Strategic Directions in Computational Geometry Working Group Report

Roberto Tamassia (editor and working group chair),
Pankaj K. Agarwal, Nancy Amato, Danny Z. Chen, David Dobkin,
Robert L. Scot Drysdale, Steven Fortune, Michael T. Goodrich, John Hershberger,
Joseph O'Rourke, Franco P. Preparata, Jörg-R. Sack, Subhash Suri,
Ioannis G. Tollis, Jeffrey S. Vitter, and Sue Whitesides

ACM Computing Surveys 28(4), 591–606, December 1996.

This report outlines the evolution of computational geometry, highlights its past accomplishments, and discusses strategic directions for the future. It complements a recent report by Chazelle et al. on the impact of computational geometry on applied fields.

1. INTRODUCTION

Computational geometry investigates algorithms for geometric problems. For an introduction to the field, see the textbooks [Edelsbrunner 1987; Mulmuley 1994; O'Rourke 1994; Preparata and Shamos 1985] and the forthcoming handbooks [Goodman and O'Rourke 1997; Sack and Urrutia 1997]. Resources on the World Wide Web are also available (see, e.g., the *Directory of Computational Geometry Software* [Amenta], the *Computational Geometry Pages* [Erickson]), and *Geometry in Action* [Eppstein]. This report outlines the evolution of computational geometry, discusses strategic research directions with emphasis on methodological issues, and proposes a framework for interaction between computational geometry and related applied fields.

A preliminary version of this document is available on the World Wide Web at `http://www.cs.brown.edu/people/rt/sdcr/report.html`.
Address: Department of Computer Science, Brown University, 115 Waterman Street, Providence, RI 02912-1910, USA. `rt@cs.brown.edu`

1.1 Evolution of the Discipline

Motivated by the need for geometric computing in science and engineering applications that deal with the physical world, about twenty years ago a community of researchers started forming around the study of algorithms for geometric problems. A new discipline, christened *computational geometry*, was soon chartered with the dual mission of investigating the combinatorial structure of geometric objects and providing practical tools and techniques for the analysis and solution of fundamental geometric problems.

As a testimony to the original success of this mission, the initial body of computational geometry literature had a prominent presence both in the field of theory of computing and in applied areas, such as graphics, robotics, mechanical engineering, and pattern matching (see, e.g., the 1984 survey of the area [Lee and Preparata 1984]).

As the discipline came of age through the establishment of specialized conferences and journals, powerful new techniques of considerable mathematical sophistication were added to the existing repertory. With this formal strengthening, however, came an increased emphasis on the combinatorial aspects of computational geometry, which, in a sense, softened the original link with applications.

Such an inward research orientation freed computational geometers from the unpleasantness of modeling the complexity and imperfections of the physical world and of coping with the limitations of realistic computing devices. It allowed them to focus on the analysis of a Platonic world of simple, well-behaved, geometric objects that can be manipulated by idealized computing machines with unbounded memory space and real number arithmetic.

As mentioned above, the outcome of this research trend (see Section 2) was an impressive wealth of results on the combinatorics of objects with simple shape (such as points, lines, and polygons) in low-dimensional space (mostly two and three dimensions), and on the asymptotic complexity of fundamental geometric computations (e.g., convex hull, intersection reporting, point location, and proximity queries). The impact of computational geometry was especially strong in the field of design and analysis of algorithms. Indeed, major progress on general techniques for searching (e.g., fractional cascading), dynamic data structures (e.g., incremental rebuilding), randomized computing (e.g., random sampling), and parallel computing (e.g., cascading divide-and-conquer) was effected within the computational geometry community.

However, at the same time as the accomplishments of computational geometers were celebrated within the general field of theory of computing, they also became less understood and appreciated in applied circles. The simplifying models that enabled theoretical research to flourish turned out to be major impediments to technology transfer, and hindered computational geometry from accomplishing in full its dual mission. In particular, the following apparently innocent assumptions appear to be the main culprits:

—the reliance on asymptotic analysis as the ultimate gauge for estimating the performance of geometric algorithms, disregarding more practical aspects of efficiency;

—the adoption of real arithmetic, disregarding numerical finite-precision issues;

—the neglect of degenerate configurations, disregarding the difficulty of taking them into account in implementations;

—the model of uniform access to data in memory, disregarding the huge gap between the speed of main memory and disks.

As a consequence, the excitement of the computational geometry community at its theoretical accomplishments was mitigated by a sense of discomfort at the perceived loosening of ties with the very same applications that motivated the establishment of the discipline. The "pipeline" towards graphics, robotics, GIS, etc. continued to work successfully, but the rate of technology transfer lagged behind the growth of the "reservoir" of theoretical results.

It became clear within the community that a correction of course was needed to achieve a more balanced evolution of the field. Indeed, the last couple of years have witnessed an increasing consensus in the computational geometry community towards a renovation of the discipline that would reconcile theory with practice and reaffirm the original dual mission. It is interesting to notice that a renewed interest in applications is also developing in the theory of computation community (see, e.g., the report on "Strategic Directions in Theory of Computing" [Loui et al. 1996]).

1.2 Goals of the Report

A recent report entitled "Application Challenges to Computational Geometry" [Chazelle et al. 1996], by the Computational Geometry Impact Task Force chaired by Bernard Chazelle, stresses the importance of a reorientation of the field towards providing practical solutions to the specific needs of the applications that use geometric computing. That report makes several recommendations aimed at strengthening the pipeline with geometric computing applications, and identifies ten problem areas where computational geometry can have a major impact.

This report aims at complementing the Impact Task Force Report by identifying key research directions for the computational geometry community. Our focus is methodological rather than on specific problem areas.

1.3 Organization of the Report

In Section 2, we highlight selected past accomplishments of computational geometry, which illustrate the richness and depth of combinatorial and algorithmic results obtained so far. Sections 3–4 discuss strategic directions for the field. In particular, Section 3 addresses the methodological frameworks of robustness, finer-grain complexity analysis, implementation, and experimentation, while Section 4 deals with realistic computational paradigms such as parallel and distributed computing, external-memory algorithms, real-time computing, and randomized and approximation algorithms. In Section 5, we describe the emerging application domain (not mentioned in the Impact Task Force Report) of information visualization. A framework for interaction between computational geometry and related applied fields, such as graphics and geographic information systems, is presented in Section 6. Final remarks are made in Section 7.

2. PAST CONTRIBUTIONS

In this section, we outline some of the major achievements of computational geometry, and provide a visual depiction of the state of advancement of the discipline.

2.1 Selected Major Accomplishments

The purpose of the the following list of accomplishments is to provide *examples* of fundamental results in various subfields of computational geometry. As discussed in Sections 1.1 and 2.2, the majority of such results are combinatorial in nature and deal with asymptotic time complexity. Less explored are practical implementations and numerical robustness issues. Due to the nature of this report, all technical references have been omitted. Most of them can be found in [O'Rourke 1996] and by searching the Geometry Literature Database [Jones].

(1) *Simple polygons.* For nearly every problem based on simple polygons, asymptotically optimal algorithms have been found (e.g., finding the kernel). One of the last to succumb was triangulation: a simple polygon of n vertices can be triangulated in $O(n)$ time.

(2) *Segment intersection.* After many attempts, an output-size optimal algorithm was constructed for intersecting n segments in $O(n \log n + k)$ time, where k is the number of pairs of intersecting segments.

(3) *Convex hulls.* The convex hull of n points in d dimensions has $O(n^{\lfloor d/2 \rfloor})$ facets, and asymptotically worst-case optimal $O(n \log n + n^{\lfloor d/2 \rfloor})$ algorithms were found both for even and odd dimensions.

(4) *Voronoi diagrams and Delaunay triangulations.* An optimal and practical sweepline algorithm was discovered for constructing Voronoi diagrams in the plane; implementations are now widely distributed. The deep insight that Delaunay triangulations, the duals of Voronoi diagrams, are projections of convex hulls from one higher dimension unified two principal lines of research.

(5) *Linear programming.* Remarkably, it was established that linear programming could be accomplished in linear time (in the number of constraints) for fixed dimension d. The doubly-exponential dependence on d was steadily improved and eventually simple randomized algorithms were found whose expected dependence on d is subexponential. These results generalize to other optimization problems, such as finding the minimum spanning ellipsoid.

(6) *Point location.* Point location is a classical geometric searching problem, and is used as a subroutine in a variety of geometric algorithms. Various optimal algorithms for point location in a planar map have been devised. They use $O(n)$ space and support point location queries in $O(\log n)$ time, where n is the size of the map. Simpler data structures that are not asymptotically optimal but are very efficient in practice also exist. Progress has also been made on three-dimensional point location.

(7) *Range searching.* A significant achievement in the last decade was the near-complete resolution of the range searching problem, with almost-matching upper and lower bounds. The introduction of ε-nets led to linear-size data structures with near-optimal query times for simplex range searching (reporting points inside query simplices). Trading off space with query time permits

achieving faster (logarithmic) query times, using "$1/r$-cuttings": given an arrangement of n lines and a parameter $r < n$, the plane may be quickly (and deterministically) decomposed into $O(r^2)$ "triangles" (some unbounded), so that no triangle meets more than $O(n/r)$ of the lines. A similar result holds for arrangements of hyperplanes in d dimensions.

(8) *Complexity of arrangements.* Great strides were made in establishing the complexity of arrangements: The "Zone Theorem" established that the "neighborhood" of any one hyperplane in an arrangement of n hyperplanes in d-dimensions has complexity $O(n^{d-1})$. A difficult technical achievement was showing that any m faces in an arrangement of n lines in the plane have total complexity $O(m^{2/3}n^{2/3} + m + n)$. This bound applies, for example, to the number of incidences between n points and m lines. One long-sought result is that the complexity of the lower envelope of n surface patches in d-dimensions is $O(n^{d-1+\epsilon})$ (for any $\epsilon > 0$). The beautiful and intricate theory of Davenport-Schinzel sequences was shown to be central to the combinatorics of arrangements, establishing, for example, that the complexity of the lower envelope of n (perhaps interpenetrating) segments in the plane is $\Theta(n\alpha(n))$, where α is the inverse Ackermann function.

(9) *Visibility graphs.* Efficient computation of visibility graphs has been achieved in a polygon and in a polygonal environment: for a visibility graph with n vertices and k edges, $O(n+k)$ time suffices to find all shortest path trees (from a vertex to all others) in a polygon, and $O(n \log n + k)$ is achievable for construction of the visibility graph among obstacles in the plane.

(10) *Motion planning.* Many motion planning algorithms have been proved to be NP- or PSPACE-hard. Two general algorithms for solving any motion planning problem have been developed: cell decomposition and the roadmap algorithm, which run in time exponential with respect to the number of degrees of freedom of the robot, in the Turing machine model. For many special cases more efficient algorithms have been found, most notably for a polygon translating and rotating in the plane, for which a nearly-quadratic algorithm was developed.

(11) *Graph drawing.* A variety of techniques for constructing geometric representations of graphs have been devised. Major theoretical achievements include showing that the problems of upward planarity testing and representing a tree as a Euclidean minimum spanning tree are NP-hard. Algorithms have been discovered for upward drawings of trees with linear-area, planar straight-line drawings with integer coordinates and quadratic area, convex drawings in two and three dimensions, visibility representations and orthogonal drawings with the minimum number of bends, and upward planarity testing of embedded digraphs. In addition, algorithms with good performance in practice exist for trees, directed graphs, and undirected graphs.

(12) *Randomized geometric algorithms.* The introduction of random sampling techniques showed that many complex geometric problems have startlingly simple and efficient randomized solutions. Notable is the simple randomized incremental algorithm for construction of the convex hull, whose expected running time in d dimensions is asymptotically optimal. Simple optimal expected-time algorithms were also found for segment intersection (item 2 above). Deran-

domization led to several advances in deterministic running times, the even-d optimal hull algorithm (item 3 above) being a prominent instance. Finally, the introduction of the "backwards analysis" technique led to running-time analyses for randomized algorithms as simple as the algorithms themselves.

(13) *Dynamic geometric algorithms.* The study of dynamic algorithms and data structures has received major momentum from computational geometry. Basic dynamic geometric data structures include the segment tree, range tree, and interval tree. Based on them, efficient dynamic data structures have been devised for several fundamental geometric problems, including convex hull, point location, proximity, intersection, range searching, and path problems.

(14) *Parallel geometric algorithms.* Optimal or near-optimal work bounds were realized for many geometry problems under a variety of parallel computing models. For example, algorithms for the convex hull (cf. item 3 above) achieve in the EREW PRAM model an optimal runtime-processors product of $O(n^{\lfloor d/2 \rfloor})$ for even d, and a polylog(n) factor more for odd d.

2.2 A Visual Depiction of the State of Advancement

The space of geometric problems so far explored can be loosely characterized by two parameters: the dimension of the geometric space and the shape complexity of the objects in that space. Most of the computational geometry accomplishments of the past deal with low-dimensional space (especially two and three dimensions), and simple objects, such as points, polygons, and subdivisions (planar maps, three-dimensional cell complexes). Higher dimensions and curved objects remain relatively unexplored. A third parameter can be used to characterize the methodology used by researchers. The majority of the computational geometry literature deals with combinatorial analysis and asymptotic computational complexity. As discussed in the next section, the pursuit of numerical robustness and the development of practical implementations appear to be strategic methodological choices for the evolution of the field.

Figure 1 schematically illustrates the state of advancement of computational geometry research as a portion of a virtual "cube" whose x-, y- and z-axes are associated with methodology, dimensionality, and shape complexity. A vast portion of the cube remains to be explored.

3. METHODOLOGIES

We believe that the most important strategic direction for computational geometry is to substantially enlarge its arsenal of tools to include methods that can handle the practical aspects of geometric computing. While combinatorial and asymptotic analysis remains a cornerstone of the discipline, it is essential to start an extensive reexamination of geometric problems from the following viewpoints:

—*robustness*;

—*finer-grain complexity analysis*;

—*implementation*; and

—*experimental evaluation.*

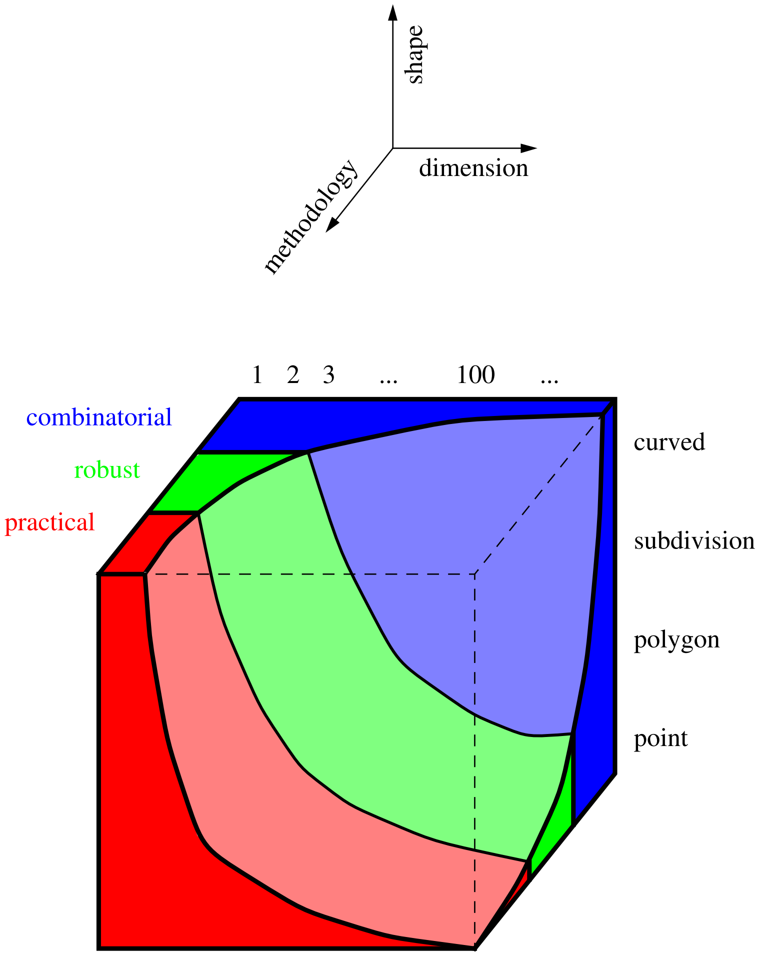

Fig. 1. State of the art of computational geometry.

Problems considered completely solved and no longer interesting from the viewpoint of asymptotic complexity reveal unexpected challenges when studied under a new light. For example, conventional asymptotically optimal algorithms for minimum link paths in a simple polygon and proximity queries on set of planar point sites perform poorly with respect to the arithmetic precision of the numerical computations, and new data structures and approximation schemes are needed to reconcile efficiency with robustness.

3.1 Robustness

Geometric algorithms are usually described in the conceptual model of the real numbers, with unit-cost exact arithmetic operations. However, the original assumption of a computational model obtained by extending the traditional RAM to real-number arithmetic proved less innocent than originally thought. Implementers often substitute floating-point arithmetic for real arithmetic. This leads to the well-known problem of numerical robustness, since geometric predicates depend upon sign evaluation, which is unreliable if expression evaluation is approximate. To equate floating-point arithmetic to real-number arithmetic turned out to be indefensible in practical applications.

Another convenient assumption has been the hypothesis of "general position," which dispenses with the detailed consideration of special cases. Unfortunately, degenerate conditions (colinearity, cocircularity) which are likely to be generated by coarse-grid data as they occur in practice, give rise to numerically critical events. Failures originating from these assumptions have fundamentally hindered the adoption of computational geometry by practitioners.

Over the years several approaches have been proposed to remedy these shortcomings. It is likely that no single approach may be capable of conferring robustness to geometric algorithms. Presumably, several tools may be included in an arsenal designed to achieve robust computations.

However, an approach that has the potential to yield a useful methodology is the following. The numerical computations of a geometric algorithm are basically of two types, which we may designate as *tests* and *constructions*. These two types have clearly distinct roles. Tests are evaluations of geometric predicates associated with branching decisions in the algorithm that determine the flow of control, whereas constructions are used to produce the geometric objects that normally represent the output of the application.

Approximations in the execution of the constructions give rise to approximate results, which may nevertheless be entirely acceptable as long as the maximum error does not exceed the resolution required by the application (in all cases, some more or less coarse grid). On the other hand, approximations in the execution of tests may produce incorrect branchings, which may have catastrophic consequences, since they may yield *structurally* (i.e., topologically) incorrect results (such as a missed intersection or an open polygon). Therefore, tests have much more stringent requirements, which leads to the conclusion that they must be carried out with complete accuracy, whereas some tolerance is permitted for constructions. (It must be observed, however, that such tolerance must be consistent with the topological structure of the result as provided by the tests.)

Complete accuracy would seem to require the infinite precision implied by real-number arithmetic. Fortunately, the inherently coarse nature of the input data comes to the rescue in this connection. Each predicate is expressible as the sign of a multivariate polynomial in the input variables. If input variables are assumed of degree 1, the degree of such a polynomial specifies the maximum precision required by the test in question. Of course, the maximum precision may have to be deployed only in near-degenerate cases. In typical cases, much lower precision may be sufficient to confidently evaluate the predicate. It is therefore the function of an "arithmetic filter" to identify the adequate precision. In this framework, the development of cost-effective filters may be one of the major challenges in the quest for robustness.

There is considerable evidence that adaptive-precision arithmetic, if engineered carefully, can substantially reduce the effective cost of extended-precision evaluation. Performance comparable to floating-point arithmetic has been achieved for algorithms with relatively modest precision requirements, e.g. evaluating simple predicates on points, lines, and planes in dimensions two and three.

A related issue is the problem of rounding geometric structures. The goal is to represent derived geometric structures in fixed precision, so that the key combinatorial/topological properties of the structures are preserved.

3.2 Finer-Grain Complexity Analysis

Asymptotic performance, rather than acting as a powerful and useful analysis tool, has frequently become the ultimate focus of computational geometry research. Unfortunately, the very nature of asymptotic "big-Oh" worst-case analysis carries its own inadequacy: the suppression of multiplicative constants from performance functions and the overemphasis on pathological scenarios. It is quite common for algorithms that have been declared "asymptotically optimal" in the Random-Access Machine (RAM) computational model to be inferior to "suboptimal" algorithms in practice. Focusing exclusively on asymptotic analysis discounts the importance of developing practically-efficient computational tools.

Remedying this shortcoming is an important and difficult task. A promising approach is to isolate significant primitives appearing in the execution of algorithms, such as pointer updates, evaluations of fixed-dimension determinants, and other data management operations (including those occurring in the handling of external and hierarchical memories), and to express performance not as a single function of application parameters (problem size, memory size, number of processors, etc.), but rather as a vector of such functions, each component of which quantifies the use of such primitives. It may even be desirable or necessary to precisely quantify some components of the performance vector (forfeiting the comforts of the big-Oh notation), in order to provide a realistic comparison between competing algorithms.

3.3 Implementation

Existing computational geometry algorithms are often directly relevant in industrial applications. However, the knowledge of those algorithms is not widespread, and robust, easy-to-use, well-publicized implementations are rare. The computational geometry community should strive to package its best geometric algorithms into easy-to-use software tools that can be used by non-specialists. Such tools would dramatically enlarge the set of potential users of geometric algorithms.

A handful of programs—implementations of convex hull and Voronoi diagram algorithms—have been distributed successfully. In fact, these popular programs have been distributed more widely than any computational geometry publication except perhaps Preparata and Shamos's textbook [Preparata and Shamos 1985]. The programs have been used in a wide variety of applications, most of them unanticipated by the program authors. These successes hint at the potential influence of computational geometry in practice, and should encourage further implementation efforts.

It is instructive to compare the teaching of computational geometry to that of sorting or hashing. Computational geometry is typically taught in specialized courses, or as a single short section in algorithms classes. Similarly, detailed analyses of sorting and hashing are taught in specialized algorithms courses. The crucial difference is that sorting and hashing are widely applied in other disciplines (e.g., systems programming), and are presented as black-box tools in non-algorithmic courses.

Except in a few cases, computational geometry has not provided the simple, flexible tools that would enable the kind of widespread use that sorting and hashing receive. Published libraries of geometric routines (see, for example, the *Directory of Computational Geometry Software* [Amenta]) often have a large granularity of adoption: potential users must adopt the whole library and its data models if they want to use any part of it. Writers of geometric software should provide lowest-common-denominator interfaces, as well as more efficient interfaces for sophisticated users. In particular, tools for geometric software development should enable non-specialists to specify their geometric problems in a straightforward manner, and to create programs for solving them by combining basic building blocks in a simple fashion.

Two efforts to build libraries of geometric software are underway—CGAL [Overmars 1997] in Europe and GeomLib [Agarwal et al. 1995] in the United States.

Successful examples of software tools, libraries, and repositories from the numerical, scientific, statistical, and symbolic computing communities include Netlib, LAPACK, SPSS, Mathematica, and Maple.

Other models of successful publication of algorithmic results include the popular *Numerical Recipes* and *Graphics Gems* books. In the same vein, computational geometry needs books and Web sites with titles like *Geometric Recipes* and *Geometric Tools*, with accompanying software. Further ideas for dissemination of algorithms and software appear in Section 6.

3.4 Experimental Evaluation

A prerequisite to deciding which algorithms to implement in geometric libraries is knowing which ones perform well in practice. The last couple of SCG (*ACM Symposium on Computational Geometry*) conferences have encouraged papers that do experimental work, as has SODA (*ACM-SIAM Symposium on Discrete Algorithms*). However, comparative studies of algorithms for solving given geometric problems are just beginning to appear. We need to encourage and reward such studies.

The algorithms should be implemented to share data structures and primitives whenever possible (so that the relative speeds of the algorithms and not the cleverness of the implementers is being tested). The programs need to be ported to a range of machines, because differences in architecture seem to greatly influence the relative speeds of algorithms.

A sub-area of experimental evaluation is choosing appropriate test data. We at least have some idea of what "random" point sets might be (uniform, normal, etc.), even if we do not know how realistically these distributions model the real world. But what is a "random" collection of non-intersecting line segments or rectangles or polygons? We have barely begun to consider such questions. Furthermore, how do these correspond to "real world" data?

Using "real world" data is not a panacea, either, because applications differ widely in what sort of data they use. What is needed is a collection of benchmark data drawn from a wide range of applications, analogous to the SPEC benchmarks used in computer architecture. No such collections of benchmark data are currently available, and creating such collections would be a valuable service to the community.

4. COMPUTATIONAL PARADIGMS

To be effective, the proposed renovation of computational geometry must be accomplished within the context of today's complex technology and computing environments. In particular, it is a key strategic issue to take into account the following realistic computational paradigms:

—*parallel and distributed computing*;
—*external-memory algorithms*;
—*dynamic and real-time computing*; and
—*approximation and randomized algorithms.*

4.1 Parallel and Distributed Computing

For time-critical applications, multiple processors may be needed to perform the specified computations in a short amount of time, and the data inputs for the computations may be distributed geographically. This viewpoint conforms with a general trend toward a more extensive deployment of concurrency and distributed computation. The programming ease deriving from the use of a uniform address space (shared memory) must be carefully weighed against the potentially better performance of a distributed-memory network.

Particularly attractive is algorithmic research within the so-called *coarse-grain parallel* model, which seems particularly attuned to a variety of geometric computations. The coarse-grain model reflects very closely the most plausible parallel/distributed computing technology of today or of the near future. In this model a system consists of relatively few (typically, tens/hundreds) processors (typically, off-the-shelf microprocessors), each equipped with a sizable private memory. The processors are either interconnected according to one of the conventional networks, or are part of a distributed network (as it turns out, since the number of processors is very small, the interconnection does not play a central role). For p processors and problem size n, the coarse-grain algorithmic approach consists of identifying p subproblems of size n/p whose solutions can be combined to solve the original problem. Thus, initially all processors act serially on their respective subproblems (solitary parallelism), and subsequently interact (cooperative parallelism) to combine the results of the first phase. For n much larger than p (a very realistic situation) the parallel time of the first phase dominates the time of the second phase, and optimal speed-ups are achievable. Problems that lend themselves to this approach are those possessing substantial data-locality.

4.2 External-Memory Algorithms

In the large-scale geometric databases and other data-intensive processing encountered in many applications, the main (random-access) memory of the processor is not large enough for the requirements of the application. This limitation faced by large-scale applications was recognized very early on in the computer era, and its correction (hierarchical memory) represents the first serious revision of the von Neumann model. As it happens, *Input/Output* communication (*I/O*) between levels of hierarchical memory is the bottleneck in many large-scale geometric applications. Algorithms designed specifically to make efficient use of two or more levels of memory are often called *external-memory algorithms* to emphasize the explicit use of memory beyond random-access main memory. The I/O bottleneck gets accentuated as processors become faster with respect to disks (currently the typical medium of external storage) or when multiple processors are used, prompting several researchers and companies to deploy external storage systems with parallel capabilities. The issues here are closely related to those outlined in the preceding subsection in connection with the coarse-grain model, when the private memories of the processors do not satisfy the requirements of the subproblems.

In many practical situations, we can restrict attention to the case of two levels; for such purposes the fairly realistic two-level multiple-disk I/O model covers both uniprocessor and multiprocessor systems: there are several disks and several processors (both currently in the range 1–2^{10}); each processor is equipped with a main memory that can store a limmited amount of data (currently in the range 2^{16}–2^{28}). The processors and the disks are connected by a network (such as a shared-memory interconnection, hypercube, or cube-connected cycles, as also postulated in the previously described coarse-grain model) that allows for efficient execution of some basic operations like sorting.

4.3 Dynamic and Real-Time Computing

Dynamic (or incremental) computation considers updating the solution of a problem when the problem instance is modified. Many applications are incremental (or operation-by-operation) in nature and the typical run involves *on-line* processing of a mixed sequence of *queries* and *updates*. In a real-time environment, it is essential that a query be answered within a fixed time bound t. If the size of the problem is such that the query time exceeds t, we should at least ensure that at time t the query algorithm has produced some useful results, i.e., an approximation of the query answer. An algorithm is called *interruptible* if it converges toward the exact solution by incrementally producing better and better approximations. Interruptible algorithms should be explored for a variety of geometric problems that arise in time-critical applications.

With respect to real-time applications, it is also interesting to devise approximation algorithms that use substantially fewer time and space resources than exact ones, with an ensuing performance/approximation trade-off. For example, consider the convex hull problem. Its exact solution needs $O(n \log n)$ time, but in a real-time environment we may be able to afford only $O(n)$-time computations. What is the best approximation of the convex hull that can be achieved with $O(n)$ time? Most previous research has been devoted to the study of polynomial-time approximation algorithms for NP-hard problems (e.g., for the traveling salesman problem). Approximation algorithms with $O(n)$ or $O(n \log n)$ time complexity should be studied for problems whose exact solution seems to require substantially more time (e.g., $O(n^2)$).

Motion is common with objects in the physical world and is a primary concern in geometric applications such as collision detection in robotics and visibility determination in computer graphics. Another facet of dynamic computation is dealing with continuously changing data. A *kinetic data structure* maintains attributes of mobile objects (e.g., convex hull and closest pair of a set of points in continuous motion). Previous research on this subject has focused on the case where the full motion of the objects is known in advance. Work is needed on a more realistic scenario where objects can change their motion on-line because of external impulses and interactions with each other.

4.4 Approximation and Randomized Algorithms

The lack of fast, simple, deterministic algorithms for many fundamental problems has motivated the study of *randomized* algorithms. In the last few years, a number of randomized geometric algorithms, based on elegant probability theory, have been developed that are significantly simpler and, in several cases, faster than their deterministic counterparts. Randomization is also useful for dynamic data structures, on-line algorithms, and intractable problems.

A number of interesting geometric problems are intractable, including certain instances of navigation, machine learning, target acquisition and tracking, and visualization. In many applications, it suffices to obtain a good but fast solution. For example, in a typical navigation problem, it is desirable to have a real-time algorithm that computes a reasonably short path instead of one that computes an optimal path but takes longer time. These factors initiated the study of *approximation* algorithms, which always determine a solution that is close to optimal. Approximation algorithms are also useful for higher dimensional problems because the running time of typical geometric algorithms increases exponentially with the dimension.

5. INFORMATION VISUALIZATION

The Impact Task Force Report [Chazelle et al. 1996] is an excellent resource on geometric computing problems within the following ten key application domains: computer graphics and imaging, shape reconstruction, computer vision, geographical information systems, mesh generation, robotics, manufacturing and product design, robustness, molecular biology, and astrophysics.

In this section, we describe the emerging application domain of *information visualization*, where computational geometry can have a major impact. Two specific fields are covered within information visualization:

—*graph drawing*; and
—*algorithm animation*.

Information visualization is identified as a strategic research direction also in the report on "Strategic Directions in Human Computer Interaction" [Myers et al. 1996].

5.1 Graph Drawing

The visualization of complex conceptual structures is a key component of support tools for many applications in science and engineering. Examples include software engineering (call graphs, class hierarchies), database systems (entity-relationship diagrams), digital libraries and WWW browsing (hypermedia documents), distributed computation (reachability graphs of communicating processes), VLSI (symbolic layout), electronic systems (block diagrams, circuit schematics), project management (PERT diagrams, organization charts), decision support (scheduling and logistic diagrams), medicine (concept lattices), telecommunications (ring covers of networks), and law (conceptual nets).

Foremost among the visual representations used are drawings of networks, graphs, and hypergraphs. A variety of graph drawing algorithms have been developed in the last decade [Di Battista et al. 1994]. Several graphic standards such as straight-line, polyline, and orthogonal, have been used to represent graphs, depending on the application. Major geometric problems in information visualization include:

—optimization of measures of quality of drawings, such as number of bends, number of crossings, and area;
—trade-offs between quality measures of drawings, e.g., between area and angular resolution of planar straight-line drawings;
—conceptual "fisheye views" for displaying large graphs;
—label placement for nodes and edges of a drawing;
—incremental and interactive layout maintenance;
—three-dimensional representations.

Detailed experimental studies on the practical performance of layout algorithms are also much needed.

5.2 Algorithm Animation

The visual nature of geometry makes it a natural area where visualization can be an effective tool for communicating ideas. This is enhanced by the observation that much research in computational geometry occurs in two and three dimensions, where visualization is highly plausible. Given these observations, it is not surprising that there has been noticeable progress during the past few years in the production of visualizations of geometric algorithms and concepts [Hausner and Dobkin 1997]. There is every reason to believe that this will continue and even accelerate in the future.

As anyone who has tried to implement a complex geometric algorithm knows, implementing geometric algorithms is a difficult task. Conventional tools are limited as aids in this process. The programmer spends time with pen and pencil drawing the geometry and data structures the program is developing. This problem could be solved by the use of visualization tools. In the ideal world, this visualization would be used for three purposes: demonstration, debugging, and isolation of degeneracies. Ideally, we would like to use the same tools for all three functions. For example, we would use the tool to help us debug the implementation of an algorithm by providing visual interaction during the debugging process. Next, we would like to use the same tool to create a visualization of the algorithm with which the user can interact. This interaction could be either passive or active. For example, a video tape provides passive interaction since the viewer's controls are limited to the VCR controls. Active interactions allow the viewer greater control over the visualization. Finally, there is the issue of isolating problems in code that is symbolically correct. Typically, such bugs come from degeneracies either in the data or in the computational model. Visualization has the potential to be a great help here as a tool allowing the user to jump into the code at (or preferably before) the point at which it breaks.

The problem of creating active interactions remains largely unsolved. It is still the case that a visualization demonstrates the behavior of an algorithm on one sample input and explains the behavior of the algorithm on that input. A better scenario would allow the user not only to specify the input, but also to interact with the view (and possibly even the input data) as the algorithm is running. There are a few existing systems that allow the user to interact with a running animation. However, the interactions come at a price. The viewer must typically have the hardware that was used to develop the interaction. This limits the ability to integrate such animations into hypermedia documents. There is hope that the emergence of Java and VRML will help remove this limitation.

6. INTERACTION WITH OTHER DISCIPLINES

Computational geometry has established itself both as a discipline and as a community of researchers. To realize the discipline's potential for usefulness to others and to maintain its vigor, the community now seeks closer collaboration with application domains that inspire geometry problems. At the same time, it wishes to maintain both its identity and its traditional contacts with mathematics.

Previous sections have suggested computational paradigms and methodologies for computational geometers to adopt in order for their research to become usable by others. However, adopting new methods is not enough: real-world problems are often inherently interdisciplinary in nature and international in impact. Hence mechanisms are needed that facilitate the crossing of boundaries between academic disciplines, between countries and continents, and between universities, government research labs and industries.

The organization of effective mechanisms for interaction is a job not only for individuals (e.g., students, researchers, university-industry liaison officers, administrators), but also for computational geometry community groups (e.g., program committees, boards) and organizations (e.g., professional societies, funding agencies). Hence the remarks below are addressed to a wide audience, and while stated in the context of geometry, many apply to other subfields of computer science as well.

6.1 Education for Collaboration

Certainly the horizons of computational geometers can be broadened by guest speakers at seminars and conferences who come from other, related fields. Interdisciplinary workshops and short courses can go much further. International workshops such as those run by the Dagstuhl Research Center in Germany and the summer joint research conference program run by AMS-IMS-SIAM provide models.

Graduate education in computational geometry should provide an opportunity for interdisciplinary study and, where possible, industrial collaboration. This could be facilitated for example by university-industry internships, degree programs with minor options in other fields, and special topics or projects courses taught by industrial researchers. Students in disciplines that have a geometric component should be informed of the possible relevance of computational geometry courses to their program of study.

6.2 Fostering Collaboration Across Boundaries

Opportunities for professionals to share geometry problems across academic disciplines and across university/industry boundaries should be fostered. This might take the form of short or long term visits for study or consulting, including consulting by academics within their own universities. The value of such exchanges should be recognized by agencies and institutions.

Mechanisms to ease publication of interdisciplinary research, and to promote publication of research outside the boundaries of traditional, narrow subdisciplines, should be designed and considered. Researchers should be able to build and maintain a reputation within the computational geometry community, while at the same time making their applied results known to the community of the application area. Possibilities for double-publishing might be explored (e.g., a research summary for one community accompanied by a full paper for the other).

6.3 Dissemination of Knowledge

Unfortunately, many computational geometry algorithms are either completely unknown or otherwise inaccessible to researchers and practitioners outside the community. Computational geometers should continue to address this problem by contributing expository writing such as handbooks. Handbooks currently in preparation include [Goodman and O'Rourke 1997; Sack and Urrutia 1997]. Other possibilities include creating collections modeled after *Graphics Gems*, and contributing survey articles to publications such as *ACM Computing Surveys*, *Communications of the ACM*, or *Scientific American*. Such expository literature should provide potential users with pointers to implementation advice, performance results, and any available code.

The availability of systems that contain libraries of well-documented code for geometric problems forms an important aspect of knowledge dissemination. The availability of good programming environments and usable code facilitates the development of geometric applications for both specialists and non-specialists. Hence such systems should allow easy export of code to other systems, such as a geographical information system. In return, the design and implementation of such environments give rise to a host of interesting research problems for computational geometers. Current efforts include the *Workbench for Computational Geometry* (at

Carleton University, Ottawa), *XYZ-Geobench* at ETH Zürich, *LEDA* at Max Planck Institut für Informatik, Saarbrücken, the *CGAL* project involving a consortium of seven European sites (Utrecht University, ETH Zürich, Free University Berlin, INRIA Sophia-Antipolis, Max Planck Institut für Informatik at Saarbrücken, RISC Linz, and Tel Aviv University), and the GeomLib project of the Center for Geometric Computing, a consortium of three US sites (Brown University, Duke University, and The Johns Hopkins University).

Journals can publish implementation-oriented research articles by developing standards for refereeing code and by making the code associated with accepted articles available over the Web. Code accepted by a journal would become a citable, refereed journal item. Already several conferences (e.g., ESA, SCG, SODA, WADS) have started accepting papers of the above nature. Also, journals such as the *International Journal on Computational Geometry and Applications* (ed. D.T. Lee), *Computational Geometry: Theory and Applications* (eds. J.-R. Sack and J. Urrutia), the *ACM Journal of Experimental Algorithmics* (ed. Bernard Moret) as well as several special issues in other journals are currently seeking such papers.

6.4 Information Links

Theoretical and applied areas in which computational geometry could play a role span several disciplines; indeed they span the scope of several professional societies. For example, research in fields such as computer vision, automation, manufacturing, CAD/CAM, robotics, computer graphics, topology, geometry, tomography, medical imaging, polyhedral combinatorics, combinatorial optimization, cartography, and geographic information systems is reported in the conferences and journals of professional societies such as ACM, AMS, SIAM, IEEE, ASME, ORSA, and AGI (Association for Geographic Information).

Technical terms are not standardized: roughly the same problem may have different names in different disciplines; conversely, different problems may receive the same name. Furthermore, applied problems typically elude precise, tidy, mathematical definition. Mechanisms are needed for the rapid cross-referencing, across discipline and professional society boundaries, of geometric concepts, problems, keywords, solutions methods, and implementations.

For problems that do admit precise description, practitioners should be able to find relevant information easily, including pointers to literature and code. They should also be able to pose geometric problems, whether of a general or a specific nature, to the computational geometry community at large.

The computational geometry community should consider ways to maintain, build on and cross-link resources (especially electronic ones) for geometry problems. A number of individuals and research groups have initiated efforts to create electronic sources of information. Perhaps such efforts should be imitated at the level of professional societies, to insure continuity and ease of access across disciplines.

A kind of electronic encyclopedia of geometry might be designed, with mechanisms for "looking up" and/or posing geometry problems in words and images, with cross-referencing between problems and application domains, and with pointers to code libraries and researchers.

6.5 Rewarding Experimental and Applied Research

The computational geometry community should continue to design strategies to encourage and evaluate experimental and applied work. However, strategies such as creating separate categories for theoretical and applied papers at conferences and such as creating bench marks and standard data sets should be constantly monitored for desired effect.

Clearly, providing implementation results and comparisons requires substantial research effort as well as time investment, so suitable reward structures should be devised. These might take the form of publication in journals or established, public geometric libraries as discussed above.

The commercial potential for geometric libraries provides major incentives for researchers to work on implementation issues: the possibility of financial reward, and also, the satisfaction of making something that works and gets used. Since the availability of libraries will inspire and facilitate yet more implementation and application-oriented research, the price structure for commercial code should differentiate between academic and industrial/commercial users.

6.6 A Vision for Interaction

The vision for many in the computational geometry community is that computational geometry emerge as the discipline where for geometry, theory meets practice, where problems of an applied nature inspire and inform research problems in computational geometry and mathematics, where theoretical results are implemented, made usable, and disseminated to application domains.

7. CONCLUSIONS

Computational geometry is a lively discipline that is undergoing a crucial phase of its evolution. We have identified methodologies and computing paradigms that we consider of strategic importance for the growth of the discipline and its impact on applications. The key message is that computational geometry should reaffirm its dual mission of investigating the combinatorial structure of geometric objects and providing practical tools and techniques for the analysis and solution of fundamental geometric problems

8. CONTRIBUTORS AND ACKNOWLEDGMENTS

This report represents the efforts of the Computational Geometry Working Group formed as part of the *ACM Workshop on Strategic Directions in Computing Research*, held at the Massachusetts Institute of Technology Laboratory for Computer Science, Cambridge, Massachusetts, USA, on June 14–15, 1996. The main ideas presented in this document were discussed at the working group meetings scheduled during the workshop. The material contained in this report originates in part from the participants' position statements, from the "Computational Geometry Column 29" [O'Rourke 1996], and from the *Center for Geometric Computing* proposal on "Applicable and Robust Geometric Computing" [Agarwal et al. 1995].

This report benefited from a white paper on "Exact Computation and Reliable Geometric Software" contributed to the working group by Chee Yap, and from a previous report entitled "Application Challenges to Computational Geometry," [Chazelle et al. 1996] by the Computational Geometry Impact Task Force chaired by Bernard Chazelle.

Comments and suggestions from Leo Guibas, Chris Hankin, and Michael Loui are gratefully acknowledged.

Finally, we would like to thank Peter Wegner for encouraging the formation of this working group and for many useful discussions.

REFERENCES

AGARWAL, P. K., GOODRICH, M. T., KOSARAJU, S. R., PREPARATA, F. P., TAMASSIA, R., AND VITTER, J. S. 1995. Applicable and robust geometric computing. http://www.cs.brown.edu/cgc/.

AMENTA, N. Directory of computational geometry software. http://www.geom.umn.edu/software/cglist/.

CHAZELLE, B. ET AL. 1996. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96 (April), Princeton Univ. http://www.cs.duke.edu/~jeffe/compgeom/taskforce.html.

DI BATTISTA, G., EADES, P., TAMASSIA, R., AND TOLLIS, I. G. 1994. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl. 4*, 235–282.

EDELSBRUNNER, H. 1987. *Algorithms in Combinatorial Geometry*, Volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Heidelberg, West Germany.

EPPSTEIN, D. Geometry in action. http://www.ics.uci.edu/~eppstein/geom.html.

ERICKSON, J. Computational geometry pages. http://www.cs.duke.edu/~jeffe/compgeom/.

GOODMAN, E. AND O'ROURKE, J. Eds. 1997. *Handbook of Discrete and Computational Geometry*. CRC Press. To appear.

HAUSNER, A. AND DOBKIN, D. 1997. Making geometry visible: An introduction to the animation of geometric algorithms. In J.-R. SACK AND J. URRUTIA Eds., *Handbook on Computational Geometry*, pp. ??–?? North Holland. To appear.

JONES, W. Geometry literature database. http://www.cs.duke.edu/~jeffe/compgeom/biblios.html#geombib.

LEE, D. T. AND PREPARATA, F. P. 1984. Computational geometry: a survey. *IEEE Trans. Comput. C-33*, 1072–1101.

LOUI, M. C. ET AL. 1996. Strategic directions in theory of computing. *ACM Computing Surveys 28*, 4. http://geisel.csl.uiuc.edu/~loui/complete.html.

MULMULEY, K. 1994. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ.

MYERS, B., HOLLAN, J., CRUZ, I., ET AL. 1996. Strategic directions in human computer interaction. *ACM Computing Surveys 28*, 4. http://www.cs.cmu.edu/~bam/nsfworkshop/hcireport.html.

O'ROURKE, J. 1994. *Computational Geometry in C*. Cambridge University Press.

O'ROURKE, J. 1996. Computational geometry column 29. *Internat. J. Comput. Geom. Appl. ??*, ?–? Also in SIGACT News, 27:3 (1996), Issue 100, to appear.

OVERMARS, M. H. 1997. Designing the computational geometry algorithms library CGAL. In *Applied Computational Geometry (Proc. WACG '96)*, Lecture Notes in Computer Science (1997). Springer-Verlag.

PREPARATA, F. P. AND SHAMOS, M. I. 1985. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY.

SACK, J.-R. AND URRUTIA, J. Eds. 1997. *Handbook on Computational Geometry*. North Holland. To appear.