# 一种改进 Ball Pivoting 的散乱点云数据重建算法

胡丝兰[1,2]，周明全[1,2]，税午阳[1,2]，武仲科[1,2]
（1.教育部虚拟现实应用工程研究中心，北京 100875；2.北京市文化遗产数字化保护重点实验室，北京 100875）

**摘要：**Ball Pivoting 算法主要是用来对点云数据进行表面重建，算法是从一个种子三角形开始的，球沿着这个种子三角形的一个边进行旋转，直到这个球接触到下一个点，该边与该点组成一个三角形。该算法主要的缺点是当点云数据不均匀的时候球在滚动的过程当中不会接触到点，因此会产生洞。*提出了一种对散乱点云数据进行重建的改进的 Ball Pivoting 算法；对初始点云数据构建 k-d 树，使得搜索区域算法时间效率提高；使用可变半径搜索改进算法，使得算法能够处理不均匀的点云数据而不会产生洞；优化了网格的拓扑结构。*实验结果证明该算法相对于 Ball Pivoting 算法效率高，且不会生成洞，三角网格的拓扑结构好。

**关键词：**Ball Pivoting 三角化；三角网格；散乱点云数据；k-d 树
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2015) 10-2446-07
DOI:10.16182/j.cnki.joss.2015.10.033

## Improved Pivoting Ball Algorithm for Nonuniform Point Cloud Data

*Hu Silan*[1,2], *Zhou Mingquan*[1,2], *Shui Wuyang*[1,2], *Wu Zhongke*[1,2],
(1. Engineering Research Center of Virtual Reality and Applications, Beijing 100875, China;
2. Beijing Key Laboratory of Digital Preservation and Virtual Reality for Cultural Heritage, Beijing 100875, China)

**Abstract:** Ball Pivoting algorithm is mainly used for surface reconstruction of point cloud data, the algorithm starts from a seed triangle, the ball along the seed triangle a side of rotation until the ball is exposed to a point and the edge and the composition of a triangle. The main disadvantage of this algorithm is that when the point cloud data is not uniform when the ball in the rolling process does not contact the point, and therefore will have a hole. *A non-uniform point cloud data for improved reconstruction of the Ball Pivoting algorithm of the original point cloud data was proposed to construct k-d tree, allowing searching area algorithm time efficiency; variable radius search algorithm could make the algorithm deal with non-uniform point cloud data and did not produce hole, putting forward the more being a searching area algorithm, to optimize the topology structure of the grid. The experimental results show that the algorithm is efficient and cannot generate the holes and the topological structure of the triangle mesh is better than the Ball Pivoting algorithm.*

**Keywords:** Ball Pivoting triangulation; mesh triangular; nonuniform point cloud; k-d tree

## 引言

随着计算机在各个方面应用的渗入和三维建模技术的发展，基于点云的曲面重建技术广泛应用于逆向工程、计算机视觉、虚拟现实、工业制造、医学、游戏、博物馆文物保护、CAD 等领域。使用三维扫描仪对三维真实物体进行扫描得到点云数据，然后对该点云数据进行三维网格化被称为表面重建。在现在有的很多的点云三维重建算法当中，三角剖分算法由于其实用的表达形式和易于渲

收稿日期：2015-06-14 修回日期：2015-07-31；
基金项目：自然科学基金(61202198, 61402042)；中央高校基本科研业务费(2013YB70, 2013YB72)；国家科技支撑计划(2012BAH33F04)；
作者简介：胡丝兰(1991-)，女，湖南，硕士生，研究方向虚拟现实与可视化技术。

染和添加纹理的特点得到的广泛的应用。对于很多应用来说, 三维网格化的鲁棒性、可靠性和三角化速度是很重要的指标。同样, 重建之后和真实物体的相似度也很重要。其中 Delaunay 三角网[1]具有的空圆特性和最大最小角特性使得其在众多三角网格化算法中, Delaunay 三角网格化得到的网格质量是最优的。Delaunay 三角剖分的方法可以使用各种不同的方法去实现 Voronoi 图和 Delaunay 三角化。其中具有代表性的算法有 Ball Pivoting 算法[2]、Crust 算法[3]以及 Cocone 算法[4]。

Ball Pivoting 算法的核心思想是：从一个种子三角形开始, 以该种子三角形的三个边为网格 G 的边界边, 加入到备选边队列 Q 当中, 然后从队列 Q 当中依次选择一个边, 以该边 ef 为(端点分别为 d1 和 d2)轴, 以输入的参数 r 为半径, 生成一个球 S, 使用该球 S 在点云数据的表面进行滚动, 当在滚动的过程当中遇到一个新的点 d3, 则将给点 d 和边 ef 组合成一个新的三角形加入到网格 G 当中形成新的网格, 将新形成的边(d2 和 d3 组成新的边 e23 和 d2 和 d1 组成新的边 e12)加入到备选队列 Q 当中, 并将 ef 从备选队列 Q 当中移除；然后再从备选队列 Q 当中选择一个边重复上述操作直到所有的边都被选择或者是所有的点云数据都被遍历完。

但是该算法的主要缺点是如果点云数据不够密集, 那么在球滚动的过程当中就不会遇到新的点, 这样的话就会形成小洞。本文使用改进的自适应 k-d 树来实现对点的邻近区域的搜索, 该处理能够提高搜索的时间效率 ;在搜索的过程当中使用自适应的半径(本文依次选择的是 1.2r, 1.4r, 1.6r, 1.8r, 2r)来增加搜索过程当中得到的备选点, 这个能够有效的避免 Ball Pivoting 算法当中使用单一半径 r 进行搜索而留下的洞。

# 1 相关工作

在很多的综述[5-6]当中都将三维表面重建分成两类：隐式算法和显式算法。

## 1.1 隐式算法

隐式算法主要是建立一个隐函数 F(p)=0, 其中 p 或许是全部点云数据或者是一部分点云数据, 最后的三角化是又函数 F(p)=0 提取出来的。最常见的隐函数有如下几种：

(1) 径向基函数的和是处于点云的中心[7-8]；

(2) 建立一系列约束, 使得函数满足给定的值, 且使得其向上的梯度与样本点的法线相匹配[9-10]；

(3) 泊松问题[11]。

隐式方法在稀疏点云数据和有噪声点的点云数据的情况下对于表面重建也是无懈可击的。然而, 这些算法需要处理很多计算, 有些算法甚至需要每个点的表面法向。更严重的是得到的三角网格或许不能包括了所有的点, 这意味着重建之后的三维网格有可能缺失了原始重建对象表面的一些细节特征。而且隐式函数使用的点越多, 那么得到的结果越好。

## 1.2 显式算法

显式算法是直接处理点云数据。与隐式算法相比显式算法重建速度快, 但是显式算法对噪声点的鲁棒性差。显式算法最常见的是被分成两类：网格增长算法和 Voronoi/Delaunay 算法。

### 1.2.1 网格增长算法

网格增长算法是从一个种子三角形开始, 然后逐渐增加点, 直至遍历完所有的点。Bernardini 在文献[2]提出了 Ball Pivoting 算法, 算法的基本概念：M 是流型三维物体的表面, S 是该物体的采样点。假设点足够密集以至于半径为 p 的球不能在不接触到样本点的情况下通过表面。我们从一个种子三角形组成的网格开始, 保持球与这个种子三角形的边界上的每一条边的两个点接触, 然后我们旋转该球直到遇到新的点, 然后将该边和点进行连接形成新的三角形和新的网格边界。

### 1.2.2 Voronoi/Delaunay 算法

这一组的算法首先计算点云数据的四面体, 然

后再在四面体上建立三角网格。文献[3-4,12-16]的主要区别是怎么删除四面体和建立外部三角网格。Crust 算法[3]里候选三角形的集合(称为 Crust)这些三角形的三个顶点不是端点。由于 Crust 不是流型的, 为了矫正表面的拓扑结构, 采用竞走算法来处理候选三角形。Crust 算法的最坏时间复杂度是 O(m2), 其中 m 是点个数和端点个数。为了减少运行时间和内存消耗, 文献[4]提出了改进的 Crust 算法, 被称为 Cocone 算法 :备选三角形是处于 Cocone 区域里面的, 最坏时间复杂度是 O((n2), 其中 n 是点的个数。文献[14]证明了只有当样本点是采样很好的时候 Crust 算法和 Cocone 算法才能获得一个很好的三维重建结果。Dey 和 Goswami 提出了 Super Cocone 算法, 这个算法能够处理大量的点云数据, 在 Super Cocone 算法里面, 整个样本点使用八叉树被分为很多小簇, 然后使用 Cocone 算法处理每一小簇。Power Crust 算法[15]是 Crust 算法的改进, 这个算法能够将拥有足够样本点的点云数据重建成一个完全封闭的三角网格, 但是该算法不是流型的且锋锐的边界和噪声点对结果会产生差的影响。

## 2 相关理论

### 2.1 Ball Pivoting 算法简介

Ball Pivoting 算法(简称 BPA)主要是用来对点云数据进行表面重建。用户首先定义一个球的半径 r, 当球在滚动的时候接触到点, 并且在球中不包含其他的点, 那么就将该点与旋转轴组成一个三角形。BPA 是从一个种子三角形开始的, 球沿着这个种子三角形的一个边进行旋转(沿着边进行旋转但是保持与边的端点接触)直到这个球接触到下一个点, 然后该边与该点组成一个三角形。

假设 M 是流型三维物体的表面, S 是该物体的采样点。假设点足够密集以至于半径为 p 的球不能在不接触到样本点的情况下通过表面。从一个种子三角形组成的网格开始, 保持球与这个种子三角形的边界上的每一条边的两个点接触, 然后旋转该球直到遇到新的点, 然后将该边和点进行连接形成新的三角形和新的网格边界, 然后再重复上述方法, 直到所有的边都访问完, 若还有未被访问到的点, 那么就从未被访问的点集当中再选出一个种子三角形, 继续重复上述方法。生成的三角形的球半径是小于 r 的, 因此 BPA 计算出来的子集也是 3D Delaunay 三角剖分的子集。而且 BPA 计算出来的子集也是流型表面的一个子集。

BPA 对采样点有两个要求：

1. 半径 p 的球与流型的交集是一个拓扑圆盘。

2. 半径为 p 的球至少包含了一个样本点。

第一个条件保证了流型的曲率半径大于 0, 半径为 p 的球也可以通过洞和其他的凹面。第二个条件保证了采样密度足够的大, 球可以再样本点走不留孔。

但是在现实中, 我们必须经常处理不理想的采样, 也就是说样本密度不均匀的、缺失了的点云数据。这样的话就会在进行网格化的过程当中产生洞, 如图 1 所示。

图 1 样本密度不均匀的点云数据有一些边就不会生成, 因此会留下洞

### 2.2 k-d 树简介

k-d 树(k-dimensional 树的简称), 是一种分割 k 维数据空间的数据结构。主要应用于多维空间关键数据的搜索(如：范围搜索和最近邻搜索), 本文主要用来进行范围搜索。它是一棵具有如下性质的树

(1) 若它的左子树不为空, 那么左子树上所有节点的值均小于它的根节点的值。

(2) 若它的右子树不为空。那么右子树上所有节点的值均大于它的根节点的值。

(3) 它的左右子树也分别是一棵二叉搜索树。

在实例随机分布的情况下, 找到最近邻的时间复杂度为 O(logn), 当点云数据量大的时候可以大大提高搜索效率。

## 3 算法描述

我们从一个种子三角形开始, 以该种子三角形(本文当中选取的种子三角形是位于点云数据中心并且在该三角形形成的外接圆当中不包含其余的点)为起始网格, 以网格的每个边缘边为球的轴, 以 r 为球的半径, 出于每个球上和球内的点位该边缘边的候选点集, 然后我们从该候选点集当中选择一个点和该网格的边缘边的两个端点组成一个新的三角形加入到网格当中, 形成新的网格, 然后再以该新的网格的每个边缘边递归的重复上述过程直到所有的点都处理完。

由于使用 Ball Pivoting 算法对散乱点云数据进行三维网格重建会产生洞, 本文的主要改进是使用可变参数在散乱点云数据使用 Ball Pivoting 进行三维重建的时候不会产生洞, 从而避免了后续的填洞处理。主要的改进为：使用改进的自适应的 k-d 树对点云数据进行搜索 ;在搜索过程当中使用可变半径进行自适应搜索。该算法使用三维点云作为输入并输出一个三角网格, 具体步骤如下：

步骤 1.导入点云数据；

步骤 2.将点云数据构建成一个 k-d 树；

步骤 3.将点云数据三角网格化。

### 3.1 改进的自适应 k-d 树

由于 Ball Pivoting 算法使用的是固定 r 为半径, 以已形成的三角网格的边界边为旋转轴而形成的球来对点云数据进行搜索, 如果球旋转的过程当中能够遇到点, 那么就将该点加入到三角网格当中。但是对于散乱点云数据来说, 球在旋转的过程当中很可能就不会遇到点, 这样就会形成小的洞, 而这些洞本身不属于原始重建对象, 本文采用改进的自适应 k-d 树来对点云数据进行搜索, 当以 r 为半径的球在旋转的过程当中没有遇到点, 那么我们就增加该球的半径直到 2r(这个时候还不能遇到点那么我们就将该区域视为边界区域), 这样可以有效的减少洞的产生。, 相对于传统的 k-d 树的时间的复杂度 O(logn), 本文改进的 k-d 树的时间复杂度为 O(logn)取下整数；且本文的 k-d 树是自适应的, 在本文的算法当中我们可以依据当前的包围盒当中是否有点来增加包围盒的大小(本文当中球半径大小依次选择 r, 1.2r, 1.4r, 1.6r, 1.8r, 2.0r)。

算法原理 :由根节点的包围盒子来确定树中数据的范围, 在搜索过程当中使用划分包围盒子(划分方法如图 2 所示, 以二维空间为例, 说明父节点包围盒与子节点包围盒的关系, 其中 L1 和 L2 表示的是左右划分平面；L1 和 L2 将包围盒 B 划分为 B1, B2 和 B33 个部分, 左右子节点的包围盒是 B1 和 B3。)来缩小搜索范围, 并且递归的计算查询点到包围盒的距离来提高搜索效率。

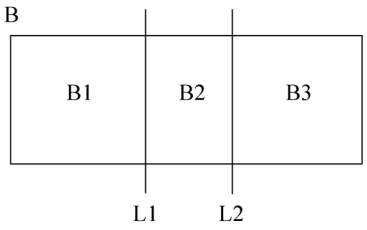


图 2 包围盒划分图

设输入的点云数据为 P, 点与点之间的距离使用欧式距离 ;设 l=[l1, l2, l3]T 和 h=[h1, h2, h3]T, 则 l 和 h 所定义的包围盒 B 是以 h 和 l 两点之间的连线为轴的半径为 2r 的球体, 判断点 q 是否在 B 内部可以使用 q 到包围盒(也就是球)的中心的距离是否小于或等于 r 来进行判断。

k-d 树的搜索方法(图 3)是从根节点开始, 依次计算查询点到左右子节点的包围盒的距离, 然后判断该点到包围盒的距离于半径 r 的距离进行比较：若该点的距离到包围盒中心(球中心)的距离小于或等于 r 则加入到备选点当中；若该点的距离到包围中心(球中心)的距离大于 r 则对该子树进行减枝。

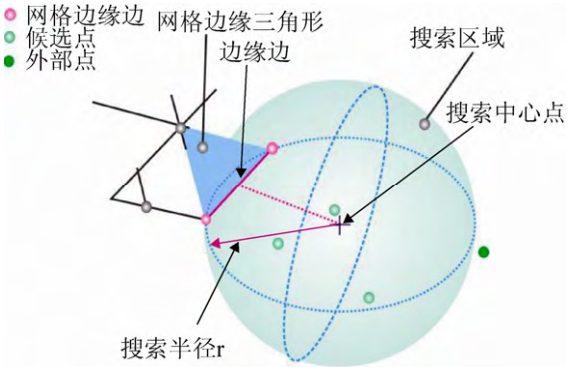


图 3　搜索区域展示

### 3.2 备选点筛选法

当从 3.1 当中得到了备选点之后，我们需要从众多的备选点当中选择出一个最优点和边缘边组成新的三角形。

如图 4(a)所示 :在搜索区域当中有三个点(cp1、cp2 和 cp3)，我们需要在这三个点当中选择一个点来和边缘边组合成一个新的三角形, 首先我们随机选取一个点(图 4(b)中选择了 cp2)，然后做包含该点的最小球 ;如果该球内还有其他的点则我们在该球内随机选择一点重复上述操作(如图 4(c))；如果球内没有任何点了那就选择该点做为最优备选点(如图 4(d))，该点和边缘边组成的三角形就是备选三角形。

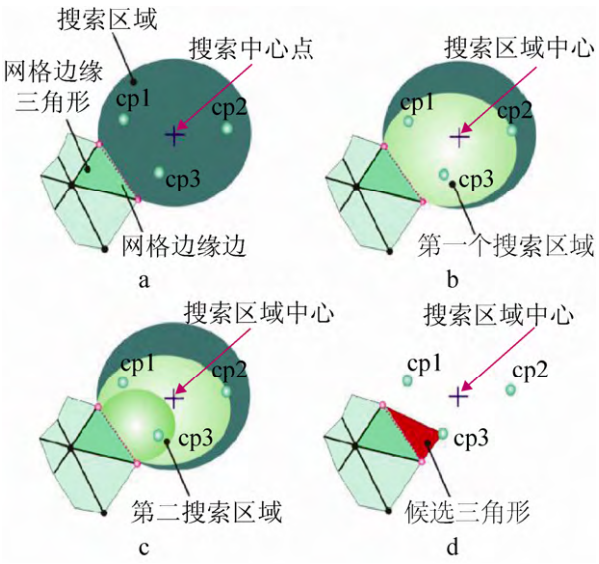


图 4　在区域当中选择合适的点的过程

### 3.3 拓扑结构测试

当从 3.2 当中得到了备选三角形后，我们需要测试该三角形的拓扑结构, 如果形成的新的边没有被其他三角形占有那么我们直接将该三角形加入到网格当中 ;如果新的边已经被网格当中的其他三角形给占有了, 那么我们需要测试备选三角形和已有三角形所共享的同一个边的方向是否是一致, 如果一致则将该备选三角形加入到网格当中, 如果不一致则不加入。

## 4　实验结果

本文选取了若干常用点云数据模型进行实验。实验的环境为 Intel I5(3210M, 2.50GHz)CPU，4GB 内存和 Windows 7(64 位)。本文的参数为：球半径 r。半径 r 参数是设置滚球的初始半径大小，半径大的得到的网格的三角片会大一些, 半径小的得到的网格的三角片会小一些。

图 5 当中显示了设置不同的半径参数得到的网格的细密度也会不一样。Ball Pivoting 算法设置的半径参数大一些能够避免小洞的产生, 但是同时也造成了生成的网格过于粗，造成重建效果不好。所以本文当中我们会使用改进的 Ball Pivoting 算法设置适当的半径参数来加强重建效果, 因为我们的方法当中不会生成小洞。

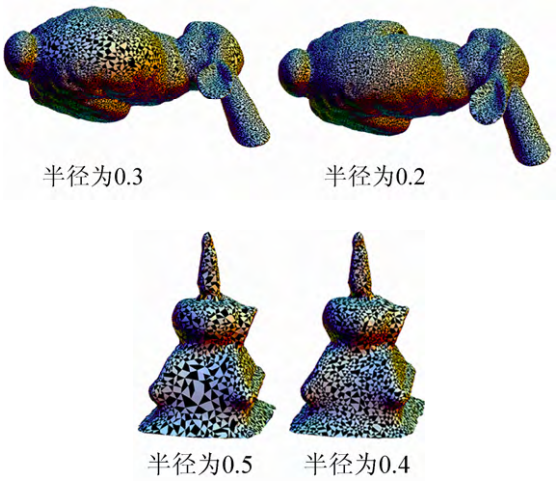


图 5　设置参数半径 r 不同生成的网格密度不同

图 6 的(a)和(b)当中显示了使用 Ball Pivoting 算法生成的三角网格, 其中使用红色圈圈标记出来的表明该地方会产生洞；图 6 的(c)和(d)当中展示了使用本文改进的 Ball Pivoting 生成的三角网格, 其中我们能明显的看到(c)和(d)当中没有洞, 相对于(a)和(b)效果好很多。

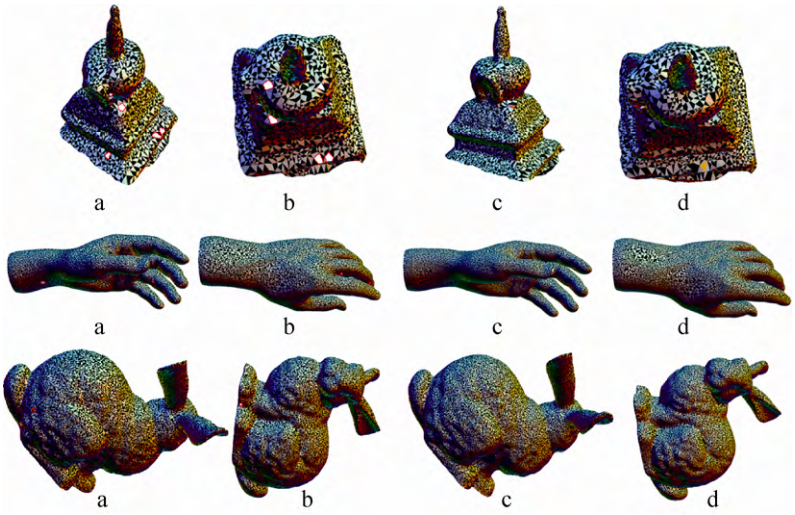


图 6 Ball Pivoting 算法生成的 a, b 图和本文的算法生成的 c, d 图结果比较。

表 1 点云数据化 Ball Pivting 算法和本文的算法时间效率和洞的个数的比较

| 方法 | 塔 | 手 | 兔子 |
|---|---|---|---|
| Ball Pivoting 算法 | 共 7 786 个点, 运行时间为 40 s, 产生 32 个洞 | 共 39 235 个点, 运行时间为 159 s, 产生 10 个洞 | 共 35 947 个点, 运行时间为 153 s, 产生 14 个洞 |
| 本文改进的 Ball Pivoting 算法 | 共 7 786 个点, 运行时间为 36 s, 产生 5 个洞 | 共 39 235 个点, 运行时间为 132 s, 产生 0 个洞 | 共 35 947 个点, 运行时间为 153 s, 产生 3 个洞 |

## 5 结论

本文主要的改进是使用改进的自适应 k-d 树来实现对点云数据的邻近区域的搜索, 且在搜索的过程当中使用自适应半径来增加搜索到备选点的概率, 由于 Ball Pivoting 算法是使用固定半径, 所以使得在三角网格形成的过程当中会生成洞, 使用本文改进的 Ball Pivoting 算法能有效的避免洞的生成。实验结果(表 1)证明本文的算法较 Ball Pivoting 算法有较大的改进, 时间效率提高了、洞的个数明显的减少了。

由于本文在进行重建之前还得手动设置半径 r, 下一步研究方向是能够自动根据周围的数据设置 r 值, 从而实现全自动的点云数据重建, 而且自适应的半径设置会使得点云重建的效果更加好, 对于重建对象表面的细微形状重建结果会更好。

## 感谢

本文特别感谢周明全教授的指导和大力支持, 他渊博的知识和开阔的视野给了我很大的启迪, 论文也凝聚着他的汗水 ;同时也感谢武仲科教授和税午阳老师对我的论文的细节和框架的指导, 并提出了很多宝贵的意见和建议。

## 参考文献:

[1] Cazals F, Giesen J. Delaunay triangulation based surface reconstruction. [M]// Boissonnat J, Teillaud M, editors. Effective computational geometry for curves and surfaces. Math. and visualization. Germany: Springer-Verlag, 2006: 231-76.

[2] Bernardini F, Mittleman J, Rushmeier H, *et al*. The Ball Pivoting algorithm for surface reconstruction [J]. IEEE Transactions on Visualization and Computer Graphics (S1001-395), 1999, 5(4): 349-359.

[3] Amenta N, Bern M, Kamvysselis M. A new Voronoi-based surface reconstruction algorithm [C]// The Proceeding of Computer Graphics, SIGGRAPH'98. USA: ACM, 1998: 415-421.

[4] Amenta N, Choi S, Dey TK, *et al*. A simple algorithm for homeomorphic surface reconstruction [J]. International Journal of Computational Geometry & Applications (S1002-8331), 2002, 12(1/2): 125-141.

[5] Li X, Han CY, Wee WG. On surface reconstruction: a priority driven approach [J]. Computer-Aided Design (S1003-6221), 2009, 41(9): 626-640.

[6] Chang MC, Leymarie FF, Kimia BB. Surface reconstruction from point clouds by transforming the medial scaffold [J]. Computer Vision and Image Understanding (S1002-0708), 2009, 113(11): 1130-1146.

[7] Carr J, Beatson R, Cherrie H, *et al*. Reconstruction and representation of 3D objects with radial basis functions [C]// SIGGRAPH, 2001. USA: ACM, 2001: 67-76.

[8] Turk G, O'Brien J. Modelling with implicit surfaces that interpolate TOG, 2002: 855-73.

[9] Dey Tamal K, Sun Jian. An adaptive MLS surface for reconstruction with guarantees [C]// Proceedings of the Third Eurographics Symposium on Geometry Processing, 2005. (TALG), 2010, 4(2): 812-78.

[10] Kolluri R. Provably good moving least squares [J]. ACM Transactions on Algorithms (TALG) (S1001-8843),2008, 4(2): 832-73

[11] Kazhdan M, Bolitho M, Hoppe H. Poisson surface reconstruction [C]// Symposium on Geometry Processing, The IEEE Symposium on Parallel and Large-data Visualization and Graphics. USA: IEEE, 2006: 61-70.

[12] Dey TK, Goswami S. Tight cocone: a watertight surface reconstructor [J]. Journal of Computing and Information Science in Engineering (S1002-4688), 2003, 3(4): 302-307.

[13] Dey TK, Goswami S. Provable surface reconstruction from noisy samples [J]. Computational Geometry (S1003-2254), 2006, 35(1/2): 124-141.

[14] Dey TK, Giesen J, Hudson J. Delaunay based shape reconstruction from large data [C]// The Proceeding of The IEEE Symposium on Parallel and Large-data Visualization and Graphics. USA: IEEE, 2001: 19-27.

[15] Amenda N, Choi S, Kolluri R. The power crust [C]// The Proceeding of the ACM Symposium on Solid Modeling and Applications. USA: ACM, 2001: 249-60.

[16] Cohen-Steiner D, Da F. A greedy Delaunay-based surface reconstruction algorithm [J]. Visual Computer (S1002-3386), 2004, 20(1): 4-16.

[17] Gopi M, Krishnan S, Silva C. Surface reconstruction using lower dimensional localized Delaunay triangulation [J]. The Proceeding of Eurographics (S1000-2432), 2000, 19(3): 467-78.

[18] Huang J, Menq CH. Combinatorial manifold mesh reconstruction and optimization from unorganized points with arbitrary topology [J]. Computer Aided Design (S1001-4432), 2002, 34(2): 149-65.

(上接第 2445 页)

[5] Schechter H, Bridson R. Ghost SPH for animating water [J]. ACM Transactions on Graphics (TOG) (S0730-0301), 2012, 31(4)：61:1–61:8.

[6] 邵绪强, 周忠, 张劲松, 等. 微可压缩 SPH 流体的稳定性固体边界处理算法[J]. 计算机辅助设计与图形学学报, 2014, 26(11): 1915-1922.

[7] Boyles M, Fang S. Slicing-based volumetric collision detection [J]. Journal of Graphics Tools(S1086-7651), 1999, 4(4): 23-32.

[8] He J, Chen X, Wang Z Y, Cao C, Yan H, Peng Q S. Real-time adaptive fluid simulation with complex boundaries [J]. The Visual Computer(S1432-2315), 2010, 26(4): 243-252.

[9] Harada T, Koshizuka S, Kawaguchi Y. Smoothed Particle Hydrodynamics on GPUs[J]. Proc of Computer Graphics International, 2007, 4(4):671-691.

[10] 陈曦, 王章野, 何戬, 等. GPU 中的流体场景实时模拟算法[J]. 计算机辅助设计与图形学学报, 2010, 22(3): 396-405.

[11] 温婵娟, 欧嘉蔚, 贾金原. GPU 通用计算平台上的 SPH 流体模拟[J]. 计算机辅助设计与图形学学报, 2010, 22 (3): 406-411.

[12] 肖苗苗, 刘箴, 史佳宾, 等. 基于 OpenCL 加速的 SPH 流体仿真[J]. 系统仿真学报, 2015, 27(4): 803-809.