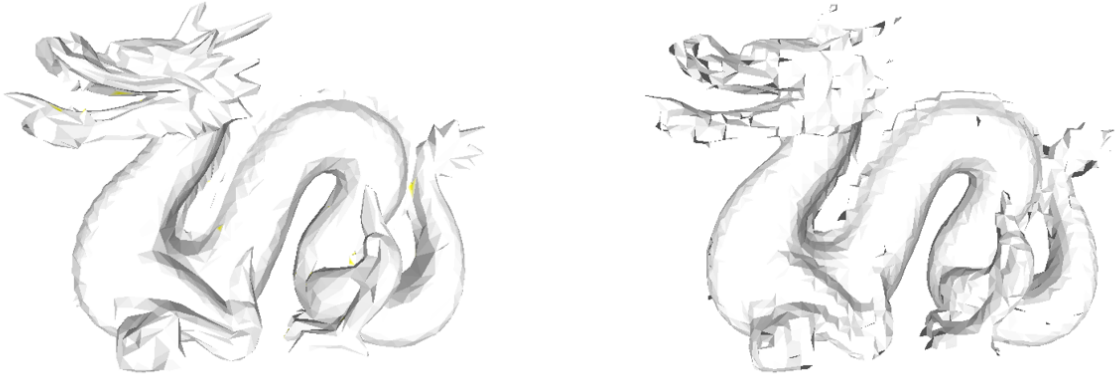
- 并不是所有的mesh都能放入到HalfEdge中，比如一些非流形等，使得每个边可能会有3个临接三角形。因此我调整了HalfEdge，使得它在读取时候会舍弃掉超过两个三角形的情况。另外还有一些单独的点，在放入HalfEdge中会自动被删除掉。
- 为了提高效率，避免每次简化一个边后都需要重新整理mesh，我对于每个三角形维护一个脏位（dirty flag）。当一个边坍塌时，该边的两个顶点相接的三角形脏位都置1，这样如果需要删除的边连接到这些三角形了，就会直接跳过，因为它们的顶点还有Q矩阵没有更新，因此计算的最新顶点可能不再是最优位置了。
- 在边坍塌算法中，我维护了一个优先队列。同时还有一个error threshold，当队列中弹出来的边的error大于error threshold时，就对mesh进行一个compact，也就是重新计算。那么为什么还要使用优先队列呢？直接将教育error threshold的边全部进行坍塌就好了？实际上在运行时候还会进行一个是否达到目标值的判断，也就是算法是随时可能退出的，不一定能达到error threshold。使用优先队列保证了不到达error threshold前收缩的边error是有小到大的。
- 对于HalfEdge坍塌之后，边，面的关系的维护需要一些耐心。

本次作业也实现了聚类的简化算法，相比于二次误差的简化算法来说，聚类算法更简单，因此也更快，相应的它的效果也更差。实现聚类的简化算法时，只要将落入同一个voxel的点聚合为一个点即可，这时候维护一个HalfEdge反而是一个效率比较低的操作，使用空间的VoxelHash是一个更高效的做法。

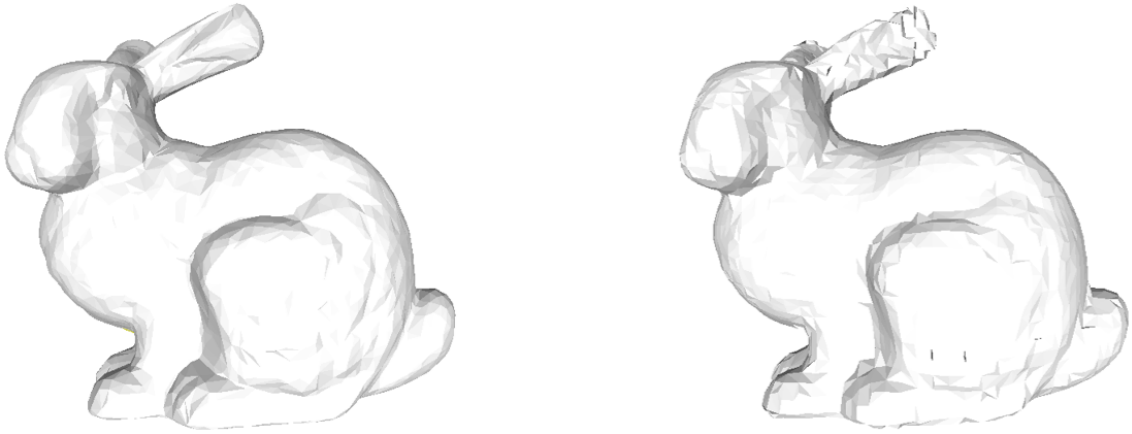
下面是简化效果：



Armadillo, Quadrics (*left, 34594 triangles*) and Clustering (*right, 31656, triangles*) Decimation.



Dragon, Quadrics (*left, 8714 triangles*) and Clustering (*right, 8592 triangles*) Decimation.



Bunny Quadrics (*left, 6082 triangles*) and Clustering (*right, 6083 triangles*) Decimation.

可以看到，简化数量级一致的情况下，二次方简化的效果更好，它更好地保留住了几何的细节特征，而且可以避免flip，而体素聚类的简化算法会造成flip，简化效果也更差。但是另一方面来说，它的速度会更快。

代码的实现在 https://github.com/MyEvolution/Dragon/blob/main/src/Geometry/TriangleMesh/Processing/Decimation.h，更多的内容可以查看视频。

谢谢老师与助教。