Contemporary Mathematics
Volume 223, 1999

The Early Years of Computational Geometry—a Personal Memoir

Michael Ian Shamos

ABSTRACT. A first-person account of the development of geometric algorithms at the Yale University Department of Computer Science and Carnegie Mellon University during 1972-1978 by one of the architects of computational geometry through his interactions with other researchers such as Dan Hoey, Stanley Eisenstat, H. R. Strong, Gideon Yuval and Jon Bentley.

Computational geometry (CG) as a discipline is now a quarter-century old and has given rise to several entire journals, scores of conferences and thousands of publications. Because many sources credit me with having organized the subject and formulated and solved some of its early problems [**S**], it seems appropriate for me to relate the story of how computational geometry was born.

Most practitioners of CG are not aware that all of my work was motivated by a single problem that I tried to solve while a graduate student at Yale in the early 1970s. The problem was difficult because not only did the necessary tools for solving it not exist, but the problem itself had never been examined from the point of view of computational complexity. However, I cannot explain how I came upon the problem, let alone why I became obsessed with building a new discipline of computer science, without offering some biographical material.

My upbringing in New York City was science-oriented. My father was Chairman of the Physics Department at New York University and is now emeritus. He was an experimental physicist with a facility for mathematics and I spent a lot of time tinkering in his cosmic ray laboratory on the roof of NYU's Main Building. A Geiger counter was always on to warn of any spill of radioactive material, which contributed to the excitement. The place was loaded with test equipment for making sensitive electrical measurements and constant pain with trying to make it work turned me eventually to a career in theory.

My mathematical education was in the hands of my grandfather who, as a civil engineer and surveyor, laid out rights of way for the New York Central Railroad. His training was classical and in weekly geometry sessions he taught me the algorithmic

---

1991 *Mathematics Subject Classification.* Primary 68U05; Secondary 68-03, 01-08.

Key words and phrases. Computational geometry, analysis of algorithms, history, closest-points problem, Voronoi diagram, geometric intersection.

The author thanks an anonymous referee for pointing out embarrassing errors and making important suggestions to improve the presentation.

methods of Euclid. I was fascinated then, as I am now, at the power of ruler and compass constructions and was struck by the close connection in Greek geometry between an algorithm and its corresponding proof of correctness. Sundays with him were a world of reverse curves, drafting instruments and seven-place log tables. My grandfather showed me the workings of his planimeter, which I still possess. This device, now remembered only in museums, could figure the area enclosed by a simple curve just by tracing a stylus over the curve's boundary. I was amazed that a mechanical device could do this, and I still remember the machine's dials turning as the stylus moved over a piece of paper. At the time, I couldn't imagine how this was enough to calculate the area. This was perhaps my first tangible experience with geometry and computation. It was with great satisfaction 20 years later that I found an optimal algorithm to compute the area of a plane polygon,[1] using the computer as a "discrete planimeter."

I entered the Horace Mann-Barnard School (then the Horace Mann School for Boys) in 1958 and was lucky to have Bob Moses as my first formal math teacher. He left the school a few years later to become a civil-rights worker and now runs the Algebra Project in the Boston area, a grass-roots effort to teach inner-city children the wonders of mathematics. He was and is a true educator. During my freshman year, he gathered up a few students, tossed away the textbook, and had us experimenting with matrix algebra while our less fortunate colleagues were struggling with quadratic equations. This early exposure to matrices was important background for me in solving the polygon area problem, which for a century had been formulated as computing the determinant of a matrix of special form.

While at Horace Mann, I was selected to participate in the Columbia University Science Honors Program, which provided high-school students from all over New York City with advanced science courses taught by Columbia faculty members. It was there that I was introduced to computers by two researchers from NASA, Danny Gardner and Paul Schneck. We were allowed access to the IBM 650 at the T. J. Watson Computation Laboratory. This machine had no disk memory and its only input and output were in the form of punched cards, yet it could compute much faster than mechanical calculators. We soon graduated to the IBM 7094, which at least was transistorized and my friend Allan Cooper, who now works for Microsoft, took the bold step one day of walking into the IBM Data Center at 59th Street and Madison Avenue and asking whether IBM would let the two of us use the machine for free. In 1962, IBM was charging $900 an hour for a 7094. The manager was so astounded at the request that he granted it, and for the next year I spent my off-hours there on a variety of programming projects.

NYU also had a 7094 that was being run by one of my father's faculty acquaintances, Jack Schwartz. His name is now familiar to you as the author of many papers in CG, but you may not know that he was a computer science pioneer 35 years ago. Jack was always bubbling with ideas. In 1963, many years before parallel computing was taken seriously, he circulated a memo describing an interconnection of 1024 mainframes to allow them to cooperate in the solution of numerical problems. He gave Allan and me summer jobs at NYU's Courant Institute. Jack had noticed that most programs written in assembly language ran for a few milliseconds before causing the machine to crash. It was nearly impossible to determine

[1] This is easy to do in linear time. The problem is to reduce the number of multiplications required to a minimum.

where the program had gone wrong because of a lack of debugging tools. (The concept of a debugging tool did not even exist then.) He asked us to write a system program, which we named SENTRY, that would monitor assembly language programs to trap illegal operations and address references. This was essentially the first software debugger, the concept of which you may credit to Jack Schwartz.

All of this computing activity managed to get me admitted to Princeton. I went there to study physics, but was sidetracked somewhat by mathematics. The Physics Department did not accept majors until junior year, while Math allowed sophomore concentration, so I majored in Physics and minored in Mathematics. This created something of a course overload, which I solved by managing never to take any courses in the English Department. My mother to this day wonders how I got through Princeton's liberal arts curriculum without taking a literature course.

The quadrangle housing physics and math was alive with geniuses whom one would pass without fanfare on the sidewalk. Alonzo Church, who was technically in the Philosophy Department, was there. (Back then, when professors spoke reverently of Church's Thesis, I thought it was the document he wrote to get a Ph.D.!) C. C. Gillespie, the historian of science, Solomon Lefschetz, a founder of topology, A. S. Wightman, the mathematical physicist, Marvin Goldberger and Sam Treiman of the Goldberger-Treiman relations (the phrase had nothing to do with their personal lives) could all be found within a few yards of one another. If knowledge could be acquired through osmosis, this was the place to be.

As a sophomore, I enrolled in Albert W. Tucker's Topology Seminar. He was a pioneer in the field, and made major contributions to game theory and operations research as well. He is remembered fondly by everyone except me. For the seminar I wrote a paper in which I tried to define a metric on plane closed curves that would correspond to an intuitive notion of closeness of fit. The work was, alas, more intuitive than rigorous, a defect to which I also readily admit in my later research. Tucker would have none of it. I was summoned to his office by the University police. While this sounds frightening, it was the standard method of delivering important messages to students on campus. It never occurred to me that there might be anything amiss and I even dreamed that he might want to collaborate on extending the results. This fantasy was quickly dispelled. He asked me to explain my paper to him, and questioned me sternly on every statement in it. After an hour, he sat back, apparently pacified, and said, "The reason I called you in here was to determine if you were pulling my leg or whether you are just stupid. I am relieved to find it was the latter, so you will receive a passing grade. If you had been trying to fool me, you would have failed the course." While he was caustic, Tucker actually did me a favor by demonstrating conclusively that I had no future in topology, a subject of which I had no inherent grasp.

My seminal undergraduate experience was working as a research assistant to John Wheeler, who was best known as Feynman's Ph.D. advisor and for inventing the term "black hole." Long hours in laboratory courses had turned my taste decidedly toward the theoretical. But how I could I get Wheeler to accept me as a student? His field was general relativity, which was not even taught until graduate school. For this reason, although he had a very active research group that met weekly, he had only one other undergraduate assistant.

I prepared in secret by spending the first semester of junior year writing a review of variational methods in general relativity and made an appointment to show it to Wheeler. He leafed through the paper, but it was impossible to tell

whether it had made an impression. As a review, it contained no new results. He then said, "Let's take a walk." I later learned that this meant he was about to pose a problem—Wheeler did most of his work with colleagues while on foot. He continued, "Back in 1945, Feynman and I proved that electrodynamics can be derived without the use of fields. That is, everything can be explained in terms of bodies acting on one another at a distance. I have always wondered whether one could develop a similar theory of gravitation. You can find our paper in the *Reviews of Modern Physics* that year. Read it through and then come see me so I can help you get started."[**WF**] I was dumbfounded. Here Wheeler had been wondering about a question for 20 years, and he expected me to answer it.

That was the critical moment in my research life. If Wheeler thought I could do it, he must be right. It was as if he had held on to the problem for two decades waiting for someone at the right level to work on it. It wasn't difficult enough for a Ph.D. student, and didn't merit a significant amount of Wheeler's time, but it was a good challenge for an advanced undergraduate. It was invigorating to think that a famous scientist felt I could create new theory when I had never done so before.

Since I didn't have any original ideas about action-at-a-distance gravitation at the time, I made a huge effort studying the work of others. Unfortunately (or fortunately), there were no published papers that even mentioned the concept. However, I developed the habit of reading scientific journals obsessively, an addiction fueled by the fact that the Math-Physics library was open 24 hours a day. I even skimmed over papers I couldn't begin to understand, just to absorb their notation and remember their results.

This behavior came in handy one day in the Spring of 1968. At the regular meeting of Wheeler's research group, someone posed the question whether the existence of matter necessarily implied the existence of electromagnetic force. It was of course known that matter implies gravitation—that is what general relativity is all about. But could one have a universe entirely of neutrons, for example, devoid of electromagnetism? I immediately answered that electromagnetism must exist in any universe, since Einstein had proven it. The group eyed me suspiciously. They were on intimate terms with Einstein's work; some of them, like Wheeler, had been on intimate terms with Einstein himself. Nobody recalled such a result. I said, "Oh, yes, it appeared in a paper of his in the *Revistas Mexicanas de Física* in the early 50s. I'll be happy to run over to the library and bring back a copy." Wheeler said they would all be obliged if I would do so. I returned with it about 20 minutes later and the paper was passed around the table. It indeed contained the assertion that electromagnetism must exist. The group was able to spot an error in the paper rather quickly, and they assured me the problem was still unsolved despite Einstein's claim. From this episode I acquired a reputation for having an encyclopedic familiarity with the published literature. Ritualistic reading of publications would stand me in good stead five years later as I tried to assemble hundreds of geometric problems for computational attack.

Wheeler's intuition about action-at-a-distance was correct. There is a formulation of gravity theory that does not require fields and is perfectly consistent with "classical" general relativity. This work formed the subject of my undergraduate thesis [**Sh**]. The problem was rather abstruse, as the very existence of gravitational radiation, the subject of the thesis, had not been demonstrated at the time and it is still not known whether the phenomenon exists. Later, with the help of Morton Tavel at the Vassar College Physics Department, I extended the work to

acoustics [**ST**]. Tavel was fascinated because the acoustic result is completely counterintuitive. Neither electrodynamics nor gravitation requires a medium other than vacuous space in which waves can travel. Acoustics requires a medium; otherwise pressure waves cannot exist. It therefore seems impossible to have action-at-a-distance in acoustics. Yet it *is* possible; however, the velocity of the waves is zero unless there is a medium present, so the result is consistent with classical acoustics.

My hobby for a time at Princeton was computing, which then was under the firm control of the engineering school, a foreign domain to us physicists. Because of my earlier work as a system programmer for Jack Schwartz and a somewhat irreverent attitude toward rules and regulations, I engaged in some activity that would now be considered "hacking," and was expelled for the rest of my undergraduate days from the university computing center for a prank that was innocently intended but had the effect of wiping out a month's worth of computer accounting records. The cost to the university to reconstruct the lost data was approximately equal to my year's tuition. The incident did not diminish my enthusiasm for computers, but extinguished any opportunity I might have had to act on it.

Upon graduation in 1968, I went to work immediately for IBM in East Fishkill, New York at their semiconductor manufacturing plant. This was IBM's largest facility in the world, with over 20,000 employees, dwarfing the more famous "Main Plant" in Poughkeepsie where mid-range mainframes were assembled. Fishkill turned sand into logic. Literally. They grew silicon crystals for computer modules right there in the plant. My job was to write quality control software to analyze electronic test results performed in real-time on semiconductor circuits. Familiarity with test equipment gained in my father's laboratory was useful here; none of the other programmers had any idea what an ohm was. The assignment may not sound fascinating, but it was. Our resources were essentially unlimited, and the problem was practical and vast. The factory turned out the logic devices used in a majority of IBM computers (then a majority of the world's computers) and costs, yields and quality were essential concerns. I labored happily at the task until Vietnam intervened and I was obliged in 1970 to go into government service.

What for many turned into tragedy was fortuitous for me and probably led directly to the development of CG. I attempted to enlist in the Army, but for a variety of reasons, including poor eyesight, I wound up as a commissioned officer in the U.S. Public Health Service at the National Institutes of Health (NIH) in Bethesda, Maryland. I was assigned to the National Cancer Institute (NCI), to the division responsible for developing and testing chemotherapy drugs.

The trick in chemotherapy is to kill cancer cells without destroying too many normal cells. This is a sensitive matter of toxicity and involves getting the right dose of the right compounds delivered to the right place in the body for the right amount of time. NCI spent huge sums (in those days their budget was about $1 billion a year) studying chemical structures, trying to determine which portions of a compound acted on cancer cells and which portions that caused healthy cells to die could be eliminated. Richard Feldmann of the NIH Division of Computer Research and Technology (DCRT) had developed an interactive system to display and search chemical substructures. This was revolutionary work. Indexing of chemical compounds had previously been primitive, and there was no effective way to look up, for example, all known compounds that contained a particular configuration of atoms or radicals. In Feldmann's system, all you had to do was draw a structure on

the computer screen with a light pen and you were presented with a list of chemicals that incorporated that structure.

I was assigned as the liaison between Feldmann and NCI. One of the problems he had not yet solved was how to draw chemical structures on the computer screen in a presentable way. They were technically accurate in the sense that all of the correct bonds were shown, but they didn't look nice. I was seduced by the problem. If you open any book on organic chemistry, you will find it filled with beautiful drawings of chemical structures. These are prepared by artists. The issue is how to get a computer to draw them that way. Feldmann explained to me that chemical structures are graphs with labeled nodes, so if I wanted to figure out how to do it I had better read up on graph theory.

I had no acquaintance with that subject, although I did know what it was. There were very few texts available, but the most accessible appeared to be Harary [**H**]. I obtained a copy and spent nearly a year working through most of the exercises. The book contains this intriguing statement on p. 106: "One of the most fascinating areas of study in the theory of planar graphs is the interplay between considering a graph as a combinatorial object and as a geometric figure." That was exactly what we were studying at NIH. Harary went on to mention the result of Istvan Fáry that every planar graph can be embedded in the plane so that all of its edges are straight line segments [**F**]. It started me wondering how one would go about actually finding such an embedding.

I attended courses at DCRT that made me realize that I needed to go to graduate school in computer science. My fiancée, Julie Van Allen, learned that Yale had just started a department and was looking for students. I applied and was accepted under the unusual condition that I would have no fellowship and had to pay graduate tuition. I was, however, eligible for Veteran's benefits that would pay living expenses. For my first two years at Yale, I labored under the detriment of being the only graduate student in the Computer Science Department who actually paid tuition to be there.

The department at Yale was small, filled with bright faculty members and extremely research oriented. Where else could one work with Alan Perlis, the creator of Algol, and Ned Irons, inventor of syntax-directed compiling, Martin Schultz, the numerical analyst who wrote the fundamental text on spline theory, and attend lectures by Sam Winograd, a pioneer in arithmetic complexity? Yale's approach to starting new departments is that they are expected to be the equal of any in the University and certainly the equal of any outside the University!

Yale's attitude toward Ph.D. qualifying exams reflected this seriousness of purpose. We were expected to complete them all during the first year and then devote every moment thereafter to research. The Computer Science Ph.D. qualifiers were exams like no other. We were tested in three subjects: Numerical Analysis, Hardware and Programming, and Computational Complexity. Each exam was take-home and lasted six days, to be turned in at noon. Upon handing in one exam, we received the next, and so forth—18 straight days of exams each semester, for a total of 36 days of qualifiers in a nine-month year! And don't imagine that the fact that the exams were take-home was any help; all of the problems given were unsolved and the answers could not be found in any book or paper. Eating and sleeping were our own affair; if we chose to engage in these activities it was at the expense of the exams. In 1975, when I came to Carnegie Mellon, I learned that their qualifying exam lasted 24 hours, which they thought was quite a challenge.

After Yale, I still laugh at the concept. I passed all the qualifiers with distinction during the first year, which enhanced my reputation in the Department, and I was free to do research full-time.

During that first year, three events took place that altered my view of computer science. The first was the publication of Tarjan's linear-time algorithm for determining graph planarity [**T**]. It is difficult to appreciate now how revolutionary this result was back then. All previous work on planarity testing had been based on the mathematically elegant but computationally clumsy characterization due to Kuratowski that a graph is planar if it does not contain a subgraph homeomorphic to the complete graph K_5 or the complete bipartite graph $K_{3,3}$. Direct application of the theorem leads (with some difficulty) to an $O(n^5)$ algorithm. To do it in linear time was mind-boggling, but it wasn't just the speed of Tarjan's method that caught my attention. It was that he did not base his algorithm on any of the classical theorems characterizing planar graphs. His approach was completely constructivist—he just started to lay out the graph so that if it were planar he would produce a plane embedding and if the graph were not planar the algorithm would report failure. This taught me that developing fast algorithms sometimes requires going beyond traditional mathematics.

The second important event was the appearance of Richard Karp's paper, "Reducibility Among Combinatorial Problems"[**K**], in which he showed that a large collection of important and seemingly unrelated problems could be solved in polynomial time if any one of them could. His method of mutual reducibility seemed a powerful and efficient way (from the point of view of the theorist's time, not computational effort) to use algorithms in multiple contexts. My plan for CG was modeled directly after Karp's structure.

The third inspiration was the publication of Volume III of Knuth, "Sorting and Searching"[**Kn**]. Here was a huge collection of potential thesis topics! In those days, it was commonly accepted that if you could solve a level-50 problem in Knuth, you would get a Ph.D. What amazed me about these unsolved problems was how simple some of them appeared to be. I do not, of course, mean simple to solve, but simple to state and apparently quite fundamental. One of them was the "post-office" problem: Given n points in the plane, how quickly can it be determined which of them is closest to a new given point p? I couldn't believe this hadn't been solved, since it seemed so important for geographic information systems, but it hadn't. An intimately related problem is the closest-points problem: given n points in the plane, which two are closest together? It was a fixation with solving this simple problem that ultimately led to the algorithmic development of the Voronoi diagram.

I joined the graph theory seminar, which met weekly to discuss current problems and papers. I was assigned to present Tarjan's algorithm to the group and suggest a research problem related to it. When I thought of planarity, I immediately recalled the straight-line result of Fáry, and decided that it might be time to develop a fast algorithm to produce straight-line embeddings of planar graphs. The seminar advisor, Stan Eisenstat, thought this was a worthy problem, and I began working on it. Tarjan's algorithm did not seem to be useful immediately, so I went back to Fáry's original paper, which gave a constructive method, and tried to implement it.

Fáry's construction requires repeatedly finding a point interior to the kernel of a star-shaped polygon. I had never heard of a kernel before, nor even of a star-shaped polygon, so I set out to find more about them in the literature. The most

lucid explanation was in Yaglom and Boltyanskii [**YB**]. A star-shaped polygon is one having an interior point p with the property that the straight line segment joining p with each boundary point lies wholly within the polygon. The kernel of the polygon is the union of all such points p. A star-shaped polygon thus is one having a non-empty kernel.

Essentially the only classical result on star-shaped polygons at the time was Krasnoselsky's Theorem, which states that a polygon P is star-shaped if and only if, for every four vertices of P there is a point such that the segment joining the point to each of the four vertices lies within P. This is a combinatorial characterization, and although it gives an algorithmic test for star-shapedness, it provides no assistance in constructing the kernel. The material in Yaglom and Boltyanskii made it clear that the kernel is just the intersection of the interior half-planes of P. (Each edge e of a simple polygon P determines two half-planes. The interior half-plane is the one that lies to the interior side of e.) I therefore had a simple algorithmic prescription for the kernel. An n-gon has n edges, so finding the kernel of an n-gon is no more difficult than forming the intersection of n half-planes.

In 1973 I thought it would be a simple matter to go to the literature to find an algorithm for intersecting half-planes and the problem would be solved. So I started looking. To say that I searched high and low would be an understatement. Beginning with Collected Algorithms from CACM, I then went through every computer journal in Yale's substantial collection. There was nothing. I couldn't even find a good algorithm for determining whether a two line segments intersect, a seemingly trivial calculation that presents some unexpected pitfalls. At this point the choice was to abandon the problem or develop my own tools for solving it. Giving up was not a genuine option. Instead, I developed the obvious algorithm, which was to start with one half-plane, and intersect it repeatedly with each other half-plane in turn. This required observing that the intersection of k half-planes is a convex (but possibly unbounded) convex figure having at most k edges. The intersection of one of these objects with a half-plane can easily be formed in $O(k)$ time, which results in an $O(n^2)$ algorithm for the kernel of an n-gon. Applying the result to Fáry's construction gave an $O(n^3)$ algorithm for finding a straight-line embedding of a planar graph. Viewed today, this is an extremely crude algorithm, but it was the first time anyone had performed any algorithmic analysis of the problem. I had no insight then into how the procedure might be improved.

In the Spring of 1973, I presented this result to the Yale Computer Science Department (CSD) at a seminar in which each graduate student was required to report his research progress. Alan Perlis, the late spiritual leader of the Department, attended these seminars faithfully. Perlis exuded genius. He had founded and built the CMU Computer Science Department and was persuaded by Yale to assist in the establishment of a department there. His statements on even the most mundane topics were fascinating, and he peppered his lectures with pithy statements that his students still remember today. (One of them is "ALGOL is a blight."—quite a statement coming from one of its creators.) It was speculated among the graduate students that Perlis was the only Yale faculty member who would be able to pass all the Ph.D. qualifying exams if he had to.[2]

[2] I learned later that this sort of postulating is common among graduate students everywhere. When I arrived at CMU, I was immediately told that Alan Newell was the only faculty member who could pass the qualifying exams there.

Perlis, who had a particular interest in geometry problems, listened attentively to my talk. Toward the end, I made the mistake of saying that I thought the $O(n^2)$ algorithm for intersection of half-planes might be optimal. Perlis instantly made a face at me, and it was clear he was skeptical. He was certainly justified, and not only because faster algorithms have subsequently been found. At the time, there was no computational problem having at most n inputs and n outputs for which a polynomial-time algorithm was known and for which anyone had proven a lower bound worse than $\Omega(n \log n)$. After the talk was over, I went to Stan Eisenstat and confessed that Perlis's reaction had shaken me. Eisenstat had an advance proof copy of Aho, Hopcroft and Ullman's *Analysis of Computer Algorithms*, which contained extensive material on divide-and-conquer. Why not try it?—Eisenstat suggested—nothing else had worked. Divide the n half-planes into two sets, each having $n/2$ half-planes, form their intersection separately, and see where that goes, he urged.

The next day I went back to Eisenstat and said that the critical merge step was intersecting two convex $n/2$-gons P and Q, but there appeared to be no published literature on that problem. He thought about it, pronounced it trivial to solve in linear time, and sketched an algorithm on the blackboard. Either P and Q have intersecting boundaries, they are disjoint, or one contains the other. In the first case, Eisenstat gave a procedure that started from an intersection point of the boundaries of P and Q and stitched together the boundary of their intersection, using two pointers to keep track of progress along P and Q. I remember it was very elegant, but neither of us was ever able to prove it correct, and it ignored the question of how to find that crucial initial intersection point. However, Eisenstat convinced me that a linear algorithm was possible, and then it was fairly simple to find one.

In the spring of 1973 I was fortuitously introduced to Dan Hoey, an encounter that certainly influenced the course of the field. Hoey was a Yale undergraduate in Computer Science with a profound understanding of things computational. He was always in the machine room nursing the PDP-10 and keeping the Department's Mergenthaler typesetting machine in working order. He was brilliant and friendly but somewhat abstruse. He obviously had deep ideas, but was a bit shy about explaining them orally and even less willing to write anything down. His taste ran to mathematical puzzles and games having a mathematical component, like backgammon. (Along these lines, Hoey later produced extensive analyses of the problem of Rubik's cube. Together we computed tables of bearing-off strategies in backgammon.) He was interested in geometry problems, and saw funny diagrams of mine coming out of the Versatec plotter. We struck up a friendship and spent the next two years challenging each other with problems, inventing algorithms, and writing a host of computer programs, only some of which related to geometry.

One day Shimon Even, a guest lecturer from Haifa, presented a seminar on algorithms for finding minimum spanning trees in graphs. He went over the algorithms of Kruskal and Prim, and even discussed the problem of the maximum spanning tree, which provoked some debate about why anyone would ever want to find one. By that time, I was viewing the entire world through geometric glasses. In my scheme, every problem involving numbers was a geometry problem. This is correct in a rather trivial way. Every problem involving scalars or n-vectors can be considered as a problem involving points in n dimensions. The only issue is whether it is useful in a particular case to take this view. After Even's lecture, I wanted

to look at the spanning tree problem in the Euclidean plane, that is, where geometric points are taken to be vertices of the complete graph and the edge weights correspond to pairwise Euclidean distance.

The traditional approach forces one to consider all $n(n-1)/2$ edges of the graph and leads to an $O(n^2 \log n)$ algorithm. I felt that the Euclidean problem was so highly constrained, the edge weights not being independent but determined only by the $2n$ coordinates of the points (even fewer degrees of freedom if allowance is made for rotation, translation and change of scale). The question was how to use these constraints to give a faster algorithm. I introduced the problem to Hoey and said that there was a lot more about MST's we needed to know. For a mathematician this would have been a summons to engage in some deep thought. For computer scientists, it was an invitation to start programming. I suggested coding up the Kruskal and Prim algorithms to see how they worked for the Euclidean case. We did this on the PDP-10, using a Tektronix display for output, so we could see the MST's actually being constructed an edge at a time.

The results were productive. We quickly saw that the maximum degree of any vertex in a Euclidean MST was six. This fact is easily proven once it is known, but it had not previously been published. (We later learned that the result, along with many others, appeared in an internal Bell Laboratories technical memorandum by Ron Graham.) Although we discovered many interesting MST properties, we made no progress in developing a fast algorithm. With the appropriate data structure, Prim's algorithm leads to a quadratic algorithm. Because Kruskal proved that the MST always contains a shortest edge of its associated graph, solving the MST problem is at least as hard as solving the closest-points problem, a useful fact later on.

In the summer of 1973, Julie and I got married and rented a house in the woods of Hamden, CT, a northern suburb of New Haven, and acquired a large Belgian sheepdog named Nassau. The property was next to a state land preserve and was extremely quiet. During the year that we lived there, I do not recall hearing a sound. This environment was conducive to contemplative research. After Julie went to sleep at night, the dog and I would stay up working until about 4:00 a.m. He was very patient with me, and listened with head cocked as I explained the difficulties presented by some new problem. Our honeymoon year (we're now in our 24th) was blissful, and her encouragement was crucial in keeping me going when it would have been so easy to give up geometry for more traditional problems.

It was around this time that I started calling my ragtag collection of results "Computational Geometry," unaware that the term had been used at least twice before for different but related disciplines. In their 1969 book *Perceptrons*, Marvin Minsky and Seymour Papert used the phrase to refer to analysis of raster images, and Robin Forrest in England used it in the context of surface modelling and *CAD/CAM*. All of these disciplines can properly be called computational geometry because they all involve solving geometric problems by computer. In this memoir, I use CG to mean the subclass of discrete algorithmic problems in [**PS**] and the journal *Discrete & Computational Geometry*.

In 1973, David Dobkin and Richard Lipton joined the faculty. They worked on a host of problem in computational complexity, including some geometric ones. Their chief contribution to my work at the time was a short preprint entitled "On Some Generalizations of Binary Search," in which they showed how to determine in which region of the plane formed by n intersecting lines a particular point p was

located [**DL**]. They gave an $O(\log n)$ algorithm, if polynomial-time preprocessing of the lines is allowed. The idea is central to geometric searching, and it spurred CG considerably. Lipton and Dobkin have by now had long distinguished careers in theoretical computer science, and are responsible for many geometric results.

Dobkin was the head of my thesis committee, and my scientific relationship with him has been the subject of some speculation. Our personal interaction was often stormy, but we never had any disagreement over the significance of geometry problems or whether the work deserved a doctorate. I felt I was up against a Catch-22 situation—no matter how many new results I came up with, the committee wanted additional ones. Such is the risk of defining a new field for one's Ph.D. thesis—it's difficult to tell when you're done. From my vantage point now, the fact that I was prodded to produce more was beneficial. It was important for the thesis to have enough material and rigorous proofs to convince those outside Yale that computational geometry was a worthy subject.

I devoted the fall semester of 1973 to tidying up work on convex polygons and convex hulls. The planar convex hull problem was known by folklore in the graduate computer science departments until Ron Graham published an elegant paper on the problem in 1972 [**G**]. His was the first complexity analysis of a problem in computational geometry, and he gave an $O(n \log n)$ algorithm. It became apparent early that finding the convex hull of n points in the plane is equivalent to sorting, and I showed that every sorting algorithm has a natural interpretation as a convex hull algorithm. Because of this equivalence, it is easy to prove that hull-finding requires $\Omega(n \log n)$ in the worst case. However, it is not immediately clear that simply identifying the hull points, as opposed to listing them in sequential order, is as difficult, but Andy Yao later showed that to be the case also [**Y**]. With respect to convex polygons, I showed how to find the diameter and width (the shortest possible distance between two parallel lines enclosing the polygon) in linear time, which through the use of the convex hull leads to $O(n \log n)$ algorithms for the diameter and width of point sets in the plane. These results were completely new. In hindsight they seem rather elementary, but the question of how difficult it is computationally to find the diameter of a set of points had never even been asked, let alone answered.

I formulated a large number of problems that seemed interesting and whose solutions could be used to attack other problems. Given two polygons P and Q, are they congruent? Can P be made to fit inside Q? What is the smallest triangle (in a area) containing P? What is the largest triangle (in perimeter) that lies entirely within Q? How many unit circles are required to cover P? What is the smallest number of disjoint convex polygons whose union is Q? Is P simple? Some of these I had no idea how to solve, but I thought they should be looked at from the viewpoint of algorithmic complexity.

I also decided to do a complete review of the literature. I manually scanned every issue of the *Communications of the ACM*, *Journal of the ACM*, *SIAM Review*, *Journal of Combinatorial Theory*, *Discrete Mathematics*, *American Mathematical Monthly*, and dozens of textbooks, looking for geometry problems to solve by computer.

While I was at Yale, the University awarded John Wheeler, my undergraduate advisor, an honorary degree. He appeared at the same ceremony at which I was to receive a master's degree in computer science, so we had a chance to reminisce. Wheeler had always taken a geometric approach to physics, especially cosmology,

and was intrigued by CG. We briefly discussed the possibility of using combinatorial geometry as the basis of quantization of space, but nothing has yet come of the idea.

By June 1974, I had collected or posed about 75 problems, which I described in a typewritten manuscript entitled, "Problems in Computational Geometry" [**Sh4**]. This was meant primarily as a workbook for me, and it set forth a short statement of each problem, gave a brief description of the obvious, or naive, algorithm for the problem, followed by a discussion of the current state of the art. A secondary objective of this collection was to convince the CSD that computational geometry was a field, and not just a disconnected set of problems. What I did not count on was the power of the Xerox machine. I sent out about five copies of the manuscript to colleagues at other universities and research institutions. I estimate that several hundred more were made thereafter by the recipients and those to whom they passed copies. This was a boon to the field, for it caused many people to begin thinking about and working on these problems. Science Citation Index reveals that the workbook has been cited scores of times in the literature even though it was never formally published.

Yale saw that Hoey and I were getting along well and arranged for us to have office space together on the top floor of Dunham Laboratory on Hillhouse Avenue, a stone's throw from the building in which the Department is now located. This was providential. One day in the spring of 1974, I walked into the office to see Hoey drawing some strange diagrams with colored markers on acetate. As usual, he was excited and intense and had a cigarette dangling from his mouth. When I asked what he was doing he said, "Given n points in the plane, each point is surrounded by a convex polygon, not necessarily closed, such that if you're in a polygon than the closest point of the set to you is the one that owns that polygon. I call them proximal polygons." Hoey had independently discovered Voronoi diagrams!

He had marker ink all over his hands and showed me his drawings. The pictures themselves were messy, but their underlying structure was beautiful. They looked somewhat familiar to me, and I told Hoey that they looked like objects from solid-state physics that we called Wigner-Seitz cells. This set me off on a literature search to see what had been written about these structures. Hoey and I spent the next several months investigating their properties. I eventually found that they had been discovered by G. Voronoi and described by him in a 1908 paper [**V**]. They have been rediscovered independently many times since. Hoey and I needed some name for the overall structure. He liked the term "proximal diagram," but I knew that whatever name we adopted would soon be in common use and I thought the name of their inventor should be included, so I lobbied for "Voronoi diagram," and the term has stuck. It had never been used before. Despite my reverence for his work, I have never seen Voronoi's paper, since the particular volume that contained it was missing from Yale's Mathematics Library. Likewise "Delaunay triangulation" was a phrase of my own creation.

The Voronoi diagram (VD) looks complicated, but Hoey realized that it only contains a linear number of pieces. Thus in principle a lot of fast algorithms could be obtained if the VD could be found quickly. For example, the two closest points of a set can be found in linear time once the VD is known.

We had tried everything to cut down the time to find closest points in the plane. It was maddening that the two farthest points (the diameter) could be found in $O(n \log n)$ time, but we could do no better that quadratic for closest points. Why was "close" difficult when "far" was easy? I spent a lot of time worrying about that

one. The answer, as I suspected, is that there is no real difference between near and far in a very general sense, as was later shown by Kevin Brown in his CMU Ph.D. thesis [**B**]. Hoey and I tried cutting sets of points in half, in quarters, into a $\sqrt{n}$ by $\sqrt{n}$ array of rectangles, and other bizarre divisions. In the worst case, none yielded anything faster than quadratic. We would have been satisfied with $O(n^{1.9})$! At least that would have provided some hope, instead of the brick wall that we faced.

Likewise, Hoey and I were unable initially to produce any fast algorithms for the Voronoi diagram. Using the algorithm for intersecting half-planes, it is easy but tedious to develop an $O(n^2 \log n)$ algorithm. We were able to shave this down to $O(n^2)$ through careful housekeeping, but then progress ground to a dead halt. The difficulty was largely psychological. Since we weren't sure that any subquadratic algorithm for closest points was possible, there wasn't much point in looking for a fast VD procedure. Nevertheless, I was so taken with the potential of the Voronoi diagram that I continued to study its properties, and found that it could be used to find all closest points, smallest perimeter triangle, convex hull, minimum spanning tree and a triangulation of a point set. I also showed that it could also be used to give the first $O(\log n)$ solution to Knuth's post-office problem.

Since even Yale embraced the idea that solving a level-50 problem in Knuth was worth a Ph.D., I knew I was done. In January 1975, I presented a thesis proposal to the Yale CSD called, simply, "Computational Geometry." By that time, I had obtained so many results that it was apparent that the work was sufficient for a thesis, and the proposal was well received. I felt confident that I would be able to finish the thesis by May and made plans to return to IBM, from which I had been on leave for five years. I had no desire to return to East Fishkill—it's a lovely town, but not a center for algorithm research. I wanted to work at IBM's Research Center at Yorktown Heights, NY, so I requested the opportunity to give a research talk as part of my job interview there.

The talk took place on January 28, 1975, with many acquaintances from the IBM Computer Science and Mathematics Departments in attendance. One of them was Ray Strong, whom I knew from a Yale seminar on Petri nets. During the talk, I described efforts to solve the closest points problem and said that I felt irked that we had not be able to do better than $O(n^2)$, despite having "tried everything." After the talk, Strong invited me back to his office to discuss an idea he came up with during my lecture. When we got there, he drew a set of points on the board and outlined a procedure. First find the median x-coordinate of the n points in linear time by the method of Blum et al. The vertical line M having this coordinate divides the set in half. Now find the closest pair of points in the left set and in the right set and let the smaller of the two distances be δ. Now draw lines at distance δ parallel to and on either side of the line M. Strong said it was sufficient to consider only points that lie in the two vertical slabs formed by these lines.

Well, Hoey and I had seen these slabs many times as we had tried the same construction in applying divide-and-conquer to the problem. So I immediately said to Strong, "But in the worst case all n points will lie in the slabs, and you get no improvement." He said, "That's true, but the points in the slabs now have a special structure. They're sparse. Within a slab, no two points are closer together than δ." My response was, "So what? How can you use that fact to do anything?" There followed one of those moments that scientists live for—a flash of insight that produces eternal clarity. Strong said, "That means that for every point in the left

slab you only have to look at points in the right slab that are within distance δ, and there can be at most a *constant* number of them. For every point in the left you don't have to look at every point in the right, so the merge step won't be quadratic." This was the whole key to the closest-point problem.

There was a little more work to do; we had to find a procedure for locating that constant number of points (at most six, it turned out) to be examined for each point in each slab. This we did by sorting points by y-coordinate at each recursive step, which led to an $O(n \log^2 n)$ algorithm. I wasn't happy about the extra factor of $\log n$, but I drove back to New Haven ecstatic in the knowledge that the closest-point problem was subquadratic. That suggested that a fast algorithm for the Voronoi diagram might be possible. I showed Strong's result to Hoey and we huddled to see whether we could squeeze out that extra factor of $\log n$. It turns out that if the set of points is sorted initially by y-coordinate, no further sorting steps are required, and the merge step of the divide-and-conquer can be performed in linear time, which gives an $O(n \log n)$ algorithm.

After this, I made a renewed assault on constructing the Voronoi diagram. With Eisenstat's encouragement, I took another look at divide-and-conquer. Suppose we divide a set of n points into two sets, L and R, splitting at the median x-coordinate. Now form the Voronoi diagrams of L and R separately. I had gotten this far many months earlier; the big stumbling block lay in the merge step: how do you join the two diagrams together quickly? By playing with colored pens on transparency film, I could see that a polygonal boundary P always exists between L and R such that all of the Voronoi diagram of L lying to the right of P can be discarded and all of the Voronoi diagram of R lying to the left of P can be discarded. The final diagram consists of the surviving portions plus P. But how to find P?

The answer is that P starts at the convex hull of the set, where it is formed by the perpendicular bisector of a hull edge joining a point of L and a point of R. It then proceeds monotonically downward. Monotonicity is important; it allows P to be constructed without backtracking. Since the Voronoi diagram contains only a linear number of points and edges, a non-backtracking algorithm must finish in linear time. These were the only insights required and the collection of closest-point problems was completed about a week after my visit to IBM.

This was none too soon. In February of each year the ACM Computer Science Conference was held. Its principal function, rumor had it, was to serve as a job fair. Department chairmen from all over the country attended to watch potential faculty candidates describe their research. It's a very good way to judge both the depth of a person's work and her ability to communicate. In 1975 the conference was held at the Statler Hilton in Washington, DC, to which I ventured in hope of a decent faculty appointment. Yale had a policy of not making offers to its own graduate students, and in any event the transition from having been a student in a department to faculty in the same department is difficult to manage. IBM, on which I had pinned my hopes, surprised me by telling me that they were "reducing their head count," and it would be much more convenient for them if I just resigned instead of coming back. So, after seven years with IBM, five on full-time unpaid leave, I had to look for a job.

I prepared a short talk on the Voronoi diagram to deliver at the conference, complete with the color drawings that are so effective in communicating its properties. When I arrived at the hotel, I got a copy of the conference proceedings to see

who else would be speaking at my session. To my surprise, there was another paper on a geometry problem. Two operations researchers were studying a maximin facilities locations problem: given n points in the plane, where should a new point be located, within the convex hull of the others, so it is as far as possible from any of the n original points? An obvious application of this problem has to do with locating competing stores or noxious facilities such as sewage treatment plants. The authors, I learned from their abstract, would present an $O(n^3)$ algorithm the next day in my session.

I had never heard of the problem, but anything related to closest points screamed "Voronoi diagram" to me. The abstract in the conference program was tantalizing because it described only the result, not the method used to obtain it. I felt that this would be a trial by fire for Voronoi diagrams, since it was the first closest-point problem I hadn't known about at the time I found an $O(n \log n)$ algorithm for its construction. If the VD could be used to solve an entirely new problem, it was clearly a powerful tool.

The night before the talk, I stayed up to work on the facilities location problem. It was immediately clear that if the facility could be located at a point interior to the convex hull, then it must be at a vertex of the Voronoi diagram. (If you move away from a Voronoi point, you're getting *closer* to some point, not farther away.) The troublesome part was that if the point lay on the convex hull, it might be anywhere on the boundary. It took another hour to prove that in that case it must lie at an intersection of a convex hull edge and a Voronoi edge. That completed, I had an $O(n \log n)$ algorithm for a problem whose posers were to present a cubic algorithm the next day. While this was exciting, I worried about whether it was fair to the other authors to present it. I did so as politely as possible, and Joseph Traub, the chairman of the Computer Science Department at Carnegie Mellon, who was in the audience, invited me for a job interview.

When I came to CMU for the interview, Traub had arranged for me to see George Fix, the chairman of the Mathematics Department. Fix was a former colleague of Martin Schultz and had worked on triangulation algorithms for the finite element method, so he was receptive to CG. Later that day, after a hasty conference with Fix, Traub offered me a joint appointment in the Computer Science and Mathematics Departments. I was thrilled at this honor, but later learned the bad news—I had to maintain a full teaching load in both departments! I looked at the catalog of the Math Department and noticed that Combinatorics had not been taught during the preceding five years. I immediately proposed to resurrect it. Fix said he would have to put it to a vote of the faculty. It turned out that he had quite a lobbying job. The CMU Math Department was heavily dominated by analysts. When Fix explained to them that combinatorics was only being taught every five years, one of them grumbled, "Why so often?"

I had a few months left at Yale, and returned to a problem that had been nagging me for two years: how can you tell whether a polygon is self-intersecting? It is easy to check by examining each pair of edges to determine whether they intersection but this gives a quadratic algorithm. Could it be done faster? I posed the problem to Hoey, and after a week of intense introspection fueled by coffee and cigarettes and interrupted only by an occasional game of backgammon, Hoey had a beautiful idea. He first generalized the problem to that of determining whether any two of a set S of n line segments intersect, and then observed that if none of the segments is vertical, then with respect to any given value of x (say, $x = b$),

all segments of S that intersect the line $x = b$ are totally ordered by the "above" relation. That is, of any pair of segments c and d that intersect $x = b$ at abscissas e and f, respectively, then $c > d$ if $e > f$ and $c < d$ if $e < f$, while $c = d$ if $e = f$. As the value of b changes, two segments can only change places with respect to one another in the total order if they intersect somewhere. Hoey also realized that two segments intersect iff they are adjacent in the total ordering for some value of b. At this point in my thesis research, I was familiar enough with the subject to know that this was a totally original concept. No one had previously defined such a relation.

Hoey suggested sweeping a vertical line through S from left to right, maintaining a list of segments that intersect the line in the order determined by the "above" relation. A segment is added to the list when its left endpoint is encountered and is removed when its right endpoint in encountered. Before a segment is added, it is checked against its neighbors in the ordering to see if it intersects either of them. When a segment is removed, any pair of segments that then become adjacent must be checked for intersection. If each segment is added and eventually deleted from the list without an intersection being found, then S is intersection-free.

It remained to find a data structure in which the total ordering could be maintained, in which n insertions and n deletions could be made, and the successor and predecessor of each segment in the total ordering could be found n times, all in less than quadratic time. This task fell to me, along with proving the algorithm correct. At the time, AVL trees were barely known, and then only through a marginal reference in Knuth. 2-3 trees had just been invented. Either of these data structures would fill the bill, but 2-3 trees were easier to code, so we set about programming. The algorithm worked like a charm and was amazingly fast. It seemed clear that the algorithm worked, but my job of proving it was somewhat tricky because of the need to be careful about degeneracies. I was able to show that it always finds a leftmost endpoint, and wrote up the paper [**SH2**]. This sweep-line algorithm turned out to be a fundamental technique in CG that had no real progenitor. It formalized the tools necessary to solve the hidden-line problem, which had previously been attacked through a hodge-podge of ad hoc methods.

At about the same time, Hoey developed the generalized Voronoi diagram. I had earlier shown him the farthest point VD along with a proof that, unlike the closest-point VD, its graph is outerplanar. We had also experimented with Voronoi diagrams of line segments, polygons, sets of points and line segments, circles and other geometric figures. We also had made such diagrams in the and metrics. Many of these were later rediscovered and published by other researchers. But no one had yet considered the locus of closeness to multiple points. Hoey invented the higher-order VD and gave an argument that the sum of the number of Voronoi edges in all VD's of all orders on a given set of n points in the plane is only $O(n^3)$. (This was a slight surprise, since the total number of subsets of n points is 2^n.) I produced a precise enumeration, which appeared in our Voronoi diagram paper [**SH**]. This is the paper of which I am most proud, since it is so closely modeled after Karp's and achieved the desired result of unifying problems that had not even been mentioned previously in the same breath.

My thesis was not complete when I left Yale in June 1975, but I didn't see the point in remaining a graduate student. I didn't realize that I would get sidetracked by other research and responsibilities, and wouldn't actually turn in a thesis for more than another two years [**Sh2**]. I finally received a Ph.D. from Yale in 1978,

exactly three years after leaving. My grandfather lived to be 93 years old, and turned 88 shortly before I got my diploma. He made the trip to New Haven for the graduation ceremony with pride and understood quite well that the degree was for geometric work inspired by our Sunday lessons with his planimeter.

Hoey stayed on at Yale for a time and then became a graduate student at CMU. He had many other interests beside geometry, and now works for the Naval Research Laboratory in Washington. In the meantime, I met two additional collaborators. The first was Jon Bentley, who in 1975 was a graduate student at the University of North Carolina. As his thesis work, he undertook to generalize the sparsity algorithm for finding closest points into higher-dimensional space. His insight was that you could solve an n-point problem in d dimensions by solving two $n/2$ point problems in d dimensions (the divide step) and one n-point problem in d-1 dimensions (the merge step). This gave an $O(n \log^{d-1} n)$ algorithm instead of the rote $O(dn^2)$, and was a major advance [**BS2**]. I told Traub that we needed Bentley at CMU and his appointment was arranged in short order.

Bentley and I worked on a host of problems in geometry, statistics, and combinatorics. Most of them are described in his prodigious published output. Within a few years, he produced over 50 papers. He and Thomas Ottmann generalized the Shamos-Hoey line segment intersection algorithm to enumerate the intersections in $O((k + n) \log n)$ time, a result subsequently improved by others. Bentley and I taught and worked with students all day and then assembled at my house each night at 11:30 to watch reruns of the television series *Kojak* and do research with my dog Nassau, the world's best canine geometer. We were devoted fans of the show, and Bentley, who was constantly finding analogies between search algorithms and detective work, was not above quoting dialogue from *Kojak* in his scientific papers.

Bentley and I worked primarily on blazingly fast algorithms, since we were interested in large practical problems involving huge amounts of data, such as satellite images. For such work even $O(n \log n)$ algorithms are too slow—real-time processing must take no more than linear time. This means in some cases having to accept approximate answers. But not in all. Bentley and I spent a lot of time on average-case analysis. It bothered us that so many such analyses depended on unreasonable assumptions about the statistical distribution of the inputs. We finally found a way to construct the convex hull of a plane set in average linear time regardless of the form of the distribution of the points, provided that the distribution is continuous [**BS**]. From then on we were hooked, and hardly ever looked at another algorithm unless it was $O(n)$.

Jay Kadane of the CMU Statistics Department liked our work on average-case analysis and discrete algorithms for statistical problems and arranged for me to be appointed to a faculty position there also. I thus became the only person at CMU ever to hold appointments in the Computer Science, Mathematics and Statistics Departments simultaneously. (Don't worry, my first experience with a joint appointment left scars. I was promised that I wouldn't have to teach any extra courses, just an annual seminar in computational statistics.) In my view, all statistical problems are geometry problems. Some of my colleagues felt just the reverse was true.

My other collaborator was Gideon Yuval, a visitor to CMU from Israel. Yuval was an intuitive thinker, much like Hoey, and had no taste for formal proofs. He had flashes of great insight, and when he turned his mind to a problem for any length of time one could be sure that interesting ideas would emerge. One day I mentioned

to him a stumbling block in the theory of lower bounds. At the time, all non-trivial lower bounds on computation time were based either on fan- in, algebraic independence or information-theoretic arguments. For example, sorting requires $\Omega(n \log n)$ time because every comparison tree containing all $n!$ permutations of n objects has depth at least $n \log n$. There was no problem in existence having n inputs and a single number as output for which a polynomial-time algorithm was known for which any non-linear lower bound had been proven. This bothered me tremendously, since many geometry problems yield single numbers as results, and I couldn't demonstrate any good lower bounds for them.

Yuval asked me which problem I had in mind, and I said the mean distance among n points in the plane. (That is, the sum of the pairwise distances divided by the number of pairs.) This was a maddening problem because the average square of the pairwise distances can be found trivially in linear time. It was also difficult since no techniques were known for dealing with complexity in terms of square roots, except some old results of Hilbert not immediately applicable to this particular problem. Yuval went off to think about it. The next day he said, "Of course, it must take quadratic time in the worst case." This was striking news, since no quadratic lower bound on any polynomial-time problem was then known. (Quadratic, that is, in the number of inputs.) I challenged him to prove it, but proof was not Yuval's strong point. He instead gave an intuitive argument that dazzled me. He said to consider any algorithm that computes the function D that is the sum of the pairwise distances from the $2n$ coordinates of the points. Each distance is a square root. Now consider D as a function of $2n$ complex rather than real variables. The Riemann surface of D must have $\Omega(2^{n(n-1)/2})$ sheets, and therefore at least $n(n-1)/2$ square root operations must be required to produce a Riemann surface of this complexity.

I was familiar enough with Riemann surfaces from undergraduate school, but I had never heard the term used in connection with computational complexity. It seemed clear that Yuval's idea was either very profound or totally ridiculous. After all, what did an ordinary geometry problem have to do with complex variables? I struggled over the next weeks not only to produce a proof, but also to develop a model of computation in which the results would make sense. The "square root" as an elemental operation existed in no known model of computation. The key is having a set of operations in which only the square root operation can cause a bifurcation of the Riemann surface. Though unexpected, the Shamos-Yuval result is as tight as can be. For n points in the plane, exactly $n(n-1)/2$ square roots are necessary and sufficient to compute the mean distance [**SY**]. The result is also very robust in the sense that all kinds of auxiliary operations can be added to the model without reducing the lower bound. In addition to arithmetic operations, one may feel free to add exponentiation, trigonometric functions, hyperbolic functions, Bessel functions and many other special functions. None of these is sufficient to reduce the number of square roots required even by one.

It has always been my practice in looking at research problems to examine the simplest, almost trivial, cases to obtain a full understanding. In geometry, this often meant looking at one-dimensional problems. How difficult is it to compute the mean distance among n points on a line? Here no square roots are involved. If the points are sorted, it is easy. The interval between point k and point $k+1$ is part of the interval connecting all points whose left endpoint is between 1 and k and whose right endpoint is between $k+1$ and n, inclusive. This means that

the sum of the distances is just a weighted sum of the intervals between adjacent points. However, since the final result is a single number, how can we prove that sorting is required?

Yuval also provided the key to this one-dimensional case. He observed that in the arithmetic decision tree model, one can always compute partial derivatives of any computed quantity with respect to any of the input variables without any additional comparisons. He then explained that the partial derivatives of the mean distance with respect to the x-coordinates of the points can be used to reconstruct the values of k, that is, the indexes of the points in sorted order. Therefore, any mean distance algorithm can be transformed into a sorting algorithm without performing even one extra comparison. This gives an $\Omega(n \log n)$ lower bound for the mean distance on one dimension. This result is very powerful, for it is a virtual generator of lower bound results. Perhaps because it lies buried in a conference proceedings, it has been utilized only rarely and appears to be unknown to most researchers [**SY**].

During my second three-year appointment at CMU, I began to grow restless. It was difficult to obtain funding for geometry research because so few people were engaged in it that the government agencies assumed it was not important. A shining exception was the Office of Naval Research (ONR), which was a firm supporter from the beginning. Even while I was a graduate student at Yale, ONR allowed me to serve as co-principal investigator with David Dobkin. I was disappointed at the apparently slow uptake of CG by computer scientists, even though I had enthusiastic colleagues in Jon Bentley, Ron Graham, Franco Preparata, D.T. Lee, Leo Guibas, Gottfried Toussaint and Bernard Chazelle, and many others in Europe, such as Herbert Edelsbrunner and Thomas Ottmann.

Being at CMU, I was exposed to a lot of forefront software that was under development there. I noticed that after a project's final report was turned in, nothing else would be done with the resulting software. I thought this was wasteful and in 1979 I became a reluctant entrepreneur. I formed a software company to market the Scribe document production system, which was developed by Brian Reid, then a graduate student at CMU. His system was the forerunner of desktop publishing, and he eventually received the Grace Murray Hopper Award of the ACM for it. Growing the company turned my attention to business matters, and I eventually found it useful to obtain a law degree. These activities were inconsistent with being a full-time faculty member, so in 1981 I shifted to adjunct status and spent the next 17 years combining a legal career with starting software companies. At the beginning of 1998, I returned to CMU to lead a project called the Universal Library, which can be thought of as an electronic "library of libraries" a gateway to all digital Internet collections. A license to practice law, once obtained, is rarely relinquished, so I am still a partner in the Webb Law Firm, the largest intellectual property practice in Western Pennsylvania, where my concentration, logically, is in computer law.

The existence of my textbook with Franco Preparata [**PS**] is due to the intercession of Ron Graham. Every new discipline needs some time to mature before a sensible introductory book can be written, and indeed in the case of computational geometry none had appeared during the decade following 1975, a year in which many results were published. Graham, an generous and unselfish supporter of research in combinatorics and discrete geometry, thought that it was about time for a book. He would have been perfectly qualified to write one, but he felt that the

first book should come from the architects of the subject. After 1981 I was in no position to take on the task, so Graham cajoled Preparata into doing it.

I had met Franco at the SIGACT conference in Albuquerque in 1975 when I presented my first geometry paper [**Sh3**]. We subsequently attended numerous conferences together but had never actually worked on a problem together. He was kind enough to invite me to teach at the Winter School of Computational Geometry in Erice, Italy in 1978. (By curious coincidence, this was the last time I saw Yuval—on the ancient Carthaginian island of Motya, off the western coast of Sicily.) When Franco took up the book project, he proposed to start with the text of my Ph.D. thesis, supplementing it with additional material and adding new chapters. Even though the resulting book contains many of my words and results, it was really written by Preparata, and he is now working on a new edition.

I have no regrets about having retired from active work in computational geometry. I did as much as I could, and feel pleased about having laid out the field and constructed its first tools. Many recent results by others are so complex that I am certain I could never have discovered them myself, and the subject is certainly flourishing. Those who are quick to link cause and effect might note that CG did not really take off until I left it!

Like all fishermen, I sigh at the ones that got away. I wish I had been a participant in some results that escaped me completely: Megiddo's linear-time algorithm for linear programming in two and three dimensions, Lee and Preparata's linear-time algorithm for the kernel of a simple polygon, the results of Tarjan, Chazelle et al. on triangulating polygons, and the Lipton-Tarjan algorithm for separating planar graphs. I came so close with Megiddo. He was visiting CMU for a year in the Graduate School of Industrial Administration, where operations research was done. He called me up one day from across campus and asked whether I knew that my $O(n \log n)$ algorithm for the smallest circle enclosing n points in the place was not optimal. I had never heard of him and thought the call was a prank! Now there was a missed opportunity! And it would have been nice to have appreciated the importance of motion planning back then. However, I made my contribution and moved on, and it is a pleasure now to watch the oak trees that have grown from acorns planted in the 1970s.

I marvel that everything in my life preceding the work in computational geometry seemed to have prepared me for it, and am grateful for the confluence of people who contributed their time, ideas and encouragement. They did it because they loved the problems, and worked on them without any thought of having proper credit ascribed to them. Therefore I hope this account clarifies who did what in the early years.

References

[BS] J. L. Bentley and M. I. Shamos, *Divide and conquer for linear expected time.* Info. Proc. Lett. 7:87–91 (1977).

[BS2] J. L. Bentley and M. I. Shamos, *Divide and conquer in multidimensional space.* Proc. Eighth Annual ACM Symposium on Automata and Theory of Computing (May, 1976) 220–230.

[B] K. Q. Brown, *Geometric Transformations for Fast Geometric Algorithms.* Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University (1979).

[DL] D. P. Dobkin and R. J. Lipton, *On some generalizations of binary search.* Yale University Computer Science Department Technical Report (1974).

[F] I. Fáry, *On straight line representations of planar graphs.* Acta Sci. Math. Szeged. 11:229–233 (1948).

[G] R. L. Graham, *An efficient algorithm for determining the convex hull of a finite planar set.* Info. Proc. Lett. 1:132–133 (1972).

[H] F. Harary, Graph Theory. Addison-Wesley, Reading, MA (1971).

[K] R. M. Karp, *Reducibility among combinatorial problems.* In Complexity of Computer Computations. Edited by R. E. Miller and J. W. Thatcher. Plenum Press, New York (1972).

[Kn] D. E. Knuth, The Art of Computer Programming Volume III: Sorting and Searching. Addison-Wesley, Reading, MA (1968).

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

[S] R. Sedgewick, Algorithms. Addison-Wesley, Reading, MA (1983).

[Sh] M. I. Shamos, *An Absorber Theory of Gravitational Radiation.* Senior undergraduate thesis, Physics Department, Princeton University (1968).

[Sh2] M. I. Shamos, *Computational Geometry.* Ph.D. thesis, Department of Computer Science, Yale University, 1978. Available from University Microfilms, Ann Arbor, MI.

[Sh3] M. I. Shamos, *Geometric complexity.* Proc. Seventh Annual ACM Symposium on Automata and Theory of Computation (May, 1975) 224–233.

[Sh4] M. I. Shamos, Problems in Computational Geometry. Unpublished book manuscript (1974, revised 1977). Distributed in photocopy.

[SH] M. I. Shamos and D. Hoey, *Closest-point problems.* Proc. Sixteenth IEEE Symposium on Foundations of Computer Science (Oct., 1975) 151–162.

[SH2] M. I. Shamos and D. Hoey, *Geometric intersection problems.* Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science (Oct., 1976) 208–215.

[ST] M. I. Shamos and M. A. Tavel, *An absorber theory of acoustic radiation.* J. Acoust. Soc. of America 54:46–49 (1973).

[SY] M. I. Shamos and G. Yuval, *Lower bounds from complex function theory.* Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science (Oct., 1976) 268–273.

[T] R. E. Tarjan, *Depth-first search and linear graph algorithms.* SIAM J. Comput. 1:146–160 (1972).

[V] G. Voronoi, *Nouvelles applications des paramètres continus à la théorie des formes quadratiques.* J. reine angew. Math. 134, 198–287 (1908).

[WF] J. A. Wheeler and R. P. Feynman, *Interaction with the absorber as the mechanism of radiation.* Rev. Mod. Phys. 17:157 (1945).

[YB] I. M. Yaglom and V. G. Boltyanskii, Convex Figures. Holt, Reinhart and Winston, New York (1961).

[Y] A. C. Yao, *A lower bound to finding convex hulls.* J. ACM 28:780–787 (1981).

SCHOOL OF COMPUTER SCIENCE, CARNEGIE MELLON UNIVERSITY, PITTSBURGH, PA 15213
E-mail address: shamos@cs.cmu.edu