网格简化期末报告

王怡力
2016012642
wangyili16@mails.tsinghua.edu.cn

1 摘要

网格简化算法处理的对象是用来对 3d 物体建模的网格数据结构，网格简化算法的目的是将原来的网格 $\mathcal{M} = (\mathcal{V}, \mathcal{F})$ 转化为具有比原来网格结构更少边和点的新网格 $\mathcal{M}' = (\mathcal{V}', \mathcal{F}')$，且满足如下两个条件：

1. 给定 $\|M - M'\| < \epsilon$, $\min \|V'\|$
2. 给定 $\|V'\|$, $\min \|M - M'\|$

我在这次实验中，主要实现了以下几点：

1. 用 C++ 实现了基于边坍缩的网格简化算法。
2. 实现了初始化 pair 的 kd-tree 加速。
3. 实现了基于边坍缩算法的拓扑保持，边界保持。

2 算法

2.1 基于边坍缩的网格简化算法

边坍缩类型算法的基本思想就是，通过一**定的标准**，选择一条边，把它的两个端点变成一个新的点，更新与它相关的拓扑结构，从而实现网格的减面和减点操作。我实现了由Garland and Heckbert [1997] 提出的基于二次误差的边坍缩网格简化算法。简单来说，该算法**选边的标准**就是二次误差。

- 算法描述

  首先，算法缩小了可选坍缩边的范围，规定了一个特殊的 “pair” 结构，给定 pair (p, q), 满足：(p, q) 是一条边，或 $\|p - q\| < \epsilon$, ϵ 由用户给定。

  接下来就是根据二次误差去选边，数学上，一个三维空间中的平面可以被一个四维向量 $\mathbf{p} = [a, b, c, d]^T$ 表示，$[a, b, c]$ 为该平面法向量 $\mathbf{n}$, $d = -(\mathbf{n} \cdot \mathbf{v}), v \in \mathbf{p}$。

二次误差具体而言就是，对于一个平面集, 任意一点 $\mathbf{v}$ 到平面集 $\mathbf{p}_1, .., \mathbf{p}_N$ 的距离平方和：

$$\Delta(\mathbf{v}) = \sum_{i=1}^{N} \hat{\mathbf{v}}^T (\mathbf{p}_i \mathbf{p}_i^T) \hat{\mathbf{v}} = \hat{\mathbf{v}}^T (\sum_{i=1}^{N} K_i) \hat{\mathbf{v}}$$

$$\Delta(\mathbf{v}) = \hat{\mathbf{v}}^T \mathbf{Q} \hat{\mathbf{v}},\ where\ \hat{\mathbf{v}} = [v_x, v_y, v_z, 1]$$

如果我们把网格的端点 $\mathbf{v}$ 理解为一个平面集 $planes(\mathbf{v})$ 的交，那么一个 pair$(\mathbf{v}_1, \mathbf{v}_2)$ 的二次误差为：

$$\Delta(v) = \mathbf{v}^T (\mathbf{Q}_{\mathbf{v}_1} + \mathbf{Q}_{\mathbf{v}_2}) \mathbf{v}$$

选取使得二次误差最小的收缩点 $\mathbf{v}^*$ 作为最优收缩点，这个 pair 的收缩代价就可以确定为 $\Delta(\mathbf{v}^*)$

最后，我们选出收缩代价最小的 pair 进行合并，更新网格。

该过程不断重复直到网格满足用户的简化标准。

边界保持: 在原 paper 里，作者通过初始化 pair 的时候判断是否在边界上，若在边界上，则将 boundary point 加一个垂直的平面，这样，坍缩点自然会被吸引到边界点的附近，pair 的 delta(v) 也会相应的大很多。而我在本次实验中是直接 hack 的，用了一个比较 naive 的方法，具体的说，我在 pair 合并时而不是初始化 pair 的时候判断合并点是否在边界上：若都在边界上，不进行合并，若有一点在边界上，则合并的新点重新设置为边界点再合并，如果二者均不在边界上, 则进行常规合并。判断是否在边界的方法: 统计点的邻面与该点关联的边的数量, 若存在一边只出现了一次，则该点在边界上。

拓扑保持：更新一个新点后，检查其邻面原法向量和新法向量的夹角的 cosine 值，若小于零，则不进行合并，撤销更新操作。

- 数据结构

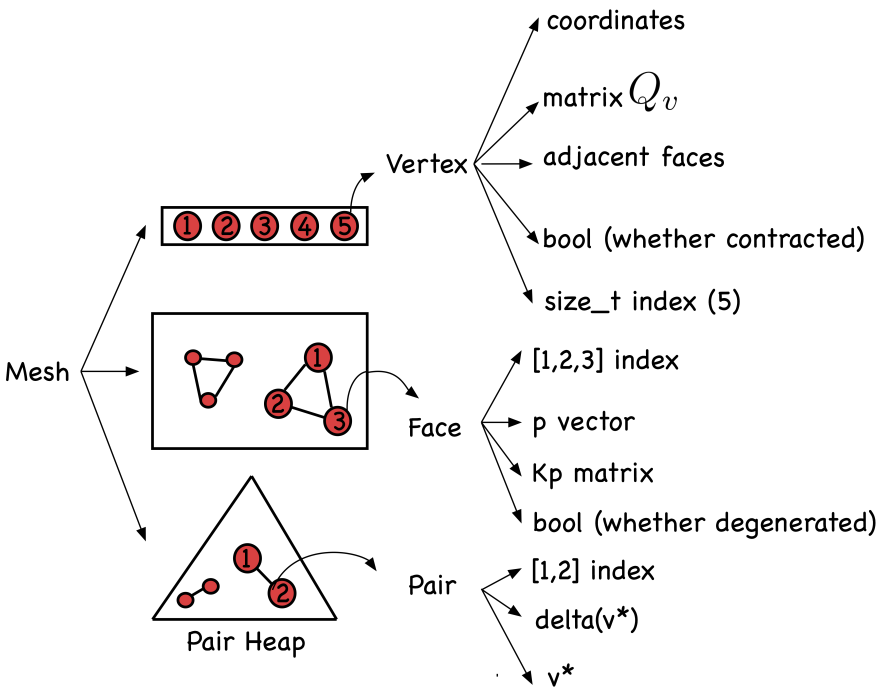


Figure 1: Mesh

- 算法实现

  1. 初始化网格
     - 读入 obj 文件，初始化 vertex 的坐标成员，flag 设为 true，记录其在 Vertices 里所在的位置信息。初始化 face 的 index 成员，flag 设为 true。
     - 进一步初始化 face，通过检索 Vertices，计算每一个 face 对应的向量 $\mathbf{p}$ 表示和 $\mathbf{K_p}$ 矩阵。
     - 进一步初始化 vertex，通过检索 face 列表，建立 adjacent face 的索引，对所有 adjacent face 进行矩阵求和得到 Q_v。
     - 初始化 pair 最小堆。**在这里我实现了 Kd-tree 加速**。首先建立所有 vertex 的 Kd-tree，然后通过邻近搜索找到小于 threshold 的所有 Vertex 结点；再遍历所有的 face，找到所有的边。最后得到所有 pair，初始化 index，$\Delta(v^*)$, v^*，依次入堆。这里通过 set 避免重复入堆操作。
  2. 选边，更新
     - 选择 $\Delta(v^*)$ 最小的 pair，**处理边界情况**，如果该 pair 的 point 未被收缩，则继续操作，得到该 pair 的最优收缩点坐标.
     - 找到 $planes(\mathbf{v}_1), planes(\mathbf{v}_2)$（neighbor faces）以及与其相关的所有点 (neightbor points)。
     - init 一个新的点，assign 最优收缩点坐标，加入 Vertices 里。

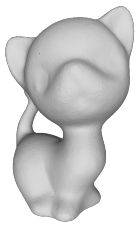
- 对所有 neighbor face 进行更新。如果一个 face 有两个点都是收缩点，则 flag 设置为 false，如果有一个点是收缩点，则将该 face 的 vertex index 修改收缩点 index 为更新点 index。重新计算其 $\mathbf{p}$ 和 $\mathbf{K_p}$。
- 对所有 neighbor points 进行更新，得到新的 adjacent face，$\mathbf{Q_v}$。
- 对所有相关 pair 进行更新，对所有含有 neighbor point 的边，建立新的 pair，压入堆中。

3. 重复操作 2，直到现有 face 的数量和原 face 数量之比小于规定的比例。

3 结果和比较

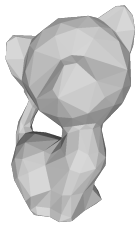
3.1 简化比例

(a) Kitten



(b) ratio = 0.5

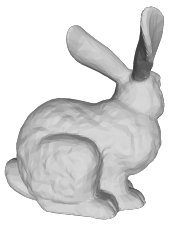
(a) ratio = 0.1



(b) ratio = 0.01

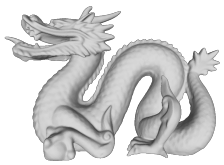
(a) Bunny

(b) ratio = 0.5



(a) ratio = 0.1

(b) ratio = 0.01



(a) Dragon

(b) ratio = 0.5

3.2 简化时间

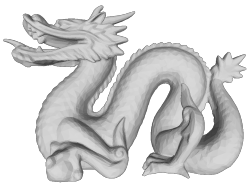
threshold = 0.01		**ratio = 0.5**	**ratio = 0.1**	**ratio = 0.01**
kitten	初始化时间	0.236586s		
	简化时间	1.506360s	4.465262s	5.735434s
bunny	初始化时间	7.826981s		
	简化时间	2.268373s	11.725651s	29.085398s
dragon	初始化时间	20.294218s		
	简化时间	6.753470s	26.143818s	34.805448s

3.3 K-d 树加速比较

初始化时间	**加速前**	**加速后**
kitten	2.412293s	0.236586s
bunny	14.478038s	7.826981s
dragon	52.495232s	20.294218s

4 心得体会

本次作业一共做了四天，从刚开始对算法的一知半解，到设计出 naive 的数据结构，实现了基本的功能并且能够熟练调用 C++ 的 STL 库和 opencv 库，最后随着功能的增加和思路逐渐清晰对数据结构进行完善。大大小小的 bug 被我一个个 de 完之后，总算得到了不错的简化效果。写报告的过程中，我再一次回顾了整个算法的思路，整理了自



(a) ratio = 0.1

(b) ratio = 0.01

已实现的框架。总体上，这次作业提高我的 C++ 码力和写 code 的耐心，感谢助教和老师这学期的悉心指导！

References

Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7. doi: 10.1145/258734.258849. URL `https://doi.org/10.1145/258734.258849`.