网格简化实验报告

计 24 黄晨 2012011337

目录

---

1 程序描述

本程序实现了网格简化，采用边坍塌的方式，基于二次误差的度量实现了网格简化。采用堆、红黑树（set）和 map 进行了优化，使初始化网格与简化更加迅速。

使用说明：通过命令行传递参数，第一个参数为输入文件名，第二个参数为输出文件名，第三个参数为简化率，第四个为可选参数，为 t 值，默认为 0.01

2 程序类介绍

2.1 Vec3f 类

2.1.1 表示坐标、颜色的类

2.1.2 提供了计算 2 阶距的函数和单位化的函数

2.1.3 重载了[]运算符以方便使用

2.2 SimpleObject 类

2.2.1 简单 obj 类，进行 obj 文件的读取（仅读取顶点 v 和面片 f）

2.2.2 顶点坐标以 Vec3f 类存储，面片信息用三维数组表示

2.3 Matrix 类

2.3.1 矩阵类，用以存储矩阵

2.3.2 重载了加、减和乘运算符以实现矩阵运算

2.3.3 提供了 transposition()方法实现矩阵的转置，方便运算

2.3.4 重载了[]操作符以方便使用

2.4 EdgeCollapse 类

2.4.1 边坍塌算法的实现类，继承自 SimpleObject，方便读取 obj 文件

2.4.2 成员

2.4.2.1 heap：堆，用以存储各 pair 的二次误差值和 pair 的点的信息

2.4.2.2 order：map，记录各 pair 在堆中的位置，方便修改和删除

2.4.2.3 v_Q：各点的 Q 方阵数组

2.4.2.4 pairs：与各点组成 pair 的点的列表，其中对每个点使用 set 结构进行存储

2.4.2.5 faces：各点所属面片的列表，对每个点使用 set 结构存储面片编号

2.4.2.6 deletedPoints, deletedFaces：算法过程中被删除的点和面片的列表

2.4.2.7 projection：点编号的映射，用于最后将结果进行转换时使用

2.4.2.8 initialize()：初始化方法，主要功能为获取 pair 信息，计算各点的 Q 方阵，建立最小堆，更新 pairs、faces 和 order 的信息

2.4.2.9 belong(…)：判断某点是否属于某个面

2.4.2.10 simplify()：网格简化的外部接口，其中会调用_simplify()进行简化的步骤

2.4.2.11 其余方法属于辅助函数，在此不在赘述其功能

3 程序算法描述

3.1 初始化

3.1.1 定义 2 各 set：v_edge_pairs, v_noedge_pairs，分别表示有边相连的 pair 集合和无边相连（距离小于 t）的 pair 的集合

3.1.2 遍历所有的面，对每个面 f，计算 f 的 K 方阵，将属于 f 的每个顶点的 Q 方阵加上 K。同时，将该面中由边连接产生的 pair 加入 v_edge_pairs，加入时注意使第一个顶点的编号小于第二个顶点以方便后续计算，更新 faces 列表

3.1.3 将所有顶点按照其到原点的距离从小到大排序，距离相等按点编号从小到大排序，注意在排序时记录顶点编号，得到顶点列表 v_sort

3.1.4 依下标从小到大遍历 v_sort 中的所有顶点，对每个顶点 v_i，依下标从小到大遍历从 i+1 开始的所有顶点，对满足 $|v_k-v_i|<t$ 的顶点，若 (v_i,v_k) 不在 v_edge_pairs 中，则将其加入 v_noedge_pairs。当遍历结束，或 $|v_k|-|v_i|>t$ 时停止遍历。

3.1.5 遍历 v_edge_pairs 和 v_noedge_pairs 中的每一个点对，计算点对的二次误差值，将(delta, v_1, v_2)加入堆中，更新 v_1 和 v_2 在 pairs 中的信息

3.1.6 调用 stl 的标准函数将 heap 建成最小堆

3.1.7 遍历 heap 中的每一个元素，更新 map 中各点对在 heap 中的位置

3.2 网格简化

3.2.1 若已删除面片数达到目标值，返回

3.2.2 取出堆顶元素(delta, v_1, v_2)

3.2.3 将 v_2 标记为已删除点，v_1 将在之后更新为坍塌后的点。本程序为简化计算，直接去中点为坍塌后的点。

3.2.4 将 v_1 的 Q 矩阵清零

3.2.5 建立 set：changePoints，用以存储此次简化中矩阵发生变化的点

3.2.6 遍历所有与 v_1 相关联的面

3.2.6.1 若其不包含 v_2，则将其包含的不同于 v_1 的点加入 changePoints，更新其 Q 矩阵。这里采用先剪去原来的面的 K 矩阵，再加上新的面的 K 矩阵的方式更新。同时更新 v_1 的 Q 矩阵。

3.2.6.2 若其包含 v_2，则将该面加入 deletedFaces，更新另一不同于 v_1 和 v_2 的点的 Q 矩阵（减去原来该面的 K 矩阵即可），将该面从 v_2 的 faces 中去除

3.2.7 遍历所有与 v_2 相关联的面，类似 3.2.5 中的更新步骤更新各关联顶点，其后将面列表中 v_2 的标号替换为 v_1 的标号，并更新 v_1 的 Q 矩阵

3.2.8 将所有与 v_2 关联的面加入到与 v_1 相关面的列表中
3.2.9 在 pairs 中删除点对(v_1, v_2)（这里注意需要更新两个点的信息）
3.2.10 根据 order 和 pairs 的信息在堆中更新 changePoints 中各点相关联的 pair 的二次误差，并对堆进行维护，使其保持堆序性。注意这里需要将与 v_2 相关的 pair 替换为与 v_1 相关，若堆中本身就有该点与 v_1 构成的 pair，则直接删除该点与 v_2 的 pair 即可，否则需要替换
3.2.11 将 pair 中与 v_2 相关的点更新为与 v_1 相关
3.2.12 将与 v_2 相关的点全部加入 v_1 的 pairs 中
3.2.13 更新 v_1 坐标为 v_1 和 v_2 中点
3.2.14 将点 v_2 映射至 v_1（修改 projection 中的值为 v_1 的编号）
3.2.15 回 3.2.1

3.3 结果转换
3.3.1 将所有未删除的顶点加入新的顶点列表，并记录顶点在新顶点列表的编号，更新 projection
3.3.2 对每个未被删除的面片
3.3.2.1 对其每个顶点 v
3.3.2.1.1 若 v 在被删除顶点列表中，则将 v 置为其在 projection 中的值，回 3.3.2.1
3.3.2.1.2 否则，将 v 置为其在 projection 中的值

4 程序亮点
4.1 初始化过程优化
4.1.1 先对顶点排序，然后再进行遍历
4.1.2 在点不是非常密集的情况下，可以有效地加快初始化效率
4.2 简化过程优化，通过牺牲空间提高简化效率
4.2.1 一个 obj 所产生的 pair 非常多，若每次依次遍历将非常耗时，特别在 t 值比较大的情况下
4.2.2 通过 faces 和 pairs 两个列表减少每次遍历的次数，有效减少了每次简化的时间，提高了简化效率
4.2.3 实际测试发现，在不优化的情况下，简化速率约为 6-12 片/s，优化后速率可打 500-1000 片/s，即使在 t 值大于物体尺寸 1/15 时简化效率仍有 100 片/s

5 程序运行结果
5.1 见附件打包的 obj，每个 obj 的明明方式为 name_p.obj，其中 p 表示简化率

6 总结与收获
6.1 在空间允许的条件下，可以通过牺牲空间以提高程序性能
6.2 对于简化结果的现实，可以使用光线跟踪的程序对其进行显示，结合键盘操作即可实现视角变换

7 参考资料
7.1 课件
7.2 C++ Reference