# 1

# Computational Geometry

**H. Edelsbrunner**

CONTENTS



Herbert Edelsbrunner
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.
e-mail: edels@p.cs.uiuc.edu
fax: +1-217-3333501

# Introductory Remarks

The original idea of this column was to present interesting topics in computational geometry in an informal and informative manner. This means blending discussions about new and old trends in the field with reports on major events, presentations of technical results, and reflections of people working in the field. Following these guidelines eight sequels have been written and are collected in this book. There are only minor differences to the original publications in the EATCS Bulletin. For convenience, each sequel is now assigned a descriptive title. The references are kept the same, by and large, to properly represent the time the sequels have been written.

H. Edelsbrunner



# What Is Computational Geometry?

What exactly does the term "computational geometry" represent? The lack of answering this question is possibly a sign of wisdom, or interpreted as such. Nevertheless we attempt to offer some answers. It does *not* coincide with the areas usually referred to as "computer graphics" and "computer aided design (CAD)", although there are many problems common to these three areas. Typical problems in computational geometry have a more theoretical touch than typical problems in the two areas mentioned. If we consult a library and search for books whose titles contain both important keywords, that is,

*computational* and *geometry*,

then we find at least four items:[1] [2] [3] [4]. In [2], *computational geometry* represents an area also know as *geometric modeling*. Among other things it is concerned with modeling curves and surfaces by means of splines. This area is closer in spirit to numerical analysis than to geometry. Nevertheless, there is considerable overlap between this area and *computational geometry* as it will be defined below. In [1], the complexity of predicates that recognize geometric properties, such as connectivity and convexity, is investigated. The aim of the study is to determine the power of large arrays composed of simple circuits in performing pattern recognition tasks. The corresponding notion of *computational geometry* is unrelated to ours, which coincides with the definition adopted in [3] and [4]:

> *computational geometry* is the study of the computational complexity of well defined geometric problems.

Thus, we are talking algorithms, data structures, analysis of time and storage, lower and upper bounds, and also geometric objects, geometric operations, and combinatorial complexity of structures.

[1] Minski, M. I. and Papert, S. *Perceptrons. An Introduction to Computational Geometry*. MIT Press, Amherst, Mass., 1969.

[2] Faux, I. D. and Pratt, M. J. *Computational Geometry for Design and Manufacture*. John Wiley & Sons, New York, 1981.

[3] Mehlhorn, K. *Multidimensional Searching and Computational Geometry*. Springer-Verlag, Heidelberg, 1984.

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

A different way to specify a research area is by means of exhaustively enumerating all subareas. In this column, we take a similar approach, that is, we present a tree of keywords that classifies work done in computational geometry. Besides for defining *computational geometry*, this tree can also be used to structure a bibliography in the area – and I hope that everybody agrees that a structured bibliography would be an awfully useful thing to have. If you are interested in obtaining a copy of such a bibliography, then it is vital that you continue to read.

To the best of my knowledge, there was only one attempt to maintain a bibliography in computational geometry,[5] and this project ended a few years ago. It lead to the compilation of a list of papers that was reasonably complete until the summer of 1982. There are two reasons why it was not continued beyond 1982. One was the increasing amount of time necessary to maintain the list, which was caused by the growing interest in the area and the thus increasing number of publications per year. The other reason was that besides alphabetical order and an attached author list it was entirely unstructured. A bibliography that is not structured into sublists in some way or the other becomes increasingly clumsy to use if it grows beyond some crucial size. This is exactly what happened in 1982.

Now, here is a proposal that describes how a bibliography in computational geometry should be maintained today. We believe it is absolutely crucial that as many people as possible contribute to such an undertaking; still it is a good idea to keep the bibliography at a central location and to have somebody in charge of maintaining it there. The structure of the bibliography can be obtained by means of keywords associated with each paper. Assigning appropriate keywords to a large number of publications is a major task which can only be completed by the community of researchers itself. To achieve a uniform assignment of keywords, we thus need a scheme of keywords – to be maintained through time as well – that we all agree upon. The suggested scheme is a tree of height four whose leaves are the keywords (italics) and whose interior nodes represent categories of keywords. Below, we give a pre-order traversal of the suggested tree.

Keywords in computational geometry
  Keywords that refer to research areas
    Theory of computation
      *complexity theory, parallel computation, network algorithms, design of, algorithms, data structuring, implementing algorithms.*
    Computer applications
      *computer aided design (CAD), VLSI design, computer graphics, robotics, motion planning, computer vision, image processing, pattern recognition, cluster analysis, database theory, artificial intelligence (AI), software.*
    Geometry
      *elementary geometry, combinatorial geometry, discrete geometry, probabilistic geometry, differential geometry, algebraic geometry, linear, algebra, topology.*
    Non-geometric mathematics
      *probability theory, statistics, graph theory, algebra, calculus, numerical, analysis, operations research.*
  Keywords addressing problem characteristics
    Algorithmic characteristics
      *optimization, construction, detection, sorting, reporting, counting, searching, probing, approximation.*
    Geometric characteristics
      *decomposition, packing, covering, separation, partition, proximity, intersection, visibility, domination, diameter, area, volume, length, distance, geodesic distance, measure, feature, shape.*
    Particular problems
      *convex hull, range search, point location, linear programming, path planning, hidden line/surface elimination.*
  Keywords addressing algorithmic aspects
    Algorithmic methods
      *binary search, graph traversal, plane-sweep, space-sweep, divide-and-conquer, prune-and-search, dynamic programming, incrementation, branch-and-bound, heuristics, bucketing, repeated search; geometric transformations, locus approach, dynamizing data structures.*
    Data structures
      *trees, range trees, segment trees, interval trees, priority search trees, quad trees, oct trees, kd-trees, tries, directed acyclic graphs (DAG), arrays, hashing, implicit data structures, plane graph representations, incidence graph; constructive solid geometry (CSG).*
    Complexity analysis
      *worst-case analysis, amortized analysis, expected case analysis, Monte Carlo, Las Vegas, complexity classes, NP-completeness, output dependent, computational model, numerical stability, numerical precision, lower bounds.*
  Keywords addressing geometric aspects
    Geometric structures
      *subdivisions, cell complexes, arrangements, Voronoi diagrams, Delaunay triangulations, triangulations, polygons, polyhedra; geometric graphs, traveling salesman tours (TST), minimum spanning trees (MST), Steiner trees, matchings.*

[5] Edelsbrunner, H. and van Leeuwen, J. Multidimensional algorithms and data structures – bibliography. *Bull. EATCS* **11** (1980), 46–74. Supplement to "Multidimensional algorithms and data structures – bibliography". *Bull. EATCS* **13** (1981), 76–85. Multidimensional data structures and algorithms: a bibliography. Report F104, Inst. Informationsverarb., Techn. Univ. Graz, Austria, 1983.

Geometric objects

*points, lines, planes, hyperplanes, rays, half-planes, half-spaces, line segments; intervals, rectangles, hyperrectangles, circles, disks, spheres, balls; triangles, tetrahedra, simplices; curves, splines; translates, homothets.*

Geometric attributes

*one-dimensional, two-dimensional, three-dimensional, d-dimensional, convex, star-shaped, connected, simply connected, congruent, symmetric; weighted; topological.*

Keywords for non-research papers

*book, doctoral thesis, survey paper, bibliography, problem collection.*

The main objective in the construction of this tree of keywords is that it allows a useful clustering of the bibliography into not necessarily disjoint but significantly different sublists. It is planned to publish such sublists periodically – not in this column, however. We therefore encourage everybody who is interested in contributing to the bibliography to raise his or her voice, to criticize the proposed plan, and/or to suggest changes and improvements. Besides publishing sublists, we hope to be able to offer on-line access to the bibliography in the near future so that everybody can search the bibliography, extract information from it, and create his or her own sublists.

Currently, a preliminary version of the bibliography exists in bibref format in the Computer Science Department in Urbana, Illinois. It was initially created by merging [6] with a list of references collected over the years by Leonidas Guibas and Jorge Stolfi, Department of Computer Science, Stanford University. Currently, the bibliography contains about fourteen hundred entries. The next step is to provide every interested author with a sublist of his or her co-authored papers and to leave it up to him or her to correct and extend this sublist and to provide keywords for each of his or her publications. If you are interested, then let me know and I will provide you with a sublist that contains your papers already entered into the bibliography. We hope to be able to make copies of the complete list available sometimes not far in the future.

[6] Edelsbrunner, H. and van Leeuwen, J. Multidimensional data structures and algorithms: a bibliography. Report F104, Inst. Informationsverarb., Techn. Univ. Graz, Austria, 1983.



# Publishing in Computational Geometry

It takes a lot of time and effort on the side of the people working in it before a scientific discipline gets established as a respected member of the family. Many new areas do not make it that far and there are a number of reasons that can be responsible for falling short. As it stands now, computational geometry seems to be among the lucky ones in the battle for a sunny place. Over 10 years ago, Michael Shamos worked on his influential thesis that shaped the area for quite some time, and there are many signs that testify excellent health to this flourishing field.

In this issue, I would like to talk about ways the computational geometry community expresses itself, which most prominently includes the publication means it uses. Of course, there are conferences, journals, and books, and we will chat a little about each one of the three categories.

**Conferences.** For many years, computational geometry papers got presented at theory of computing conferences such as the *International Colloquium on Automata, Languages, and Programming (ICALP)*, the *Symposium on Theory of Computing (STOC)*, and the *Symposium on the Foundations of Computer Science (FOCS)*, just to name a few. It is still the case that many computational geometry papers make it into these conferences, in particular those that have a strong algorithmic component of general interest. One could even say that there is at least one computational geometry session by tradition. For example, the next years *Symposium on Mathematical Programming* in Japan includes an organized special session on computational geometry.

The main occasion, however, is the *Symposium on Computational Geometry* which is now annually held as an ACM symposium supported by SIGACT and SIGGRAPH, two suborganizations of the ACM. I can warmly recommend the attendance at this yearly event to everybody who has an interest in new developments or who wants to get in touch with active researchers in the area. Next year, 1988, the fourth conference is going to be held in Urbana, Illinois, and I sign responsible for the local organization. For historical interest I mention that the first three conferences were held in Baltimore, Maryland, organized by Joseph O'Rourke, in Yorktown Heights, New York, organized by Alok Aggarwal, and this year in Waterloo, Ontario, with Derick Wood as the local chairman. We are all indebted to Joseph O'Rourke who took the first and daring step to start this successful series of conferences.

Aside from these annual events, there are workshops and other meetings that enhance the communication within the community as well as with other communities. For example, the *AMS Workshop on Discrete and Computational Geometry* held 1986 in Santa Cruz, California, was organized to build a bridge between researchers in discrete geometry and in computational geometry. Another such effort was constituted by the *Workshop on Polyhedral Complexity and Geometric Complexity* which took place in Minneapolis, Minnesota. Its emphasis is on problems in the interface between combinatorial optimization and computational geometry.

**Journals**. I remember David Dobkin, Michael Shamos's thesis advisor, once mentioning that at the beginning it was not clear at all to what journals computational geometry papers should be sent. Many issues in geometry have numerical aspects which suggested that numerical analysis journals were, maybe, the right forum. Because of the strong algorithmic contents, however, the decision was made to seek the theory of computer science audience. Partly because of this decision, we find many of the computational geometry papers in journals such as the *Communications of the ACM*, *IEEE Transactions on Computers*, *Information Processing Letters*, *Journal of Algorithms*, *Journal of the ACM*, *SIAM Journal on Computing*, *Theoretical Computer Science*, and others. We can also find papers in the mathematical programming literature, such as in the *Journal of Mathematical Programming*, and in more applied journals, such as *Computer Aided Design* and *Computer Graphics, Vision, and Image Processing*.

I deliberately leave out the *Journal on Discrete and Computational Geometry* which is run by Jacob Goodman and Richard Pollack. This journal is unique in the choice of topics it covers. Its purpose is to offer a forum for computational geometry papers with strong combinatorial component as well as for discrete geometry papers with algorithmic potential. Currently, among all journals this is probably the one that publishes the largest number of computational geometry papers – without compromising the quality of its selection. We should also mention two new journals, *Algorithmica* and the *Journal on Symbolic Computation*, which emphasize that they consider computational geometry as one of their main areas of interest.

**Books**. By now, there are four textbooks on the market documenting the maturity of computational geometry as a scientific discipline. Nevertheless, it is still true that the dynamics of the area make it very difficult to write such a text. On the other hand, computational geometry is rich enough to provide sufficient space for a few more textbooks which do not interfere in contents or style with the current four. Naturally, the four books share some of the material, but everything considered they are reasonably disjoint and emphasize different aspects of computational geometry. The book published first, Kurt Mehlhorn's *Data Structures and Algorithms 3: Multidimensional Data Structures and Computational Geometry*, Springer-Verlag 1984, concentrates on the data structuring aspects of computational geometry. It nicely fits into his series of three data structures and algorithms texts. In 1985, the



long awaited *Computational Geometry – an Introduction*, by Franco Preparata and Michael Shamos, was published by Springer-Verlag. It offers a broad view of what can be called the classic part of computational geometry. In 1987, two books entered the scene of computational geometry texts. Joseph O'Rourke's *Art Gallery Theorems and Algorithms*, Oxford Press, concentrates on problems for polygons and gives a nice collection of mathematical as well as algorithmic results. The other book, *Algorithms in Combinatorial Geometry*, Springer-Verlag, is by myself. Its goal is to present fundamental results both in combinatorial and in computational geometry. Ignoring my personal bias, I think it is fair to say that this book is the most comprehensive text available in the combinatorially influenced part of computational geometry.

# Shaving Logs

**The Topic.** "The disease of the log-factor" or "Why is it that some complexity functions are common and some are not?".

What is the reason that so many researchers in design of algorithms pick on log-factors, our beloved friends that seem to be integral parts of so many time-complexity analyzes. Indeed, log-factors are being "shaved off" the time-complexities of all kinds of problems these days[1]. The ultimate goal seems to be the elimination of all log-factors – as if they were contagious (well, maybe they are; just think of the polylog which describes both small families to large populations of logs by one deceivingly short word). When we attempt to shave off a log-factor we might have the unpleasant experience that this superficial operation leaves little pieces of the poor fellow behind. These pieces come in different sizes such as $\frac{\log}{\log\log}$, $\sqrt{\log}$, $\log\log$, $\log^*$, and $\alpha$; we call these the *descendents*[2] of the log.

We consider three questions, the first of which is often asked by people who like logs because they are so easy to get and because they are too small to be real trouble: "Why bother?". The second looks at the most successful insecticides in current use against logs. The final question has to do with the widespread belief that the world is simple – we just don't look at it from the right angle. In terms of algorithms and complexities this can only imply that the "real" complexities of problems are simple and "natural". It seems that there are a few complexity formulas that contain respected log-factors, but most often their presence raises suspicion. In any case, if a log-factor shows signs of weakness it should be terminated without mercy – which means leaving no descendents behind, for if we cannot trust the log, how can we trust its descendents? To counter this belief, section 3 will show a few examples where honest descendents of the log were covered underneath dishonest logs or other (bigger) descendents of it.

[1] I thank Mikhail Atallah for pointing out that this colloquial phrase is commonly used for improvements by log-factors, as opposed to *improvements* by factors bigger than log.

[2] The log log of something is equal to the log of what remains if we take the log of the something. If we iterate until what is left is smaller than 2 then the number of iterations is known as the $\log^*$ of this something. $\alpha$ is also known as the inverse of Ackermann's function; it pretends to decreases everything down to size at most 4 but it is actually true that $\alpha$ of something goes to infinity when the something does. However, one must be real patient to see this with ones own eyes.



## 1 Why don't we leave the log alone?

Much of the today's reputation of the log is based on the time-complexity of sorting. There are a number of algorithms that show that $n$ numbers can be sorted in time $O(n \log n)$[3] [4] [5] and $\Omega(n \log n)$ is a lower bound on the complexity under a very general although comparison based model of computing[6]. This result implies the existence of honest logs in the complexity formulas on many problems, namely those that can be solved in time $O(n \log n)$ and that can be shown to be at least as difficult as sorting. Examples in computational geometry include the construction of the convex hull and the Voronoi diagram of a set of $n$ points in the plane[7] [8]. Although $\Theta(n \log n)$ is the worst-case complexity of the convex hull problem, Kirkpatrick and Seidel[9] have shown that time $O(n \log h)$ time suffices if only $h$ of the $n$ points lie on (rather than within) the convex hull. Is this a sign of weakness of the log? Not really. The same paper proves that $\Omega(n \log h)$ is a lower bound for this problem.

So what are examples of dishonest logs, that is, for which problems can we avoid the log-factor we have to pay for sorting? In this section we mention three such examples:

(i) finding the median of an unsorted list of numbers[10],

(ii) constructing the convex hull of a simple polygon in the plane[11], and

(iii) sorting the intersections of a line with a Jordan curve if the intersections are given in their order along the curve[12].

[3] Knuth, D. E. *Sorting and Searching – the Art of Computer Programming III.* Addison-Wesley, Reading, Mass., 1973.

[4] Aho, A. V. Hopcroft, J. E. and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.

[5] Mehlhorn, K. *Data Structures and Algorithms 1: Sorting and Searching.* Springer-Verlag, Heidelberg, Germany, 1984.

[6] Ben-Or, M. Lower bounds for algebraic computation trees. In "Proc. 15th Ann. ACM Sympos. Theory Comput. 1983", 80–86.

[7] Preparata, F. P. and Shamos, M. I. *Computational Geometry – an Introduction.* Springer-Verlag, New York, 1985.

[8] Edelsbrunner, H. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, Germany, 1985.

[9] Kirkpatrick, D. G. and Seidel, R. The ultimate convex hull algorithm? *SIAM J. Comput.* 15 (1986), 287–299.

[10] Blum, M., Floyd, R. W., Pratt, V. R., Rivest, R. L. and Tarjan, R. E. Time bounds for selection. *J. Comput. System Sci.* 7 (1972), 448–461.

[11] McCallum, D. and Avis, D. A linear algorithm for finding the convex hull of a simple polygon. *Inform. Process. Lett.* 9 (1979), 201–206.

[12] Hoffmann, K., Mehlhorn, K., Rosenstiehl, P. and Tarjan, R. E. Sorting Jordan sequences in linear time using level-link search trees. *Inform. and Control* 68 (1986), 170–184.

Many other examples can be found in section 2. There we also touch upon the properties of problems that admit solutions without sorting.

The real question in the mind of a practitioner is whether these new algorithms are really any faster. After all, they are only a factor of $\log n$ faster in the theoretical sense, which doesn't mean much if the constant hidden by the big-Oh increases significantly. The answer to this question depends on the case. Sometimes, the improvement is based on new insights into the combinatorics of the problem and the resulting algorithm is indeed faster than previous log-infected algorithms. Sometimes the algorithm remains the same and only the analysis improves. In other cases the new algorithms are more difficult to understand but that the resulting programs are significantly simpler than those generated from the old algorithms. Indeed this phenomenon occurs unexpectedly often, which may tell us something about the adequateness of our current understanding of algorithms.

## 2 What makes the log disappear?

One of the most common methods for avoiding log-infested complexities is to exploit a priori sorting information. This sounds simple enough but can be exceedingly difficult. A well-known case where this method is successful is the problem of sorting a list that consists of two already sorted lists. To sort the entire list, we scan the two sublists and, at each step, we take the smaller of the two current numbers and advance the corresponding pointer. This scheme is known as *merging* sorted lists. Similar although more sophisticated techniques are necessary to get linear algorithms for constructing the kernel of a simple polygon[13] [14], for constructing the convex hull of a simple polygon[11], and for finding the nearest neighbor of each vertex of a convex polygon[15]. All these algorithms follow the same philosophy as merge sort: exploit the order information of the input or of intermediate structures. The order information in the first two cases is that the polygon is simple, which excludes many otherwise possible point sequences. In the third case, the order information is even stronger – the points are the vertices of a convex polygon, given in the order around the polygon. The additional structure is compensated by the more demanding requirements to compute a piece of information for *each* vertex of the polygon.

To illustrate that exploiting sorted input is not always as simple as it sounds, we mention the problem of sorting the $n^2$ numbers in an $n$-by-$n$ matrix whose rows and columns are all sorted in increasing order. Indeed, it seems that the degree of presortedness is substantial. Contrary to our intuition, it can be shown that this type of input admits $2^{cn^2 \log n}$ different sorting permutations, which implies an $\Omega(n^2 \log n)$ lower bound for comparison based algorithms[16]. The situation is different if there are sorted vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ so that the matrix is equal to $X + Y$, that is, its entry in row $i$ and column $j$ is equal to $x_i + y_j$. In this case, only $2^{cn \log n}$ sorting permutations exist, and Fredman[17] proves that there is an algorithm that takes only $O(n^2)$ comparisons to sort $X + Y$. However, it is not known whether there is an $O(n^2)$ time algorithm for sorting $X + Y$.

[13]The *kernel* of a simple polygon is the set of points that can be connected by straight line segments contained in the polygon to all other points within the polygon.

[14]Lee, D. T. and Preparata, F. P. An optimal algorithm for finding the kernel of a polygon. *J. Assoc. Comput. Mach.* **26** (1979), 415–421.

[15]Lee, D. T. and Preparata, F. P. The all-nearest neighbor problem for convex polygons. *Inform. Process. Lett.* **7** (1978), 189–192.



Another popular method for eliminating logs is the use of intricate recursive schemes. If we have an algorithm whose time-complexity follows a recurrence relations of the form

$$T(n) = T(a \cdot n) + T(b \cdot n) + O(n),$$

with $a+b < 1$, we have won. The reason is that $a+b < 1$ implies that the homogeneous solution to the recurrence relation is of the form $n^\gamma$, with $\gamma < 1$. Thus, the additive term, which is linear in $n$, determines the solution, which is $O(n)$. Possibly the first algorithm that made use of this observation is the linear median finding algorithm in[10], see also[4,5]. Hence, finding the median of an unsorted list of numbers is easier than sorting the list. In computational geometry, algorithms using the same paradigm have been found for linear programming in a fixed number of dimensions[18] [19] [20], finding the furthest neighbor of every vertex of a convex polygon[21] [22], and constructing the Voronoi diagram of the vertices of a convex polygon[23]. For each one of the three problems it is necessary to bring in non-trivial ideas which lead to three inherently different generalizations of the by now classic median finding scheme. Each one of these generalized schemes has been applied to a number of other geometric problems. These can be found in the mentioned papers.

[16]Harper, L. H., Payne, T. H., Savage, J. E. and Straus, E. Sorting $X + Y$. *Comm. ACM* **18** (1975), 347–349.

[17]Fredman, M. L. On the information theoretic lower bound. *Theoret. Comput. Sci.* **1** (1976), 355–361.

[18]Megiddo, N. Linear-time algorithms for linear programming in $R^3$ and related problems. *SIAM J. Comput.* **12** (1983), 759–776.

[19]Megiddo, N. Linear programming in linear time when the dimension is fixed. *J. Assoc. Comput. Mach.* **31** (1984), 114–127.

[20]Dyer, M. Linear time algorithms for two- and three-variable linear programs. *SIAM J. Comput.* **13** (1984), 31–45.

[21]The *furthest neighbor* of a vertex, $u$, is the vertex of the same polygon that maximizes the distance to $u$.

[22]Aggarwal, A., Klawe, M. M., Moran, S., Shor, P. and Wilber, R. Geometric applications of a matrix sorting algorithm. In "Proc. 2nd Ann. Sympos. Comput. Geom. 1986", 285–292.

[23]Aggarwal, A., Guibas, L. J., Saxe, J. and Shor, P. A linear time algorithm for computing the Voronoi diagram of a convex polygon. In "Proc. 19th Ann. ACM Sympos. Theory Comput. 1987", 39–45.

Balanced trees have been a bastion of the log since searching and updating trees requires $\Omega(\log n)$ time per operation, even on the average. This is true since otherwise we could sort in asymptotically less than $O(n \log n)$ time. Without contradicting this lower bound, it is still possible to take asymptotically less than logarithmic time per operation - the catch is that the operations performed cannot quite be used for sorting. The difference to the common collection of operations (searching, inserting, deleting) is often subtle. The key is the use of advanced balanced tree schemes, such as 2,4-trees[5]. If we perform $n$ arbitrary insertions and deletions in a 2,4-tree we use only constant time for rebalancing per operation - in the amortized sense, that is, $O(n)$ time for the entire sequence. Of course, this does not account for the amount of time needed to find the entry to be deleted or to find the right location for an entry that is to be inserted. The way around this difficulty is the use of pointers (so-called *fingers*) that point from the outside into the tree.

Using families of such *finger trees* (which can also be split in amortized constant time), Hoffmann et al.[12] solve the Jordan sorting problem mentioned in section 1 in time linear in the number of intersections. Another problem where the a priori order information can be exploited using finger trees is the triangulation of a simple polygon; in this case, the early $O(n \log n)$ time algorithms have been improved to $O(n \log \log n)$[24]. Surely, the log was not of the honest type, but is the log log fake too? The safest bet still is the existence of a linear time algorithm for triangulating a simple $n$-gon - but one cannot be sure.

The technique to design data structures that are efficient in the amortized sense but not necessarily in the worst-case per operation sense has a wide range of applications not restricted to binary search trees. The construction of an arrangement of $n$ lines in the plane is probably the closest one can get to a two-dimensional generalization of sorting[25]. Their $O(n^2 \log n)$ time algorithm was improved to $O(n^2)$ in[26] [27]. The improvement was made possible not by intricate algorithmic techniques and data structures (those could have been used but were not necessary) but by proving a certain combinatorial property of line arrangements. The message here is that the right kind of combinatorial insight leads to simple algorithms which can be optimal nevertheless. More recently, Edelsbrunner and Guibas[28] showed by an amortized time analysis argument that $n$ lines in the plane can be swept in $O(n^2)$ time and only $O(n)$

[24]Tarjan, R. E. and Van Wyk, Ch. J. An $O(n \log \log n)$-time algorithm for triangulating simple polygons. *SIAM J. Comput.*, to appear.

[25]Goodman, J. E. and Pollack, R. Multidimensional sorting. *SIAM J. Comput.* **12** (1983), 484–507.

[26]Chazelle, B., Guibas, L. J. and Lee, D. T. The power of geometric duality. *BIT* **25** (1985), 76–90.

[27]Edelsbrunner, H., O'Rourke, J. and Seidel, R. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.* **15** (1986), 341–363.

[28]Edelsbrunner, H. and Guibas, L. J. Topologically sweeping an arrangement. In "Proc. 18th Ann. ACM Sympos. Theory Comput. 1986", 389–403.



storage. One of the applications of the line arrangement algorithms is a quadratic solution to the following $n$-fold sorting problem. Given $n$ points in the plane, for each point $p$ construct the list of the other $n-1$ points sorted around $p$. Because of the sorting lower bound, the construction of only one such list takes $\Omega(n \log n)$ time.

## 3 Also dishonest logs can have honest descendents

The probably best known example of a problem that can be solved in time asymptotically less than $O(n \log n)$ but still takes asymptotically more than $\Omega(n)$ is the so-called union-find problem[4,5]. The execution of a sequence of $n$ union and find operations[29] takes time $\Theta(n\alpha(n))$. The lower bound is proved for the pointer machine model[30]. So $\alpha$ is of the honest type here and we should respect it. There is, however, this nagging feeling that $n\alpha(n)$ is not really a natural function, and how come it is the right complexity function for the union-find problem. Could it be that the union-find problem itself is unnatural? Probably not, because it has many applications including such natural problems as finding minimum spanning trees of graphs. Can it be that the notion of time-complexity or maybe of computing altogether is unnatural in the way we currently understand it?

Fortunately, there is a way out of this dilemma: $n\alpha(n)$ is indeed a natural function - how could it otherwise be the answer to many other problems which describe rather concrete things in our world. For example, there is the notion of a Davenport-Schinzel sequence of order 3 and it is known that the longest such sequence consists of $\Theta(n\alpha(n))$ letters[31], where $n$ is the number of different letters in its alphabet[32]. Using this result, Wiernik[33] shows that the largest number of linear pieces of the upper envelope of $n$ possibly intersecting line segments in the plane[34] has length $\Theta(n\alpha(n))$. The upper envelope of line segments is something very concrete which leads the author of this column to believe that $n\alpha(n)$ is a natural function. Davenport-Schinzel sequences can be generalized to orders $s \geq 1$ by disallowing alternating sequences of length

[29]Before performing the operations we have a collection of $n$ singleton sets. Each *union* operation merges two sets of the system to one set, and each *find* operation tells us for a given element to what set it currently belongs.

[30]Tarjan, R. E. On the efficiency of a good but no linear set union algorithm. *J. Assoc. Comput. Mach.* **22** (1975), 215–225.

[31]Hart, S. and Sharir, M. Nonlinearity of Davenport-Schinzel sequences and of a generalized path compression scheme. *Combinatorica* **6** (1986), 151–177.

[32]A sequence of letters $a_1, a_2, \ldots, a_m$, where the letters are taken from the set $\{1, 2, \ldots, n\}$, is a *Davenport-Schinzel sequence* of order 3 if there are no five indices $i_1 < i_2 < i_3 < i_4 < i_5$ so that $a_{i_1} = a_{i_3} = a_{i_5} \neq a_{i_2} = a_{i_4}$.

[33]Wiernik, A. Planar realizations of nonlinear Davenport-Schinzel sequences by segments. In "Proc. 27th Ann. IEEE Sympos. Found. Comput. Sci. 1986", 97–106.

[34]Think of the line segments as partial univariate functions. The *upper envelope* of the line segments is the pointwise maximum of these functions.

$s + 2$. For each constant order, the maximum length is bounded from above by $O(n \log^* n)$[35]. Recent advances show that this upper bound is not asymptotically tight[36], but except for $s \leq 4$ the known bounds are not known to be tight. For $s = 4$, the bound is $\Theta(n \cdot 2^{\alpha(n)})$[37].

An interesting case is also the $\frac{\log}{\log\log}$, which appears to be a rather unnatural function. There are surprisingly many cases for which it is the honest part of a complexity bound. Blum[38] proves that the worst-case per operation complexity of the union-find problem is $\Theta(\frac{\log n}{\log\log n})$. For the range search problem in the plane[39] Chazelle[40] proves that in order to get polylogarithmic search time the data structure must use $\Omega(n\frac{\log n}{\log\log n})$ memory space, and this bound can be achieved. Finally, $\frac{\log}{\log\log}$ appears as a tight bound of a combinatorial problem. Consider a convex polytope with $n$ faces in three dimensions. We take an orthogonal projection, which is a convex polygon, and let the *size* of this projection be the number of vertices of the polygon. The minimum, taken over all convex polytopes with $n$ faces, of the maximum projection, taken over all directions, is $\Theta(\frac{\log n}{\log\log n})$[41].

Why does $\frac{\log}{\log\log}$ appear as a tight bound for so many unrelated problems? An answer can be given when we realize that the complexity of a problem is usually born out of a compromise between sufficiency and necessity. As such, the bounds are the roots of equations describing this compromise. For example, $\log n$ is the root of

$$2^x = n,$$

an equation that takes only four different symbols each appearing once. The equation whose solution is $\frac{\log n}{\log\log n}$, or some constant multiple of it, is

$$x^x = n.$$

The number of symbols is still four but there are only three different symbols in the equation. Doesn't this mean that the $\frac{\log}{\log\log}$ is more natural than log?

[35]Szemerédi, E. On a problem by Davenport and Schinzel. *Acta Arithmetica* **25** (1974), 213–224.

[36]Sharir, M. Almost linear upper bounds on the length of general Davenport-Schinzel sequences. Report 29/85, The Eskenasy Inst. Comput. Sci., Tel Aviv Univ., Israel, 1985.

[37]Agarwal, P., Sharir, M. and Shor, P. Improved upper and lower bounds for the length of general Davenport-Schinzel sequences. Manuscript, Courant Inst., New York Univ., 1987.

[38]Blum, N. On the single-operation worst-case time complexity of the disjoint set union problem. Report 84/04, Fachbereich 10, Univ. Saarlandes, Germany, 1984.

[39]Given a set of $n$ points in the plane, the *range search problem* asks for a data structure that admits reporting the points inside a query rectangle in (hopefully) little time. The *search time* is defined as the overhead needed on top of the time required for reporting the points.

[40]Chazelle, B. Lower bounds on the complexity of multidimensional searching. In "Proc. 27th Ann. IEEE Sympos. Found. Comput. Sci. 1986", 87–96.

[41]Chazelle, B., Edelsbrunner, H. and Guibas, L. J. The complexity of cutting convex polytopes. In "Proc. 19th Ann. ACM Sympos. Theory Comput. 1987", 66–76.



**The moral.** "Keep this in mind the next time you get hit on the head" or "Some strange functions sit on lower bounds like a bump on a log"[42].

[42]I thank Steven Skiena for his wise words which summarize this column.

# The 4-th Computational Geometry Symposium

The 4-th Annual Symposium on Computational Geometry is coming up. By the way, did you know that the ACM distinction between a workshop, a symposium, and a conference is that the expected number of attendees is fewer than 100 for a workshop, between 100 and 300 for a symposium, and more than 300 for a conference? Anyway, the dates for the Computational Geometry Symposium are June 6, 7, 8 (Monday, Tuesday, Wednesday), the place is Champaign-Urbana in Illinois, the program chairman is Bernard Chazelle, and the local chairman is the writer of this column. As you can imagine the latter is spending much time worrying and looking forward to this event. This column contains a few random thoughts about the conference (excuse me, symposium) in general and about its program in particular. In addition, it remarks on new trends in computational geometry (as derived from the statistics of the program).

There are two invited addresses, the first by Bruno Buchberger (Johannes Kepler Universität Linz, Austria), the other by Thomas Banchoff (Brown University, Rhode Island). Buchberger will deliver a survey on new results in symbolic computation. Just in case the connection between computational geometry and symbolic computation is not obvious: every geometric algorithm must, at some point (in fact most of its running time), do primitive geometric calculations such as intersecting two line segments or determining which one of two triangles in three-dimensional space is closer to the viewer, etc. Such computations can be done in various ways, including symbolic manipulation of the data. In a nut-shell, the symbolic approach would not compute the intersection of the two line segments explicitly but store the intersection implicitly, e.g. by pointers to the two line segments plus, maybe, a bit that indicates whether the line segments intersect at all.

Banchoff is scheduled to talk about visualizations of four- and higher-dimensional objects. Few people can visualize even four-dimensional geometric objects (can anybody?) but there is hope to get some feel for what they really are. (Yes, I insist that four and higher dimensions are real.) Take, for example, a hyperplane in four dimensions (this is a three-dimensional space) that sweeps slowly through a four-dimensional object. At every point in time the hyperplane cuts the object in a three-dimensional cross-section, which *can be* understood by human beings. If, for example, the object is a four-dimensional convex polytope, the cross-section is a three-dimensional convex polytope. Initially (when the hyperplane does not intersect the object), the cross-



section is empty, then it is a point, and then it grows and becomes a three-dimensional polytope. Eventually, it becomes empty again. At discrete points in time, this polytope gains and looses faces. Between those events, the polytope changes continually by increasing and decreasing the sizes of its edges and faces. A graphics package that simulates such a sweep would be a fantastic tool, no? Of course, many other aids for visualizing (getting feel for) objects in four and higher dimensions are conceivable.

I believe that by inviting talks on those two subjects the program committee sends the message that its members consider the two areas as part of or related to computational geometry and that they encourage submissions from these areas in the future. Generally, it is rather difficult to attract good papers from areas which are not considered mainstream. For one thing, the strongest papers in these areas tend to go to specialized meetings. There is a rather unfortunate corollary of this fact. If the program committee decides not to accept a paper from some area $X$ because the quality of the paper does not live up to the usual standards, it is often perceived as a statement that the committee is not interested in area $X$. Such a statement is usually not intended.

Two interesting pieces of statistics. The number of different authors is up – it is now 73 – and so is the average number of authors per paper – it is 2.46 (2.53 if the two invited talks are not counted). The number of talks is 41 (including the two invited talks). A quick calculation shows that the number of not necessarily different authors is therefore 100.83 which suggests that it is really 101.

This brings us to the first trend we observe when we browse through the program: there are quite a number of papers concentrating on robustness issues in implementing geometric algorithms. Of course, there are several aspects of a program that contribute to its robustness or non-robustness. However, the main source for non-robust or even non-correct behavior of a program is the necessary finiteness of the representation of numbers. Assume you design a car on a computer and the car is represented by a polytope with integer coordinate vertices. This can be nicely drawn on a graphics output device. Of course, you and your friends would like to admire your creation from more than one point of view, in fact from any point of view they please to choose. To satisfy these desires, you set up a rotation matrix that, if applied to the coordinates, maps the points to their new locations. Unfortunately, these new locations do not necessarily have integer coordinates. So you take the nearest integer and everything seems fine. Only after iterating the rotation a couple of times, the car takes on strange and surprising shapes. This may or may not be to the advantage of the design. It is clear what went wrong – what is less clear is how to avoid this effect. A partial answer is that we should always keep the original representation around. Every rotation should be calculated directly from this original rather than from intermediate results. But what do we do if we want to intersect the car (which is a polytope) with a copy of itself rotated by one degree about some line? Not many automated design systems will be able to produce a meaningful result, in particular

when we let the angle be even smaller.

Many instances of a geometric operation are not this subtle. But imagine a program that performs a million geometric operations of some kind to finish its task. Now, there might be one that does not quite work because of numerical instabilities – this one instance is enough to make the program fail. I do not claim to know a satisfying answer to all these questions and refer to the Tuesday morning session of the symposium for new results on this topic.

The other trend in computational geometry apparent from the program is the increased interest in randomized algorithms. Does the term "randomized" needs some clarification? We certainly do not mean the type of "random" behavior that results from numerical instability as discussed above, nor do we refer to algorithms that use "random" access memory (what a thought). We mean algorithms that make use of a string of random bits – given the string, the behavior of the algorithm is deterministic. Of course, truly random bitstrings are hard to get (do they exist?) and must be simulated by the use of random number generators. That no random number generator is perfect will not be our concern here.

There is a classical randomized algorithm[1] for primality testing. It fails only with a constant probability smaller than one. If we repeat the algorithm we can get the probability of a wrong answer (failure all the time) arbitrarily close to 0. So in this case, the randomization influences the output of the algorithm, which is *not always correct.*

The randomized algorithms that will be presented at this year's computational geometry symposium are of a different breed though. They *always* compute the (or a) correct result, but it is not quite clear what the amount of time or memory space is the algorithms need. So we will hear statements of the kind "the randomized time-complexity of this algorithm is", etc. What does this mean? Here is my interpretation. Take a worst-case input (it will become clear in a second what "worst-case" means in this context) and, for every (random) bitstring of the desired length, estimate the amount of time the algorithm takes. The mean of these amounts is called the *randomized* time-complexity of the algorithm. So this is really the *expected* time the algorithm takes, right? There is, however, one difference to the usual meaning of expected complexity which is that the expectation is computed for a single input example over all bitstrings and not over all sets of input data.

As far as I can see there are really two reasons that make randomization so attractive (at least in computational geometry). One is that better time-bounds than in the deterministic model with the worst-case measure can be obtained with simpler algorithms (and "simplicity" often means "efficiency" in practice). The other is that

[1]See for example Brassard, G. and Bratley, P. *Algorithmics – Theory & Practice.* Prentice Hall, Englewood Cliffs, New Jersey, 1988.



randomization is closely related to certain combinatorial properties of sets of geometric objects. It is at least plausible that a reasonably large sample of a set of objects faithfully represents the distribution of the objects in the underlying set. The meaning of "faithfully representing" and "reasonably large" can be made concrete using probabilistic counting arguments[2]. Well, to put everything in perspective, not every sample faithfully represent the distribution of the underlying set but an overwhelming number of samples do so. That is to say, a randomly chosen sample is very likely to faithfully represent the distribution (whatever this means exactly). But this also means that there is at least one sample which faithfully represents the distribution. A number of new combinatorial results have been obtained by means of such samples whose existence can be established via probabilistic counting arguments.

Enough idle talk. I hope that many of you are interested in these and other issues, and I certainly hope that many of you will be able to attend the symposium.

[2]See for example Spencer, J. *Ten Lectures on the Probabilistic Method.* SIAM, Philadelphia. Penn., 1987 for an account of probabilistic counting methods.

# Two Weeks Worth of Teaching

For the good of an area such as computational geometry it is wise to sometimes ask what its impact on a higher level is or can be. Will computational geometry change (parts of) computer science or will it disappear after a while? – remember that nobody seemed to have needed computational geometry 20 years ago (silly argument?, well, I guess so). And there are always people, mostly not from within the area, who ask the question – and rightfully so.

Well, there are many meanings one can assign to the above question and there are answers in many different directions. In this column I would like to discuss the possible impact of computational geometry in the classroom. More particularly, I will talk about adding topics in computational geometry to by now standard graduate level algorithm courses. Graph algorithms have become a standard part of algorithm courses, and so should geometric algorithms. There is only one difficulty, namely that the area of geometric algorithms is much more diverse than graph algorithms so that it is difficult for a non-specialist to make a good choice of material if, say, only two to three weeks of lecture time are available. Sedgewick's algorithm text[1] realizes this idea and contains a chapter on geometric algorithms – next to chapters on sorting, searching, string processing, graph algorithms, and mathematical algorithms. I guess it is not too unusual for someone working in the area to disagree with Sedgewick's choice of topics. In graph algorithms there is a well-defined core of problems that everybody should know, such as depth-first and breadth-first search, shortest path finding, minimum spanning trees, etc., but there is no such set of established problems in geometric algorithms. My personal explanation for this state of affairs is that the area of geometric algorithms is, by definition, much larger than graph algorithms. After all, a graph is a discrete object, whereas geometry is an area that contains an inexhaustible collection of objects which are sometimes less and sometimes more complicated than graphs.

Below, I present a possible choice of material with the aim of showing as many aspects of geometric programming as possible. That is, there should be some elegant geometric reasoning, some numerical considerations, some interesting algorithmic paradigms, all grouped together in a coherent package. The idea is to give a taste of what computational geometry is and not to present the problems that are considered most important by any standard (assuming such problems can be identified at all).

[1] Sedgewick, R. *Algorithms*. Addison-Wesley, Reading, Mass., 2nd edition, 1988.



Let us start by briefly discussing what some of the different aspects of a geometric algorithm are or can be. In other words, what are the right questions we can ask and spend some well-used class-time to answer them? Here is a group of questions which will be answered later.

(i) What is the data and how can it be represented?

(ii) How can we implement geometric primitives?

(iii) How can we distinguish the combinatorial from the numerical part of a geometric program?

(iv) What mathematical understanding is necessary or useful in computational geometry?

We propose to discuss geometric algorithms in class after the chapter on graph algorithms (this is a different ordering than suggested in[1]) so that basic notions and manipulations on graphs can be assumed. With this assumption it is natural to talk about geometric graphs, such as subdivisions or alike. The material proposed below indeed focuses on general triangulations, Delaunay triangulations, and Voronoi diagrams which are different types of subdivisions of the plane. We now enter the more technical part of the discussion. It consists of eight sections:

1. Planar graphs, Euler's relation, and maximal planar graphs.
2. Triangulating a point set by plane-sweep.
3. Delaunay triangulations and Voronoi diagrams in the plane.
4. A few applications of Delaunay triangulations.
5. Lawson's diagonal-flip algorithm for Delaunay triangulations.
6. Testing the local Delaunayhood of an edge.
7. Three-dimensional interpretation of triangulations and of Lawson's algorithm.
8. A conceptual perturbation for coping with degenerate cases.

Each section addresses a different aspect of computational geometry, but together they form a well rounded discussion of one theme: Delaunay triangulations of finite point sets in the plane.

**1. Planar graphs, Euler's relation, and maximal planar graphs.** One advantage of talking about geometric algorithms after the discussion of graph algorithms

is the possibility of a gradual transition from graphs to geometry. A natural choice for the transition are planar graphs.

So we can introduce the notions of *plane embeddings* of graphs and that of *planar graphs*. An embedding of a planar graph generates a *subdivision* of the plane into *faces* or *regions*. Let $\mathcal{G} = (V, E)$ be a planar graph that is simple (no multiedges, no loops) and write $v$ and $e$ for the cardinalities of the vertex set, $V$, and the edge set, $E$. Using induction over the number of edges, it is straightforward to prove Euler's relation which is

$$v - e + f = 2,$$

if $\mathcal{G}$ is connected and $f$ is the number of faces of any plane embedding of $\mathcal{G}$. The extension of this relation to the case where $\mathcal{G}$ is not connected can now be shown if so desired.

A *maximal* planar graph is one where no edge can be added without violating planarity; thus, all faces of a maximal planar graph are bounded by exactly three edges. It follows that $3f = 2e$, and, together with Euler's relation, this can be used to show that

$$e \leq 3v - 6 \quad \text{and} \quad f \leq 2v - 4$$

for arbitrary planar graphs if $v \geq 3$ (why are the inequalities not true if $v \leq 2$?). From an averaging argument we also get that every planar graph has a vertex of degree at most 5. The above topological results will be useful later when geometric questions are considered. □

**2. Triangulating a point set by plane-sweep.** For a given finite point set, $P$, in the plane, a *triangulation* is a planar graph that is maximal contingent upon $V = P$ and the edges can be embedded as non-intersecting straight line segments. This is a natural place to introduce notions such as *convexity* and the *convex hull* of a point set. In fact, every triangulation of $P$ is a subdivision of the convex hull of $P$ into triangles; only the unbounded face is not necessarily a triangle.

Now comes the first geometric algorithm: given $P$, construct a triangulation of $P$ (any triangulation will do). The probably simplest algorithm that solves this problem sorts the points from left to right (increasing $x_1$-coordinates) and adds one point after the other according to this ordering. Let $p_1, p_2, \ldots, p_n$ be the $n$ points from left to right. To get the algorithm started, we can first construct the triangle spanned by points $p_1$, $p_2$, and $p_3$. Then we add $p_4$, next $p_5$, and so on and so forth. What does it mean to "add point $p_i$" to the current triangulation? It means that we connect $p_i$ to all vertices of the current triangulation that are "visible" from $p_i$. How can we find these visible points? For one thing, $p_{i-1}$ is visible from $p_i$ because it is the rightmost of all points to the left of $p_i$. After connecting $p_i$ to $p_{i-1}$ we traverse the unbounded face of the current triangulation, starting at $p_{i-1}$, in counterclockwise order and add the edge from $p_i$ to the current point $q$ if $(p_i, q, succ(q))$ is a right-turn, where $succ(q)$ is the next point after $q$ on the boundary of the unbounded face (yes, the unbounded



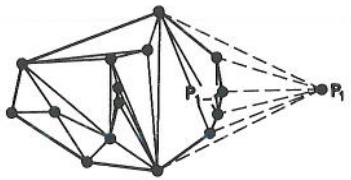


Figure 0.1: When we add $p_i$ we connect it to $p_{i-1}$ and to all other visible vertices by scanning the current unbounded face, once in counterclockwise and once in clockwise order.

face has a boundary too). After that we do the same scan, again starting at $p_{i-1}$, in clockwise order (see Figure 0.1). This algorithm was generalized to $d \geq 3$ dimensions by Raimund Seidel and can be found in[2].

There are several interesting questions that we have to answer now. First, what do we do if two or more points have the same $x_1$-coordinate, if three points are collinear, etc. We propose to defer all degenerate position problems to later and to temporarily pretend that such cases do not occur.

Second, how do we represent a triangulation? We could use the adjacency structure of the underlying graph (introduced earlier in the graph algorithms section), but the quad-edge data structure[3] is a more elegant choice which supports all the geometric operations one needs (such as walking along the boundary of a face, or walking around a vertex). No fear, it is not necessary to present the full-fledged quad-edge structure plus procedures as given in[3] because we talk only about planar graphs in the (one-sided) plane.

Third, what is a good way to decide whether a sequence of three points, $(p, q, r)$, forms a left- or a right-turn? The most elegant way uses determinants. Write $p = (\pi_1, \pi_2)$, $q = (\psi_1, \psi_2)$, and $r = (\rho_1, \rho_2)$. Then, $(p, q, r)$ is a left-turn iff the determinant of

$$\begin{pmatrix} \pi_1 & \pi_2 & 1 \\ \psi_1 & \psi_2 & 1 \\ \rho_1 & \rho_2 & 1 \end{pmatrix}$$

is positive. If the determinant is negative then the three points form a right-turn, and if it is zero then they are collinear. As mentioned above we assume that no three points are collinear and thus design our algorithms without any branch for the case the determinant vanishes. Later we introduce a conceptual perturbation method (a

[2]Edelsbrunner, H. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, Germany, 1987.

[3]Guibas, L. J. and Stolfi, J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graphics* 4 (1985), 74–123.

systematic way to treat degenerate cases) that takes care of all those temporarily undesired cases.

For the analysis of the above algorithm note that $O(n \log n)$ time is needed to sort the points; the additional time is $O(n)$ because each step creates a new edge and from Euler's relation we know that the total number of edges of a triangulation cannot be more than $3n - 6$. □

**3. Delaunay triangulations and Voronoi diagrams in the plane.** The Delaunay triangulation of a point set $P$, $\mathcal{D}(P)$, is named after the Russian mathematician Boris Delaunay[4], also Delone. It is rather straightforward to give a definition if $P$ is finite:

> an edge connecting $p, q \in P$ belongs to $\mathcal{D}(P)$ if there is a circle through $p$ and $q$ so that no point of $P$ lies inside and no other point of $P$ lies on the circle (see Figure 0.2).

It is less straightforward to see that this really defines a triangulation – in fact, it does not necessarily if four or more cocircular points are allowed in $P$, but if we exclude these degenerate cases (and we do exclude them) then $\mathcal{D}(P)$ is always a triangulation of $P$. How can we see that? Maybe the easiest way is by first introducing so-called Voronoi diagrams.

> The *Voronoi region* of a point $p \in P$, $V(p)$, is the set of points $x$ in the plane that are closer to $p$ than to any other point in $P$. The *Voronoi diagram* of $P$, $\mathcal{V}(P)$, is the subdivision defined by the Voronoi regions of the points of $P$ (see Figure 0.2).

By definition of Delaunay triangulation and Voronoi diagram we have $V(p)$ and $V(q)$ adjacent iff the edge connecting $p$ and $q$ is an edge of $\mathcal{D}(P)$. This implies that every vertex of $\mathcal{V}(P)$ corresponds to a triangle of $\mathcal{D}(P)$, every edge of $\mathcal{V}(P)$ corresponds to an edge of $\mathcal{D}(P)$, and every region of $\mathcal{V}(P)$ corresponds to a vertex of $\mathcal{D}(P)$ (see Figure 0.2). From our consideration of planar graphs we know that $\mathcal{D}(P)$ has at most $3n - 6$ edges and at most $2n - 5$ triangles, $n = |P|$. By the "dual" correspondence between $\mathcal{V}(P)$ and $\mathcal{D}(P)$ we thus conclude that $\mathcal{V}(P)$ has at most $3n - 6$ edges and at most $2n - 5$ vertices. □

**4. A few applications of Delaunay triangulations.** Maybe the best justification for choosing Delaunay triangulations as the central topic of the chapter on geometric algorithms is the elegance of the concept and the inherent magic that reveals a lot of the beauty of geometry (as we will see a little later). Still, it is difficult to present

[4]Delaunay, B. Sur la sphère vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* **7** (1934), 793-800.



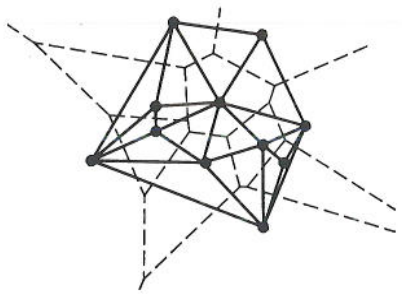

Figure 0.2: The solid edges show the Delaunay triangulation of $P$ and the broken edges belong to the Voronoi diagram of $P$.

this as a convincing argument to novices in geometry for which reason it might be helpful to briefly discuss a few applications of Delaunay triangulations.

A tree with vertex set $P$ is a *minimum spanning tree* of $P$ if the total edge length is a minimum. For minimum spanning trees of finite points sets in the plane we have the following result.

> Every minimum spanning tree of $P$ is a subgraph of $\mathcal{D}(P)$.

To prove this result one can use the lemma that if $e = \{a, b\}$ is an edge of a minimum spanning tree then there is a partition of $P$ into sets $A$ and $B$, $a \in A$ and $b \in B$, so that $e$ is a shortest edge between $A$ and $B$. It follows that $e$ is an edge of $\mathcal{D}(P)$ since there is no point of $B$ inside the circle through $b$ centered at $a$ and no point of $A$ inside the circle through $a$ centered at $b$; thus, the circle through $a$ and $b$ centered at $\frac{a+b}{2}$ is a witness for the Delaunayhood of $e$. With this result we can construct a minimum spanning tree of $P$ by first constructing $\mathcal{D}(P)$ and second running any fast minimum spanning tree algorithm on $\mathcal{D}(P)$. Indeed, this is the only known $O(n \log n)$ time algorithm for the problem. Note, however, that the algorithm for constructing $\mathcal{D}(P)$ that we will describe below does not run in $O(n \log n)$ time, but there are others that do, see [2] or[5].

The *Gabriel graph* of $P$, $\mathcal{G}(P)$, is the graph with vertex set $P$ that contains an edge $\{p, q\}$ if the circle through $p$ and $q$ with center $\frac{p+q}{2}$ is a witness of its Delaunayhood. This graph has found applications in geography[6] and other areas. It is straightforward from the definition that $\mathcal{G}(P)$ is a subgraph of $\mathcal{D}(P)$ and that it contains every minimum spanning tree of $P$. To construct $\mathcal{G}(P)$ we can construct $\mathcal{D}(P)$ first and then delete all edges that do not intersect their dual Voronoi edges.

[5]Fortune, S. A sweepline algorithm for Voronoi diagrams. *Algorithmica* **2** (1987), 153–174.

[6]Matula, D. W. and Sokal, R. R. Properties of Gabriel graphs relevant to geographic variation research and clustering of points. *Geogr. Analysis* **12** (1980), 205–222.

The Delaunay triangulation of $P$ somehow captures the relative nearness of the points in $P$, and this is really the reason why the Gabriel graph of $P$ and every minimum spanning tree of $P$ are subgraphs of $\mathcal{D}(P)$. It is thus not surprising that $\mathcal{D}(P)$ also contains the *nearest neighbor graph* of $P$ that contains a directed edge $(p, q)$ if no other point of $P$ is closer to $p$ than $q$. To compute all nearest neighbor pairs we first construct $\mathcal{D}(P)$ and then, for each $p \in P$, test all Delaunay neighbors of $p$. Since $\mathcal{D}(P)$ has at most $3n - 6$ edges the second step takes only time $O(n)$.

Finally, we mention the application of Delaunay triangulations to finite element methods. These methods can be based on a triangulation of the domain of interest. A sufficient condition for the effectiveness of the triangulation is that $\alpha + \beta < \pi$ for every interior edge $e$, where $\alpha$ and $\beta$ are the angles of the two incident triangles that are opposite to $e$. As we will see later, this condition is satisfied only if the triangulation is a Delaunay triangulation. □

**5. Lawson's diagonal-flip algorithm for Delaunay triangulations.** This is not the most efficient algorithm for constructing $\mathcal{D}(P)$ but one of the simplest and most interesting ones. It starts with an arbitrary triangulation of $P$ (for example with the one constructed by the left-to-right sweep as described above) and modifies this triangulation through a sequence of local changes. Let $\mathcal{T}$ be an arbitrary triangulation of $P$ and let $e = \{p, q\}$ be an edge of $\mathcal{T}$. We say that $e$ is *locally Delaunay* if $e$ is a convex hull edge or there is a circle through $p$ and $q$ so that the respective third vertices of the two incident triangles lie outside the circle. Note that a locally Delaunay edge is not necessarily an edge of the Delaunay triangulation, but every edge of $\mathcal{D}(P)$ is also locally Delaunay. An elementary geometry lemma implies that if $e$ is locally Delaunay but not a convex hull edge then $\alpha + \beta < \pi$ with $\alpha$ and $\beta$ as defined above. The algorithm can now be described.

**Step 1.** Construct an arbitrary triangulation of $P$.
**Step 2.** Check all edges and push non-locally Delaunay edges onto a stack.
**Step 3.** **while** the stack is non-empty **do**
  $\{p, q\}$ := POP;
  **if** $\{p, q\}$ is non-locally Delaunay **then** replace $\{p, q\}$ by the edge connecting the third vertices of the two incident triangles
  **endif**;
  Push the other four edges of the two triangles onto the stack, unless they are already there
**endwhile**.

We will talk about the correctness of Lawson's algorithm and how to implement the test for local Delaunayhood later. For now let us just convince ourselves that the algorithm eventually halts. Whenever we do a diagonal-flip we retriangulate a convex quadrilateral. Such a triangulation consists of two triangles and thus uses six angles.



Using the elementary geometry lemma mentioned before it is not difficult to show that the smallest of the six new angles is larger than the smallest of the six old angles. Thus, the non-decreasing angle vector of the triangulation increases lexicographically if we do a diagonal-flip. This implies that the algorithm does not run into a cycle, and since there are only finitely many triangulations of $P$, it must finally halt. □

**6. Testing the local Delaunayhood of an edge.** An edge, $e$, incident to two triangles, $t_1$ and $t_2$, is locally Delaunay if there is a circle through $a$ and $b$, the two endpoints of $e$, so that $c$ and $d$, the other vertices of $t_1$ and $t_2$, lie outside the circle. Such a circle exists iff $d$ lies outside the unique circle through $a$, $b$, and $c$. Thus, to decide whether or not an edge is locally Delaunay we can perform a so-called *in-circle test*[3]:

> given a sequence of four points, $a$, $b$, $c$, $d$, decide whether $d$ lies inside, on, or outside the circle spanned by $a$, $b$, and $c$.

A formula for this test consisting of one four-by-four and one three-by-three determinant can be developed using the following geometric transform. For a point $p = (\pi_1, \pi_2)$ define $p^+ = (\pi_1, \pi_2, \pi_1^2 + \pi_2^2)$ ($p^+$ is the vertical projection of $p$ to the paraboloid $U : x_3 = x_1^2 + x_2^2$). This transform turns out to be quite useful for our needs because the vertical projection into the $x_1x_2$-plane of $h \cap U$, $h$ a non-vertical plane, is a (possibly empty) circle – this is real easy to prove using analytic geometry. It follows that point $d$ lies inside (on, outside) the circle through $a$, $b$, and $c$ iff $d^+$ lies vertically below (on, vertically above) the plane through $a^+$, $b^+$, and $c^+$. Now we can use elementary analytic geometry and eventually arrive at the following statement which ignores the degenerate case where the four points are cocircular.

> Point $d = (\delta_1, \delta_2)$ lies inside (outside) the circle through points $a = (\alpha_1, \alpha_2)$, $b = (\beta_1, \beta_2)$, and $c = (\gamma_1, \gamma_2)$ iff the sign of the determinant of the matrix
>
> $$\begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_1^2 + \alpha_2^2 & 1 \\ \beta_1 & \beta_2 & \beta_1^2 + \beta_2^2 & 1 \\ \gamma_1 & \gamma_2 & \gamma_1^2 + \gamma_2^2 & 1 \\ \delta_1 & \delta_2 & \delta_1^2 + \delta_2^2 & 1 \end{pmatrix}$$
>
> is equal to (different from) the sign of the determinant of the matrix
>
> $$\begin{pmatrix} \alpha_1 & \alpha_2 & 1 \\ \beta_1 & \beta_2 & 1 \\ \gamma_1 & \gamma_2 & 1 \end{pmatrix}.$$

We need the determinant of the second matrix in order to compensate for permutations of the first three points. For example, notice that the four-by-four matrix changes the sign of the determinant if we swap points $a$ and $b$. Still, the result should

be the same since swapping $a$ with $b$ does not change the circle through $a$, $b$, and $c$. □

**7. Three-dimensional interpretation of triangulations and of Lawson's algorithm.** The mapping from two to three dimensions used to develop the in-circle test can be applied more globally. If $P$ is the set of points in the plane then define $P^+ = \{p^+ \mid p \in P\}$, that is, $P^+$ is the vertical projection of $P$ to the paraboloid $U$. Let $\mathcal{P}^+$ be the convex hull of $P^+$, and call a vertex, edge, and facet of $\mathcal{P}^+$ *lower* if it can be seen from underneath, that is, from point $(0, 0, -\infty)$. Three points $a^+$, $b^+$, $c^+$ form a lower facet of $\mathcal{P}^+$ iff all other points of $P^+$ lie vertically above the plane through $a^+$, $b^+$, $c^+$, which is the case iff all other points of $P$ lie outside the circle through $a$, $b$, $c$. In other words, the vertical projection of all lower vertices, edges, and facets of $\mathcal{P}^+$ yields the Delaunay triangulation of $P$. Algorithmically, this could be used to construct $\mathcal{D}(P)$ via a three-dimensional convex hull algorithm, but this is not our goal. We are going to use the transform to prove Lawson's algorithm correct, to give a time analysis, and to derive an optimality property of Delaunay triangulations.

The only way Lawson's algorithm could fail to construct $\mathcal{D}(P)$ is if it gets stuck with a triangulation $\mathcal{T} \neq \mathcal{D}(P)$ which has no locally non-Delaunay edge. If we raise $\mathcal{T}$ up to three dimensions by projecting $P$ to $U$ we get a piecewise linear surface which is not convex, for otherwise $\mathcal{T}$ would be $\mathcal{D}(P)$. But then there is also an edge so that the dihedral angle below the edge is less than $\pi$. Such an edge corresponds to a locally non-Delaunay edge in the $x_1x_2$-plane, a contradiction.

Let us think about this argument for a minute. It means that whatever triangulation $\mathcal{T}$ of $P$ we have, if $\mathcal{T} \neq \mathcal{D}(P)$ we can apply a diagonal-flip step which lexicographically increases the sorted angle vector of the triangulation. This immediately implies that $\mathcal{D}(P)$ maximizes the angle vector. In particular,

> over all triangulations of $P$, the smallest angle over all triangles of the triangulation is a maximum for the Delaunay triangulation of $P$.

Another corollary of the above considerations is a characterization of $\mathcal{D}(P)$: a triangulation $\mathcal{T}$ of $P$ is the Delaunay triangulation of $P$ iff every edge is locally Delaunay or, equivalently, $\alpha + \beta < \pi$ for every interior edge of $\mathcal{T}$.

We have seen how an arbitrary triangulation $\mathcal{T}$ of $P$ corresponds to a piecewise linear surface in three dimensions. In this space, what does it mean to perform a diagonal-flip? Such a flip is performed only at an edge, $e$, whose lower dihedral angle is less than $\pi$. The diagonal-flip can be interpreted as gluing the tetrahedron spanned by the two incident triangles of $e$ to the piecewise linear surface. When viewed from underneath, $e$ cannot be seen any longer; instead we see the other edge of the quadrilateral. Thus, every step of Lawson's algorithm adds another tetrahedron and eventually constructs a cell complex of tetrahedra in three dimensions. This cell



complex has $n$ vertices and can therefore have at most $O(n^2)$ tetrahedra. (Why is that? Because every vertex can be incident to at most $2n - 6$ tetrahedra.) This implies that Lawson's algorithm runs in $O(n^2)$ time in the worst case. Indeed, there are examples where it takes $\Theta(n^2)$ time[7], but it is not known how many diagonal-flips it takes "on the average". □

**8. A conceptual perturbation for coping with degenerate cases.** This concludes the chapter on geometric algorithms – except that we did not yet cover degenerate cases, and an algorithm is not correct unless it works for all cases, no? The idea is to use a tiny bit of symbolic computation that simulates non-degeneracy if degenerate input is fed to the program. This method, developed in[8] is termed the Simulation of Simplicity, where "simplicity" is used as a synonym for "non-degeneracy". Here is how it works.

Let $P = \{p_1, p_2, \ldots, p_n\}$ be the input, with $p_i = (\pi_{i,1}, \pi_{i,2})$. There is a degeneracy if three points are collinear, that is,

$$\det \begin{pmatrix} \pi_{i,1} & \pi_{i,2} & 1 \\ \pi_{j,1} & \pi_{j,2} & 1 \\ \pi_{k,1} & \pi_{k,2} & 1 \end{pmatrix} = 0,$$

or if four points are cocircular, that is,

$$\det \begin{pmatrix} \pi_{i,1} & \pi_{i,2} & \pi_{i,1}^2 + \pi_{i,2}^2 & 1 \\ \pi_{j,1} & \pi_{j,2} & \pi_{j,1}^2 + \pi_{j,2}^2 & 1 \\ \pi_{k,1} & \pi_{k,2} & \pi_{k,1}^2 + \pi_{k,2}^2 & 1 \\ \pi_{\ell,1} & \pi_{\ell,2} & \pi_{\ell,1}^2 + \pi_{\ell,2}^2 & 1 \end{pmatrix} = 0.$$

We replace the coordinates by polynomials in $\epsilon$, where $\epsilon$ is an indeterminant that is supposed to be a positive by sufficiently (in fact, arbitrarily) small real number. The polynomials are chosen so that the new point set is non-degenerate and so that the primitive operations (left/right-turn and in-circle tests) can still be computed fast. Both goals can be met if we replace

$$p_i = (\pi_{i,1}, \pi_{i,2}) \quad \text{by} \quad p_i(\epsilon) = (\pi_{i,1} + \epsilon^{2^{2i-1}}, \pi_{i,2} + \epsilon^{2^{2i-2}}).$$

It is a simple exercise to show that for $\epsilon > 0$ sufficiently small no three perturbed points can be collinear and no four perturbed points can be cocircular.

So how about the fast simulation? The polynomials seem rather complicated; even calculating the exponents of $\epsilon$ would be too costly. Fortunately, there is a way around it. Consider a left/right-turn test for points $(p_i, p_j, p_k)$ and assume $i < j < k$

[7] Fortune, S. A note on Delaunay diagonal flips. Manuscript, AT&T Bell Lab., Murray Hill, New Jersey, 1987.

[8] Edelsbrunner, H. and Mücke, E. P. Simulation of Simplicity: a technique to cope with degenerate cases in geometric algorithms. In "Proc. 4th Ann. Sympos. Comput. Geom. 1988", 118–133.

(otherwise, exchange points and remember the parity of the number of exchanges). As mentioned above we have to compute the sign of

$$D = \det \begin{pmatrix} \pi_{i,1} + \epsilon^{2^{2i-1}} & \pi_{i,2} + \epsilon^{2^{2i-2}} & 1 \\ \pi_{j,1} + \epsilon^{2^{2j-1}} & \pi_{j,2} + \epsilon^{2^{2j-2}} & 1 \\ \pi_{k,1} + \epsilon^{2^{2k-1}} & \pi_{k,2} + \epsilon^{2^{2k-2}} & 1 \end{pmatrix}.$$

This determinant is itself a polynomial in $\epsilon$. If we sort the terms in order of decreasing exponents we get

$$D = \det \begin{pmatrix} \pi_{i,1} & \pi_{i,2} & 1 \\ \pi_{j,1} & \pi_{j,2} & 1 \\ \pi_{k,1} & \pi_{k,2} & 1 \end{pmatrix} - \epsilon^{2^{2i-2}} \cdot \det \begin{pmatrix} \pi_{j,1} & 1 \\ \pi_{k,1} & 1 \end{pmatrix} + \epsilon^{2^{2i-1}} \cdot \det \begin{pmatrix} \pi_{j,2} & 1 \\ \pi_{k,2} & 1 \end{pmatrix} +$$

$$+\epsilon^{2^{2j-2}} \cdot \det \begin{pmatrix} \pi_{i,1} & 1 \\ \pi_{k,1} & 1 \end{pmatrix} + \epsilon^{2^{2j-2}} \cdot \epsilon^{2^{2i-1}} + \ldots$$

Note that the sign of the first non-zero coefficient decides the sign of $D$ because all later terms are astronomically smaller than this term. Thus, the simulation consists of computing, in turn, the coefficients of the polynomial $D$ and stopping when the first non-zero coefficient is found. It is reassuring that the first term is exactly the test for the unperturbed points; thus, if the unperturbed points are non-degenerate then the test is just the same as before. □

The above eight points contain material on geometric algorithms that can be worked into a course on algorithms. To present the material in class takes about two to three weeks. Except for minor changes, the above is exactly what I taught in the fall semester of 1988, and it took me about seven full hours of lecture time. Upon request I will be happy to provide additional details and possibly copies of my class notes. Some of the important concepts in geometric algorithms covered by the above material are

> planarity, Euler's relation, convex hull, triangulation, quad edge structure, Delaunay triangulation, Voronoi diagram, plane-sweep algorithm, geometric transform, primitive geometric operation, conceptual perturbation.

Most importantly, the above collection of material demonstrates the richness of the area and the relations between seemingly different concepts (such as three-dimensional convex hulls and two-dimensional Delaunay triangulations).



# Two Small Results

In every area of research there are results that fall through the cracks ... I guess. In any case, some of these results turn out to be useful and often several people thought of them independently. Then they become common knowledge – inside a limited community. Often, such results are extensions or modifications of published work, or corollaries thereof, that did not quite make the deadline. Now they are wondering about in the wilderness. A column like this seems like a good shelter for such orphans and we can put them on display in case anybody wants to adopt them. The shelter may not be as comfortable or respected a place as a refereed journal, but sometimes it is not appropriate to be overly picky.

I use this issue of the computational geometry column to present two such orphans and explain how they came about and why their family status is somewhat questionable. The reader is invited to comment on the presented orphans, to curiously explore them, and to bring other orphans to my attention if he or she feels that this shelter is an appropriate and warm home. It is understood that we can give a home only to orphans that fall into the categories known as "discrete geometry", sometimes called "combinatorial geometry", and "computational geometry", also known as "algorithmic geometry".

The first result we discuss is on triangulations in the plane, the second deals with lines that stab a collection of triangles in the plane. In both cases we start by introducing some terminology and then proceed to describe the result. We also briefly sketch the history, to the extent known to me, and review some of the relatives with more respected social status.

## 1 Non-Obtuse Triangulations

A *triangulation* of a finite point set $S$ in the plane is an embedding of a planar graph with vertex set $S$ so that

1. each edge is a straight line segment, and
2. every face of the embedding is a triangle, except the unbounded face which is equal to the complement of the convex hull of $S$.

By Euler's formula for planar graphs, we have

$$n - e + t = 1,$$

where $n = |S|$, $e$ is the number of edges of the triangulation, and $t$ is the number of triangles (we do not include the unbounded face). By the way, from this equation and because every triangle is bounded by three edges and every edge bounds at most two triangles, we get $e \leq 3n - 6$ and $t \leq 2n - 5$. So every triangulation of a set of $n$ points consists of only $O(n)$ edges and triangles.

There is a special triangulation of $S$, called the *Delaunay triangulation*, $\mathcal{D}(S)$, of $S$, which is rather important because of various applications including the generation of meshes for finite element methods. $\mathcal{D}(S)$ can be defined as follows.

> An edge $pq$ is in $\mathcal{D}(S)$ if and only if there is a circle through points $p$ and $q$ so that all other points of $S$ lie strictly outside this circle.

Notice that the edges that belong to the triangulation determine the entire triangulation, that is, the triangles do not have to be specified separately. Note also that the convex hull edges belong to $\mathcal{D}(S)$ because there is always a large enough circle that approximates the line through the edge in the neighborhood of $S$, and there is no other point of $S$ on or inside the circle because one open half-plane bounded by the line through the edge contains no point of $S$.

The critical reader may wonder whether or not this definition really specifies a triangulation. In fact, it does if we assume general position of the points which in this case means that no four points are cocircular; still it is not trivial to see why this is so. If $S$ is not in general position then it can be that some of the bounded faces of $\mathcal{D}(S)$ are bounded by more than three edges. For example, if $S$ contains $k \geq 3$ points on a circle and no point of $S$ lies inside the circle, then the $k$ points define a face with $k$ edges in $\mathcal{D}(S)$. To force a triangulation we can either decompose such faces into triangles by drawing appropriate diagonals or we can simulate a perturbation of the points to guarantee general position. Below, we assume general position of the points.

The Delaunay triangulation of $S$ satisfies a certain angle condition which turns out to be important for finite element applications.

**Lemma.** Let $\mathcal{T}$ be a triangulation of $S$ and for each edge $e$ that does not lie on the boundary of the convex hull of $S$ let $\alpha_e$ and $\beta_e$ be the two opposite angles of $e$ (see Figure 1.1). Then $\mathcal{T} = \mathcal{D}(S)$ if and only if $\alpha_e + \beta_e < \pi$ for each such edge $e$ of $\mathcal{T}$.

In the finite element literature it is proven that this condition is exactly what is needed for certain approximation methods. Before the event of this proof it was known that a sufficient condition for the well-behavior of the approximation methods



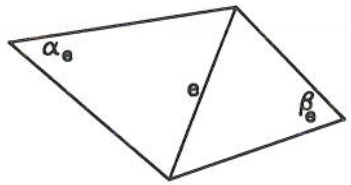


Figure 1.1: The two opposite angles of an edge.

is that all angles of the triangulation are *non-obtuse*, that is, at most $\frac{\pi}{2}$. The existence and construction of *non-obtuse* triangulations, triangulations with non-obtuse angles only, is interesting from a theoretic point of view too. Below, we present a result which will enable us to decide, for a given set $S$, whether or not there is a non-obtuse triangulation of $S$.

**Baby theorem.** There is a non-obtuse triangulation, $\mathcal{T}$, of $S$ iff $\mathcal{T} = \mathcal{D}(S)$ and $\mathcal{D}(S)$ is non-obtuse.

**Proof.** Notice that if $\mathcal{T}$ is non-obtuse then $\alpha_e + \beta_e < \pi$ for every edge incident to two triangles. Therefore $\mathcal{T} = \mathcal{D}(S)$, if it exists. □

So in order to construct a non-obtuse triangulation of $S$ just compute the Delaunay triangulation of $S$ – and there are many efficient algorithms known for this problem[1] [2] – and test whether it is non-obtuse. If it is not, then there does not exist any non-obtuse triangulation of $S$.

It is also interesting to ask how many angles of $\mathcal{D}(S)$, on the average, are non-obtuse. One answer to this question is that if $S$ is a set of $n$ points chosen independently and uniformly in the unit-square, then about one sixth of the at most $6n - 15$ angles are obtuse. This follows from a result of Matula and Sokal[3] that about one third of all edges of $\mathcal{D}(S)$ do not intersect the dual Voronoi edge (this is equivalent to saying that for each such edge $pq$ not all other points of $S$ lie outside the circle through $p$ and $q$ whose center is the midpoint of $p$ and $q$). The result follows because there is a one-to-one correspondence between such edges and obtuse angles.

There are very few papers in the literature that address questions about triangulations that satisfy certain criteria, such as all angles be non-obtuse. The only paper I am aware of that discusses the non-obtuse angle criterion is[4] where it is shown that

[1] Edelsbrunner, H. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg, Germany, 1987.

[2] Fortune, S. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2 (1987), 153–174.

[3] Matula, D. W. and Sokal, R. R. Properties of Gabriel graphs relevant to geographic variation research and clustering of points in the plane. *Geographical Analysis* **12** (1980), 205–222.

[4] Baker, B. S., Grosse, E. and Rafferty, C. S. Nonobtuse triangulation of polygons. *Discrete Comput. Geom.* **3** (1988), 147–168.

every simple polygon in the plane can be triangulated so that no angle exceeds $\frac{\pi}{2}$. However, the paper does not give bounds on the number of new vertices that have to be introduced in order to obtain such a triangulation. To my knowledge, related problems such as minimizing the maximum angle for a given point set, or minimizing the maximum number of obtuse angles are open. On the other hand, it is known that the Delaunay triangulation maximizes the minimum angle over all triangulations of the same point set.

## 2 Stabbing Triangles

Let $S$ be a set of $n$ triangles in the plane, not necessarily disjoint. For simplicity we assume non-degenerate position, that is, no three collinear vertices and no two vertices on a common vertical line. A *transversal* of $S$ is a line that meets *all* triangles in $S$.

For many sets $S$ there will be no transversal, but if there exists one, how many other transversals can there be? This, of course, is a silly question because there are infinitely many other transversals as we assume non-degeneracy of the triangles which implies that for any transversal there is an entire "neighborhood" of transversals. Nevertheless, the question makes sense if we are careful about how we state it. Here is a finite representation of the (possibly infinite) set of transversals of $S$:

> map every line $\ell: \ y = \lambda_1 x + \lambda_2$ to the point $\ell^* = (\lambda_1, -\lambda_2)$, and define $\mathcal{T}(S)$, the *transversal region* of $S$, as the set of points $\ell^*$ so that $\ell$ is a transversal of $S$.

Never mind that the map $^*$ is not applicable to vertical lines.

How does $\mathcal{T}(S)$ look like? Is it connected? Is its boundary piecewise linear? The answer to the third question is affirmative while the answer to the second question is not. Let us take a look at the case of a single triangle after extending the map $^*$ to points:

> a point $p = (\pi_1, \pi_2)$ is mapped to the line $p^*: \ y = \pi_1 x - \pi_2$.

The map preserves order, that is, $p$ lies (vertically) above a line $\ell$ if and only if the point $\ell^*$ lies (vertically) above the line $p^*$.

The three vertices of a triangle $t$, points $p$, $q$, and $r$, are mapped to three lines, $p^*$, $q^*$, and $r^*$, and a line $\ell$ intersects $t$ if and only if the point $\ell^*$ does not lie above all three lines or below all three lines (see Figure 2.2). Call the set of all such points the *transversal region* of $t$, $\mathcal{T}(t)$. The boundary of $\mathcal{T}(t)$ consists of two polygonal



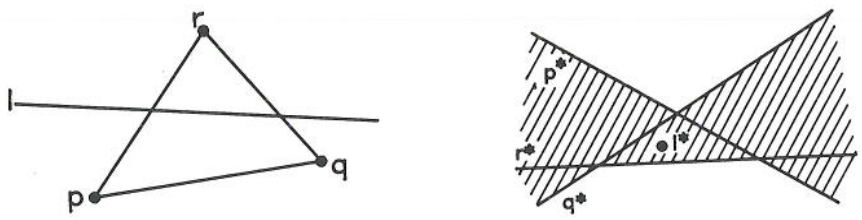


Figure 2.2: The shaded region is the set of all points $\ell^*$ so that $\ell$ intersects the triangle.

paths, both unbounded (the "unbounded boundary" ... never mind), monotone in $x$, and consisting of two or three edges each. Call the upper boundary $\mathcal{U}(t)$, for *upper envelope* of $p^*$, $q^*$, $r^*$, and the lower boundary $\mathcal{L}(t)$, for *lower envelope* of $p^*$, $q^*$, $r^*$.

So how about a set $S$ that contains more than just one triangle. Well, if $S = \{t_1, t_2, \ldots, t_n\}$ then

$$\mathcal{T}(S) = \bigcap_{i=1}^{n} \mathcal{T}(t_i).$$

In other words, it is the set of points below every $\mathcal{U}(t_i)$ and above every $\mathcal{L}(t_i)$. Now it should be easy for the reader to construct examples so that $\mathcal{T}(S)$ is not connected. What is the maximum number of edges $\mathcal{T}(S)$ can have? Here is the somewhat surprising answer to this question.

**Baby theorem.** The maximum number of edges in the boundary of $\mathcal{T}(S)$, over all sets $S$ of $n$ triangles in the plane, is $\Theta(n\alpha(n))$.

Recall that $\alpha(n)$ is the inverse of the extremely fast growing Ackermann function[5]. It is notorious for growing slowly and it is often ridiculed for this property – still, it goes to infinity, eventually.

Oh well ... where does the upper bound come from? Of course, Davenport-Schinzel sequences of order 3 can be used[6]. Take the lower envelope of the collection of $n$ upper envelopes $\mathcal{U}(t_i)$ and read the names of its edges from left to right. An edge can occur repeatedly, actually quite often sometimes, but there is no not necessarily contiguous subsequence of the form

$$\ldots a \ldots b \ldots a \ldots b \ldots a \ldots$$

with $a \neq b$. A sequence with this property is called a *Davenport-Schinzel sequence* of order 3 if, in addition, no two adjacent symbols are equal. The quite sophisticated

[5] Ackermann, W. Zum Hilbertschen Aufbau der reellen Zahlen. *Math. Ann.* 99 (1928), 118–133.

[6] Hart, S. and Sharir, M. Nonlinearity of Davenport-Schinzel sequences and of a generalized path compression scheme. *Combinatorica* 6 (1986), 151–177.

analysis in[6] implies that the maximum length of such a sequence with $n$ different symbols is $O(n\alpha(n))$.

But then, maybe not every Davenport-Schinzel sequence is realizable as the lower envelope of upper envelopes $\mathcal{U}(t)$. Indeed this is a possibility. So why do we claim $\Omega(n\alpha(n))$ as a lower bound? There is a construction of $n$ line segments in the plane[7] so that the lower envelope of these line segments consists of $\Omega(n\alpha(n))$ pieces. We can use this construction as follows. Let $s$ be a line segment in the construction. Choose the triangle $t$ so that $\mathcal{U}(t)$ consists of $s$ plus a very very steep half-line with positive slope emanating from the right endpoint of $s$ and a very very steep half-line with negative slope emanating from the left endpoint of $s$. If we do this for every line segment in the construction we get a lower envelope of the $\mathcal{U}(t)$ which is the same as the lower envelope of the line segments, except that vertical edges of the latter are replaced by very very steep edges in the former.

Ok, so we understand the result, but where does it come from and why was it not adequately published? Some years ago[8] the problem of counting the edges of the transversal region of $n$ line segments in the plane has been studied. The maximum number of edges in this case is $O(n)$ which is a striking though subtle difference to the case of triangles. It is this subtlety that prevented the authors of this paper to look for a non-linear lower or upper bound. A few years later, the study of Davenport-Schinzel sequences was pursued by Hart and Sharir[6]. Their results imply an $O(n\alpha(n))$ upper bound for the transversal problem, as explained above, but it was not before the construction in[7] that it was realized that $O(n\alpha(n))$ is in fact tight. Who would have guessed that 7 years ago? You ask "and who would have cared"? Well, maybe $O(n)$ and $O(n\alpha(n))$ are not any different for any practical purpose, but if you try to prove a linear upper bound on something that grows proportional to $n\alpha(n)$ you have a frustrating time in front of you, and examples that suggest non-linear growth are difficult to get because they tend to be at least enormous.

All right, so how about an algorithm that constructs $\mathcal{T}(S)$, or better, the lower envelope of a set $S$ of $n$ line segments? The most obvious approach is by divide-and-conquer.

1. Split $S$ into two about equally large sets $S_1$ and $S_2$.
2. Compute the lower envelopes, $\mathcal{L}_1$ and $\mathcal{L}_2$, of $S_1$ and $S_2$ recursively.
3. Construct the lower envelope of $S$ by scanning $\mathcal{L}_1$ and $\mathcal{L}_2$ from left to right.

[7]Wiernik, A. and Sharir, M. Planar realizations of non-linear Davenport-Schinzel sequences by segments. *Discrete Comput. Geom.* 3 (1988), 15–47.

[8]Edelsbrunner, H., Maurer, H. A., Preparata, F. P., Rosenberg, R. L., Welzl, E. and Wood, D. Stabbing line segments. *BIT* 22 (1982), 274–281.



For the time, $T(n)$, we get $T(n) = 2T(\frac{n}{2}) + O(n\alpha(n)) = O(n\alpha(n)\log n)$. Hershberger[9] showed recently how to modify this algorithm so that it runs in time $O(n \log n)$.

The result in this section received brief mention in [7] and is stated as Exercise 15.7 in [1]. It is also mentioned in [10] where extensions to three and higher dimensions are considered. To my knowledge, there is no place in the literature that adequately describes the result and its place in discrete and computational geometry.

[9]Hershberger, J. Finding the upper envelope of $n$ line segments in $O(n \log n)$ time. Manuscript, 1989.

[10]Edelsbrunner, H., Guibas, L. J. and Sharir, M. The upper envelope of piecewise linear functions: algorithms and applications. *Discrete Comput. Geom.*, to appear.

# Computational Geometry Research in Japan

Because of the rapid development of computational geometry in Japan I thought it is a good idea to have a report in this column that gives some information about the people involved and the problems they are interested in. I am grateful to Hiroshi Imai[1] for agreeing to write such a report. Here it is.

Computational geometry is now one of the most active areas within the field of theoretical computer science in Japan. This can easily be seen from the large number of papers on computational geometry that have recently been presented at domestic research meetings. In the following I will list abstracts of some computational geometry papers presented at these meetings and a related conference in 1989 to illustrate the state-of-the-art of computational geometry research in Japan.

Before describing the abstracts, I explain the domestic and international activities of Japanese academic societies to encourage researchers in Europe, North America, and other parts of this world to visit Japan. There are now two organized research groups in theoretical computer science, one is the Special Interest Group on Algorithms (SIGAL) within the Information Processing Society of Japan and the other is the Technical Group on Computation (COMP) within the Institute of Electronics, Information and Communication Engineering of Japan. Each group has 8 to 10 research meetings per year, which are held at main cities all over Japan (Japan is small enough for researchers to travel frequently). In 1989, SIGAL organized 7 domestic research meetings and one international workshop, the latter jointly with COMP. At these seven domestic research meetings, 50 papers were presented 16 of which fall into the area of computational geometry. At the International Workshop on Discrete Algorithms and Complexity jointly organized by SIGAL and COMP and held in November 20–22, 1989, there were 36 talks (15 from Japan, 11 from Asia, 7 from North America, 2 from Europe, and 1 from Australia), and among them 12 were on topics in computational geometry. There is still a limited number of the proceedings still available and those who are interested in getting a copy may contact me directly.

Like the international workshop mentioned above, there were two international conferences organized in 1987 and 1988, each of which included a session consisting of about 10 papers in computational geometry. This year, 1990, from August 16 through

[1]Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka 812, Japan.



18, there will be the SIGAL International Symposium on Algorithms in Tokyo. This symposium will be mentioned again at the end of this report.

Now, I list abstracts of some computational geometry papers presented at the domestic research meetings and one presented at some international meeting. Implicitly, this list provides you with (a non-exhaustive set of) names of Japanese researchers in computational geometry and some indication of their special interest.

From SIGAL Research Meetings:

SIGAL 5-19

Takeshi Tokuyama (IBM Tokyo Research Laboratory, Tokyo), Takao Asano (Sophia University, Tokyo), Shuji Tsukiyama (Chuo University, Tokyo): *A Dynamic Algorithm for Placing Rectangles without Intersection.*

Construction of the contour of the union of rectangles is an important problem in computational geometry. In this paper, we define a special class of arrangements of rectangles, called *FIFO arrangements*, whose contour is constructed in $O(n \log \log n)$ time. Using the result, we solve the following dynamic allocation problem of rectangles efficiently: Given a rectangle and an orthogonal region with $n$ non-intersecting rectangular holes, find a possible placement of the rectangle in the region and create a new hole. We present an algorithm that takes $O(n \log \log n)$ time and $O(n)$ space; this algorithm finds not only one but all possible placements.

SIGAL 5-20

Masato Edahiro (NEC C&C Systems Research Laboratories, Kawasaki): *An Assignment Algorithm in VLSI Layout Design Using Voronoi Diagram.*

This paper presents a new algorithm for the driver-flip-flop reassignment problem after the placement of the semi-custom layout design. This algorithm, by using the Voronoi diagram, is quite efficient and gives a better solution than the existing techniques. The efficiency and quality of the solution are shown in experimental results.

SIGAL 6-3

Xuehou Tan, Tomio Hirata and Yasuyoshi Inagaki (Nagoya University, Nagoya): *The C-Oriented Polygon Intersection Problem.*

We examine the problem of reporting all intersecting pairs in a set of c-oriented polygons in the plane. A set of polygons is called c-oriented if the edges of all polygons have only a constant number of orientations. The problem arises in many applications such as the VLSI design rule checking and architecture or furniture databases. We present an optimal algorithm that runs in $O(n \log n + t)$ time and $O(n)$ space, where $n$ is the number of polygons and $t$ the number of intersecting pairs.

SIGAL 7-4

Tetsuo Asano (Osaka Electro-Communication University, Osaka) and Takeshi Tokuyama (IBM Tokyo Research Laboratory, Tokyo): *A Geometric Construction of Optimal Hash Function.*

Given a fixed set $S$ of keys and two arbitrary function $h_1$ and $h_2$ associated with each key, we want to find a hash function by linear combination of those functions that is optimal in some criterion. Various criteria may be considered, such as minimization of the maximum number of keys to be included in one bucket or minimization of the number of empty buckets. This paper presents three different efficient algorithms for finding such a hash function based on dual transform between points and lines in the plane. They improved the previous best result both in time and space complexities.

SIGAL 8-3

Naoki Katoh (Kobe University of Commerce, Kobe): *Finding a Bipartition of Points in the Plane by $L_2$-Intraset Metric Criteria.*

We consider problems of clustering a set of $n$ points in the plane into two subsets under three different optimization criteria based on $L_2$-intraset metric. The first is to minimize the square sum of the $L_2$ intraset metrics of partitioned subsets. The second is to minimize the sum of the $L_2$ intraset metrics of partitioned subsets. The third is to minimize the maximum of the $L_2$ intraset metrics of partitioned subsets. We present $O(n^3)$ algorithms for the first and second problems. Both algorithms make use of higher order Voronoi diagrams for $n$ points in the plane. A generalization of these problems to higher dimensions is also discussed. We present a fully polynomial time approximation scheme for the third problem.

SIGAL 9-4

Keiko Imai (Kyushu Institute of Technology, Iizuka): *On the Linearization Technique in Computing Multidimensional Davenport-Schinzel Sequences and Its Applications.*

We consider the problem of computing multidimensional Davenport-Schinzel sequences by using the linearization technique. This problem has strong connection with the combinatorial complexity of the lower (or upper) envelope of $n$ multivariate functions. In this paper, we present the linearization technique and its applications to the dynamic enclosing circle problem of moving points in the plane and the minimax geometric fitting problem of two corresponding sets of points.

SIGAL 10-5

Hiromi Aonuma, Hiroshi Imai (Kyushu University, Fukuoka) and Takeshi Tokuyama (IBM Tokyo Research Laboratory, Tokyo): *Some Voronoi Diagrams for Character Placing Problems in Map Databases.*



In map databases, it is often required to label regions (e.g., states, lakes) of maps automatically in a clear and unambiguous manner. In this paper, we focus on the problem of placing character strings approximately at the centers of their regions. By considering these regions as polygons and strings as rectangles, we may (1) find a maximum empty square inside a polygon, or (2) find a position of a rectangle such that the minimum distance between the rectangle and the boundary of the polygon is maximized. Here, the sides of the square and rectangles are restricted to lie parallel to a given set of axes. We use the $L_\infty$-Voronoi diagram of line segments to solve the first problem and introduce a new Voronoi diagram to solve the second. It is shown that both problems can be solved in $O(n \log n)$ time.

SIGAL 11-2

Toshiyuki Imai and Kokichi Sugihara (University of Tokyo, Tokyo): *A Combinatorial-Structure Oriented Algorithm for Voronoi Diagrams of Line Segments.*

An algorithm to construct Voronoi diagrams whose generators are line segments is shown. In this algorithm combinatorial structures of the diagrams are considered as higher priority information than numerical values. If the precision in computation is enough, this algorithm constructs the correct Voronoi diagrams. Moreover, even when precision in computation is so poor that conventional algorithms do not work well, this algorithm produces some output that is topologically consistent. In this sense, this algorithm is robust against numerical error. This algorithm is designed on the assumption that numerical error exists; hence it has not any exceptional rules for degeneracy but still works for degenerate cases.

Abstract from the First Canadian Conference on Computational Geometry:

Hiroyuki Ebara, Noriyuki Fukuyama, Hideo Nakano and Yoshiro Nakanishi (Osaka University, Osaka):

The roundness problem is stated as follows: Given $n$ points in the Euclidean plane, find the center of the concentric circles enclosing all given points between outer and inner circles and minimizing the difference between radii of the outer and inner circles. For this problem, we have never known optimal algorithms whose time complexity is bounded by a polynomial. We present a new optimal roundness algorithm, whose time complexity is bounded by a polynomial. This algorithm is based on the property that the optimal solution for the roundness problem exists only on the vertices in the union of the nearest-point Voronoi diagram and the farthest-point Voronoi diagram. We show that the time complexity of this algorithm is $O(n^2)$.

This list of abstracts gives you some idea about results obtained by computational geometers in Japan in the last year. In concluding my report, I would like to mention the international symposium to be held in Japan this summer. The SIGAL Inter-

national Symposium on Algorithms will be held at the CSK Information Education Center, Tokyo, Japan in August 16–18, 1990. The symposium is organized by SIGAL. Computational geometry is one of the main topics of interest of the symposium, and we would like to have many computational geometry papers again like in the recent international conferences on theoretical computer science in Japan. Authors are requested to send twelve copies of a detailed abstract of at most ten double-spaced pages by February 15, 1990, to Tetsuo Asano, Osaka Electro-Communication University, Hatsu-cho, Neyagawa, Osaka 572, Japan; e-mail: asano@iscb.osakac.ac.jp. Proceedings will possibly be published as Lecture Notes in Computer Science. I hope to see many of you this summer in Japan. Also, if you have any questions related to this report, please do not hesitate to contact me.



# Meetings in 1991

The following few pages feature a short and biased review of the two major computational geometry meetings of this year. In my opinion, these are the Symposium on Computational Geometry, this year held in Berkeley, California, and the Canadian Conference on Computational Geometry, organized in Ottawa, Ontario. I hope that calling these two the "major two meetings" does not offend organizers or attendees of many other computational geometry meetings, such as the annual Computational Geometry Workshop that alternates between Switzerland and Germany, the week long meeting in Dagstuhl, a new conference center for computer science in Germany, and others. I add that computational geometry is typically also well represented in major computer science conferences, such as STOC and FOCS in America, ICALP in Europe, and SIGAL in Asia.

Whatever the situation, this column will give a short review of the two meetings and offer a few personal observations and thoughts to guarantee a proper bias.

On a general level it should be mentioned that the two meetings differ widely in their characters. The Symposium on Computational Geometry is certainly the more established forum being organized for the 6th time already. It is also the more serious of the two meetings, with a tight three-day schedule and a tough selection modus that guarantees high quality papers. Would it be fair to call it the more elitist of the two meetings? The Canadian Conference on Computational Geometry, on the other hand, has been organized only twice so far, last year in Montreal and this year in Ottawa. I believe it is intentional that the refereeing process for this meeting is kept to a minimum. In my opinion the two meetings complement each other nicely and respond to different needs of the community.

Surely it is possible to read trends in the area from the selection of papers presented at the two meetings. I admit that it is an open question whether I can read them, but everybody who has some confidence, or maybe merely some interest in my opinion is invited to continue reading this column. I will talk about the two meeting, the one in Berkeley first (because it precedes the Ottawa meeting chronologically), and mention some results that caught my interest.

**The Berkeley meeting.** Frances Yao as the program chair and Raimund Seidel as the local organizer are two names that promised an exquisite symposium, and they delivered. The meeting took place on the Berkeley campus with the local dormitories,

vacant of students during the summer, serving as convenient residences. The highlight of the social program was an outdoor party (the banquet) featuring superb food and sports activities. Thanks to both for the good work.

The technical program was spread over three days, June 6 through 8, with an invited talk each. On Wednesday, June 6, Shiing-Shin Chern delivered an enjoyable talk on differential geometry, linking this area to computing by pointing out many problems where the use of computers promises to lead to new insights. On Thursday, James Blinn talked about the demands on computational geometry made by image rendering. Indeed, computer graphics is one of the main areas that provide motivation for the study of computational problems in geometry. The third invited talk was delivered on Friday by Dana Scott. He uses the computer for class-room teaching and shared his experience with the audience by presenting how he implemented projective geometry via symbolic computation.

Among the technical contributions I would like to mention Jiři Matoušek's paper on cutting hyperplane arrangements. In this paper he extends his two-dimensional cutting results to three and higher dimensions. These results are very important in computational geometry because they are fundamental to many algorithmic and also combinatorial geometry problems. Roughly, what Matoušek shows is that for every arrangement of $n$ hyperplanes in $d$ dimensions, $d$ a fixed constant, and for every $r \leq n$ there is a decomposition of the $d$-dimensional space into $O(r^d)$ simplices so that each simplex intersects roughly on the order of $\frac{n}{r}$ hyperplanes. The paper also presents efficient algorithms for constructing the decompositions.

Another highlight was Bernard Chazelle's outside-the-program presentation of his brand new linear time triangulation algorithm for simple polygons. Indeed, he obtained the result well after the deadline for submissions and his talk was appended to the end of the technical program because of the general interest created by the announcement of his result. Another triangulation result which the author of this column humbly believes is worthwhile mentioning is the $O(n^2 \log n)$ time algorithm for finding a minmax angle triangulation for a set of $n$ points in the plane by Herbert Edelsbrunner, Tiow-Seng Tan and Roman Waupotitsch. This problem has a long history in the finite element literature, and their paper gives the first polynomial time algorithm for the problem.

I also liked the paper by Gert Vegter and Chee Yap on the computational complexity of constructing the fundamental domain, a $2g$-gon, of a closed triangulated surface and embedding it on the surface. There are still only few papers that study the computational aspects of topology problems, but it seems clear that this area has great potential for the future. Maybe it is still not clear what the right questions to ask are, but it is important that some steps in this direction are done.

I end my contemplation of the Berkeley meeting with a few general remarks. Judging from the program, and the programs of the previous five meeting in this



series, the Symposium on Computational Geometry is very successful in attracting the best papers in the area. Its proceedings can therefore be strongly recommended to everyone interested in computational geometry in general or in certain application problems that fall into this area. What the conference needs though is a supplement of services for the technology transfer from computational geometry to areas whose problems it claims to solve.

**The Ottawa meeting.** Jorge Urrutia was in charge of the program and the local organization of this meeting which featured four invited talks and more than seventy contributed talks spread over five days, from August 6 through 10. The talks were scheduled in two parallel sessions to provide sufficient time for each talk and for the participants to exchange ideas and work together. The invited talks were of course exceptions to this rule and were delivered by Selim Akl, Franco Preparata, Godfried Toussaint and the author of this column. The location of the meeting was in walking distance from Ottawa downtown on campus of the University of Ottawa. I would like to thank Jorge Urrutia for making this meeting successful and enjoyable.

It is interesting to notice the difference in the choice of invited speakers between the two meetings. At the Symposium the emphasis is on branching out, creating ties to related areas, in mathematics as well as engineering. In contrast to this, the Canadian Conference invites experts in the main core of computational geometry, and throughout they choose to give survey talks on subareas of computational geometry.

Among the technical contributions I liked particularly a paper by Chi-Yuan Lo and William Steiger which showed that a ham-sandwich cut of two not necessarily separated sets of points in the plane can be found in time proportional to the number of points. This solves a problem that was open for a few years.

Vasilis Capoyleas, Günter Rote and Gerhard Wöginger showed that for $n$ points in the plane and a fixed integer constant $k$, an optimal partitioning of the set into $k$ subsets can be found in polynomial time. The notions of optimality covered by their result include the sum and the maximum of the diameters of the subsets being minimal. The degree of the polynomial depends on $k$.

Some nice combinatorial geometry results about the separability by lines of a set of $n$ pairwise disjoint convex objects in the plane were presented in a paper by Jurek Czyzowicz, Eduardo Rivera-Campo, Jorge Urrutia and Joseph Zaks. They show that in every set of at least $12k$ such objects there is one that can be separated by a straight line from at least $k$ of the other objects. Such a result is impossible, in general, for pairs of objects.

Let me finally rephrase a shelling problem that came up in a paper on constructing triangulations by Isabel Beichl and Francis Sullivan. Suppose you have a Delaunay triangulation of a finite point set in $d$ dimensions. Now compute a shelling of this triangulation as follows. Pick an arbitrary initial simplex and extend the set of visited

simplices by choosing an arbitrary unvisited simplex so that the union of the visited simplices (plus the one new simplex) is topologically the same as a ball. The question is whether this algorithm always succeeds. I believe it does succeed in the plane, but I do not know the answer to this question even in three dimensions. It is important to know that unlike an arbitrary triangulation the Delaunay triangulation always admits a shelling, but it is not clear whether a process that extends the shelling fairly arbitrarily always succeeds to construct one.

**Next years meeting.** I close this report by urging you to submit your best computational geometry result to the 7th Symposium on Computational Geometry in North Conway, New Hampshire. The location promises a beautiful environment that offers the possibility to combine business with pleasure by adding a few vacation days before or after the spiritual recreation offered at the meeting. Of course, it is less important to have a paper there than to attend and learn about all the new things that will have happened by then.

# 2

## Al
## H.

CONT



Hartmut
Institute f
Technical
Franklinst
1000 Berl