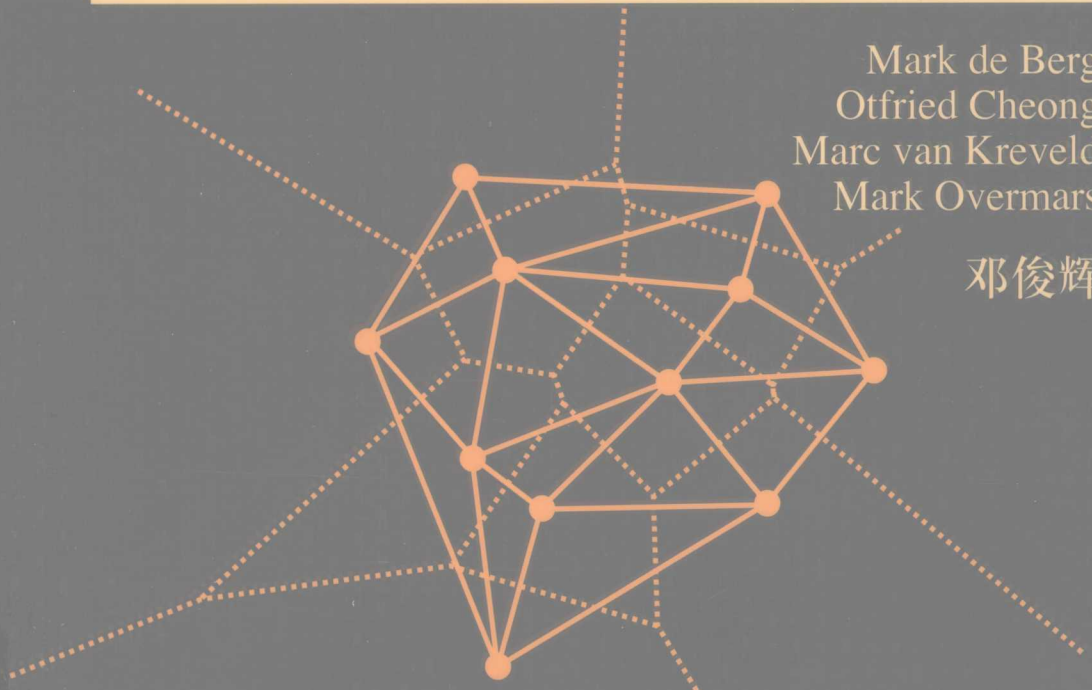
世界著名计算机教材精选

计算几何

算法与应用（第3版）

Mark de Berg
Otfried Cheong
Marc van Kreveld
Mark Overmars
著

邓俊辉 译



COMPUTATIONAL GEOMETRY

ALGORITHMS AND APPLICATIONS

Third Edition

清华大学出版社
Springer

世界著名计算机教材精选

Computational Geometry

Algorithms and Applications

Third Edition

计 算 几 何

算法与应用

（第3版）

清 华 大 学 出 版 社

北 京

图书在版编目（CIP）数据

计算几何：算法与应用(第 3 版)/(德)伯格(Berg, M. D.)等著；邓俊辉译. —北京：清华大学出版社，2009.8
(世界著名计算机教材精选)
书名原文：Computational Geometry: Algorithms and Applications, Third Edition
ISBN 978-7-302-19938-0

Ⅰ. 计…　Ⅱ. ①伯…　②邓…　Ⅲ. 计算几何-教材　Ⅳ. O18

中国版本图书馆 CIP 数据核字(2009)第 058589 号

责任编辑：龙啟铭
责任校对：徐俊伟
责任印制：何　芊

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京市清华园胶印厂
装 订 者：三河市李旗庄少明装订厂
经　　销：全国新华书店
开　　本：185×260　　**印　张**：26.25　　**字　数**：636 千字
版　　次：2009 年 8 月第 1 版　　**印　次**：2009 年 8 月第 1 次印刷
印　　数：1～3000
定　　价：49.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。
联系电话：010-62770177 转 3103　　产品编号：029950-01

前言

20 世纪 70 年代末，计算几何学（computational geometry）从算法设计与分析中孕育而生。今天，它不仅拥有自己的学术刊物和学术会议，而且形成了一个由众多活跃的研究人员组成的学术群体，因此已经成长为一个被广泛认同的学科。该领域作为一个研究学科之所以会取得成功，一方面是由于其涉及的问题及其解答本身所具有的美感，而另一方面，也是由于在（诸如计算机图形学、地理信息系统和机器人学等）众多的应用领域中，几何算法都发挥了重要的作用。

许多解决几何问题的早期算法，要么速度很慢，要么难于理解与实现。随着近年来一些新的算法技术的发展，此前的很多方法都得到了改进与简化。这本教材力图使得这些现代的算法能够为更广泛的读者理解和接受。本书既是面向计算几何课程的一本教材，同时也可用于自学。

本书的结构。除导言外，这 16 章中的每一章都从来自应用领域的某一实际问题入手。这个问题将被转化为一个纯粹的几何问题，进而通过计算几何所提供的方法加以解决。每章所讨论的，实质上就是对应的那个几何问题，以及解决该问题所需要的概念与方法。我们根据所希望覆盖的计算几何专题，来选取有关的应用；而就具体的应用领域而言，这些介绍还远远不够全面。引入这些应用的目的，只是为了激发读者的兴趣；而各章本身的目的，并不在于为这些问题提供现成可用的解决方法。虽然如此，我们还是认为，为有效地解决应用中的几何问题，计算几何方面的知识是非常重要的。希望本书不仅能够吸引来自算法学术圈的那些人，而且对来自应用领域的人们亦是如此。

同一几何问题，可能有好几种不同的解决方法，不过，在论述大多数几何问题时，我们将只给出其中一种。我们通常所选取的，都是最易于理解与实现的方法。我们也十分注意，尽力使本书能够涵盖更多的方法，比如分治策略、平面扫描以及随机算法（randomized algorithm）等等。对每个问题可能的种种变型，我们也不打算面面俱到；我们觉得，更重要的是首先对计算

几何中的各个主要问题做一介绍，而不是过于深入地去探究少数专题的细枝末节。

某些章的若干节标有星号。这些节的内容涉及解法的改进与扩展，或者解释了不同问题之间的相互关联。就对后续章节的理解而言，它们并不十分重要。

每章最后，都由名为“注释及评论”的一节进行概括总结。这些节会给出对应各章所介绍结果的来龙去脉，概述其他的解决方法、一般化处理方法及改进，并给出参考文献。虽然这些节可以被跳过，但是对于那些希望就某一章的专题做进一步了解的读者来说，其中的材料都是非常有用的。

每章后面，都附有一定数量的习题。其中一些旨在检查读者对内容的理解程度，也有些是对书中内容的推广，需要精心解答。高难度的问题以及对应于标有星号各节的问题，也被标上星号。

课程大纲。尽管在很大程度上，本书各章之间是相互独立的，但在进行介绍时，最好还是不要随意打乱其次序。例如，第2章介绍了平面扫描算法，故在阅读采用了这一方法的其他各章之前，最好首先了解该章的内容。出于同样考虑，在进入有关随机算法的各章之前，也应该首先阅读第4章。

如果是作为计算几何的第一门课程，建议（教师）按照书中的次序来讲授前十章。根据我们的经验，这十章覆盖了任何一门计算几何课程都必须介绍的概念和方法。如果还有可能顾及更多的内容，可以在后面六章中进行挑选。

先修要求。作为教材，本书既适用于高年级本科生课程，也适用于低年级研究生课程，具体安排视课程的其他要求而定。读者应具备算法设计与分析、数据结构的基本知识：必须熟知大O记号，以及诸如排序、二分查找和平衡查找树等基本的算法技术。读者不需要对这里所涉及的应用领域有所了解，也几乎不需要什么几何知识。在对随机算法进行分析时，会用到一些非常基本的概率理论。

实现。本书中的算法都是以伪代码的形式给出，虽略显概括笼统，但也算详尽，实现起来相对容易。值得一提的是，我们还尝试着介绍了处理退化情况的方法，在具体实现过程中如不能解决好这一问题，往往会使整个计划落空。

我们认为，动手实现其中一个或多个算法将十分有益；这可以令你获得对算法复杂度的实际感受。每一章都可以当成一个编程训练的课程项目。根据可利用时间的多少，你既可以只实现算法本身，也可以连同应用系统一起完成。

为了实现一个几何算法，若干基本的数据类型——点、直线和多边形等——以及对其实施操作的一些基本例程都是必需的。实现这些基本例程并使之具有鲁棒性，绝非易事，为此需要投入大量的时间。自己动手这样做一次不无裨益，然而如果能够找到一个提供基本数据类型及其操作例程的现成的软件库，将很有帮助。在我们的万维网页面上，可以找到指向这类软件库的链接。

万维网站。本书还附有一个万维网站，该网站提供了本书各个版本的勘误、所有插图、所有算法的伪代码，以及一些其他资源。其地址是：

http://www.cs.uu.nl/geobook/

如果您发现了书中的错误，或是对本书有何建议，可以通过该页面与我们联系。

关于第3版。第3版的改动主要有两处：第7章“Voronoi图：邮局问题”中，增加了关于线段Voronoi图、最远点Voronoi图的讨论；第12章“空间二分：画家算法”中，针

对低密度场景的 BSP 树，作为实际输入模型的导论，增加了一节。此外，更正了大量瑕疵与错误（请参阅网站提供的第 2 版勘误）。每章的“注释及评论”一节也做了更新，以体现新的研究成果及相关文献。为不致影响学生继续在课程学习中沿用第 2 版，第 3 版尽可能没有改动原先各节与各习题的编号。

致谢。编写教材是一项耗时的工作，即便有四位作者共同合作，也不例外。在过去几年中我们得到了很多人的帮助：关于本书应该包括、不应该包括哪些内容，有些人提供了有益的建议，有些人在阅读初稿后对如何修改提出了建议，另一些人则指出并更正了前两版中的错误。感谢所有这些人，特别要感谢 Pankaj Agarwal、Helmut Alt、Marshall Bern、Jit Bose、Hazel Everett、Gerald Farin、Steve Fortune、Geert-Jan Giezeman、Mordecai Golin、Dan Halperin、Richard Karp、Matthew Katz、Klara Kedem、Nelson Max、Joseph S. B. Mitchell、Rene van Oostrum、Gunter Rote、Henry Shapiro、Sven Skyum、Jack Snoeyink、Gert Vegter、Peter Widmayer、Chee Yap 和 Gunther Ziegler。感谢 Springer-Verlag 出版社给予的建议和支持，使得本书各版本得以出版，并被译成日文、中文及波兰文。

最后，还要感谢荷兰科学研究组织（Netherlands Organization for Scientific Research-N. W. O.）与韩国研究基金（Korea Research Foundation-KRF）的大力支持。

Mark de Berg
Otfried Cheong
Marc van Kreveld
Mark Overmars

目录

1 计算几何：导言

正漫步于校园的你，突然需要打一个紧急电话。在遍布校园的各个公用电话中，你当然想找到离自己最近的那部。然而，哪一部才是最近的呢？一张校园地图将能帮忙，无论你身处何处，都可以在地图上找到最近的公用电话。这张地图可能会将整个校园划分成不同区域，每个区域都对应着一部最近的公用电话（如图 1-1 所示）。这些区域形状如何？又该如何计算出它们呢？

尽管这算不上一个至关重要的问题，它却简要描述了一个主要的几何概念，而这一概念在众多应用中都扮演着重要的角色。

对校园如此划分之后，就得到了所谓的 Voronoi 图（Voronoi diagram），第 7 章将详细探讨这一结构。借助该结构，可以为覆盖多个城市的商业区域建立模型，指挥机器人，甚至描述和模拟晶体的生长过程。为了构造 Voronoi 图之类的几何结构，需要一些几何算法（geometric algorithm）。这些算法就是本书的主题。

第二个例子。假设你已经找到了最近的公用电话。只要手中有一份地图，你很容易就能沿着一条很短的路径到达电话的位置，而且中途不会撞上墙或者其他障碍物（如图 1-2 所示）。然而，想要通过程序让机器人自己来完成这一任务，却要困难得多。

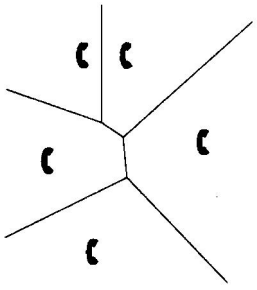


图 1-1　按照公用电话的分布，可以将校园划分为若干区域

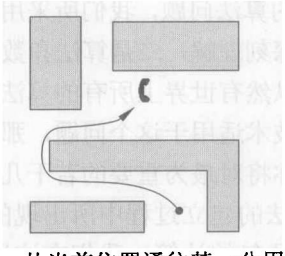


图 1-2　从当前位置通往某一公用电话的最短路径

与上例相同，这一问题的实质也是几何的：给定一组几何形状不同的障碍物，我们需要在不与任何障碍物发生碰撞的前提下，找出连接于任意两点之间的最短通路。这就是所谓的运动规划（motion planning），在机器人学中，这类问题的求解至关重要。第13和15章将针对运动规划所需的几何算法做一讨论。

第三个例子。假设你可以利用的不是一张而是两张地图：一张描述了各个建筑物，包括公用电话；另一张则画出了校园内的道路。为了规划出通往公用电话的运动路径，我们需要将这两张地图叠合（overlay）起来——也就是说，需要将这两张地图所提供的信息合并起来。在地理信息系统（geographic information system）中，地图的叠合是基本的操作之一。这种操作涉及到某张地图中的对象在另一张地图中的定位、不同特征物之间的求交计算等问题。第2章将讨论这一问题。

许多几何问题的解决都要依靠精心设计的几何算法，上面只是其中的三个例子。诞生于20世纪70年代的计算几何（computational geometry），正是旨在解决这类几何问题。这一学科可定义为“针对处理几何对象的算法及数据结构的系统化研究”，其重点在于“渐进快速的精确算法”。由几何问题带来的挑战吸引了众多的研究人员。从对问题的明确表述，到得出高效而优雅的解决方法，往往需要经历漫长的过程，其间既要克服诸多困难，也要积累一些次优的中间结果。今天，我们已经掌握了功能广泛的一整套几何算法，它们不仅高效而且相对更易理解和实现。

本书将介绍计算几何中最重要的那些概念、方法、算法以及数据结构，但愿我们的介绍方式，能够吸引那些有志于将计算几何的研究成果付诸实际应用的读者。本书的每一章都从某一实际问题入手，而这种问题的求解，都需要借助几何算法。为了说明计算几何之应用范围的广泛性，这些问题分别选自不同的应用领域：机器人学、计算机图形学、CAD/CAM以及地理信息系统等。

然而你并不能指望，在解决应用领域中的主要问题时，总是有现成的软件可以直接利用。这里的每一章，只是孤立地对计算几何中的某一特定概念进行讨论；其中所涉及的应用问题，只是作为一个例子，用以导出有关的概念，继而展开介绍。这些例子可以使我们体会到，如何才能针对工程性问题建立（数学）模型，并进而得出严谨的解答。

1.1　凸包的例子

面对具有几何本质的算法问题，我们所采用的解决方法大多需要具备两方面要素：一是对该问题的几何特性的深刻理解，二是算法和数据结构的合理应用。要是对某个问题的几何性质尚不甚了解，那么纵然有世界上所有的算法在手，你依然不能高效地解决它。反过来，如果不知道有哪些算法技术适用于这个问题，那么即使你已经对问题的几何特性烂熟于胸，也是枉然。通过本书，你将对最为重要的若干几何概念以及算法技术有个透彻的理解。

为了说明在几何算法的建立过程中所出现的问题，本节将讨论曾在计算几何中首先研究的问题之一——平面凸包的计算。我们在这里忽略该问题的来由；对此有兴趣的读者，可以阅读第11章（该章讨论的是三维凸包问题）的导言部分。

平面的一个子集S被称为是“凸”的，当且仅当对于任意两点$p, q \in S$，线段$\overline{pq}$都完

全属于 S（若图 1-3 所示）。集合 S 的凸包 $\mathcal{CH}(S)$，就是包含 S 的最小凸集——更准确地说，它是包含 S 的所有凸集的交。

这里所要讨论的，是如何计算平面上由 n 个点组成的有限集合 P 的凸包。可以借助一个虚构式实验，来想象这种凸包的模样：如图 1-4 所示，将这里的点想象成钉在平面上的钉子；取来一根橡皮绳，将它撑开围住所有的钉子，然后松开手——啪的一声，橡皮绳将紧绷到钉子上，它的总长度也将达到最小。此时，由橡皮绳围住的区域就是 P 的凸包。

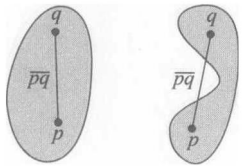


图 1-3 凸集与非凸集

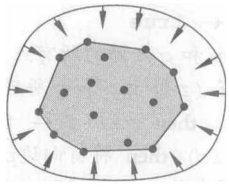


图 1-4 凸包的直观理解

因此，也可以将平面有限点集 P 的凸包定义为：顶点取自于 P 且包含 P 中所有点的那个唯一的凸多边形（convex polygon）。当然，这一定义是否有歧义（也就是说，此多边形是否唯一），以及这一定义是否等同于前面所给出的那个定义，都需要严格地予以证明，然而鉴于这是本章的导言，我们将跳过这一环节。

如何来计算凸包呢？回答这一问题之前，必须先回答另一问题：所谓“计算凸包”，到底是什么含义？正如我们已经看到的，P 的凸包是一个凸多边形。表示多边形的一种自然的方法，就是从任一顶点开始，沿顺时针方向依次列出所有顶点。因此，我们所要求解的问题就变成：

给定平面点集 $P=\{p_1, \ldots, p_n\}$，通过计算从 P 中选出若干点，它们沿顺时针方向依次对应于 $\mathcal{CH}(P)$ 的各个顶点。

input = **平面上一组点：** $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

output = **凸包的表示：** p_4, p_5, p_8, p_2, p_3（**如图 1-5 所示**）

当着手设计一个计算凸包的算法时，此前所给出的凸包定义对我们没有多少帮助。按照那个定义，需要计算出“包含 P 的所有凸集的交”，可是这种集合有无限多个。而我们所观察到的“$\mathcal{CH}(P)$是一个凸多边形”这一事实，则更有帮助。下面就来看看，$\mathcal{CH}(P)$是由哪些边构成的。

这些边的端点 p 和 q 都来自于 P；另外，只要适当地定义由 p 和 q 所确定直线的方向，使得 $\mathcal{CH}(P)$总是位于其右侧，那么 P 中的所有点也都将落在该直线的右侧（如图 1-6 所示）。反之亦然：如果相对于由 p 和 q 确定的直线，$P \setminus \{p, q\}$中的所有点都位于右侧，那么$\overline{pq}$就是构成 $\mathcal{CH}(P)$的一条边。

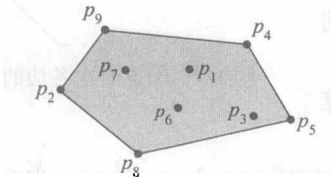


图 1-5 计算凸包

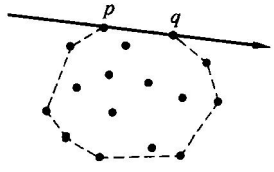


图 1-6 相对于 $\mathcal{CH}(P)$边界上任一边所在的直线，P 中所有点均居于同侧

好了，在对该问题的几何特性有了更深的理解之后，就可以构造如下一个算法了。我们通过伪代码来描述该算法，本书将统一采用这种伪代码的形式。

```
算法　SLOWCONVEXHULL(P)
输入：平面点集 P
输出：由 CH(P) 的顶点沿顺时针方向排成的队列 ℒ
1. E ← ∅
2. for(每一有序对(p, q) ∈ P × P, p ≠ q)
3.    do valid ← true
4.       for (除 p 和 q 之外的所有点 r ∈ P)
5.          do if (r 位于 p 和 q 所确定有向直线的左侧)
6.                then valid ← false
7.       if (valid) then 将有向边 pq 加入到 E
8. 根据集合 E 中的各边，找出 CH(P) 的所有顶点，并按照顺时针方向将它们组织为列表 ℒ
```

或许，你对该算法中的两个步骤还不甚清楚。

第一个出现在第5行：如何进行比较，才能判断某个点到底是位于一条有向直线的左侧，还是右侧？对大多数几何算法而言，这都是必需的基本操作之一。本书将假定这些操作都是现成的。显然，（从理论上分析）它们都可以在常数时间内完成，因此从渐进复杂度的角度看，算法的具体实现方法不会对其运行时间的数量级有何影响。但这并不等于说，这些基本操作不甚重要，或者不值一提。实际上，正确实现这些操作并非易事，而且它们对算法的实际运行时间的确会有影响。幸运的是，支持这些基本操作的软件包现已随处可得。因此总而言之，不必去担心如何实现第5行中的测试；可以假定我们已经拥有一个子函数，（通过调用该函数）可以在常数时间内完成这类测试。

该算法需要解释的另一个问题出现在最后一行。通过第2～7行的循环，可以构造出凸包的边集 E。根据 E，可以按照如下方法构造出列表 $\mathcal{L}$。E 中各边都是有向的，因此可以定义它们的起点与终点。在指定每条边的方向时，我们都使得其他所有点都位于它的右侧——这样，如果按照顺时针方向遍历（traverse）所有顶点，那么每条边的起点都会先于其终点被枚举出来。

现在如图1-7所示，在 E 中任意删除一条边 $\vec{e}_1$，将 $\vec{e}_1$ 的起点、终点分别作为第一、第二个点放入 $\mathcal{L}$；从 E 中找出以 $\vec{e}_1$ 的终点为起点的边 $\vec{e}_2$，将 $\vec{e}_2$ 从 E 中删去，并将其终点插入到当前 $\mathcal{L}$ 的末尾；再找出以 $\vec{e}_2$ 的终点为起点的边 $\vec{e}_3$，将 $\vec{e}_3$ 从 E 中删去，也将其终点插入到当前 $\mathcal{L}$ 的末尾；……。不断重复上述过程，直到 E 中只剩下最后一条边。至此已经大功告成——因为，最后这条边的终点必然就是 $\vec{e}_1$ 的起点，而该点已经加入到 $\mathcal{L}$ 中了。若直截了当地实现，这一过程需要 $O(n^2)$时间。虽然将这一复杂度改进至 $O(n\log n)$并不困难，但是毕竟算法的整体复杂度已经由其他部分决定了。

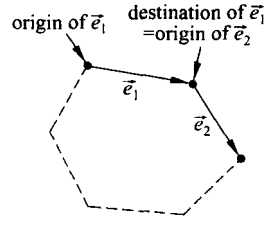


图1-7　确定 E 中各边的次序

SLOWCONVEXHULL 算法的复杂度并不难分析。总共要检查（n^2–n）对点。对每一对点，要检查其他的（n–2）个点，看看它们是否都位于（该点对所确定有向线段的）右侧。这总

共需要运行 $O(n^3)$时间。最后一步需要 $O(n^2)$时间，故总体的时间复杂度为 $O(n^3)$。在实际应用中，这样一个需要运行三次方时间的算法，除非是处理小规模的输入集，否则都会由于太慢而毫无用处。之所以会出现这种问题，是因为我们没有采用任何精巧的算法设计技术，而只是以一种蛮力的（brute-force）方式，将我们对算法的几何理解直接转换为算法。实际上，只要对该算法做进一步的审视，就不难找出改进的方法。

前面介绍了一个准则，籍以判定点对 p 和 q 是否定义了 $\mathcal{CH}(P)$的一条边。然而，在推导这个准则时，我们做得还不够细致。如图 1-8 所示，相对于由 p 和 q 所确定的直线，一个点 r 的位置并不是非左即右——有时，可能正好落在这条直线上——这就是所谓的“退化情况”（degenerate case），或者简称为“退化”（degeneracy）。对于这种情况，应该如何处理呢？在刚开始思考某个问题的时候，我们更愿意（暂时地）忽略这些情况，这样，在从问题中抽取出其几何性质的过程中，才不至于把思路搞乱。然而在实践中，这些情况都有可能发生。例如，当借助鼠标在屏幕上标定点的位置时，每个点的坐标都将局限在很窄的一段整数区间之内，因而很有可能会定义出三个共线的点。

在可能出现退化情况时，为了保证算法始终运行正确，就必须这样来重新表述上述准则：某条有向边$\overrightarrow{pq}$是 $\mathcal{CH}(P)$的一条边，当且仅当相对于由 p 和 q 所确定的有向直线，所有的其他点 $r \in P$ 或者严格地位于其右侧，或者落在开线段$\overline{pq}$上（假定 P 中没有相互重合的点）。这样，此算法第 5 行所涉及的测试，将被替换为一个更为复杂的版本。

还有一个重要的方面也被忽视了，而它却会影响到我们的算法所得出结果的正确性。不知不觉中，我们已经做了这样一个假定：只要给定一条（有向）直线，以及另外一个点，那么无论这个点是位于该直线的左侧还是右侧，我们总是能够准确地做出判断。然而，这个假设并不见得一定成立——如果各点的坐标都表示为浮点数，而且计算过程中所采用的也是浮点运算（floating point arithmetic），那么就必然存在舍入误差（rounding error），从而影响到测试的精度。

如图 1-9 所示，试想有三个点 p、q 和 r 几乎共线，而其他各点与它们都相距很远。按照上面的算法，要分别对点对(p, q)、(r, q)和(p, r)进行测试。既然这三个点几乎共线，则由于舍入误差的存在，判断的结果很有可能是：r 位于直线$\overline{pq}$的右侧，p 位于直线$\overline{rq}$的右侧，而 q 位于直线$\overline{pr}$的右侧。显然，这种几何位置关系是不可能的——然而浮点运算可不管这些！在这种情况下，算法将会把这三条边全都挑选出来。

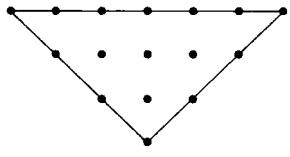


图 1-8 多点共线的退化情况

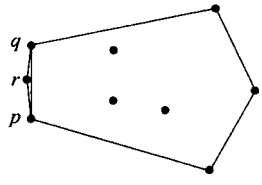


图 1-9 三点几乎共线，且与其他诸点相距足够远时，可能选出多余的边

更糟糕的情况是，这三次测试的结果也可能正好与上面相反，这样，如图 1-10 所示，算法就会将这三条边都排除掉，于是在所生成的“凸包”边界上将会出现一个缺口。

等到算法的最后一步——将凸包的顶点组织为有序表——时，这将会导致严重的错误。实际上，这一步有个假定：在凸包的每一顶点处，出边和入边都正好只有一条。然而受到舍入误差的影响，某个顶点 p 可能会有两条出边，也可能根本就没有出边。在上面那个简单的算法中，最后一行并没有考虑到对不一致数据的处理，因此，若直接按照该算法编程实现，程序就可能会崩溃。

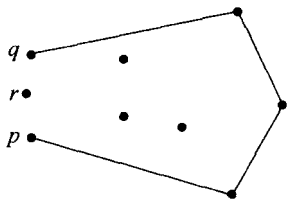


图 1-10 三点几乎共线，且与其他诸点相距足够远时，可能会遗漏边

即使已经证明该算法是正确的，而且也能够处理各种特殊情况，它可能依然称不上鲁棒（robust）——也就是说，计算过程中出现的某个微小误差，可能会导致运行失败，而且失败的形式难以预料。问题的症结在于，在证明算法正确性的时候，我们（想当然地）做了一个假设：可以精确地使用实数进行计算。

我们设计出了自己的第一个几何算法。它可以计算平面点集的凸包。然而，它的时间复杂度为 $O(n^3)$，故运行速度相当慢；它处理退化情况的能力也很差；此外，也不够鲁棒。因此，我们应该尽力去做得更好。

为此，我们将采用一种标准的算法设计模式——递增式策略——来设计一个递增式算法（incremental algorithm）。顾名思义，我们将逐一引入 P 中各点；每增加一个点，都要相应地更新目前的解。这个递增式方法将沿用几何上的习惯，按照由左到右的次序加入各点。于是，首先需要根据 x 坐标对所有点进行排序，产生一个有序的序列：$p_1,\cdots,p_n$。接下来，我们将按照这一顺序，将它们逐一引入。本来，既然是自左而右地进行处理，所以要是凸包上的顶点也能按照它们在边界上出现的次序自左向右地排列，将会更加方便。然而，情况并没有这样好。因此，我们将首先计算出构成上凸包（upper hull）的那些顶点。

如图 1-11 所示，所谓的上凸包，就是从最左端顶点 p_1 出发，沿着凸包顺时针行进到最右端顶点 p_n 之间的那段。换而言之，组成上凸包的，就是从上方界定凸包的那些边。此后，再自右向左进行一次扫描，计算出凸包的剩余部分——下凸包（lower hull）。

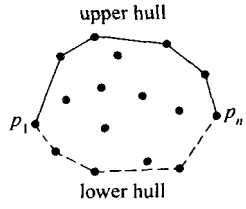


图 1-11 分别构造上凸包和下凸包

该递增式算法的基本步骤，就是在每次新引入一个点 p_i 之后，对上凸包做相应的更新。也就是说，已知点 $p_1,\cdots,p_{i-1}$ 所对应的上凸包，计算出 $p_1,\cdots,p_i$ 所对应的上凸包。可以按照如下方法进行。若按照顺时针方向沿着多边形的边界行进，则在每个顶点处都要改变方向。若是任意的多边形，则每次的转向既可能是向左，也可能向右。然而，若是凸多边形，则必然每次都是向右转。根据这一点，在新引入 p_i 之后，可以进行如下处理。令 $\mathcal{L}_{\text{upper}}$ 为从左向右存放上凸包各顶点的一个列表。首先，将 p_i 接在 $\mathcal{L}_{\text{upper}}$ 的最后——既然在目前已经加入的所有点中，p_i 是最靠右的，则它必然是（当前）上凸包的一个顶点，所以这样做无可厚非。然后，再检查 $\mathcal{L}_{\text{upper}}$ 中最末尾的三个点，看看它们是否构成一个右拐（right-turn）。若构成右拐，则大功告成，此时（更新后的）$\mathcal{L}_{\text{upper}}$ 记录了组成上凸包的各个顶点 $p_1,\cdots,p_i$，接下来，就可以继续处理下一个点——p_{i+1}。然而，若最后的三个点构成一个左拐（left-turn），就必须将中间的（即倒数第二个）顶点从上凸包中剔除出去。

若出现这种情况，需要做的可能还远不止这些——因为，此时的最后三个点可能仍然构成一个左拐（如图 1-12 所示）。果真如此，就必须再次将中间的顶点剔除掉。这一过程需要反复进行，直到位于最后的三个点构成一个右拐，或者仅剩下两个点。

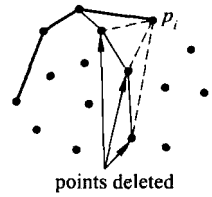


图 1-12 只要最后的三点构成左拐，即将居中的点删除

下面将给出该算法的伪代码。这段代码既计算上凸包，也计算下凸包。在完成后一项工作时，只需将各点自右向左排列，后续的计算与上凸包都是相仿的。

```
算法  ConvexHull(P)
输入：平面点集 P
输出：由 CH(P) 的所有顶点沿顺时针方向组成的一个列表
1.    根据 x 坐标，对所有点进行排序，得到序列 p1, ..., pn
2.    在 L_upper 中加入 p1 和 p2（p1 在前）
3.    for (i← 3 to n)
4.       do 在 L_upper 中加入 pi
5.          while (L_upper 中至少还有三个点，而且最末尾的三个点所构成的不是一个右拐)
6.             do 将倒数第二个顶点从 L_upper 中删去
7.    在 L_lower 中加入 pn 和 pn-1（pn 在前）
8.    for (i← n-2 downto 1)
9.       do 在 L_lower 中加入 pi
10.         while (L_lower 中至少还有三个点，而且最末尾的三个点所构成的不是一个右拐)
11.            do 将倒数第二个顶点从 L_lower 中删去
12.   将第一个和最后一个点从 L_lower 中删去
      （以免在上凸包与下凸包联接之后，出现重复顶点）
13.   将 L_lower 联接到 L_upper 后面（将由此得到的列表记为 L）
14.   return(L)
```

与上回一样，只要仔细分析，就会发现上面的算法并不正确。不用说，这里同样隐含了这样一个假设：所有点的 x 坐标互异。一旦这个假设不成立，根据 x 坐标所定义的次序就可能有歧义。幸运的是，这一问题实际上并不严重。我们只需以一种合适的方式，对这个次序进行推广——使用字典序，而不是仅仅根据各点的 x 坐标来确定其次序。也就是说，首先按照 x 坐标排序；倘若有多个点的 x 坐标雷同，则进而按照 y 坐标对它们排序。

还有一种特殊情况被忽略了：如图 1-13 所示，在对三个点进行比较，以判断它们究竟是构成一个左拐还是右拐的时候，它们有可能恰好共线。

在这种情况下，居中的那个（那些）点不应该出现在最后的凸包上——因此，应该将共线的点看成是构成一个左拐。也就是说，只有在三个点的确构成一个右拐的时候，我们的测试子程序才应返回“真”；而在其他情况下，都应返回“假”（请注意，与上面的算法所采用的测试子程序相比，在多点共线的情况下，这种测试要更加容易）。

经过如此修改，我们的算法就能够正确地计算出凸包，如图 1-14 所示，经过第一趟扫描，构造出上凸包（根据现在的定义，它是从按字典序最小的顶点出发，按照顺时针方向沿着凸包到达字典序最大顶点之间的一段路径）；第二趟扫描则构造出凸包的剩余部分。