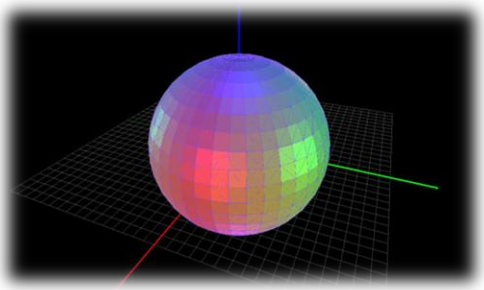
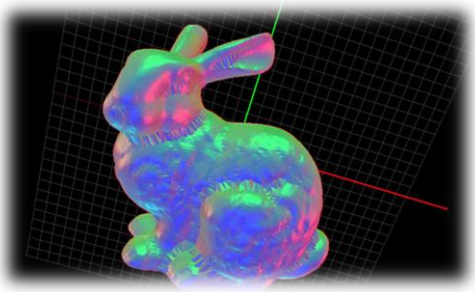
点云构建三角网格模型

——计算几何课程设计

林添 任博 杨光

一、项目背景

由点云构建三角网格模型最初的动机来源于医学方面。在对脏器，骨骼等的3维图像开始得到广泛应用之时，如何从扫描得到的点云数据生成可供进一步渲染使用的三维三角网格模型成为了人们关心的问题。表面模型的恢复，由于对细节有着一定的精度要求，生成三角网格的算法既要保证效率，又要产生能够精细反映点云结构的模型。

随着激光扫描技术的进一步成熟，这一问题的应用开始向更广的领域拓展。如扫描并制作文物古迹的三维数字模型，或是自动化模型设计等方面。

另外，由于计算机视觉的出现，诸如人脸识别，建筑物识别等研究课题的开始深入地研究。随着基于大量图片或视频的数据捕获的技术发展，通常需要在计算机视觉作为前端输入的，通过海量数据驱动地方式自动地从2维图像重构出3维模型。这些自动化的场景恢复，往往利用不同帧之间的匹配点来进行重构，所以由点云构建三角网格模型在实际研究和应用当中具有广泛的价值。

二、文献综述

在今天，有一些较为经典的由点云生成三角网格模型的算法，如 William 的 Marching Cubes 算法[1]，之后 A.Hilton 的 Marching Triangles 算法[2]，这两篇论文最

[1] Marching Cubes: A High Resolution 3D Surface Construction Algorithm.William E. Lorensen, Harway E. Cline

[2] Marching Triangles: Range Image Fusion for Complex Objet Modeling. A.Hilton, A.J.Stoddart et al.

早研究基于点云构造表面模型，它们利用了点集的空间划分，分别构造出立方体模型和三角形网格；而 Fausto 的 Ball-Pivoting 算法[3]等，在研究了以上算法之后，提出了从局部出发，迭代构造的思想。在构造表面网格的过程当中，法向量的计算尤为关键，常常会影响到后面构造的精度乃至准确性，也一些研究也特别关心如何获得表面法向等数据[4]。

三、算法原理

在本次实验设计当中，我们主要采用了 Ball-Pivoting 算法（滚球算法）[5]。选择该算法的主要原因是：

1) 算法较为稳定，速度较快 O(|E|)（|E|为最终的边数），效果较好；
2) 论文依赖的技术较为独立，具有一定的新颖性和创造性；
3) 构造较为直观，便于演示。

算法概述：

滚球算法的核心概念比较直观的。设 M 为某待求的三维物体的表面，S 是对 M 的一个扫描获得的点集。对一般的实际点集 S，通常由三维扫描得到，其实际物体的表面可以视为一封闭的集合，那么则可以合理的认为存在一个合适的半径为 r>0 的球，使得其不可能不接触任何点地穿过 S。滚球算法，则是让该球在其表面滚动，一旦球的表面触及到三个点则产生一个三角形网格，直到结束。
在一开始，我们找到一个这样的球刚好与 S 上的三个点接触的位置。之后，保持求与两个点相接触，而将球“绕两点组成的轴线滚动”，直到它接触到新的点。对所有在当前三角网格边缘的边持续这一过程，则三角形可以在此间不断地生成。

显然，对于足够大的球半径，生成的三角网格将是点集 S 的一个凸包。当球半径逐渐减小时，生成的三角网格不再是凸的，需要引入$\alpha-\text{shape}$的概念。

---

[3] The Ball-Pivoting Algorithm for Surface Reconstruction. Fausto Bernardini, Joshua Mittleman et al.

[4] Estimating Surface Normals in Noisy Point Cloud Data. Niloy J.Mitra , An Nguyen

[5] The ball-pivoting algorithm for surface reconstruction, Bernardini, F. and Mittleman, J. and Rushmeier, H. and Silva, C. and Taubin, G.

设点集 S 维度为 d，则取其大小为 k+1 的子集 T，其中 k=conv(T)是 T 的凸包的维度，一个半径为α的 d 维球 b，若有 T=∂b ∩ S，则所有这样的 conv(T)的集合即构成了 S 的α − shape边界。其中大小为 k+1 的子集 T 构成的边界称为 k-face。

滚球算法与α − shape间有着紧密联系。根据该论文，采用滚球算法生成的表面三角网格正是 S 的α − shape的 2-face 的一个子集。同时，它也是点集的三维 Delaunay 三角剖分的一个 2-skeleton 的子集。在满足以下条件时，所获得的三角网格将与原模型拓扑同构：

1. 任何半径为 r 的球与模型表面的交面与圆拓扑同构。
2. 球心在模型表面上的半径为 r 的球至少容纳一个点。

算法伪代码：

Algorithm $BPA(S, \rho)$

```
1.  while (true)

2.      while (e_(i,j) = get_active_edge(F))
3.          if (σ_k = ball_pivot(e_(i,j)) &&
                    (not_used(σ_k) || on_front(σ_k)))
4.              output_triangle(σ_i, σ_k, σ_j)
5.              join(e_(i,j), σ_k, F)
6.              if (e_(k,i) ∈ F) glue(e_(i,k), e_(k,i), F)
7.              if (e_(j,k) ∈ F) glue(e_(k,j), e_(j,k), F)
8.          else
9.              mark_as_boundary(e_(i,j))

10.     if ((σ_i, σ_j, σ_k) = find_seed_triangle())
11.         output_triangle(σ_i, σ_j, σ_k)
12.         insert_edge(e_(i,j), F)
13.         insert_edge(e_(j,k), F)
14.         insert_edge(e_(k,i), F)
15.     else
16.         return
```

算法数据结构：

在程序中，使用渐进的方法逐个构造三角网格。构造并维护三个列表，即当前的边界环表，当前的活跃边界列表，以及已生成的三角形列表。

活跃边界列表由一系列相互连接的边结构构成，每个边结构存储两断点 A,B 以及已生成的作为其边界的三角形的对顶点 O，半径为 r 的球的球心坐标 C 等信息，并与边界上前后相连的两条边结构以指针相连。

每一条边可以有状态 ACTIVE，BOUNDARY，DELETED。ACTIVE 表示这是一条可被用作滚动轴的边，BOUNDARY 则是对应不可能被使用为滚动轴生成新三角形的边。DELETED 是为维护而使用的状态，表示边结构已不需要再被考虑，可以删除。

由于在滚球算法中，三角网格的拓扑性质可能发生变化，导致边界可能存在多条，因此用边界环表来维护这样的环存在的个数。

算法流程：

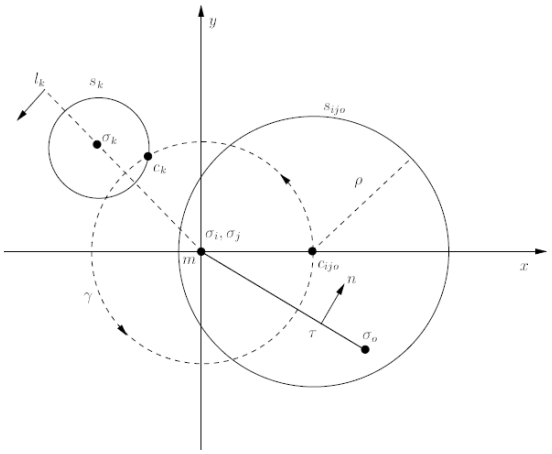
下面顺序介绍算法的流程：

A. 空间搜索

由于对于一般实际模型而言，S 的获得受扫描精度的影响，使 S 的点的空间密度有（模型无关的）固定上界，且多数为空间均匀分布，因此，本算法采用空间网格化的方法进行对点的空间搜索。网格半径采用略大于 2r，由于球半径为 r，则任何一次在网格 V 附近的搜索只涉及 V 临近的最多共 27 个网格。这样在点空间密度有固定上界的情形下，任何一次搜索的时间是 O(1)的。

B. 种子三角形选取

算法需要先选取一个种子三角形，从这一种子三角形出发不停地循环运行下去。在算法中，这一过程首先选取未使用的一个点，而后测试其周围点与其组成的三角形，如果是合法的表面三角形且半径为 r 的球不可穿过，则选定为种子三角形。对于有噪声的数据，通常存在表面外略大于 2r 处的噪声点集，为了减小这样的点集影响以及加速计算，采用如下策略，即对于每个已经有点被使用过的网格，在寻找初始点的时候直接略过。这是因为，由于实际模型的连续性一般跨



越了许多个网格，这样的策略几乎不可能漏过稍大一点的表面。当种子三角形选定之后，即进入了滚球阶段，当 ACTIVE 边列表为空时，则继续尝试找到新的种子三角形，如此循环，直到无法找到新的种子三角形为止，算法结束退出。

C. 滚球阶段

如图，在滚动的球与三角形 $\tau=(\sigma_i,\sigma_j,\sigma_o)$ 的三个顶点接触，其外法向为 n。滚动边界 $e(i,j)$ 与 z 轴重合（垂直纸面向外），其重点 m 在原点处。圆 s_{ijo} 是球与 xy 平面的交，其圆心 c_{ijo} 位于 x 轴上。在滚动过程中，半径为 ρ 的求保持与两点 σ_i,σ_j 相接触，则其圆心将构成圆心在 m，半径为 $\|c_{ijo}-m\|$ 的弧形轨迹 γ .在滚动过程中，球接触到了一个新的点 σ_k。令圆 s_k 为求心在 σ_k，半径为 ρ 的球与 xy 平面的交，则滚动球此时的球心 c_k 位于 γ 与 s_k 的交点上。

在程序中，我们考虑 m 临近 2r 范围内的所有点，取极角最小的 c_k，即为所接触到的第一个点时的球心位置。如果该点是合法点，则继续，否则（如位于已生成的网格内部，法向不符等）则将滚动轴标记为 BOUNDARY。

D. 连接、粘连边操作

在找到新接触到的点后，需要对边列表进行维护与调整，根据不同情况，需要进行边的连接或粘连操作。

边的连接是较为简单的，只需输出新生成的三角形，移除当前的滚动轴，并将新生成的两条边加入列表。

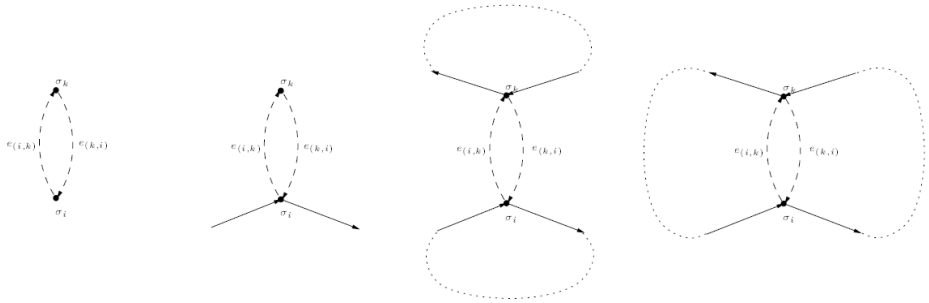
当找到的点位于边界上时，新生成的边有可能与已有的边界上的边有着共同的两端点，而方向不同。这时，由边的粘连操作来移除这一对边。这一操作可能改变已生成网格的拓扑结构，需要视情况进行相应的维护操作。情况有如下四种：

a. 两条边形成一个环。则此环被移除。

b. 两条边相连接。则它们被同时移除。

c. 两条边不相连接，且属于同一个环。它们被同时移除，环分裂为两个。

d. 两条边不相连接，不属于同一个环。它们被同时移除，环合并为一个。



(a) (b) (c) (d)

三、项目实现：

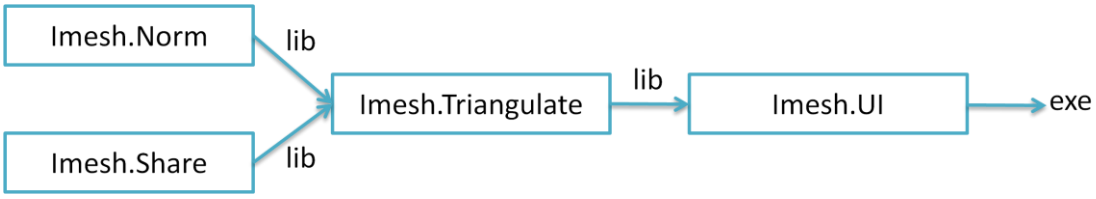
我们通过 svn 进行整个项目管理，将其作为开源项目的形式发布在 google code 上：http://code.google.com/p/imesh-thu/。

包括四个子工程：

IMesh.Share	共享的数据结构和事件机制
IMesh.Norm	算法预处理步骤求法向量，以及模型的读入导出
IMesh.Triangulate	算法核心，主要实现了 Ball-Pivoting 算法
IMesh.UI	演示部分，基于 OpenGL 渲染引擎实现的可视化框架，以及 MFC 窗口程序显示。

以及一个辅助工程：ModGenerator，用于模型的自动生成。

它们互相之间的依赖关系为：



法向部分（IMesh.Norm）：

模型文件的格式和读取

我们使用的模型文件为图形学中常用的保存了模型中顶点的位置信息的.obj文件，每个顶点的位置信息存储格式为文件以“v”开头的一行，除了表示顶点的“v”以外还有三个浮点数用来记录这个顶点的坐标，即“v [点的x坐标] [点的y坐标] [点的z坐标]”的形式。例如，某一个点的坐标是(0,1,2)，则它在文件中对应的一行即为“v 0.000000 1.000000 1.000000”。

在文件的读取部分，我们通过扫描每一行开头字母的方式识别出文件中储存顶点信息的部分，然后再读取顶点的坐标，按照这些信息建立一个点的对象存入数组。这样我们就得到了一个仅含有顶点的位置信息而不含有三角面片信息的点云模型。

我们提供的模型有：

1) bunny-full.obj: 兔子模型，共有34835个点
2) bunny.obj: 兔子模型，共有5295个点
3) horse.obj：马模型，共有5014个点
4) frog.obj：青蛙模型，共有5429个点
5) cube5.obj：立方体模型，共152个点
6) cube10.obj：立方体模型，共602个点
7) cube20.obj：立方体模型，共2402个点
8) ball10.obj：球模型，共762个点
9) cylinder8.obj：圆柱体模型，共994个点

法向量的计算

首先需要对模型进行的一项预处理就是算出每个顶点处的法向量，并且适当地调整法向量的正负符号使得所有法向量都指向模型的外侧。

每个顶点的法向量方向的计算是通过对该顶点的 k 个（k 可以取常数或者和模型点数有关的数，这里我们取 k=15）最近的邻居进行曲面拟合，并取拟合得到的曲面在该点处的法向量作为这一点的法向量的估计值[6]。

求出每个点的法向量之后，还需要对法向量的方向进行调整，使得所有法向量都指向模型的外侧。因为在我们的算法中法向量本身的精确度并不是很重要，重要的是通过法向量的方向来区分模型的内侧和外侧。这一部分我们采用了[7]中所介绍的方法，认为每个点和它的最近邻的法向量方向差别不大，所以它们的法向量内积应该是大于 0 的。这样我们就可以通过判断一个点的法向量和它的最近邻的法向量方向内积是否大于 0 来决定是否应当调整它的法向量的方向。具体实现的方法就是先找到图形的最低最左点（lowest-then-leftmost node），容易看出这一点的法向量应当指向 z 坐标为负的方向。这一点的法向量方向调整好以后，我们就可以对所有的点建一棵最小生成树（Minimum Spanning Tree，MST），以最低最左点为根深度优先地遍历整棵树，并根据每个顶点在树中的父节点调整其法向量的方向。通过这样的调整，我们就可以得到在全局意义上一致的法向量。建立最小生成树的开销为……

当然，这样的方法并非完美。例如，当模型有非常尖锐的部分时，法向量的方向往往可能会出现较大的误差；还有如果模型有薄片型的结构，薄片上一个点和薄片另一面的点在最小生成树中相邻时，按照我们的算法会把他们的法向量调整到相同的方向。（实际上薄片两面的点的法向量应该是相反的！）不过幸运的是这些问题出现的条件都比较极端。我们的算法假设模型的点是比较均匀和稠密的，所以不会出现上述两种情况。

---

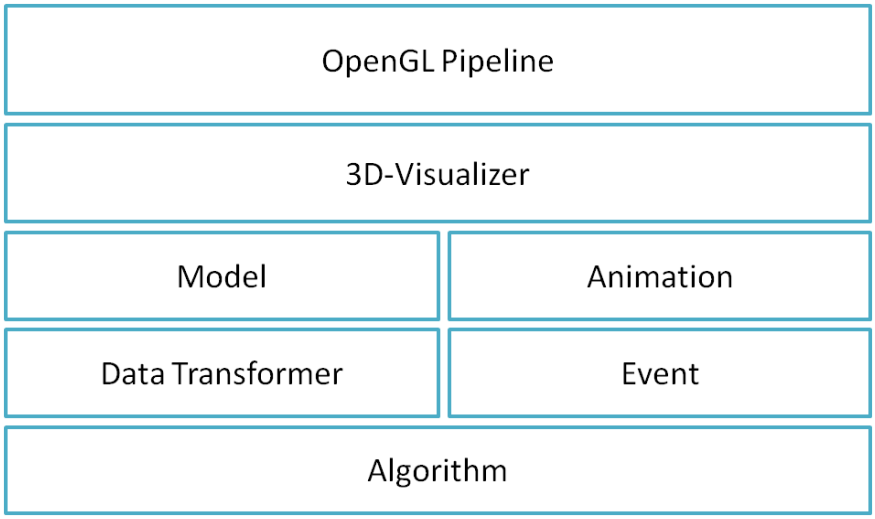
[6] Estimating Surface normal in Noisy Point Cloud Data, Niloy J.Mitra, An Nguyen

7. Surface Reconstruction from Unorganized Points Hugues Hoppe, Tony DeRose, Tom Duchamp John McDonald, Werner Stuelzle

演示部分

演示框架

主要实现了基于 OpenGL 的显示框架，它主要包括 Model, Animation 和 Visualizer，以及相应的 Data Transformer, Event 机制。最终通过 OpenGL 的渲染流水线进行渲染。其示意图如下：



在 Model 当中，我们实现了可重用的 Axis，Grid，Layer，Scene，Vertex，Edge，Triangle，Sphere 等可视化组件，它们共同具有 IModel 的接口；在 Animation 当中，为演示程序实现了 LinearAnimation 的动画组件，实现了 IAnimation 接口。在 Visualizer 当中，实现了基于 OpenGL 的渲染流水线，控制渲染、动画，以及实现交互。为了使得 OpenGL 的全局渲染机制能够同时运行多个实例，还为此实现了互斥锁和切换上下文的控制。

其中，算法过程中，我们通过多线程的方式实现了算法的阻塞消息响应机制。在算法过程当中每次产生新的事件，就会轮询消息侦听句柄是否继续，同时通过消息回调机制的更新模型。

Ball-Pivoting 算法当中的事件:

（1） OnEdgeActivated：当一条边被标记为 ACTIVE 时，即以该边作为轴时，产生该事件；

（2） OnSeedTriangleFound：当种子三角形被找到时，产生该事件。

（3） OnTriangleCreated：当创建新的三角形时，产生该事件。

（4） OnComplete：当算法结束时，产生该事件。

只有在上述操作事件，侦听句柄会做出控制，使得算法继续还是停止。

通过这个方式，算法过程与 UI 演示部分相对独立。Ball-Pivoting 算法可以不依赖而 UI 运行；同时，UI 演示部分甚至也不需要知道算法细节，因而也可以较为方便地重用到其它类似的几何相关的三维算法演示当中。

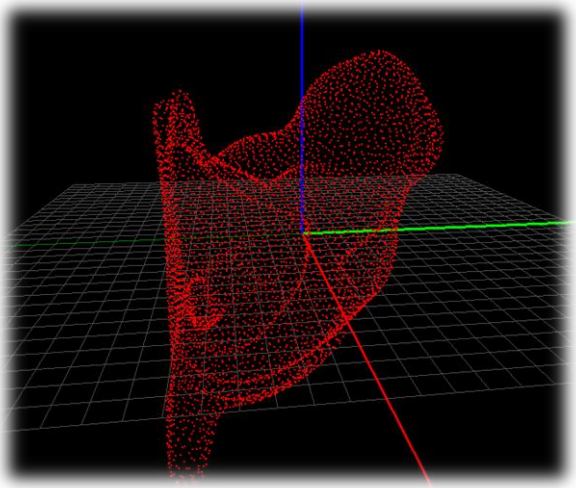
实验中遇到的问题及解决对策

在滚球算法的实现中，由于计算浮点精度总存在，因此在处理多点共平面圆的情形下，会发生拓扑错误。程序中处理这一问题的方法是对每一个点，记录一个 Taboo 列表，即周围曾与其连接过边但之后该边被删除（成为内部边）的点列表。这样可以减少拓扑错误的发生，增加程序的强壮性。

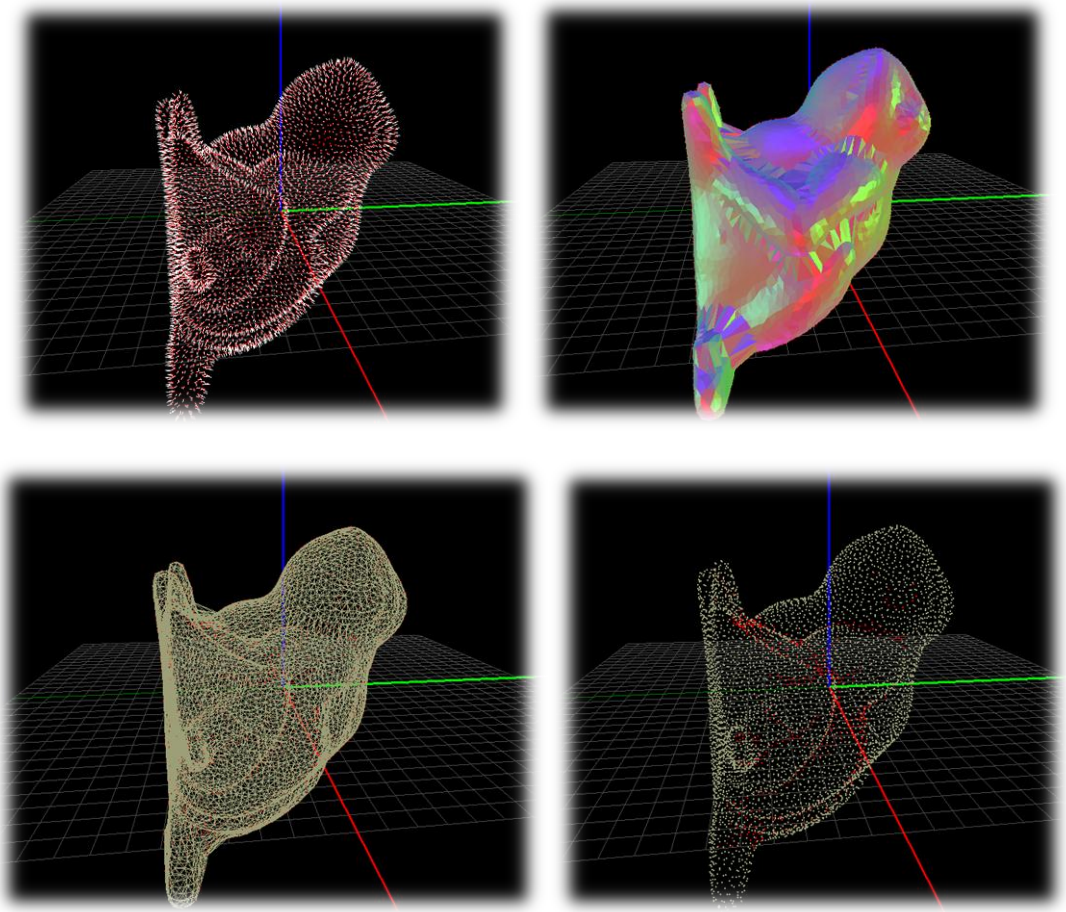
结果：

算法测试

下图为模型测试的案例，显示为 Frog.obj 的模型的点云（共 5429 个点）：



以下是运行算法后的结果：



左上图为点云及其法向量，右上图为重构后的模型。左下图仅显示重构后的三角网格，右下图显示重构的点选取（其中白色的为选中的点，红色的为误差过大而省略的点）。

性能测试

以下为性能测试：

模型	点数目	三角形数目	时间（秒）
Ball10.obj	762	1520	0.11232
Cube5.obj	152	300	0.02285
Cube20.obj	2402	4800	0.21606
Cylinder8.obj	994	1986	0.09953
Horse.obj	5014	9270	1.21005
Frog_s2.obj	1085	1795	0.31641
Frog_s4.obj	2171	3694	1.17253

Frog.obj	5429	9952	2.20690
Bunny_s6.obj	3177	5213	1.19515
Bunny.obj	5295	9297	1.06360
Bunny_full.obj	34838	64541	16.43070

其中 Ball，Cube，Cylinder 是我们便于演示而生成的模型；而 Bunny，Frog，Horse 为实际工作中扫描的模型。根据性能测试的结果，可以看到，我们实现的算法对于实际的模型运行效率良好，速度较快，能够比较好地应用到实际工作当中。

结论：

在本次课程设计当中，我们实现了基于 Ball-Pivoting（滚球）的从点云构造表面模型的算法，并将其通过自建模型和实际模型进行测试。其算法复杂度为 O(|E|)，在稳定性和效率方面表现良好。

参考文献

[1] Bernardini, F. and Mittleman, J. and Rushmeier, H. and Silva, C. and Taubin, G., The ball-pivoting algorithm for surface reconstruction, IEEE Transactions on Visualization and Computer Graphics, 2002

[2] Mitra, N.J. and Nguyen, A., Estimating Surface normal in Noisy Point Cloud Data, Proceedings of the nineteenth annual symposium on Computational geometry, ACM, 2003

[3] Hoppe, H. and DeRose, T. and Duchamp, T. and McDonald, J. and Stuetzle, W., Surface Reconstruction from Unorganized Points, Citeseer, 1994

作者联系方式：

林添　高等研究院　bolitt@gmail.com

杨光　理论计算机科学研究中心　2006yangguang@gmail.com

任博　计算机系媒体所　renboeverywhere@gmail.com