计算几何实验报告

李辉　　024937
张海涛　024996

题目

Smooth Surface reconstruction via natural neighbour interpolation of distance functions

问题背景

我们的问题是用一组无序的点以及他们的法向重构一个光滑的曲面（例如 d 维空间中的 d-1 维流形）。这样的问题已经在许多领域中有实际应用，比如：CG，逆向工程，图像处理，数学，化学等等。输入的数据有很多种来源：激光扫描，立体视觉，3D 图像（例如地震数据和医学图像），或者数学模型，几何模型压缩也是曲面重构的一个重要应用领域。大多数应用都是在三维空间中进行的（d=3），因此，我们的实验选择了三维空间中的光滑曲面重构。

算法及原理

1. 算法输入

设 S 是 d 为空间中的 d-1 维光滑流形，算法的输入就是在 S 边界上的 n 个采样点 p1,…,pn，设采样点的集合为 P。每个点 pi 都附加了一个 S 在 pi 处的单位法向 n(pi)，指向 S 外部。我们用 H(pi)通过 pi 且垂直于 n(pi)的超平面，用 Hp(pi)表示以 H(pi)为边界且与 n(pi)方向相反的半空间。算法需要的输入就是点集 P 以及相应的单位向量集 n(P)。

2. 自然邻居（Natural neighbours）

当数据点的分布不均匀时，定义邻居点便成为了一个问题。为了构造一个曲面，我们需要找到与一个给定点 x 相邻的那些点。一个简单的方法就是找到与 x 的距离小于一个指定值的那些点，显然，这样的方法效果不够好。在本实验中，我们使用 x 的自然邻居（natural neighbours）作为 x 的邻居点，下面就介绍一下如何定义自然邻居。

设采样点的集合为 P，V(pi)为 P 的 Voronoi 图中 pi 所在的 Voronoi cell，V(x)为 P∪{x} 的 Voronoi 图中 x 所在的 Voronoi cell。同时，令 B(x,pi)为垂直平分点 x 与点 pi 的超平面，令 Bp(x,pi)为以 B(x,pi)为边界的，且包含点 x 的半空间。

x 的自然邻居就定义为：在 P∪{x}的 Delaunay 三角剖分中，与 x 相邻的那些点。与它等价的定义是：x 的自然邻居就是在 P 的 Voronoi 图中插入 x 后，所在的 Voronoi cell 被“砍掉”一部分的集合 P 中的点。如果 pi 是 x 的自

然邻居，那么被 x 所“抢去”的 Voronoi region 称作 x 关于 pi 的 natural region NR(x,pi)。因此，我们可以得到：

$$NR(x, p_i) = V(x) \cap V(p_i) = V(p_i) \cap Bp(x, p_i)$$

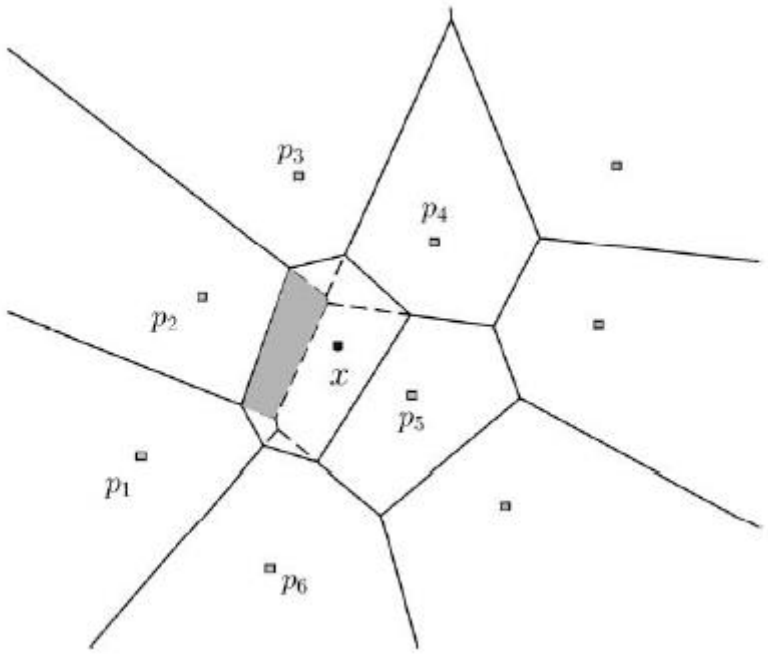


图 1　二维空间中的 natural region。x 的自然邻居是 p1,…,p6，natural region NR(x,p2)用灰色表示

用 w(pi,x)来表示 NR(x,pi)的 Lebesgue 测度——二维空间中是面积，三维空间中是体积。当 pi 不是 x 的自然邻居时，w(pi,x)=0。定义 x 关于 pi 的 natural coodinate 为：

$$\lambda(p_i, x) = \frac{w(p_i, x)}{\sum_i w(p_i, x)}$$

当 x 落在点集 P 的凸包外时，如果 pi 是凸包的一个顶点，那么 w(pi,x)就是无界的，为了使 w(pi,x)有界，我们需要在点集 P 的外面包围一个足够大的球 B。B 的边界用一系列的离散点集 Q 表示。

3. 自然邻居插值（Natural neighbour interpolation）

对于点集 P 中的每一点 pi，都附加一个连续可导的函数 h(pi,x)，函数的定义域为 d 维空间中的点集，值域为实数集，且满足 h(pi,pi)=0。我们定义自然邻居插值函数为：

$$h(x) = \sum_i \lambda(p_i, x) h(p_i, x)$$

前面提到过，我们已经加入了点集 Q 来表示球 B 的边界，我们同样将 Q 中的每个点 qi 附加一个 B 在 qi 处的单位法向，指向 B 的内部。和 pi 相同，每个 qi 也有一个函数 h(qi,x)，且 h(qi,qi)=0。

4. 曲面重构

4.1 有向距离函数

在曲面重构应用中，很多函数可以用来充当 h(pi,x)。我们选择这样一个函数：

$$h(p_i, x) = (p_i - x) \bullet n(p_i)$$

其中 n(pi)是 S 在 pi 处的单位法向。由此可见，h(pi,x)实际上是 x 到过 pi 且与 S 相切的超平面的有向距离。

我们定义插值曲面 S'为点集{x | h(x)=0}。当采样点越密集时 S'越趋近于 S，理论上，当采样点无穷多时，S'与 S 重合。

4.2 对重构表面进行三角剖分

给定一个采样点集 P，S'被 P 的 Voronoi 图和插值函数 h 唯一确定。我们需要计算一个离散的点集来近似 S'，我们使用 P 的 Delaunay 三角剖分来近似 S'。

Delaunay 三角面片 t 的对偶 Voronoi 边是一个线段，线段的两个端点是在 Delaunay 三角剖分中共享 t 的两个四面体的外接球球心。如果 Voronoi 边的一个端点的 h 值为正，而另一个端点的 h 值为负，那么我们就将这个 Voronoi 边称为 bipolar，相应的，我们将 Delaunay 三角剖分中对偶边是 bipolar 的面片成为 bipolar facet。这些 bipolar facet 就构成了对 S'的最初始的近似。

4.3 算法流程

1. 对原始采样点集 P 进行 Delaunay 三角剖分。
2. 找出 Delaunay 中的 bipolar facet。
3. 对于任意一个 bipolar facet，设其外接圆圆心为 c，将所有 f(c) > η（η为一个用户指定值）的 bipolar facet 插入一个以 f(c)为优先级的优先级队列 Q 中。
4. 若 Q 为空，算法结束；否则，从 Q 中取出一个 bipolar facet，在其对偶 Voronoi 线段中找出 f(p) = 0 的点 p，将该点作为插值点，并令 P = P∪{p}。
5. 更新 P 的 Delaunay 三角剖分，并找出新产生的 bipolar facet，转步骤 3。

需要注意的是，在算法中，需要维护两套 Delaunay 数据结构，其中一套是不变的，用来计算插值函数 h(x)，在另一套上插入新的点并更新 bipolar facet 队列。

5. 体积的快速算法

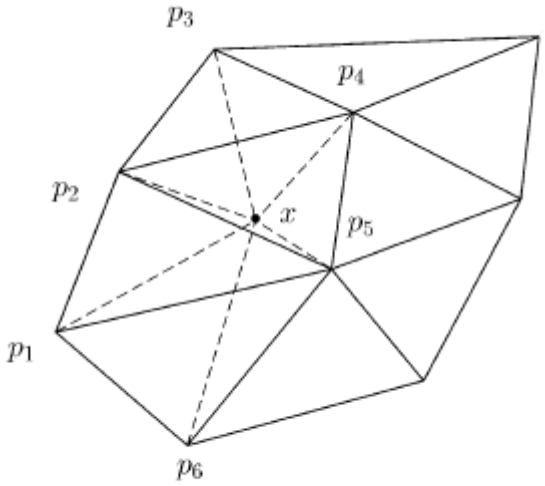


Fig. 3. Inserting a point in a 2D Delaunay triangulation.

如上图所示，为了保持 Delaunay 剖分的机制，我们同时计算所有的 natural region 通过模拟插入一个新点 x 到原有 Delaunay 三角剖分中插入。那么原有三角剖分中被 x 影响的部分（即其外接圆包含 x 点）组成一个 star-shaped 区域——洞，并且这个洞被新的三角形——以 x 为顶点，原有的边界有边的三角形所填充。推广到 3D 的情况下，进行相同的操作，将会有一系列的四面体（外接球包含 x）被破坏，并且由新的四面体（以 x 为一个顶点，被破坏部分的边界为底面）所代替。

NR(x,pi)是由 V(pi)被半平面 Bp(x,pi)切割而得，那么我们考虑 V(pi)的一个面 f，如果 f∩Bp(x,pi) = ⊙，那么我们定义 f 为 cap，如果 f∩Bp(x,pi) <> ⊙，那么我们定义 f 为 arch。那么 NR(x,pi)由若干 cap 与 arch 组成，并且 V(pi)∩Bp(x,pi)是它的一个底面。

那么我们用顶点的一个排序来表示 cap 和 arch，因为 cap 和 arch 是凸多边形，这样的一个顶点的序列就提供了一个对应的三角剖分，取 Oi 是 NR(x,pi)底面上的任一个点，NR(x,pi)的体积就是由 Oi 与所有 cap 以及 arch 三角剖分所得三角形所构成的体积之和。那么计算 NR(x,pi)体积就演变为寻找 cap 以及 arch 的顶点序列。

这就涉及到了一个算法——simulateInsert()，模拟插入算法。我们首先定义前文所提的洞的表面上的 Delaunay 边为外边（external edge），内部的边为内边（internal edge）。

那么记录 e 为任一条外边，在插入 x 之后，有一些以 e 为边的四面体被破坏掉了，记为{t1,…,tk}，并且被两个新的四面体{tp1,tp2}所取代。这两个新的四面体以 x 为顶点并且以邻接 e 的 Delaunay 面为底面。我们记录{ccp(i)}i=1,2 和{cc(i)}i=1,2,…,k 为新旧四面体外接球的球心，那么多边形{ccp1,cc1,cc2,…,cck,ccp2}就描述了 e 的对偶面与 Bp(x,pi)的交集——arch。

同样，我们任取 e 为一条内边，插入 x 之后 e 在新的三角剖分中将不存在了。但是它的对偶面是一个 cap，这个 cap 的顶点是在插入 x 之前以线段 P(i),P(j)为边的四面体的外接球球心。

这样，我们就找到了 cap 以及 arch 的顶点序列。

系统设计&数据结构

系统设计

系统采用 MVC 设计，控制与显示分离。

View(界面)部分采用 OPENGL 实现，其余不再赘述。

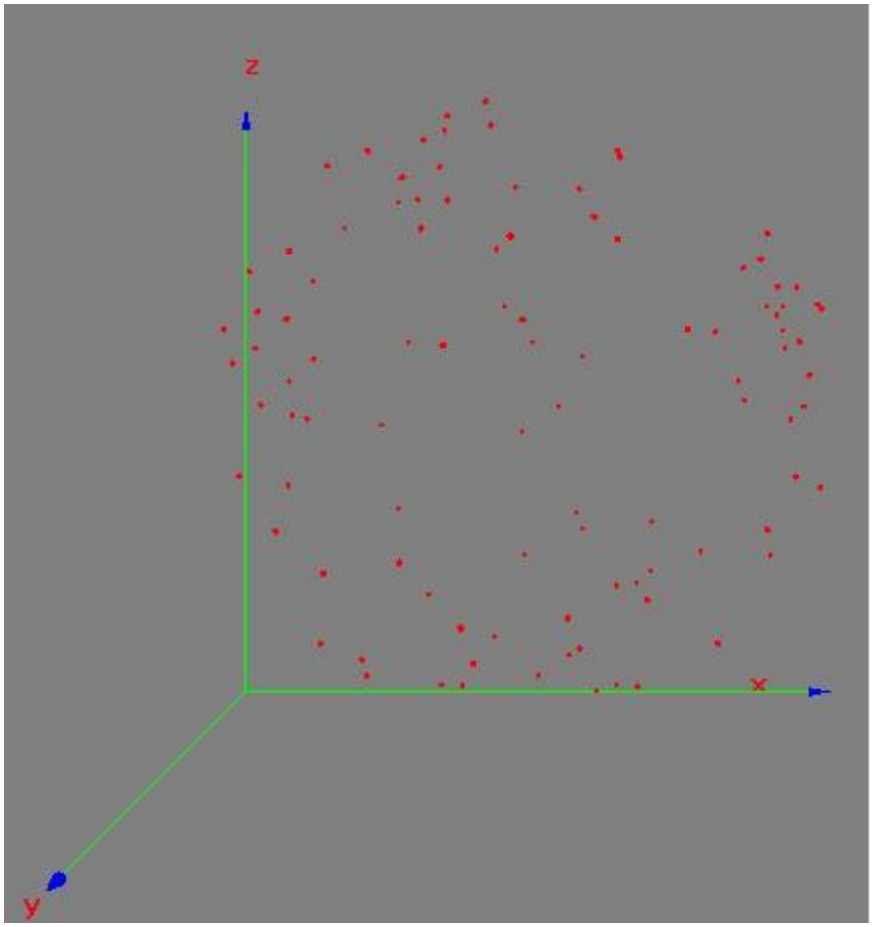
Control(控制)部分由类 SmoothSurface 实现，入口为 SmoothSurface:: Reconstruction()函数。输入参数为采样点列表，输出参数为计算完成后所有点列表。

Module(数据)部分采用 stl::list 来存储初始采样点与最终插值完成后的所有点。

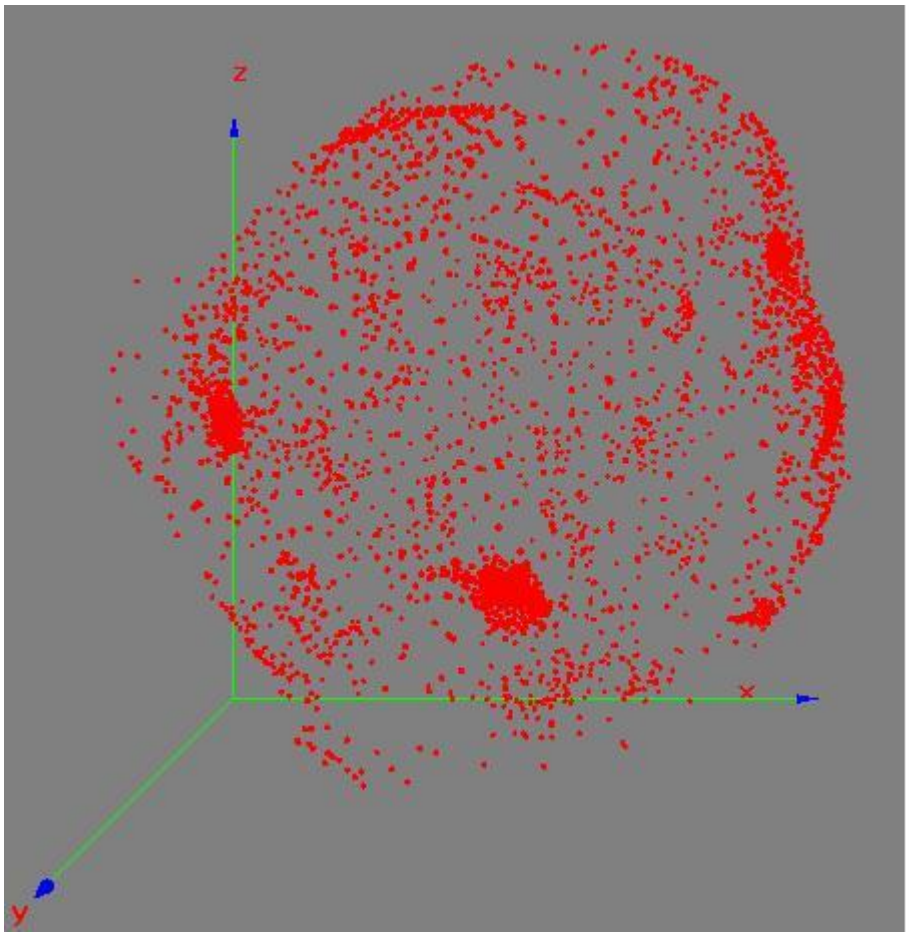
数据结构

数据结构中大量使用了 stl 库中的类型。下面详细说明一下我们认为较难理解的数据结构——即在计算 f 值时所使用到的 map<Vertex_handle, map<Vertex_handle, list<Point> > > stolen，这是用来保存 NR(pi,x)的数据结构，外层 map 的 key，即其中前一个 Vertext_handle 是当前正在处理的 natural neighbour，外层 map 的值为内层 map，内层 map 的 key，即第二个 Vertext_handle 是与当前 natural neighbour 相邻的 natural neighbour，内层 map 的值 list<Point>是位于当前星形结构内且属于当前 natural neighbour 与相邻 natural neighbour 连线的相邻 cell 的外接球的球心序列。

测试、对比结果



上图为原始采样点



上图为计算完成后的所有点

而且出于程序运行时间较长的原因，我们没能得到更多的结果以资对比……。

实现过程中遇到的问题及解决对策

- 体积计算速度极慢

  开始采用计算体积的方法是先求出凸多面体的顶点，然后对其进行三角剖分，然后将每个四面体的体积相加，这种方法速度极慢，不能忍受。于是采用了论文中提到的快速体积计算方法，极大的改善了性能。

- 数值计算的精度问题

  论文中证明 h 函数是连续的，但是我们在用二分法查找时有的时候会趋近一个非零的值，后经过研究发现这是因为数值计算的精度问题而导致的。为了解决这个问题只好在遇到这样点的时候将其直接丢弃。

- 在改进算法后依然存在的速度问题

  已经采用了快速体积计算方法，但是程序速度依然很慢，后来想到可能是因为 debug/realease 版本之间的问题，改成 realease 版本后速度提升了 **20** 倍以上。

没有解决的问题

虽然已经快了很多，运行速度依然很慢……

有关文献及代码来源

算法来自文献：Jean-Daniel Boissonnat,Frederic Cazals Smooth surface reconstruction via natural neighbour interpolation of distance functions Computational Geometry 22(2002) 185-203
使用辅助库：CGAL,http://www.cgal.org