Algorithmische Geometrie: reine Theorie?[1]

Michiel Smid

Zusammenfassung Die Algorithmische Geometrie beschäftigt sich mit dem Entwurf und der Analyse von effizienten Algorithmen für Probleme geometrischer Natur. Dieses Fachgebiet ist Mitte der siebziger Jahre als Teildisziplin der Theoretischen Informatik entstanden. In den neunziger Jahren hat sich das Fachgebiet drastisch verändert. Es werden Probleme, die durch praktische Anwendungen motiviert sind, untersucht, und immer mehr Nachdruck wird auf die Implementierung geometrischer Algorithmen gelegt. In diesem Beitrag wird eine Einführung in die Algorithmische Geometrie gegeben, und einige Forschungsergebnisse, die an der Magdeburger Universität erzielt wurden, werden kurz vorgestellt.

1 Einführung

Die Algorithmische Geometrie beschäftigt sich mit dem Entwurf und der Analyse von effizienten Algorithmen für geometrische Probleme. Die Daten, die bei solchen Problemen verarbeitet werden, sind geometrische Objekte, wie z.B. Punkte, Liniensegmente oder Dreiecke. Ein anschauliches Beispiel ist das *post-office Problem*. Hier hat man eine Menge S von Punkten (Postämter) in der Ebene gegeben, und möchte diese so in einer Datenstruktur speichern, daß *post-office queries* effizient beantwortet werden können. Bei so einem query ist ein beliebiger Anfragepunkt q gegeben, und man sucht den Punkt von S, der am nächsten bei q liegt.

Eine Lösung für dieses Problem besteht aus (i) einer Datenstruktur, die die Punkte speichert, (ii) einem Algorithmus, mit dem Anfragen beantwortet werden und (iii) einem Algorithmus, der diese Datenstruktur aufbaut.

Eine einfache Lösung besteht darin, daß die Punkte in einer Liste gespeichert werden. Ein query mit Anfragepunkt q wird dann beantwortet, indem alle Punkte der Liste nacheinander betrachtet werden, und der Punkt, der am nächsten an q liegt, zurückgegeben wird. Es ist klar, daß die *Anfragezeit*, d.h. die Anzahl der elementaren "Schritte", die benötigt werden, um eine Anfrage zu beantworten, linear von der Anzahl der Punkte der Menge S abhängt. Wenn wir die Größe von S mit n bezeichnen, dann gibt es also eine positive Konstante c, so daß eine beliebige Anfrage in höchstens cn Schritten beantwortet werden kann. Wir sagen, daß die Anfragezeit dieses einfachen Algorithmus durch $O(n)$ begrenzt ist[2].

[1] Antrittsvorlesung vom 15. Mai 1997. Der Autor ist seit Dezember 1996 als Professor für Theoretische Informatik an der Magdeburger Universität tätig. Von 1991 bis Anfang 1996 war er wissenschaftlicher Mitarbeiter am Max-Planck-Institut für Informatik in Saarbrücken, wo er sich 1995 habilitierte. In 1996 bis zu seiner Berufung in Magdeburg war er Lecturer am King's College in London.

[2] Wir sagen, daß die Laufzeit eines Algorithmus durch $O(f(n))$ begrenzt ist, wenn es positive Konstanten n_0 und c gibt, so daß für jede Eingabemenge der Größe n, wobei $n \geq n_0$, die Laufzeit höchstens $c \cdot f(n)$ ist.

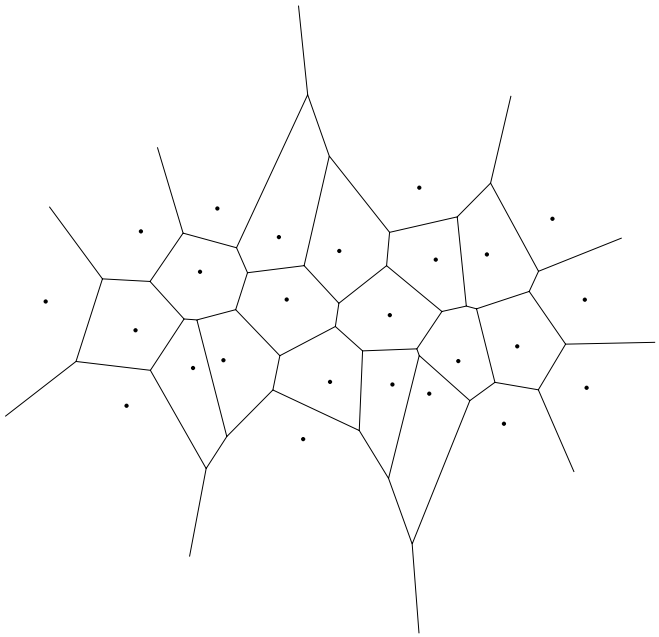

Abbildung 1: *Ein Voronoi Diagramm.*

Wenn die Anzahl der Anfragen, die beantwortet werden müssen, klein ist, ist die obengenannte Lösung für das post-office Problem akzeptabel. Wenn die Anzahl der Anfragen aber groß ist, stellt sich die Frage, ob es "bessere" Lösungen gibt. Eine bessere Lösung wird durch das *Voronoi Diagramm*, das wie folgt definiert ist, erhalten. Wir numerieren die Punkte von S als $p_1, p_2, \ldots, p_n$. Jedem Punkt p_i wird eine Region, die sogenannte *Voronoi Region* $VR(p_i)$ von p_i, zugeordnet, die aus allen Punkten in der Ebene, die näher an p_i, als an allen anderen Punkten von S liegen, besteht. Wenn wir den euklidschen Abstand zwischen zwei Punkten p und q mit $|pq|$ bezeichnen, dann gilt also,

$$VR(p_i) = \{x \in \mathbb{R}^2 : |p_i x| \leq |p_j x| \text{ für alle } j\}.$$

Das Voronoi Diagramm von S ist die Menge aller Voronoi Regionen $VR(p_i)$, $1 \leq i \leq n$. (Siehe Abbildung 1.)

Eine Voronoi Region kann auch auf folgende Weise erhalten werden. Für jedes i und j, $i \neq j$, sei H_{ij} die Menge aller Punkte in der Ebene, die näher an p_i als an p_j liegen. H_{ij} ist die Halbebene, die die Mittelsenkrechte von p_i und p_j als Rand hat, und die den Punkt p_i enthält. Es gilt

$$VR(p_i) = \bigcap_{j \neq i} H_{ij}.$$

Hieraus folgt, daß eine Voronoi Region ein (möglichst unbegrenztes) konvexes Polygon ist.

Wie kann dieses Voronoi Diagramm benutzt werden, um post-office Anfragen zu beantworten? D.h. wie beantwortet der Query-Algorithmus Anfragen? Sei q ein beliebiger Anfragepunkt. Da die Voronoi Regionen die Ebene zerlegen, gibt es einen Index i, so daß $VR(p_i)$ den Punkt q enthält. Es folgt dann direkt aus der Definition des Voronoi Diagramms, daß q näher an p_i als an allen anderen Punkten von S liegt. Der Query-Algorithmus muß also im Prinzip nur sogenannte *point location queries* lösen.

Die Behauptung, daß das Voronoi Diagramm eine gute (d.h. effiziente) Lösung für das post-office Problem ist, folgt aus den folgenden drei Tatsachen:

Wir sehen in Abbildung 1, daß das Voronoi Diagramm aus Knoten, Kanten und Regionen besteht. Die Anzahl der Regionen ist natürlich genau n. Man kann mit Hilfe der Eulerschen Formel für planare Graphen beweisen, daß das Voronoi Diagramm aus höchstens $2n-4$ Knoten und höchstens $3n-6$ Kanten besteht. Der Speicherplatzbedarf des gesamten Diagramms ist also nur linear von n abhängig.

Es gibt effiziente Algorithmen, die das Voronoi Diagramm in $O(n \log n)$ Zeit berechnen. Außerdem kann das Voronoi Diagramm so gespeichert werden, daß point location queries in $O(\log n)$ Zeit beantwortet werden können.

Das Beispiel des post-office Problems zeigt, daß für die Algorithmische Geometrie u.a. Techniken und Ergebnisse aus den folgenden Teilgebieten der Informatik und Mathematik benötigt werden:

- Geometrie: Die Daten, mit denen gerechnet wird, sind geometrischer Natur.
- Datenstrukturen: In der oben skizzierten Lösung wird der "Lösungsraum", d.h. die Ebene $\mathbb{R}^2$, in Regionen zerlegt, die dann entsprechend in einer Datenstruktur gespeichert werden. Im allgemeinen werden die Daten oder der Lösungsraum geordnet bzw. strukturiert und entsprechend in einer Datenstruktur gespeichert, so daß Anfragen effizient beantwortet werden können.
- Algorithmen: Diese werden benutzt um die geeigneten Datenstrukturen zu konstruieren und die Anfragen zu beantworten.
- Diskrete Mathematik: Die Effizienz eines Algorithmus gibt die Anzahl der elementaren Schritte an, die benötigt werden um das Problem zu lösen. Die Analyse eines Algorithmus, d.h. die Bestimmung der Effizienz, führt zu kombinatorischen Problemen. In unserem Beispiel wurde z.B. die Eulersche Formel benutzt, um nachzuweisen, daß nur linearer Speicherplatz benötigt wird, um das Voronoi Diagramm zu speichern.

2 Seit wann gibt es die Algorithmische Geometrie?

Alte Völker, wie die Chinesen, Inder, Babylonier, Ägypter und Griechen, beschäftigten sich schon mit geometrischen Problemen. Grund dafür war die Tatsache, daß diese Probleme im Alltag häufig auftraten, z.B. bei der Festlegung von Steuern von Grundstücken und dem Entwurf der Pyramiden.

In seinen *Elementen* baut Euklid u.a. die Geometrie an Hand von Konstruktionen mit Zirkel und Lineal auf. Hier sehen wir ein frühes Beispiel eines *Berechenbarkeitsmodells*, in dem die elementaren Operationen klar definiert sind. Zu Euklids Zeit stellte man sich die Frage, ob alle denkbaren geometrischen Konstruktionen mit Zirkel und Lineal ausgeführt werden können. Erst nachdem die Algebra hinreichend entwickelt worden war, konnte Abel 1828 beweisen, daß dies nicht der Fall ist: Es gibt keine Konstruktion mit Zirkel und Lineal, die einen beliebigen Winkel in drei gleiche Teile aufteilt.

Anfang dieses Jahrhunderts wurde die *Komplexität* von geometrischen Konstruktionen zum ersten Mal in Betracht gezogen. Euklid hat eine Konstruktion angegeben, die in 508 elementaren Schritten das Kreiseproblem von Apollonius löst: Konstruiere einen Kreis, der

an drei gegebenen Kreisen tangent ist. Im Jahre 1902 zeigte Lemoine, daß dieses Problem in "weniger als 200 Schritten" gelöst werden kann.

Erst mit der Entwicklung des Computers kam der Bedarf an Algorithmen, die Probleme effizient lösen. Bekannte effiziente Algorithmen aus den Anfangsjahren der Informatik sind mergesort, quicksort und die Binärsuche.

Ende der sechziger Jahre brauchten Forscher am Bell Labs einen Algorithmus, der die konvexe Hülle einer Menge von $n \approx 10.000$ Punkten berechnet. Es wurde festgestellt, daß der bis dahin benutzte Algorithmus, der eine quadratische Laufzeit hatte, zu langsam war. Daraufhin entwickelte Graham seinen berühmten *Graham's Scan*, der die konvexe Hülle in nur $O(n \log n)$ Zeit berechnet. Dieser Algorithmus wurde 1972 veröffentlicht [10], und ist somit eine der ersten Veröffentlichungen der Algorithmischen Geometrie.

Als Anfang der Algorithmischen Geometrie wird aber die Doktorarbeit *Computational Geometry* [23] von Michael Shamos aus 1978 betrachtet. Diese Arbeit enthält effiziente Algorithmen für eine Fülle von grundlegenden geometrischen Problemen. So gibt Shamos z.B. einen Algorithmus an, der in $O(n \log n)$ Zeit das Voronoi Diagramm einer Punktemenge berechnet. Aus dieser Doktorarbeit ist das inzwischen klassische Buch [20], das Shamos 1985 gemeinsam mit Preparata veröffentlichte, enstanden.

Seither hat sich das Fachgebiet sehr rasch entwickelt. In 1985 fand die erste *Annual ACM Symposium on Computational Geometry* statt. 1989 folgte die jährlich stattfindende *Canadian Conference on Computational Geometry.* Alle Tagungen, wo die neuesten Ergebnisse der Theoretischen Informatik vorgestellt werden, haben öfters mehrere Sessions über geometrische Algorithmen.

1986 erschien die erste Ausgabe der hochrangigen Zeitschrift *Discrete & Computational Geometry.* 1991 folgten dann die zwei Zeitschriften *Computational Geometry, Theory and Applications* und *International Journal of Computational Geometry & Applications.*

3 Einige Techniken der Algorithmischen Geometrie

Zuerst betrachten wir noch einige Beispiele von Problemen, die mit Hilfe von geometrischen Algorithmen effizient gelöst werden können.

Point Location: Man hat einen planaren Graphen gegeben, der in die Ebene eingebettet ist. Gesucht ist eine Datenstruktur, die diesen Graphen so speichert, daß für einen beliebigen Anfragepunkt q, die Region des Graphen, die q enthält, effizient bestimmt werden kann.

Wir haben dieses Problem bereits in Abschnitt 1 gesehen. Es tritt auch in der folgenden Situation auf. Nehmen wir an, wir haben auf unserem Bildschirm viele Fenster geöffnet. Jedes Mal, wenn wir mit der Maus klicken, muß der Rechner herausfinden, in welchem Fenster der Mauszeiger sich befindet. Dies ist natürlich genau das point location Problem.

Hidden Surface Removal: Dieses Problem stammt aus der Computergraphik. Wir haben eine Menge von dreidimensionalen Objekten gegeben und möchten die Teile der Objekte, die ein Betrachter sieht, berechnen. Das Problem wird insbesondere interessant, wenn der Betrachter sich bewegt. In diesem Fall werden Teile, die vorher nicht sichtbar waren, sichtbar, und umgekehrt. Bei einer "Fahrt" durch ein (virtuelles) Gebäude ist es wichtig, daß die Änderungen in der sichtbaren Szene in Echtzeit berechnet werden.

Range Queries: Diese Anfragen treten u.a. im Datenbankbereich auf. Betrachten wir eine Datenbank, in der alle Mitarbeiter der Universität gespeichert sind. Für jede Person wird u.a. das Alter und das Gehalt gespeichert. Eine typische Anfrage wäre "finde alle Mitarbeiter mit Alter zwischen 40 und 45 Jahren und mit Gehalt zwischen 50.000 und 60.000 DM".

Wenn jeder Mitarbeiter durch einen Punkt in der Ebene, mit x-Koordinate sein Alter und y-Koordinate sein Gehalt, dargestellt wird, dann kann unsere Anfrage wie folgt formuliert werden: Finde alle Punkte, die in dem achsenparallelen Rechteck $[40; 45] \times [50.000; 60.000]$ liegen.

Geometrische Netzwerke: Hierzu gehören die geometrischen Varianten der wohlbekannten Optimierungsprobleme der Graphentheorie. Beispiele sind die Berechnung des minimalen aufspannenden Baumes oder der kürzesten traveling salesman tour einer Menge von Punkten.

Aus den oben erwähnten Beispielen wird klar, daß geometrische Algorithmen mit unterschiedlichen geometrischen Objekten arbeiten. Somit entsteht eine erste Schwierigkeit. Algorithmen, die reelle Zahlen als Eingabe bekommen, nutzen meistens die natürliche $\leq$-Ordnung aus. Ein Beispiel dafür ist die Verwendung von binären Suchbäumen. Bei geometrischen Objekten, wie z.B. eine Menge von Dreiecken im Raum, gibt es a priori keine "natürliche" Ordnung. Es werden also Techniken (d.h. Algorithmen und Datenstrukturen) benötigt, die eine "effiziente" Ordnung konstruieren und diese dann entsprechend benutzen. In den Anfangsjahren der Algorithmischen Geometrie wurden viele solcher Techniken entwickelt. Wir nennen hier einige Beispiele.

Teile-und-Herrsche: Diese Technik ist z.B. von mergesort bekannt. Sie wurde auf folgende Weise verallgemeinert. Sei $P(n, d)$ ein Problem für n Objekte im d-dimensionalen Raum. Mit Hilfe von Teile-und-Herrsche wird dieses Problem auf zwei Probleme der Form $P(n/2, d)$ und ein "ähnliches" Problem der Form $P'(n, d-1)$ zurückgeführt.

Beispiele von Problemen, die mit dieser Technik effizient gelöst werden können, sind die Berechnung des kleinsten Abstandes in einer Menge von d-dimensionalen Punkten und die Konstruktion des Voronoi Diagramms einer Menge von Punkten in der Ebene.

Plane Sweep: Hier soll ein Problem für eine Menge von Objekten in der Ebene gelöst werden. Die Technik besteht darin, daß eine vertikale Gerade—die *sweep line*—von links nach rechts über die Objekte bewegt wird. Während dieser Bewegung gilt die Invariante, daß das Problem für alle Objekte, die links der sweep line liegen, bereits vollständig gelöst worden ist.

Die zwei obengenannte Probleme können mit dieser plane sweep Technik effizient gelöst werden. Ein anderes Beispiel ist die Berechnung der Schnittpunkte in einer Menge von Liniensegmenten.

Mehrstufige Datenstrukturen: Dies ist eine Verallgemeinerung von binären Suchbäumen. Die Datenstruktur besteht aus einem balancierten binären Suchbaum, dem sogenannten Primärbaum, in dem jeder Knoten drei Zeiger besitzt: zwei Zeiger führen zu den beiden Kindern, während der dritte Zeiger zu einem "Sekundärbaum" führt.

Wir verdeutlichen diese Technik an Hand von *range trees* [15], die 1978 von Lueker eingeführt wurden. Sei S eine Menge von Punkten in der Ebene. Der Primärbaum speichert die Punkte von S in den Blättern und zwar sortiert nach den x-Koordinaten. Für jeden Knoten

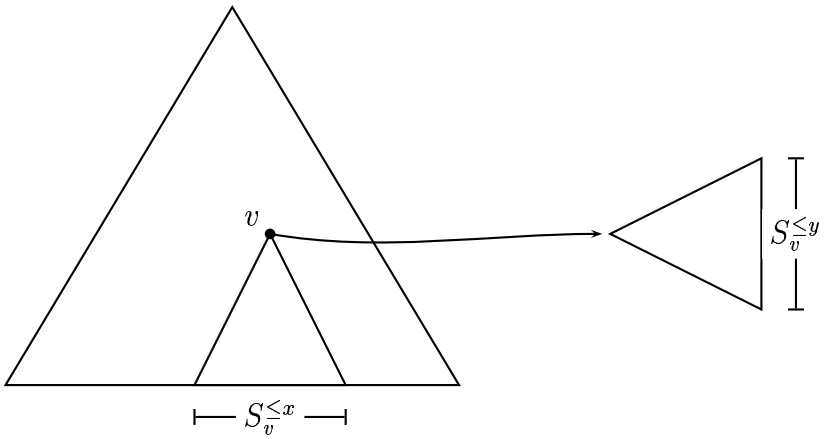

Abbildung 2: *Ein zweidimensionaler range tree. Jeder Knoten v des Primärbaumes enthält einen Zeiger auf einen Sekundärbaum.*

---

v bezeichnen wir mit S_v die Menge aller Punkte von S, die in dem Unterbaum von v gespeichert sind. Jeder innere Knoten v des Primärbaumes enthält die folgende Informationen (siehe Abbildung 2):

- Einen Zeiger auf das linke Kind und einen Zeiger auf das rechte Kind von v.
- Den Punkt von S, der im rechten Blatt des linken Unterbaumes von v gespeichert ist. Mit Hilfe dieser Information kann in dem Primärbaum ein beliebiger Punkt gesucht werden.
- Einen Sekundärbaum. Dies ist ein balancierter binärer Suchbaum, der alle Punkte von S_v speichert und zwar sortiert nach y-Koordinaten.

Mit Hilfe dieser Datenstruktur können die oben genannten range queries effizient gelöst werden. Sei n die Anzahl der Punkte von S. Das Problem, die Punkte von S, die in dem Rechteck $R = [a_1; a_2] \times [b_1; b_2]$ liegen, zu finden, wird zurückgeführt auf das Problem, in $O(\log n)$ vielen Sekundärbäumen, die Punkte, deren y-Koordinaten zwischen b_1 und b_2 liegen, zu finden. Wenn K die Gesamtzahl der Punkte, die in R liegen, bezeichnet, dann können diese Punkte in $O(\log^2 n + K)$ Zeit berechnet werden. Es ist leicht einzusehen, daß der Speicherplatzbedarf des zweidimensionalen range tree durch $O(n \log n)$ begrenzt ist.

Fractional Cascading: Wir haben gesehen, daß mit Hilfe von range trees, zweidimensionale range queries auf logarithmisch viele eindimensionale range queries reduziert werden können. Dabei soll beachtet werden, daß bei dieser Folge von eindimensionalen queries der Suchbereich $[b_1; b_2]$ immer derselbe ist. Fractional Cascading [6], 1986 von Chazelle und Guibas eingeführt, ist eine Technik, die diese Beobachtung ausnutzt: Die Sekundärstrukturen werden so verknüpft, daß die Gesamtfolge der queries in $O(\log n + K)$, statt $O(\log^2 n + K)$, Zeit beantwortet werden können.

4 Die Algorithmische Geometrie als theoretische Disziplin

Anfang der neunziger Jahre waren eine sehr große Anzahl von algorithmischen Techniken bekannt, mit deren Hilfe viele geometrischen Probleme effizient gelöst werden können. Das Fachgebiet blieb aber ein Teilgebiet der Theoretischen Informatik, und die entwickelten Algorithmen fanden kaum ihren Weg in die Praxis. Es werden einige Gründe hierfür angegeben:

- Die Algorithmen wurden immer komplizierter, indem andere Algorithmen als black-box benutzt wurden, die selbst auch wieder Algorithmen als black-box benutzen, usw. Außerdem lag der Nachdruck bei der asymptotischen Komplexität der Algorithmen. Weil diese immer komplizierter wurden, wurden die Konstanten, die in der O-Notation "versteckt" sind, immer größer. Zum Schluß waren asymptotische "Verbesserungen" manchmal sehr marginal.

  Als Nachweis dafür, daß der Autor sich auch an dieser Praxis schuldig gemacht hat, sei ein gemeinsames Ergebnis mit Sanjiv Kapoor erwähnt: In [12] entwickeln wir eine Datenstruktur der Größe $O(n \log n/(\log\log n)^k)$, die eine Menge von n Punkten so speichert, daß der kleinste Abstand nach Einfügen bzw. Löschen eines Punktes in $O(\log n \log\log n)$ Zeit neu berechnet werden kann. Hierbei ist k eine beliebige, aber konstante, natürliche Zahl.

- Es wurden manchmal künstliche Probleme, die keine praktische Relevanz haben, betrachtet.

- Als Berechenbarkeitsmodell wurde meistens das *algebraische Berechnungsbaummodell* genommen. In diesem Modell können Funktionen, wie x/y und $\sqrt{x}$, *exakt* berechnet werden. Man kümmerte sich also nicht um *numerische* Probleme.

- Oft wurde angenommen, daß die geometrische Objekte sich in *allgemeiner Lage* befinden. Als Beispiel kehren wir nochmal zu dem Voronoi Diagramm zurück. Bei der Konstruktion dieses Diagramms liefert der Fall, wenn vier oder mehr Punkte auf einem Kreis liegen, große Probleme. Insbesondere bei einer Implementierung führt dies zu Fallunterscheidungen, die getrennt behandelt werden müssen. Weil geometrische Algorithmen aber nur aus theoretischer Sicht betrachtet wurden, wurde oft einfach angenommen, daß solche Fälle nicht vorkommen.

Insbesondere die letzten beiden Punkte hatte zur Folge, daß die meisten geometrischen Algorithmen nicht implementiert wurden.

5 Die moderne Algorithmische Geometrie

Auf Grund der oben genannten Punkte, und zum Teil auch wegen des schlechten Arbeitsmarktes für Theoretiker, erschien 1996 der Bericht *Application Challenges to Computational Geometry: CG Impact Task Force Report* [5]. In diesem Bericht wird aufgerufen, sich nicht nur mit den theoretischen Aspekten von geometrischen Algorithmen zu beschäftigen, sondern auch praxisrelevante Aspekte in Betracht zu ziehen. Zum einen sollten Algorithmen entworfen werden, die (i) implementierbar sind, (ii) keine Annahme über die Eingaben machen und (iii)

numerisch stabil sind. Zum anderen sollten Algorithmen für geometrische Probleme, die für die Praxis relevant sind, entwickelt werden.

Somit hat das Fachgebiet der Algorithmischen Geometrie sich in den letzten Jahren drastisch verändert. Einige dieser Veränderungen sind:

- Die Anwendung von *Randomisierung*. Ende der achtziger Jahre haben Clarkson und Shor in [7, 8] bereits gezeigt, daß Randomisierung, d.h. Algorithmen, deren Ablauf durch Zufallszahlen bestimmt wird, zu effizienten und relativ einfachen Algorithmen führt. Für eine ausführliche Diskussion solcher Algorithmen sei auf das Buch von Mulmuley [18] verwiesen.

- Die Entwicklung von Softwarebibliotheken.

  - Am Max-Planck-Institut für Informatik in Saarbrücken wird seit 1988 die *Library of Efficient Data Types and Algorithms (LEDA)* entwickelt. Diese Bibliothek enthält Implementierungen aller Standardalgorithmen für kombinatorische Probleme. Siehe

    `http://www.mpi-sb.mpg.de/LEDA/leda.html`

  - Verschiedene Forschergruppen in der EU entwickeln gemeinsam eine *Computational Geometry Algorithms Library (CGAL)*, die alle grundlegenden Algorithmen aus der Algorithmischen Geometrie enthalten soll. Siehe

    `http://www.cs.ruu.nl/CGAL/`

- Die Entwicklung von robusten Algorithmen. Numerische Probleme und die Behandlung von speziellen Fällen werden immer mehr in Betracht gezogen. LEDA und CGAL bieten z.B. die Möglichkeit, geometrische Prädikate, wie "liegt der Punkt p unterhalb von, oberhalb von oder auf der gerichteten Geraden durch die Punkte q und r?" *exakt* zu lösen. Dies erlaubt es, bei der Implementierung von Algorithmen effiziente Datenstrukturen, deren Korrektheit wesentlich von der Implementierung solcher Prädikate abhängt, zu benutzen. Für einige Beispiele sei auf die Arbeiten [21, 22], die zum Teil an der Universität Magdeburg enstanden, verwiesen.

- In 1996 fand ein *ACM Workshop on Applied Computational Geometry* statt. Seit 1997 besteht die Annual ACM Symposium on Computational Geometry aus einem *theoretical* und einem *applied track*. Seit 1997 gibt es einen *Workshop on Algorithm Engineering*.

6 Algorithmische Geometrie an der Magdeburger Universität

In den nächsten zwei Abschnitten werden die Forschungsthemen auf dem Gebiet der Algorithmischen Geometrie, die am Institut für Simulation und Graphik bearbeitet werden, kurz vorgestellt.

6.1 Geometrische Netzwerke

Das allgemeine Problem kann wie folgt formuliert werden. Gegeben sei eine Menge S von n Punkten in der Ebene; berechne ein "gutes" Netzwerk, das diese Punkte verbindet. Beispiele

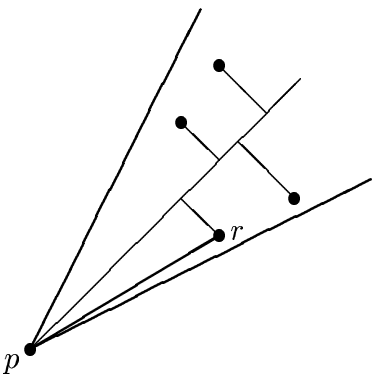

Abbildung 3: *Der θ-Graph.*

---

solcher Netzwerke sind der minimale aufspannende Baum und die traveling salesman tour; ihre Konstruktion wurde in der Vergangenheit intensiv untersucht. Unser Ziel ist, ein Netzwerk zu konstruieren, so daß jedes Paar von Punkten durch einen "kurzen" Pfad verbunden ist.

Um dies formal definieren zu können, müssen einige Begriffe eingeführt werden. Der euklidsche Abstand zwischen zwei Punkten p und q wird wieder mit $|pq|$ bezeichnet. Sei $G = (S, E)$ ein gerichteter Graph mit Knotenmenge S und Kantenmenge E. Ein *Pfad* zwischen den Knoten p und q ist eine Folge $p_0, p_1, \ldots, p_k$ von Knoten von S, so daß $p_0 = p$, $p_k = q$, und $(p_i, p_{i+1}) \in E$ für jedes i, $0 \leq i < k$. Die *Länge* dieses Pfades ist definiert als $\sum_{i=0}^{k-1} |p_i p_{i+1}|$. Der *Abstand $d_G(p, q)$ in G* zwischen den Knoten p und q ist definiert als die Länge eines kürzesten Pfades zwischen p und q.

Definition 1 Sei $t > 1$ eine reelle Konstante. Ein Graph $G = (S, E)$ heißt *t-spanner* für S, falls

$$d_G(p, q) \leq t \cdot |pq|$$

für jedes Paar p, q von Knoten.

Wenn wir von einem Knoten p zu einem anderen Knoten q reisen wollen, müssen wir mindestens den Abstand $|pq|$ zurücklegen. In einem t-spanner gibt es immer einen Pfad, der höchstens die Länge $t \cdot |pq|$ hat, d.h. der "Umweg" ist klein.

Es ist klar, daß ein t-spanner immer existiert: Wir können für G den vollständigen Graphen, in dem jedes Paar von Knoten durch eine Kante verbunden ist, nehmen. Dies hat aber als Nachteil, daß die Menge E quadratisch viele Kanten besitzt. Somit stellt sich die Frage, ob für jede Punktemenge S und jede Konstante $t > 1$, einen t-spanner mit nur *linear* vielen Kanten existiert.

Die folgende Konstruktion, von Keil und Gutwin [13] vorgestellt, zeigt, daß solche t-spanners in der Tat existieren. Die Idee besteht darin, zu garantieren, daß für jedes Paar p, q von Knoten, eine Kante von p "in die Richtung" von q existiert.

Die Kanten werden wie folgt gewählt. Sei θ ein (kleiner) Winkel. Wir betrachten für jeden Punkt $p \in S$ eine Folge von $2\pi/\theta$ vielen Kegeln mit Spitze p und Winkel θ, die die ganze Ebene überdecken. Für jeden Punkt p und jeden dieser Kegel K, wählen wir den Punkt $r \neq p$, der in K liegt, und dessen Projektion auf die Winkelhalbierende von K am nächsten an p liegt. Die Kante (p, r) wird dann in die Kantenmenge E eingefügt. (Siehe Abbildung 3.) Der so resultierende Graph $G = (S, E)$ wird θ-Graph genannt.

Es ist klar, daß der θ-Graph höchstens $(2\pi/\theta)\,n$ Kanten hat. Wenn wir den Winkel θ also als Konstante betrachten, dann hat die Kantenmenge E lineare Größe.

Betrachten wir jetzt zwei Punkte p und q. Wir konstruieren einen Pfad zwischen p und q, wie folgt. Wir nehmen den Kegel K mit Spitze p, der q enthält. Unsere Kantenmenge E enthält eine Kante (p, r), wobei r in K liegt. Wir können also von p nach r gehen. Als nächstes nehmen wir den Kegel K' mit Spitze r, der unser Ziel q enthält, und gehen dann die Kante (r, s) entlang, für den s in K' liegt. Wir gehen so weiter, bis wir den Punkt q erreichen.

Das folgende Lemma impliziert, daß der so konstruierte Pfad in der Tat ein spanner-Pfad ist.

Lemma 1 *Sei θ ein Winkel, so daß $0 < \theta < \pi/4$. Sei K der Kegel mit Spitze p, der den Punkt q enthält. Sei r ein Punkt in K, dessen Projektion auf die Winkelhalbierende von K näher an p liegt als die Projektion von q auf diese Halbierende. Dann gilt*

$$|rq| \leq |pq| - (\cos\theta - \sin\theta)|pr|.$$

Aus diesem Lemma folgt, daß der von uns konstruierte Pfad immer näher an q kommt. Insbesondere folgt, daß der Punkt q erreicht wird, d.h. der Algorithmus terminiert. Sei $p = p_0, p_1, p_2, \ldots, p_k = q$ der berechnete Pfad zwischen p und q. Dann gilt

$$|p_{i+1}q| \leq |p_iq| - (\cos\theta - \sin\theta)|p_ip_{i+1}|.$$

Hieraus folgt die folgende Abschätzung für die Länge des Pfades:

$$\begin{aligned}
\sum_{i=0}^{k-1} |p_ip_{i+1}| &\leq \sum_{i=0}^{k-1} \frac{1}{\cos\theta - \sin\theta}\,(|p_iq| - |p_{i+1}q|) \\
&= \frac{1}{\cos\theta - \sin\theta}\,(|p_0q| - |p_kq|) \\
&= \frac{1}{\cos\theta - \sin\theta}\,|pq|.
\end{aligned}$$

Der θ-Graph ist also ein t-spanner für $t = 1/(\cos\theta - \sin\theta)$. Wenn wir den Winkel θ hinreichend klein wählen, dann kommt t beliebig nahe an eins. Hiermit haben wir also gezeigt, daß es für jede Konstante $t > 1$, einen t-spanner mit linear vielen Kanten gibt. Zum Schluß sei noch erwähnt, daß der θ-Graph mit einem einfachen plane sweep Algorithmus in $O(n \log n)$ Zeit konstruiert werden kann.

Der hier vorgestellte Graph hat verschiedene Nachteile. Obwohl der Ausgangsgrad jedes Knotens durch $2\pi/\theta$ begrenzt ist, kann der Eingangsgrad groß sein. Zweitens kann die Gesamtlänge aller Kanten des Graphen sehr groß werden. Drittens kann es sein, daß der von uns konstruierte Pfad zwischen p und q aus vielen Kanten besteht.

In Zusammenarbeit mit Sunil Arya (Hong Kong), Gautam Das (Memphis), David Mount (Maryland) und Jeff Salowe (San Jose) wurden Algorithmen entwickelt, die

- t-spanners konstruieren, in dem sowohl der Eingangsgrad als auch der Ausgangsgrad jedes Knotens durch eine Konstante begrenzt ist [2, 4].

- t-spanners mit $O(n)$ vielen Kanten konstruieren, in dem jedes Paar von Knoten durch einen spanner-Pfad, der höchstens $O(\alpha(n))$ viele Kanten enthält, verbunden ist [2, 3].

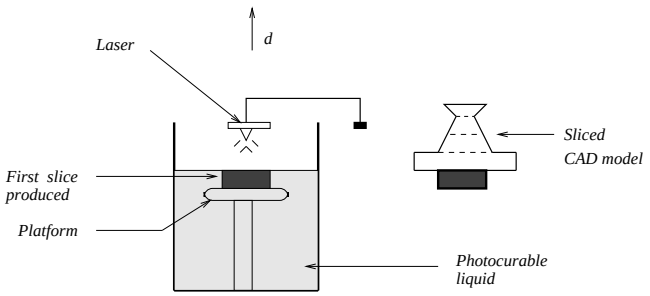

Abbildung 4: *Layered Manufacturing.*

Hierbei ist $\alpha(n)$ die Inverse der Ackermann Funktion. Obwohl diese Funktion gegen unendlich geht, gilt

$$\alpha(n) \leq 4 \text{ für alle } n \leq \underbrace{2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}}_{2048\ 2\text{'s}}^{2048} .$$

Man kann beweisen, daß dieses Ergebnis optimal ist für spanners mit linear vielen Kanten.

- t-spanners konstruieren, für den die Gesamtlänge aller Kanten proportional zu dem Gewicht des minimalen aufspannenden Baumes ist [2].

Zum Schluß seien noch einige neuere Ergebnisse erwähnt. Arikati *et al.* [1] betrachten das spanner Problem für Hindernisse. Sie zeigen, wie t-spanners konstruiert werden können, so daß jedes Paar von Knoten durch einen kurzen Pfad, der keine Hindernisse schneidet, verbunden ist. In Levcopoulos *et al.* [14] werden Algorithmen entwickelt, die *fehlertolerante* spanners konstruieren. Wenn in so einem spanner eine kleine Anzahl von Knoten oder Kanten ausfällt, dann sind die restlichen Knoten immer noch durch kurze Pfade verbunden.

Für weitere Algorithmen zur Lösung allgemeiner Abstandsprobleme wird auf die Übersichtsarbeit [24] verwiesen.

6.2 Geometrische Optimierungsprobleme für Rapid Prototyping

Im Bereich *Rapid Prototyping und Manufacturing* treten viele geometrische Optimierungsprobleme auf. Ein grundlegendes Problem, das wir in Magdeburg untersuchen, ist der Entwurf von Algorithmen für das folgende Problem. Man hat ein CAD-Modell eines Objektes gegeben, und möchte einen Prototypen mit Hilfe von *Layered Manufacturing* [11] bauen. Dieser Prototyp wird mit Hilfe eines Lasers schichtweise konstruiert. (Siehe Abbildung 4.) Da eine neue Schicht manchmal über die vorige Schicht hinaussteht, benutzt man sogenannte *supports* (Stützen), die dann später entfernt werden.

Die Zeit, die benötigt wird um den Protoypen zu bauen, und dessen Qualität, hängt u.a. von der Orientierung des Objektes ab. Diese Orientierung beeinflußt z.B.

- die Anzahl der Schichten. Da die Bauzeit proportional zu dieser Anzahl ist, ist es sinnvoll eine Baurichtung zu wählen, für die die Höhe des Objektes klein ist.

- das Volumen der supports. Weil diese supports am Ende des Prozesses weggeworfen werden, möchte man das Volumen klein halten.

- die Teile des Modells, die mit supports in Berührung sind. Weil diese Teile am Ende des Prozesses nachgearbeitet werden müssen, sucht man eine Orientierung, so daß die Gesamtfläche aller Teile, die mit supports in Berührung sind, möglichst klein ist.

- der sogenannte Treppeneffekt. Weil die Schichten eine feste Höhe haben, sehen die Flächen des Prototyps treppenförmig aus. Wieder gilt, daß dieser Treppeneffekt klein gehalten werden kann, indem das Modell geeignet orientiert wird.

In der Praxis wird die Orientierung von einem Experten "aus dem Bauch heraus" gewählt. In einem gemeinsamen Forschungsprojekt mit Ravi Janardan und Jayanth Majhi (beide aus Minneapolis), haben Jörg Schwerdt und der Autor sich das Ziel gesetzt, Algorithmen zu entwerfen, die eine gute Orientierung rechnergestützt berechnen.

Das Problem der Schichtminimierung kann wie folgt betrachtet werden. Nehmen wir an, daß das Modell ein dreidimensionales Polyeder $\mathcal{P}$ ist. Die *Breite* von $\mathcal{P}$ ist der kleinst mögliche Abstand zwischen zwei parallelen Ebenen, die $\mathcal{P}$ umschließen. Wenn diese zwei Ebenen den Vektor $\mathbf{d}$ als Normale haben, dann ist dies die Baurichtung, für die die Anzahl der Schichten minimal ist. Damit haben wir also das Schichtminimierungsproblem auf die Berechnung der Breite des Polyeders reduziert.

Es ist klar, daß die Breite von $\mathcal{P}$ gleich der Breite der konvexen Hülle KH der Knoten von $\mathcal{P}$ ist. Mit Hilfe einer *dualen Darstellung* dieser konvexen Hülle, wobei jede Fläche von KH durch seine Normale als Punkt auf der Einheitskugel dargestellt wird, kann das Breiteproblem auf die folgenden Probleme zurückgeführt werden:

- Gegeben zwei planare Graphen G_1 und G_2 auf der Einheitskugel, berechne die Schnittpunkte zwischen den Kanten von G_1 und den Kanten von G_2.

- Gegeben einen planaren Graphen G und eine Menge S von Punkten auf der Einheitskugel, berechne für jeden Punkt von S die Region von G, die diesen Punkt enthält.

Diese Probleme sind intensiv untersucht worden für den Fall, daß die Objekte in einer zweidimensionalen Ebene liegen. Für diesen Fall gibt es robuste und effizienten Algorithmen, die auf der plane sweep Technik basieren. Mittels der zentralen Projektion könnte man natürlich die obigen Probleme von der Einheitskugel auf die Ebene zurückführen. Dies hat aber als Nachteil, daß Punkte, die auf dem Äquator liegen, auf Punkte im Unendlichen abgebildet werden.

In [21] wird unsere Implementierung, die direkt auf der Einheitskugel arbeitet, vorgestellt. Diese Implementierung ist robust und exakt, d.h. es werden keine Annahmen über die Eingabe gemacht, und alle Prädikate werden mit exakter Arithmetik entschieden. Hieraus folgt, daß die Breite des Polyeders exakt berechnet wird.

Betrachten wir zum Schluß das Problem der support Minimierung. Wir nehmen zuerst an, daß das Modell ein dreidimensionales konvexes Polyeder $\mathcal{P}$ ist. Wie berechnet man die Baurichtung, für die die Gesamtfläche aller Flächen von $\mathcal{P}$, die mit supports in Berührung sind, minimal ist? Wir nennen diese Fläche die *Kontaktfläche*.

Wenn $\mathbf{d}$ eine Baurichtung ist, dann ist jede Fläche von $\mathcal{P}$, deren Normalvektor einen Winkel von mindestens $\pi/2$ mit $\mathbf{d}$ bildet, mit supports in Berührung. Wir betrachten Vektoren wieder als Punkte auf der Einheitskugel. Für jede Fläche F von $\mathcal{P}$, sei D_F die Menge aller Vektoren

$\mathbf{d}$, die mit der Normale von F einen Winkel von mindestens $\pi/2$ bilden. D_F besteht aus den Punkten der Einheitskugel, die auf einer Seite einer Ebene durch den Ursprung liegen. Diese Ebene schneidet die Kugel in einem Großkreis.

Wir berechnen alle Schnittpunkte dieser Großkreise. Das ergibt einen planaren Graphen G auf der Einheitskugel. Sei $\mathcal{F}$ eine beliebige Region dieses Graphen. Dann gilt für jede Richtung $\mathbf{d} \in \mathcal{F}$, daß die gleichen Flächen von $\mathcal{P}$ mit supports in Berührung sind. Hieraus folgt, daß das Problem der Minimierung der Kontaktfläche mit Hilfe des Graphen gelöst werden kann. Wenn das Polyeder $\mathcal{P}$ aus n Flächen besteht, dann besteht der Graph G aus $O(n^2)$ vielen Regionen. Unser Problem kann also in $O(n^2)$ Zeit gelöst werden.

Das Problem wird leider viel schwieriger für nicht-konvexe Polyeder. Wenn $\mathcal{P}$ konvex ist, dann ist eine Fläche entweder komplett oder gar nicht mit supports in Berührung. Bei nicht-konvexen Polyedern tritt der Fall, daß nur ein Teil einer Fläche mit supports in Berührung ist, auf. Der oben skizzierte Ansatz kann verallgemeinert werden. Es wird wieder ein planarer Graph G auf der Einheitskugel konstruiert, der jetzt aber aus $O(n^6)$ vielen Regionen besteht. Für jede Region $\mathcal{F}$ von G kann man die Kontaktfläche als Funktion von $\mathbf{d}$ schreiben. Diese Funktion muß dann über alle Richtungen $\mathbf{d} \in \mathcal{F}$ minimiert werden. Auf diese Weise entstehen $O(n^6)$ viele Optimierungsprobleme, die die folgende Form haben:

- Minimiere eine Summe von $O(n^2)$ Termen der Form p_i/q_i, wobei
  - p_i ein Polynom vom Grad 4 in den Variablen d_1, d_2 und d_3, mit $d_1^2 + d_2^2 + d_3^2 = 1$, ist,
  - q_i ein Polynom vom Grad 6 in den Variablen d_1, d_2 und d_3, mit $d_1^2 + d_2^2 + d_3^2 = 1$, ist,
- unter $O(n^3)$ vielen linearen Nebenbedingungen für die Variablen d_1, d_2 und d_3. (Diese Nebenbedingungen werden durch den Rand einer Region $\mathcal{F}$ definiert.)

Weil nicht bekannt ist, ob dieses Problem in Polynomialzeit lösbar ist, versuchen wir Heuristiken zu entwickeln, die die minimale Kontaktfläche approximieren und die ohne viel Aufwand implementiert werden können.

Ich hoffe, daß diese Beispiele verdeutlichen, daß im Bereich Manufacturing interessante und nichttriviale geometrische Probleme auf natürliche Weise auftauchen, und daß es eine große Herausforderung ist, effiziente und praktische Algorithmen für diese Probleme zu entwickeln und zu implementieren. Für unsere ersten Ergebnisse in diese Richtung sei auf die Arbeiten [16, 17, 21] verwiesen.

7 Weitere Verweise

In dem Text wurden bereits verschiedene Literaturverweise angegeben. Für grundlegende Einführungen in die Algorithmische Geometrie wird auf die Bücher von O'Rourke [19] und de Berg *et al.* [9] verwiesen.

Aktuelle Informationen können auf den *Computational Geometry Pages* von Jeff Erickson unter

`http://sal.cs.uiuc.edu/~jeffe/compgeom/compgeom.html`

nachgeschlagen werden. Weitere Anwendungen der Algorithmischen Geometrie können auf David Eppstein's Web-Seite *Geometry in Action* gefunden werden:

`http://www.ics.uci.edu/~eppstein/geom.html`

Literatur

[1] S. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms—ESA '96, Fourth Annual European Symposium*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 514–528. Springer-Verlag, 1996.

[2] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th Annu. ACM Sympos. Theory Comput.*, pages 489–498, 1995.

[3] S. Arya, D. M. Mount, and M. Smid. Randomized and deterministic algorithms for geometric spanners of small diameter. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 703–712, 1994.

[4] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17:33–54, 1997.

[5] B. Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., 1996.

[6] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[7] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[8] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[9] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[10] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.*, 1:132–133, 1972.

[11] P. F. Jacobs. *Rapid Prototyping & Manufacturing: Fundamentals of StereoLithography.* McGraw-Hill, New York, 1992.

[12] S. Kapoor and M. Smid. New techniques for exact and approximate dynamic closest-point problems. *SIAM J. Comput.*, 25:775–796, 1996.

[13] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete Comput. Geom.*, 7:13–28, 1992.

[14] C. Levcopoulos, G. Narasimhan, and M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 186–195, 1998.

[15] G. S. Lueker. A data structure for orthogonal range queries. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 28–34, 1978.

[16] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. In *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 136–149. Springer-Verlag, 1997.

[17] J. Majhi, R. Janardan, M. Smid, and J. Schwerdt. Multi-criteria geometric optimization problems in layered manufacturing. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 19–28, 1998.

[18] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1994.

[19] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, New York, 1994.

[20] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[21] J. Schwerdt, M. Smid, J. Majhi, and R. Janardan. Computing the width of a three-dimensional point set: an experimental study. In *Proc. 2nd Workshop on Algorithm Engineering*, 1998.

[22] J. Schwerdt, M. Smid, and S. Schirra. Computing the minimum diameter for moving points: an exact implementation using parametric search. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 466–468, 1997.

[23] M. I. Shamos. *Computational Geometry.* Ph.D. thesis, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1978.

[24] M. Smid. Closest point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry.* Elsevier Science Publishers, Amsterdam, 1998.