Application Challenges to Computational Geometry

CG Impact Task Force Report*

Abstract

With rapid advances in computer hardware and visualization systems, geometric computing is creeping into virtually every corner of science and engineering, from design and manufacturing to astrophysics to molecular biology to fluid dynamics. This report assesses the opportunities and challenges this presents for the field of computational geometry in the years ahead. Can CG meet the algorithmic needs of practitioners? Should it look to applied areas for new sources of problems? Can CG live up to its potential and become a key player in the vast and diverse world of geometric computing? These are some of the questions addressed in this document. It was prepared by a group of computer scientists, engineers, and mathematicians with extensive experience in geometric computing. This report is intended as a wake-up call rather than an agenda setter. It is hoped it will engage a community-wide discussion on the future of computational geometry.

This document is available as Technical Report TR-521-96, Princeton University, April 1996. It also accessible on the Web at URL,

http://www.cs.princeton.edu/~chazelle/taskforce/CGreport.ps

1 Preamble

The fraction of computing falling under the loosely defined rubric of "geometric computation" has been on the rise and is likely to become dominant in the next decade. Computer graphics, manufacturing, scientific visualization, computer vision, astrophysics, molecular biology, and fluid mechanics are just a few in a crowd of avid users of geometric computing. Where does computational geometry fit into all this?

Twenty-odd years ago, the nascent field of computational geometry set sail on a mission to build general tools — analytical and computational — to satisfy the algorithmic needs of geometric computing [45, 64, 98, 108, 114, 120, 129]. The intention was to create a body of knowledge to which computer programmers could turn for help when wrestling with geometric

*The Computational Geometry Impact Task Force: Nina Amenta (Xerox PARC), Tetsuo Asano (Osaka Electro-Comm. U.), Gill Barequet (Tel Aviv U.), Marshall Bern (Xerox PARC), Jean-Daniel Boissonnat (INRIA), John Canny (U.C. Berkeley), Bernard Chazelle (**Chair**, Princeton U.), Ken Clarkson (AT&T Bell Laboratories), David Dobkin (Princeton U.), Bruce Donald (Cornell U.), Scot Drysdale (Dartmouth U.), Herbert Edelsbrunner (U. Illinois at Urbana-Champaign), David Eppstein (U.C. Irvine), A. Robin Forrest (U. East Anglia), Steve Fortune (AT&T Bell Laboratories), Ken Goldberg (U.C. Berkeley), Michael Goodrich (Johns Hopkins U.), Leonidas J. Guibas (Stanford U.), Pat Hanrahan (Stanford U.), Chris M. Hoffmann (Purdue U.), Dan Huttenlocher (Cornell U.), Hiroshi Imai (U. Tokyo), David Kirkpatrick (UBC), D.T. Lee (Northwestern U.), Kurt Mehlhorn (Max Planck Inst.), Victor Milenkovic (U. Miami), Joe Mitchell (SUNY at Stony Brook), Mark Overmars (U. Utrecht), Richard Pollack (Courant Institute, NYU), Raimund Seidel (U. Saarbrücken), Micha Sharir (Tel Aviv U. and NYU), Jack Snoeyink (UBC), Godfried Toussaint (McGill U.), Seth Teller (MIT), Herb Voelcker (Cornell), Emo Welzl (ETH Zürich), and Chee Yap (Courant Institute, NYU).

problems. The vision was that of a two-way pipeline: applications areas feeding CG with important practical problems, and CG in turn providing answers in the way of algorithmic tools and mathematically sound analyses.

What is the state of the "pipeline" today? Before answering this question, one may want to step back a little. Any new field needs to build itself a home and a home needs foundations. On that score one may strike a celebratory note. Indeed, computational geometry has met with considerable success in resolving the asymptotic complexity of basic geometric problems (e.g., convex hulls, Voronoi diagrams, triangulations, low-dimensional optimization) [31]. Most of the classical problems posed in the first decade of the field have been solved. Many of them entailed building entirely new sub-areas in the field of algorithm design. For example, the twin use of randomization and derandomization has been the main force behind some truly remarkable success stories (e.g., optimal algorithms for convex hull and Voronoi diagrams, linear-time solutions of LP-type problems). Multidimensional searching, triangulation, geometric sampling, the theory of Davenport-Schinzel sequences, and zone theorems are some of the crowning intellectual achievements in the field of CG.

Computational geometry has been remarkably dynamic, productive and creative. In fact, in the area of algorithms and data structures, CG is arguably where most of the action has been in the last decade. Building theoretical foundations was undoubtedly the proper course to chart at the outset: CG can look back to its accomplishments with legitimate pride.

In the midst of its success, however, the field is standing at a crossroads. There are two options: CG can use its successes as justification for keeping the pursuit of theoretical investigations as the centerpiece of its agenda. Or it can move towards building an effective pipeline with geometric computing. In the first case, CG might simply fall out of the economic loop altogether: it might even shrink to the status of a recreational activity. In the second case, CG might grow to become as indispensable to geometric computing as, say, civil engineering is to bridge-building. The choice is for the CG community to make.

Why such alarm? The sobering reality is that, notwithstanding a few exceptions, the pursuit of practical solutions that specifically address the users' needs has lagged behind. At the same time, the computational geometry community has been largely unsuccessful in reaching out and communicating its more practical discoveries to potential users. Technology transfer has been slow and sporadic. To put it bluntly, the much-vaunted pipeline looks more like a pipe dream.

It need not be so. An effective pipeline is probably a feasible project: one certainly worth undertaking given the stakes. Among the problems to overcome first are the "obvious" ones: CG'ers must start thinking not only in terms of asymptotic complexity but also in terms of code robustness, precision, CPU times, standards, benchmarks, distribution, etc. Perhaps less obvious, though just as important, are communication problems of a more cultural nature: For example, why is it that in the graphics and computer vision literatures, rotating an object is considered (rightly so) a linear algebra problem, but computing a visibility graph or matching a geometric pattern is rarely considered a computational geometry problem? There are many computational-geometric questions in vision, graphics and robotics that are not recognized within those fields to belong in the province of CG'ers.[1] For this CG deserves most of the

[1] Defined loosely as the community gathered around such conferences as ACM SOCG, CCCG, and journals such as D&CG, IJCG&A, CGT&A, or more broadly, the geometric components of Algorithmica, J. Algorithms, SIAM J. Computing, JACM, etc. We recognize that many people not attached to that community do bona-fide computational-geometric work nevertheless. The focus of this report, however, is on the community that makes the design and analysis of geometric algorithms its primary occupation.

blame. It has failed to address the problems that people in practice want to see solved. In the absence of a pipeline, CG can never hope to be the place practitioners turn to for help. The sobering fact is that, realistically, the pipeline is mostly for CG'ers to build.

Does geometric computing need CG? It is too easy and tempting for CG'ers to convince themselves that it is the case. We try not to succumb to this temptation here. We simply state our belief that computational geometry is the *only* arena dedicated solely to algorithmic pursuits in geometric computing: discrete geometers want to understand countable geometric structures; combinatorial geometers want to count or approximate or enumerate them; practitioners want to produce working code for solving specific geometric problems. Only computational geometers make the algorithmic understanding of geometric problems their central preoccupation. We believe such pursuits can be the key to breaking current and future computational bottlenecks in many areas of engineering. This document gives abundant evidence to support this view.

If so, then why has CG broken so few "computational bottlenecks" that practitioners care about? Perhaps CG has been reluctant to cast itself in the demeaning role of "algorithm caterer" at the service of practitioners. This narrow view of CG is both false and dangerous. CG's link to the real world is essential to its intellectual vitality. The current feebleness of that link has been harmful to the field as a whole. Not only has it squelched the impact of CG in the engineering community, but research opportunities have been missed, scientific challenges have been overlooked. The intellectual landscape of CG has much to gain from close interaction with the real-world practice of geometric computing.

It is not our intent to pit theory and practice against each other, and decree the relative worthiness of each. We also want to make it absolutely clear that we are firm believers in the crucial importance of theoretical work. If anything we believe that the existence of an effective pipeline would further motivate all sorts of new theoretical investigations which the current insularity of the field has hindered. Our view is that a mature field of computational geometry should be home to a diversity of interests, some theoretical and some practical, most of them feeding on and contributing to the others.

It is particularly important to create a platform linking theoretical research at one end to production-quality, usable software at the other end. Such a platform exists in combinatorial optimization, where some of the most impressive theoretical advances (such as interior-point methods) have had tremendous impact in practice. CG is nowhere near this happy state of affairs.

To get there will require a concerted effort. The prevailing winds are favorable, however. A combination of factors, from the intellectual maturing of CG to more prosaic forces such as the job market and funding agencies, are causing a subtle but profound reorientation of the field. Instead of letting these factors alone dictate this reorientation, this document attempts to make the CG community itself the main agent of reform.

This report includes a set of specific recommendations (§**2**), followed by a sample of problem areas where computational-geometric techniques can have a major impact:

§**3** Computer Graphics and Imaging

§**4** Shape Reconstruction

§**5** Computer Vision

§**6** Geographical Information Systems

§**7** Mesh Generation

§**8** Robotics

§**9** Manufacturing and Product Design

§**10** Robustness

§**11** Molecular Biology

§**12** Astrophysics

This list of problem areas is not comprehensive by any means. It is a compendium of short case-studies, which, it is hoped, will stimulate research and in a modest way help to redirect computational geometry towards more practical ends. To illustrate what might be achieved we also include a few concrete success stories, where cutting-edge research in computational geometry has had a pivotal role in the production of geometric software.

In Section §13 we list some important resources for computational geometers (journals, conferences, Web sites, mailing lists, software).

We wish to close this section on a note of optimism. Computational geometers are blessed with a field whose scope dwarfs most other areas of computer science. Cryptography, graph algorithms, optimization, computational biology, etc., are all critically important, but they are relatively well-focused areas. By contrast, computational geometry spreads its wings over the entire computing spectrum. This breadth is an asset. But it is also a challenge. Indeed, the field is so vast that it is fragmented and has trouble recognizing itself under one roof. Computational geometry is diverse and so is its community. Within this diversity, however, we must strive to build an identity and unity of purpose. This is the main objective of our effort.

This is not meant to imply that all worthy computational-geometric research is to be done under the aegis of the CG community. Researchers in graphics, modeling, manufacturing, biology, etc, have their own traditions of fine geometric work: there is no need for this to change. What must be remedied is the lack of communication between practitioners and theoreticians of geometric computing.

It is imperative to make CG research more responsive to the needs of users. But, again let us restate our belief that not all computational geometers need to work in directly applicable research. Long-term structural explorations are also needed. Our point is not to emphasize practice to the detriment of theory. On the contrary, our chief goal is to broaden —not simply to shift—the band which CG occupies in the spectrum of computer science: Instead of a collection of small houses scattered across town and oblivious to one another, we envision CG as a large edifice with a theoretical wing and a practical wing and many aisles (the “pipeline”) connecting them. Our purpose is not to bring down one of the wings, but to consolidate the edifice.

2 Recommendations

It will take more than a change of heart to reform computational geometry. Structural changes are necessary. Happily some of them are already underway. We identify four broad categories: (1) Production and distribution of usable (and useful) geometric codes; (2) Joint forums between CG and applied areas; (3) Experimental research; (4) Reform of the reward structure in CG.

Remark: Educational matters were left out from our discussion. Teaching and research in CG at the undergraduate and graduate levels are critical issues that must be addressed. But the great differences among national education systems precluded a discussion that would have been of much meaning to the international CG community at large.

1. PRODUCTION AND DISSEMINATION OF GEOMETRIC CODE: Journals are the time-honored repository of scientific knowledge. But what about software? Journals are woefully inadequate. An on-line library of geometric code available through the Internet would be a useful starting point. Many questions need to be ironed out. Should the library conform to rigid formats and should contributions be refereed (ie, pass a number of standardized tests) before being archived? Or should it follow the freewheeling, open-door policy of most archives on the Web? What programming languages should be used? What documentation and maintenance service should be expected? Should data type conventions be enforced?

   Applied areas such as graphics and geometric modeling have produced vast amounts of geometric code over the years, some of it publicly available. Sometimes, differences in programming languages and standards might make integration difficult, but this need not be always the case. At the very least, how to tap into such resources effectively should be addressed. Our sense is that many of these issues cannot be settled once and for all before trying out various alternatives on a smaller scale. There are models to learn from in optimization, computer algebra, and numerical analysis. But it is unclear that any of them can be replicated verbatim to fit the needs of CG. For example, the uniquely complex data typing in CG places formidable hurdles in the way to correctness, robustness and portability. Only preliminary efforts have been attempted in CG so far, and to plan *the* perfect system in a vacuum is perhaps utopian. People should feel encouraged to try out their ideas and time will tell what works and what does not.

2. INTERDISCIPLINARY FORUMS: CG'ers should attend conferences in applied areas and practitioners should be invited to address the main CG conferences. This is already happening. Clearly, this is not enough. To provide fresh grist for CG's mill, means must be found so that CG'ers are given opportunities to hear about computational-geometric bottlenecks in applications areas. For example, one could organize joint conferences between CG and specialized topics. Special issues of non-CG journals could be devoted to such meetings. At the same time, the potential users of CG'ers' output must be kept informed of new developments. A series of "CG gems" books patterned after the computer graphics gems might be useful. Or at the least the graphics gems books, which already include significant amounts of geometry, should be given greater attention (and perhaps receive more contributions) by CG'ers. The current CG newsletters and mailing lists might spawn special wide-distribution issues (announcing newly released software or recent developments of interest to practitioners).

3. EXPERIMENTATION: To feed the CG pipeline with usable results, experimental work must be integrated within mainstream CG research. Actually, it is one of the great assets of CG that it lends itself so naturally to experimental work. Regrettably, that asset has been grossly neglected. The standard programming cycle (design, code, debug, test and benchmark) is painfully slow, so the number of iterations tends to be small. The field of

optimization has shown that practical algorithmic innovations tend to require repeated iterations through the programming cycle. Because this does not happen often enough in computational geometry, algorithmic innovations tend to be mostly of a theoretical nature. The unusual slowness of the implementation cycle is due to the complexity of geometric data types and the lack of adequate software tools (code libraries, visual debuggers, dataset catalogs, etc.).

Quality experimental research (as practiced in, say, biology or physics) must satisfy highly demanding criteria. Any geometric algorithm that is claimed to be the "method of choice" should be not only implemented and tested, but benchmarked against its competitors. Claims of strengths should be backed up by credible evidence and weaknesses should be identified. None of this is possible as long as every piece of code must be written from scratch, as long as every test input must be produced by hand, and as long as every debugging, visualizing, and measuring tool must be hand-crafted. There is a need for a large-scale effort in building software tools for computational-geometric experimentation. Standards should be set by which to judge the quality of experimental work.

To allow for benchmarking, representative data sets should be collected and archived. Input data should enable effective robustness and efficiency testing. Raw data should be included as well as highly structured or datatyped data.

One might ask: experimental computational geometry in applied areas has been alive and well for years. Why do we need to create anew something which already exists? The answer is that in many cases experimental work has been so closely tied to applications that more general pursuits might have been overlooked. An analogy would be that although the oil, food, and pharmaceutical industries each pursue their own brand of experimental chemical engineering tailored to their needs, few chemical engineers would deny the value of unfettered experimention. The same is true in computational geometry. Unfettered geometric experimentation will complement (not supplant) ongoing experimental work in applied areas.

4. Reward structure. Conferences, workshops and journals should be receptive (as some already are) to experimental work and software building. One possible suggestion is to run an annual workshop devoted solely to experimentation in CG, featuring geometric results as well as software tools. Researchers in relevant applied fields with a tradition of experimentation (like drug design) could be invited to share their experience with CG. For an experimental culture to take hold, it is essential that quality experimental work should be rewarded through the standard channels (journals, conferences, hiring, promotion, grant awards, etc.) For this to happen, experimental research must be judged and evaluated according to recognized standards. Of course, this is a bit of a chicken-and-egg question, and a certain amount of bootstrapping might be necessary. It probably means that quality standards should be made flexible enough at first to reflect the relative immaturity of CG experimentation. Within a few years, however, firm standards should be put into place.

   Together with an experimental culture, a software systems culture must be encouraged to grow within CG. Building novel geometric software that transforms the practice of geometric computing should be considered on par with proving a theorem that changes the mathematical landscape of the field. The reward structure must be adjusted, and

CG must learn how to judge non-theoretical research. This is not to say that one should systematically reward any attempt at writing code. On the contrary, yardsticks for distinguishing bad from good code should be introduced, and the standards should be just as rigorous as they are for evaluating mathematical work. Promotion and hiring decisions should reflect these cultural changes. Again, none of these changes can be decreed. A new way of thinking must first take hold.

Finally, CG needs not only to open up to experimental and software-building work but also to rethink its approach towards theoretical research. Several fundamental theoretical questions remain open in geometric optimization, combinatorial geometry, geometric primitives, geometric searching, etc. [31]. But the list of open problems whose centrality is so compelling as to draw a consensus –even within the CG community– is not nearly as long as one might think. We believe that many of the more interesting theoretical questions have not been formulated yet. These questions will surface as firmer bridges between CG and applied areas are created. For example, fundamental questions in computational topology might arise from geometric work in biology or astrophysics. Many of the classical computational-geometric concepts, such as convex hulls and Voronoi diagrams, arose from exposure to the natural sciences. Because CG is not a "foundational" science but a part of applied mathematics, it must draw its main inspiration from the physical world that it tries to model.

While CG must look outside for new frontiers to conquer, it must also become more critical of its current theoretical research. Problems whose only merit is to be open should probably be left as such. There are so many more open problems than what CG can ever hope to solve that it should focus on problems of identifiable importance, be they theoretical or practical.

3 Computer Graphics and Imaging

Rendering and image synthesis are two areas replete with geometric problems, some of which create significant computational bottlenecks. We discuss some of the problems encountered in rendering and we briefly report on a success story in image synthesis, where computational geometry has had a direct successful impact.

Radiosity. Graphics can be divided into two main sub-areas: *Modeling* concerns itself with the description of surfaces and their properties, light, and the media in which these are embedded, while *simulation* applies these computational models to predict behavior. One of the most important types of simulation is *rendering*, in which a description of scene and light sources is processed to generate synthetic imagery corresponding to one or a sequence of specified viewpoints. Rendering affords a rich source of computational-geometric problems. Typically, rendering a scene involves a *visibility* operation, in which the surfaces and/or surface fragments visible to the synthetic eyepoint are identified, followed by a *shading* operation, in which color values, derived from radiometric quantities, are computed at sample points on each visible surface. Computer graphics therefore combines the notion of combinatorial structures with some notion of each element's weight or importance [44].

Rendering is one of the most computationally intensive tasks in computer graphics. For example, some state-of-the-art rendering techniques use radiosity methods to compute global illumination distributions in the scene [37]. Such methods recursively divide the surfaces in the environment into smaller patches, then reduce the rendering equation to a linear system based on the pairwise interaction between these patches. At the inner loop of these methods is the computation of *form factors*. Informally, a form factor is defined for a pair of surface patches as the average solid angle spanned by one patch and visible from points of the other through the obstacles in the scene. Computing visibility in this manner is a well-studied problem in computational geometry, but one for which efficient practical solutions have yet to be found. Fast, practical methods for computing form factors could have a significant impact in rendering. Efficient approximation methods should also be sought [117]. Current methods use quadtrees or k-D trees to divide each patch into a hierarchy of patches at different resolutions. Interactions between surfaces occur at different resolutions determined by the ratio of their sizes to the distance separating them, and by some estimate of the energy flowing between them.

For the sake of concreteness, here are some specific problems. Note that in any real graphics application, a successful algorithm should be output-sensitive and incremental (as, for example, fly-throughs in complex building environments [2, 54, 55, 137]).

Given a scene of n convex polygons (or balls) in general position:

1. Determine the scene visible from a synthetic eyepoint. If z-buffer hardware is available, it might be advantageous to identify a superset of the surfaces visible from the eyepoint.

2. Given a frame rendering cost model of the form $c_1 V_k + c_2 A_k$, where c_1, c_2 are technology-dependent constants, V_k is the number of vertices of the k^{th} polygon, and A_k is its screen area, identify a superset of polygons visible from an eyepoint so as to minimize the total frame rendering cost. (This is a fairly good model of the cost to render a polygon using standard computer graphics hardware, eg, an SGI [55].)

3. Given a surface fragment with non-zero area, what (portions of) other surfaces are visible from this fragment? (This is the "area-to-area" visibility problem, which is much harder than the standard "point" visibility problem.)

4. Identify all pairs of intervisible polygons, or a tight superset. For each intervisible pair A, B, identify the "blockers" for the pair, ie, the set of polygons that block some stabline from A to B. (These problems arise in global illumination computations, which attempt to find a radiation distribution at equilibrium among the scene surfaces.)

Note that some of these problems call for hybrid solutions that mix object-space criteria (such as minimizing the combinatorial complexity of the rendered objects) image-space considerations, (such as expending more computation processing objects that occupy large screen areas), or other weighting factors (such as ignoring energy sources that have no discernible effect on the computed solution). It would be highly desirable to have algorithms that are parametrized by "knobs" that could be adjusted manually (or automatically) to reflect the particular technology in use (like the constants c_1, c_2 above).

Existing methods are very slow because they seek to produce exact solutions. Corresponding lower bounds indicate that dramatic speed-ups computing exact solutions are unlikely. Limited error tolerance is often acceptable in practice, however, and efficient approximation methods should be sought. In practice, current techniques are often based on quadtrees, octrees, or BSP trees, all elementary data structures that have been around for decades. There has been considerable progress in the theory of space partitioning recently. Can some of the new ideas be used or extended to provide simple, fast algorithms for visibility computation?

A Success Story in Image Synthesis. Thomson Training and Simulation (a subsidiary of Thomson) has developed a new generation of machines for image synthesis. The approach followed departs drastically from its predecessors. Instead of relying on massive, dedicated hardware, the new machines are mostly software-driven and use standard processors (for example, the i860). This change allowed the flexibility to provide new functionalities, such as inserting moving objects or supporting different types of animation. Expectedly, this return to software dependency entailed redesigning and reimplementing a new generation of data structures and algorithms, many of which fell squarely within computational geometry.

A new "polygon engine" was built to allow the user to view collections of thousands of nonconvex polygons with holes at interactive rates. In addition, a fully dynamic algorithm for constructing Delaunay triangulations was developed, which runs in real-time on the new machine. Many of the underlying ideas were borrowed from the recent work on randomized algorithms in computational geometry [36, 108]. In particular, the algorithm uses a hierarchical representation, the so-called Delaunay tree [19], to construct a terrain in real-time and automatically adapt the database to the specifications of the graphics device as well as to the distance of the observer from the terrain.

4 Shape Reconstruction

In many application domains, it is necessary to reconstruct 3D models of objects from 2D cross-sections: medical imaging, microscopy, geology, and aerospace manufacturing are all heavy users of shape reconstruction [107, 131]. Two main approaches have been developed. The first is inspired by image processing techniques and treats the data as a 3D image obtained by piling up the images of the cross-sections. This approach is volume- or voxel-based; the interpolation is usually done in the vicinity of each voxel, or based on a pillar of voxels of all slices with the same (x, y) coordinates [94].

The second approach is more boundary oriented; it attempts to construct a polyhedral model of the object that interpolates the boundaries of the cross-sections. It considers slice entities (contours) and, by considering each pair of successive slices in turn, it then concatenates all the layer interpolations. Hybrid methods have also been tried: for example, one might fit a surface to an unstructured cloud of points in 3D [17].

The first approach has been dominant in the past decade, despite the severe limitations caused by its storage requirements. At UCLA Medical Center, a human skull was reconstructed in this fashion: it involved 800,000 triangles, which was small enough to allow visualization on a computer screen but too large to be manufactured. Efficient geometric solutions began to emerge on the practical scene only in the late eighties.

With few exceptions, all such methods make use of recent data structuring techniques developed by computational geometers. For example, a Voronoi-diagram based method was developed at INRIA (France) [16, 18]. It involves computing the Delaunay triangulation of pairs of successive slices, then projecting each pair element against the other, thus tetrahedralizing the space between consecutive slices.

The code was commercialized by two companies. One (NOESIS) produces general-purpose software for medical imaging and industrial vision; the other (CRIL) maintains a large library for processing 3D medical images (into which the reconstruction algorithm was included). A non-commercial version can be accessed via `ftp` and has been used at over 300 sites. The program has been used for research on neurons: the cover page of the Journal of Comparative Neurology (April 1994) shows a result obtained with the commercial version of the software.

Experimental results on reconstruction of craniofacial cartilage are reported in the Anatomical Record (1991). Other applications of the code are mentioned below:

- Clinical use in radiation therapy planning software (DKFZ German center of cancer research).
- Reconstruction of a shark kidney (Max Planck Institut für Molekulare Biologie).
- Finite-element method mesh for a dog heart (Purdue University).
- Heart movement model (Yale University).
- Brain from MRI (University of Colorado).
- Body surface potential mapping (Dalhousie University).
- Volume calculation of surfaces undergoing wear (Utah Supercomputing Institute).
- Bone reconstruction (Center for Medical Robotics & Computer Assisted Surgery, Carnegie Mellon University).
- Rapid prototyping from medical data (Higher Technical Institute, Cyprus).
- Mesh generation for finite-element method (University of Waikato, New Zealand).
- Tessellations of brain surfaces (Washington University).
- Reconstruction of MRI scans of arteries (University of Waterloo).

An alternative approach was recently pursued in Israel [14] (see this paper for an extensive review of the relevant literature). This technique is based on the idea of *geometric hashing*, which was developed for object recognition problems in computer vision. This software package has been tested intensively on a variety of complex solid reconstruction problems, in medical imaging and in the reconstruction of terrains from topographic elevation contour data. It was shown to be robust, applicable in fairly complex situations, and to produce 'intuitively correct' solutions.

Here are some specific computational geometry problems related to reconstruction from parallel cross sections:

1. Given two parallel slices, when is it guaranteed that there exists a non-self-intersecting polyhedral surface that interpolates between them? How does adding Steiner points (on the edges of the polygons) help?

2. Define a geometric measure that characterizes the "goodness" of the interpolation.

3. Extend any of the existing geometric (contour-based) methods to consider more than two successive slices at each step.

Of course, shape reconstruction does not necessarily proceed from parallel cross sections. Many problems related to geometric probing, object recognition, and pattern matching use various different models of shape, and each presents its own set of problems.

Finally, it is worth reiterating that medical applications motivate a wealth of interesting geometric reconstruction and analysis problems. For example, sometimes an organ needs to be reconstructed so that its volume can be estimated [39]. Rapid prototyping of medical models is an area of growth with an abundance of challenging computational-geometric problems [85]. Advances on these problems might have a significant impact in fields such as computer-aided surgery.

5 Computer Vision

Most problems in computer vision are inherently computational-geometric. Why they have not been tackled by computational geometers is a mystery. Perhaps the answer lies in the fact that the appropriate mathematical formulations of these problems are elusive. In computer vision, to find the proper level of abstraction in formalizing questions is often an integral part of the research effort. At the same time, it is one of those rare fields of computing (like speech recognition) where the computer always takes a back seat to the human brain: put simply, humans are better at vision than computers. A direct benefit is that humans are good benchmarkers and can effectively judge the quality of a program from its output. Two areas which have made considerable use of geometric algorithms are model-based recognition (or pattern matching) and the recovery of three-dimensional structure from two-dimensional images (in particular, stereopsis and structure from motion).

Model-based Recognition. In model-based recognition problems, a model of an object undergoes some geometric transformation that maps the model into a sensor coordinate system (say, an image plane or a cylindrical coordinate system from a 3D scanner). The development of efficient algorithms for identifying such transformations is central to many model-based recognition systems. In reconstruction problems such as stereopsis and structure from motion, the geometry of multiple projections of a scene provides constraints that enable efficient algorithms.

There are several approaches to the problem which explicitly rely on results from computational geometry. For instance, in certain approaches to recognition, an object is represented implicitly by a set of two-dimensional projections. When the objects are polyhedra it is useful to be able to bound the number of combinatorially distinct projections (those with different faces, edges or vertices visible). The set of combinatorially distinct views is referred to as the *aspect graph*. Combinatorially precise descriptions of the aspect graph and algorithms for computing it were developed using techniques from computational geometry [56].

There are a number of other approaches to model-based recognition which employ nontrivial geometric algorithms, and which often draw explicitly on results from computational geometry. The affine hashing method [87] uses a redundant representation of a set of points in order to locate that point set under an affine transformation, in the presence of extraneous data points. The underlying idea is to use each triple of points in the model as an affine basis, and rewrite the other model points in terms of each basis. In order to recognize an object, triples of image points are selected, and for each triple all remaining image points are expressed in terms of the basis. When a correct basis is found, this will result in affine invariant coordinates that are the same (up to sensing error) as one of the encodings of the model. In practice this algorithm has better running time than the hypothesize-and-test approach, which consists of considering each pair of model and image bases explicitly, as is done in the alignment method [80]. However, the sensitivity of the affine hashing approach to sensor errors is difficult to characterize.

Several researchers have developed recognition methods that explicitly account for sensor errors. These methods make considerable use of results on arrangements from computational geometry. Most of these methods represent each point or line segment in an "image" set as a polygonal region (say, the Minkowski sum of the image set with a box). The matching problem is then to search for transformations bringing each point or line segment of the model into such a polygonal region in the image. These problems can be structured as sweeping arrangements, using algorithms from computational geometry [12, 28].

A different approach to model-based matching problems involves the development of cost functions for measuring the difference between two sets of points and line segments under various transformations. Cost functions based on the Hausdorff distance have been investigated in both the computational geometry [1, 3, 78] and computer vision [79, 128] literatures. The applied methods developed in the vision community are provably good approximation schemes for solving the combinatorial problems that were originally investigated in the computational geometry community.

Image Representation. Quantized images are commonly represented as sets of pixels encoding color/brightness information in matrix form. An alternative model is based on contour lines: A contour representation allows for easy retrieval of the full image in bitmap form [9]. It has been used primarily for data compression of an image. The idea is to encode, for each level l, the boundaries of connected regions of pixels at levels greater than or equal to l. It is easy to reconstruct an original image from those boundaries. There exist output-sensitive algorithms for computing contour representations.

One problem is how to store such representations in a compact manner. In practice one seldom needs the entire contour representation. Typical use is in the form of a query asking for the contours matching a given gray level. Current data structuring techniques should be called upon to provide efficient solutions. So far, they have not. Attempts to remedy this are underway.

Here is a typical problem encountered with this kind of representation. Suppose that we wish to erase wrinkles around the eyes of a person in a digitized picture given in contour representation. Because wrinkles might intersect contour lines, these might become disconnected after removal of the wrinkles. To reconnect them is not so easy. Dynamic programming might be a natural approach for this problem. In fact, it has been tried successfully in several experiments. Dynamic programming tends to be costly, however, and faster heuristics should be investigated.

Another problem ripe for a computational-geometric attack is *resolution enhancement*. Suppose we have an image scanner with 6-bit resolution for each of the three colors (RGB), and we wish to increase the resolution to, say, 8 bits. The naive approach, which consists of adding the gray levels of four pictures of the same object, has immediate flaws. Of the numerous solutions proposed in the computer vision literature, most suffer the drawback of causing blurring. As it turns out, the problem has a natural formulation in terms of weighted Voronoi diagrams, a well-studied construction in computational geometry.

Since such a Voronoi diagram is hard to compute especially in the presence of high degeneracy, a different approach might be preferable. An important observation here is that the objects are not continuous but discrete. This is the main difference with interpolation of contour lines in geographic information systems. Indeed, suppose that we want to improve the resolution on an intensity level k by a factor of M. This means partitioning the pixels at intensity level k into M different finer levels. Let S denote the set of all pixels with intensity level k, and B_k (resp. B_{k+1}) denote the set of all contour edges between pixels with intensity levels lower (resp. higher) than k. For each pixel in S we may compute the Euclidean distance to the nearest boundary edges in B_k and B_{k+1}. Using the ratio between these distances we can thus classify those pixels into M finer levels. Without getting into details, it is apparent that several variants of this heuristic can be designed. Efficient implementations and classifications of these heuristics would be very useful.

Image Segmentation. A central problem, called *segmentation*, is to distinguish objects from background [10]. For intensity images (ie, those represented by point-wise intensity levels) four popular approaches are: threshold techniques, edge-based methods, region-based techniques, and connectivity-preserving relaxation methods.

Threshold techniques, which make decisions based on local pixel information, are effective when the intensity levels of the objects fall squarely outside the range of levels in the background. Because spatial information is ignored, however, blurred region boundaries can create havoc.

Edge-based methods center around contour detection: their weakness in connecting together broken contour lines make them, too, prone to failure in the presence of blurring.

A region-based method usually proceeds as follows: the image is partitioned into connected regions by grouping neighboring pixels of similar intensity levels. Adjacent regions are then merged under some criterion involving perhaps homogeneity or sharpness of region boundaries. Overstringent criteria create fragmentation; lenient ones overlook blurred boundaries and overmerge. Hybrid techniques using a mix of the methods above are also popular.

A connectivity-preserving relaxation-based segmentation method, usually referred to as the *active contour model*, was proposed recently. The main idea is to start with some initial boundary shape represented in the form of spline curves, and iteratively modify it by applying various shrink/expansion operations according to some energy function. Although the energy-minimizing model is not new, coupling it with the maintenance of an "elastic" contour model gives it an interesting new twist. As usual with such methods, getting trapped into a local minimum is a risk against which one must guard; this is no easy task.

In contrast to the heuristic nature of these approaches, computational geometry suggests a more algorithmic tack. One would first formalize a mathematical criterion for the "goodness" of a given segmentation. This would allow us to formulate the segmentation problem as an optimization problem under certain geometric constraints.

The objective function that one would seek to optimize is the *interclass variance* that is used in discriminant analysis. Experimentation seems to suggest that this approach might be very promising. Formally, the criterion is described as follows. Imagine that an image is to be partitioned into two connected sets S_0 and S_1. The average intensity levels of the sets are denoted by μ_0 and μ_1, respectively. Then, the objective is to minimize the potential function

$$V(S_0, S_1) = |S_0||S_1|(\mu_0 - \mu_1)^2.$$

Computational geometry provides several tools (for instance, hand probing, Monge matrix searching) that lead to efficient solutions for optimal segmentation, as long as one is only interested in separation of an object defined by monotone chains. Without such assumptions the problem is NP-hard, and efficient heuristics remain to be found.

Discriminant analysis has also been applied successfully to the problem of transforming an intensity image into a binary black&white picture. This line of research relates directly to what has long been an active line of research in computational geometry, point clustering: Partition a set of points in $\mathbf{R}^d$ into k clusters so that some inter-cluster criterion is minimized.

6 Geographical Information Systems

Geographic Information Systems (GIS) [89] are increasingly being cited as a motivating application for computational geometry research. According to Unit 1 of the NGCGIA Core

Curriculum [63], "a GIS can be seen as a system of hardware, software and procedures designed to support the capture, management, manipulation, analysis, modeling and display of spatially-referenced data for solving complex planning and management problems."

The key phrase is "spatially-referenced data." A database system that combines data sets of different types from different sources about different features/objects is likely to find that converting data sets into a common form dominates the processing efforts, when it is possible at all. Data that can be assigned a spatial location, however, can be combined by simply displaying data sets simultaneously on a map or computer screen. Add a mechanism to select which data sets to display, and you have a rudimentary GIS.

Many systems have been developed in natural resource management applications—either in connection with remote sensing or from computer-aided design tools—and some have been highly successful in making up-to-date information available for management decisions in a manner that was not possible using a large inventory of paper maps. GIS use in government and marketing applications continues to expand.

Of course, a rudimentary GIS leads to the complaint [81]: "A GIS is basically a tool that mines data and displays it. It doesn't clean it up, or maintain it, and seldom even looks to see if it's reasonable." Computational geometers see their potential contribution here: in providing the algorithms and data structures to support analysis of geometric data. A forthcoming special issue of *Algorithmica*, editted by J. Snoeyink, will focus on the interface between computational geometry and GIS.

6.1 Snapshots of the parallel history of GIS and CG

While computational geometers might like to see GIS as an instance of the practical implementation of computational geometry, this is not historically accurate. The GIS and CG communities have, for the most part, developed their understanding of geometric computation independently. In some key instances ("topological structure" for storing planar subdivisions, and "TINs"—triangulated terrains, interpolation properties of the Voronoi and Delaunay), development in GIS has preceded that in CG.

Computer cartography emerged in the 1960s. Many basic concepts (map layers, topological structure, TINs) can be traced back to work done in the The Land Inventory branch of the Canadian government or the Harvard Lab for Computer Graphics and Spatial Analysis. ESRI's ARC/INFO, which is the most popular commercial GIS system, got its start with Harvard Lab technology.

Interaction between CG and GIS has increased in recent years. CG papers at the biennial International Symposium on Spatial Data Handling have been generally well received. ESRI, Caliper Corp., and other companies have been hiring from the computational geometry community and taking in research results (including, for example, nearest-neighbor properties of Delaunay triangulations, and the use of Voronoi diagrams of line segments to assist in "buffering" (offsetting) operations). A common complaint has been that publication delays and the lack of textbooks in CG hampers the adoption of research results by other fields such as GIS.

6.2 Data Model and Data Structure Debates

Which data model is best for the spatial information stored in a GIS? This question leads to several recurring debates that consider various parameterizations of the spaces of possible data models and data structures. We list three examples: the long-running raster/vector debate is limited by GIS implementation technology, the tiled/seamless debate is concerned with the practical and conceptual views of a large data set, and the object/layer/field debate appeals to human cognition of space.

Raster/Vector. Whether data should be stored and processed in raster or vector form has historically been a source of much debate. The debate is perhaps dying down slightly because most systems, at least from a user's perspective, have some provision for handling both. It is still true that not all operations are supported in both—typical systems compute buffer zones around vector features by converting to the raster domain and back.

Raster has many advantages including simplicity of concepts and algorithms, and hardware support. Remotely-sensed data, displayed output, and many digital elevation maps (DEMs) are in raster form. It is good for data that should be sampled evenly. Rasters are huge, do not scale well, and introduce aliasing effects.

Vector data is natural for linear features such as roads and boundaries, and mandatory for accuracy in instances of land survey. The speed of vector computation is most often criticized; geometers should rejoice in this as an opportunity to provide significant improvements. We should remember, however, that these improvements will need to be realizable for "typical data" and will need to provide some feature that raster processing does not provide. Paper descriptions of worst-case optimal algorithms do not constitute significant improvement in a community that wants implementations of solutions.

Tiled/Seamless. In vector databases as well as raster databases, the quantity of data becomes overwhelming. Practical considerations dictate that the data must be partitioned and considered in smaller chunks. People who converted to GIS and computer cartography systems from paper maps were accustomed to dealing with map sheets, quadrants, and tiles. Other users, however, demand a seamless structure—maybe even one that allows a wide range of levels of detail. Debates rage about what data structure is best for the partitioning: uniform grid, quadtrees, R-trees, binary space partitions, BANG files.

Object/Layer/Field. In the last few years some members of the GIS community have argued that the raster/vector debate obscures more essential issues about how people think about and manipulate space. Suppose that you want to describe the agricultural activity in a certain region. There are several spatially-referenced variables that may be of interest: owner, crop grown, soil type, elevation, rainfall, etc.

One might speak of farms in the region as geometric objects that have owner attributes. Fields could be objects with a *crop grown* attribute, and could inherit an owner from the farms that contain them. This is an example where an object-oriented approach to the data is natural: There is clear separation of objects, their representation can be encapsulated, and they form a hierarchy in which representations and attributes can be inherited.

When one adds *soil type*, however, one may find that a field can contain several soil types or that a soil type may encompass several fields. Subdividing fields into objects that can be given a single soil type attribute may or may not be a good idea—it depends on the importance of the soil type variable. It seems more natural to start a new layer. This in turn creates problems where the same object may have different representations in different layers. A stream, for example, may be represented as a change of ownership and a change in soil type.

It might be natural to think of soil type, elevation, and rainfall as functions defined over the region—cropland as a "vector field." Defining data structures to support continuous data models is difficult because data is usually obtained by discrete sampling. Even so, the amount of data can be unwieldy and data can be expensive to collect. The traditional soil map quantizes the domain into subjective categories (clay, clay with gravel, sandy, very sandy, etc.) and creates a raster or partitions the region into cells that are assigned one of these soil types to obtain an object-oriented, vector-based map. An implementation of a vector field idea might represent the proportion of each possible component (clay, gravel, sand, etc.) as elements of a vector. One then needs algorithms to quantize this vector space to present understandable output.

Arguments in this debate usually conclude by recommending raster or vector implementation so that the object/layer/field debate, too, becomes limited by current technology.

6.3 Geometric problems in GIS

Two things should be kept in mind when looking at geometric problems that arise in GIS. First, even though computational geometers are likely to see themselves as solvers of geometric problems, the concepts of space, shape, and geometric computation that develop can be a more important contribution than the solution of any particular problem. Second, the "geometric" is only one aspect of a GIS. In a survey for cartographers, Goodchild [61] lists four ways in which to view GIS; geometers may want to keep these views in mind when weighing the importance of problems motivated by GIS:

1. Automated Mapping: facilitating the production of standard maps,
2. Map Analysis: providing measurement and overlay tools that are cheaper than traditional methods,
3. Inventory: giving geographic access capabilities to existing governmental and corporate databases,
4. Spatial analysis and spatial decision support: enabling new uses for old data by giving users query and analysis tools.

Data Structure Tuning. To a GIS salesman, one of the most important features of their product is how fast it can sift through the data and put the layers that the user has selected onto the screen. Because users can select data based on spatial location, attributes, and level of detail, multidimensional access structures (B-trees, k-D trees, quadtrees, R-trees) are important and tuning these is a difficult task: van Oosterom's book on reactive data structures [139] is an example of what can be done.

Existing legacy data limits the applicability of new research. One cannot expect people to adopt a data structure if it involves expensive conversion. An emerging opportunity, however, is the partitioning of GIS tasks between server and clients. For various reasons (data management, resource needs, etc) it is natural to put spatial data on a large server and have clients request the regions and attributes for the problem that they are interested in. With this architecture, what information should be sent by the server to help clients build geometric structures for local analysis? If the input is line segments, then sending a trapezoidation helps a number of algorithms, but maintaining a trapezoidation on the server may increase the data structure size and involve data conversion. Sorting by first coordinate may be a less-costly improvement.

Generalization. Since almost all GIS systems allow a user to zoom in on data, there is no intrinsic scale associated with the computer representation of a map. Nevertheless, there is an associated level of detail which determines a range of scales at which the data can be displayed without being too crowded or too sparse. *Generalization* is the process whose goal is to extend this range [23]. Several geometric problems fall under the heading of generalization:

LINE SIMPLIFICATION: Approximate a polygonal line by one with less data (but hopefully the same information). This is has been the subject of much research in GIS, image processing, and computational geometry. Interesting problems remain for simplifying several lines (as in a contour map) or for demanding topological properties (e.g., forbidding self-intersection) from the result.

CLUSTERING: Separate geometric objects can be aggregated and represented symbolically as a polygon or as a point.

DISTORTION: Moving or changing the representation of geometric objects to enhance readability of the output. A railway that parallels a river may need to be displayed at lower resolution. A driver would prefer that a winding mountain road have some bends exaggerated on a map rather than being simplified to a line segment.

The more advanced generalization tasks are difficult to define a precise geometric optimization problems. Automating these tasks is often seen as an application for artificial intelligence techniques and "knowledge-based" heuristics. Computational geometry can offer not only efficient implementation of lower-level primitives, but also structures such as Voronoi diagrams and constrained triangulations that provide a more continuous model of space.

Digitizing. Many maps are constructed by a labor-intensive digitization process. One of the exciting applications of computational geometry in GIS is an idea of Chris Gold [58]. Rather than force someone to carefully trace what may be a fuzzy boundary anyway (eg, the edge of a forest stand), use the digitizer to spray a bunch of points near the boundary, then use the Voronoi edges that are from bisectors of points in different regions to be the boundary. This speeds up digitization by a factor of 4, avoids problems with dangling or intersecting edges, and still gives reasonable accuracy because the Voronoi edges average between the points.

Building Topology. What the GIS community calls "building topology" is to establish the ordering information for a planar subdivision—determining the edges (arcs) around faces (polygons) and vertices (nodes). This task is complicated by the fact that the input data may be huge, imprecise, and occasionally erroneous; it is usually seen as an expensive batch computation process. Gold [57] has further advocating Voronoi diagrams as way of maintaining topology as a map is constructed. Because of the increase in data and computational effort, this will work best with smaller maps. What is the best way to combine these ideas with tiling ideas to operate on large amounts of data?

Polygon Overlay and Update. The spatial data in GIS's from ARC/INFO [50] down to the Digital Chart of the World [51] is organized into layers, each of which is a subdivision of the plane with explicitly stored neighbor relations. Custom maps are produced by selecting layers of interest and overlaying them to produce new maps. This "overlay" process is the primary tool for map analysis.

As a theoretical geometric problem, computing the regions defined by a set of segments or pixels is perhaps not so difficult. But as an engineering problem, tuning algorithms is a challenge. When one considers the errors and different levels of generalization in the input, the task becomes more interesting. If the same feature, say, a stream bed, appears in more than one map, an overlay will likely have many spurious polygons in the vicinity of the stream bed. It would then be desirable to remove one copy from the overlay. Even better would be to average the two maps in some principled manner to improve overall accuracy.

A special form of the general polygon matching is to match two maps and try to determine what has changed from one to the other. This is important for observing historical change, checking map completeness, rectifying common boundaries, or using a higher resolution map to improve a lower resolution map. Again, generalization and error make this an intriguing problem.

Label placement. Label placement is a source of many geometric optimization problems. Even when labels are to be placed in fixed positions relative to point features, these are typically NP-hard. Since maps do need labels, heuristics and efficient algorithms for identifying constraints and possible positions are needed.

Spatial Accuracy. Current GIS data structures do not support error bounds. This is one of the major weaknesses of analysis carried out in a GIS—one does not know how to asses the quality of the result. One of the major challenges in GIS is a computationally efficient theory of spatial error. There has been much discussion on the topic. In 1966, Perkal [118], for example, proposed fattening boundaries by convolving them with a disk of radius ϵ; these are usually called buffer zones [146], and queries with buffer zones are an important type of overlay. See the survey [62]. Computation in floating point is one source of error. GIS data sets are riddled with degeneracies, so robust computation can be important.

Spatial Analysis. Thus far, GIS's have offered only rudimentary spatial analysis capabilities [60], such as polygon overlay and area computation. More interesting queries require more interesting data structures: e.g., Okabe et al. [112] list 35 GIS queries that can be answered using different types of Voronoi diagram.

Terrain analysis. Research on representation and computation of terrains has been an active area.

- TIN: Triangulated surfaces were suggested in GIS in 1978, and dubbed TINs or Triangulated Irregular Networks [119]. Delaunay triangulations are often suggested (and used) as good TINs. Other (data-dependent) triangulations may sometimes be more appropriate.

- Simplification: Kumler's monograph [86] compares elevation accuracy of TINs to DEMs and finds that, byte for byte, Digital Elevation Models are better. The simplification methods he uses, however, are probably not the best possible. (Also, error measure that reflect curvature or symmetric volume difference should be considered.) Simplifications with some sort of hierarchical structure are especially useful. See Heckbert and Garland's survey [68] for many citations to geometry and graphics papers on surface simplification methods.

- DRAINAGE: For many natural resource management tasks, the watershed—or region that drains to the same point—is the natural management unit. Much of the work on computing drainage networks is raster based; some geometers are looking into preprocessing TINs for drainage queries.

- VIEWSHEDS: Some forestry operations are now required to preserve the visual impact of the forest. Thus, "viewshed" algorithms become necessary. Location problems for microwave and cell phone relays also involve line-of-sight constraints imposed by the terrain.

- FLYBY: Fast graphics engines give the capability to fly-by data models to get a better feeling for the actual terrain (eg, systems such as PCI's EASY/PACE).

Adding Dimensions 3 and 4. Adding the spatial and temporal dimensions to the flat maps in a GIS is a subject of current research [125]. None of the standard GIS's support functions of three variables. In such systems, geological cross sections or floors of buildings must be treated as flat, horizontal data sets which offer no geological or architectural interconnections.

Time creates interesting geometric problems. In an agricultural region, the land will be partitioned into different fields in different years, and land ownership or use changes. Some of the dynamic data structures in computational geometry may have a role to play in allowing efficient storage, query, update, and historical summary of changing land uses.

6.4 Resources in GIS

The most relevant journals are the *International Journal of Geographic Information Systems* (IJGIS) and *Cartography and Geographic Information Systems* (CGIS). Other journals include *American Cartographer*, *CVGIP: Graphical Models and Image Processing*, *Cartographica*, *Geographical Analysis*, *Journal of Remote Sensing*. Duane Marble maintains a ftp-able bibliography on GIS (in refer and mac, but not in bibtex format) on `bastet.sbs.ohio-state.edu`

Of course, the Web is a good source of information and data. The NCGIA activities occur at the University of California, Santa Barbara, `http://www.ncgia.ucsb.edu/`, in Buffalo, NY `http://www.geog.buffalo.edu/`, and in Maine `http://ncgia.umesve.maine.edu/`. Maine has some technical reports on line, including a 1992 update of the research agenda, 92–7.ps. BC environment has the NCGIA Core Curriculum on its Web `http://www.env.gov.bc.ca/gis`.

Xerox Parc has a map program `http://pubweb.parc.xerox.com/map`. NAISMap is a GIS on the Web `http://www-nais.ccm.emr.ca/naismap/naismap.html`. GRASS (Geographic Resources Analysis Support System) is a public domain GIS developed by the US Army Corps of Engineers `http://www.cecer.army.mil/grass/GRASS.main.html`. GRASSlinks is a Web interface `http://www.regis.berkeley.edu/grasslinks/index.html`. ORES has an extensive list of data sites and other information. See for example,

`http://www.csn.net/~bthoen/ores/gis/index.html`

For GIS FAQ, see

`http://www.census.gov/geo/gis/faq-index.html`

Finally, for ESRI (ARC/Info), see: `http://www.esri.com/`.

7 Mesh Generation

A mesh is a discretization of a geometric domain, e.g., the air around a wing, into small simple shapes called elements [13, 15, 67, 75, 93, 130]. A structured mesh is usually a warped grid of boxes, while an unstructured mesh is typically a triangulation. Some advantages of structured meshes that hold generally over most applications, are simplicity, availability of code, and suitability for multigrid and finite difference methods. On the other hand, unstructured meshes conform to the domain more easily and allow element sizes to vary more dramatically. Structured meshes are currently more popular, but unstructured are catching up, especially in the more academically-inclined community.

Some mesh generation goals vary with the application. For example, long skinny elements, aligned with flow, can be quite useful in computational fluid dynamics. Moving features, such as shock fronts and vortices, require changing meshes.

Compared to GIS, mesh generation is still something of a cottage industry. Research and development groups in finite-element methods tend to write their mesh generators. There are a few public-domain codes (e.g., PLTMG, written by Randy Bank, and GEOMPACK, written by Barry Joe), and some commercially available code (e.g., ICEM CFD), but all large manufacturing companies use their own codes. There is no publicly available package that is adequate for generating 3D meshes for computational fluid dynamics. Aerospace engineering experts even go as far as admitting the lack of adequate code for generating 2D meshes for viscous flows. There lies an exciting window of opportunity for computational geometry.

Below are a list of open questions in mesh generation. Some are outstanding theoretical problems; others represent serious computational bottlenecks in practice.

- Heuristics for Point-Set Triangulations: Theoretical analysis of existing heuristics remains to be done. In practice, the "diagonal-flipping" algorithm works quite well for 2D Delaunay triangulations, but there are no theorems of the form: most initial triangulations are only $O(n)$ flips away from Delaunay. Rivara refinement (split a simplex by adding the midpoint of its longest edge, then recurse on neighbors) works well in 3D, but there is no theorem analogous to Rivara's theorem in the plane, showing that angles stay bounded away from zero.

- Surface Representation: Investigate appropriate metrics to measure how well a mesh conforms with the surface boundary, and design heuristics for constructing good meshes. When approximating highly curved surfaces, it is important to ensure that edges of the surface triangulation are aligned with the directions of principal curvatures so as to avoid "scalloping effects." One possibility is to make the surface triangulation consist mostly of right-angle triangles, with the two right sides aligned with the directions of principal curvature. It seems natural to require that lengths of the right sides should be inversely proportional to the surface curvature. A theoretical understanding of this question, and more generally, of the problem of fitting a mesh on a complex geometry, would be very useful in computational fluid dynamics.

- Volume Mesh Generation: As expected, small volume elements must be used in regions where rapid variations of the computed solution are expected. But the precise requirements vary from problem to problem. For example, for computations of the scattering of an electromagnetic wave, it is necessary to use a uniformly spaced mesh everywhere and ensure that the cell width is small compared with the wavelength of the incident radiation. For computations of fluid flow, particularly Navier-Stokes calculations at high Reynolds number, there will be extreme variations in cell size and/or aspect ratio.

The Delaunay triangulation is one of the most popular means of generating 3D meshes of tetrahedra. An effective approach is to insert a mesh node into an existing triangulation, and repeat this procedure until a sufficiently fine mesh has been achieved. Since the quality of the final volume triangulation depends on the positions at which the points are placed, it is natural to investigate point placement strategies. The two-dimensional case is relatively well understood. For example, placing each new mesh point at the circumcenter of a triangle with maximum circumradius leads to a triangulation whose minimum angle is at least 30^o and such that the ratio of maximum to minimum edge length is at most 2. Nothing similar is known in three dimensions. It is a major open problem whether point placement techniques can be used effectively to generate high quality 3D triangulations.

When dealing with viscous flow computations at high Reynolds number, extremely fine meshes are required close to the boundary surface. It is customary to generate meshes so that the point spacing normal to the boundary surface is very close while allowing a much larger spacing tangential to the surface. This may lead to poorly-shaped elements near boundary junctures. New point placement strategies are needed to ensure high-quality meshes in such extreme situations.

- Hexahedral Meshes: There are several geometric benefits to be gained from using meshes of hexahedra (ie, affinely transformed cubes) such as regularity, angle distribution, anisotropy. Some commercially available systems support such meshes. Unfortunately, the computational geometry of hexahedral mesh generation is so little understood that existing systems suffer major drawbacks. Is there a good algorithm for partitioning a solid into hexahedra that meet face-to-face? The outside-in, or advancing-front, approach starts with an initial boundary mesh and fills out the volume with hexahedra. Unfortunately, many starting configurations are unfillable. Conversely, the inside-out approach fills volume and produces boundary meshes as a side effect. This has the disadvantage of leaving us no choice in the boundary meshing. Intermediate strategies are yet to be investigated.

- Advancing front: Some tetrahedral mesh generators use the advancing-front approach as well. These generators typically place Steiner points in successive layers and then use the Delaunay triangulation as the mesh. The same difficulties arise: by meshing the boundary first this method commits to a level of detail too early in the process, and non-tetrahedralizable pockets can form when fronts collide.

- Mesh Generation and CAD Systems: Uniting mesh generation with CAD systems is much needed. The idea is to produce grids directly from the CAD geometry and at the same time retain independence of grid characteristics. Not only would this solve a compatibility issue, but it should also make the mesh generation task easier. A mesh generator has to "parse" the domain, which is hard to accomplish unless the grid generator is intimately tied to the systems used to represent the geometry. Some systems (e.g., ICEM CFD) have taken steps in that direction but much remains to be done.

- Mesh Partitioning: A mesh must often be broken up into pieces to allow for efficient parallelization or vectorization. In the latter case, for example, the number of elements in each group must be matched to the target architecture. Should processor architecture influence not just the partitioning of the mesh but also its generation?

- MULTIGRID FOR UNSTRUCTURED MESHES: Hierarchical triangulations, as in Kirkpatrick's point location algorithm, can be used to combine unstructured meshes and multigrid methods. Randomized analysis of the incremental algorithm shows that such hierarchies of small expected size form automatically for Delaunay triangulations. The hierarchy represents the history of the insertion process. The randomized result also hold in three and higher dimensions. Can similar results be obtained for triangulations other than Delaunay or the related regular triangulations?

- ROBUSTNESS: Robustness appears to be a serious issue in mesh generation. In general, the current state of robustness research in computational geometry leaves practitioners unsatisfied. In particular, many are wary of automatic degeneracy handling and prefer explicit treatment of it in their codes. (See Section 10.)

- AUTOMATIC REMESHING: A unified meshing system might be envisioned along the following lines: first, the domain is meshed, based on the geometry alone; then the differential equations are solved. Finally, remeshing occurs, based on an automatic error analysis. The last two steps can be repeated many times. The advantage of such an approach is that the final mesh is produced from an actual approximation of the solution and not an educated guess.

We close this section with a brief mention of alternative approaches to mesh generation. Meshes are used primarily to integrate partial differential equations over complicated domains. The base of numerical solution technologies is surprisingly narrow: finite-element methods, boundary-element methods, finite-difference methods, and monte-carlo methods. Are there alternatives to the methods listed above? Recent research suggests that some might lie in finding more clever representations of the underlying geometry which exploit the physics of the problem (in some cases, medial axes and Voronoi methods may come into play) and then mapping the physics onto the new representations of problem geometry. This results into a reformulation of the problem that is both geometric and numeric, as opposed to the purely numeric approaches currently in favor.

8 Robotics

Computational geometry has already had a noticeable impact in motion planning problems. Years ago the standard approach to motion planning followed one of two methodologies. The STRIPS planning formalism favored in the AI community was based on pre- and post-conditions in the propositional calculus. The lack of geometric grounding made it difficult to instantiate motion, grasping, or assembly actions and obtain guarantees of performance or correctness. In practice such approaches often showed limitations.

Techniques from control engineering were also attempted. The weakness there lay in their tendency to be local and thus to become trapped in local minima. Nevertheless, these methods have been popular in practice because of their simplicity, speed, and integration with the control layer. There has also been a flurry of interest in algorithms for non-holonomic motion planning, where motion constraints are not integrable. A textbook covering the theory of non-holonomic planning has recently appeared [109].

Path planning problems have been widely studied by computational geometers, with some spectacular theoretical successes. General motion planning problems were first shown to be solvable in polynomial time [132], then with a practical exponent (equal to the number of degrees of freedom) [26], and then with a tightened exponent in many cases with the aid of Davenport-Schinzel sequences [133]. For the case of industrial robots, the degrees of freedom range from four to six. Multi-robot systems and robot hands are much higher. AGVs (autonomous guided vehicles) have 3 degrees of freedom with non-holonomic constraints. These concrete problems are too large for the methods above to solve at this time. But even though asymptotic methods are rarely used in practical systems, many effective planners have been built using concepts from CG. Concepts such as generalized Voronoi diagrams or skeletons, critical orientations, line and plane sweep, and random sampling have been used in heuristic planners that are fast and robust. Latombe's text [88] provides a comprehensive overview.

A basic representational problem for robotics and CG more generally is that the world contains many curved objects. Curved 3D configuration spaces arise even when objects are polygonal and motion is restricted to the plane. In graphics, given today's rendering technology, it is reasonable to approximate curved surfaces at varying resolution by planar facets. However, polygonal approximations can lead to errors in mechanical analysis, for example when computing the stability of parts on a table under gravity. To what extent should CG deal with algebraic surface representations? The majority of CG work so far has abstracted away from the algebraic structure, and left the algebraic computations to a black box. There is certainly nothing intrinsic in the problems to make this be so. Indeed, "computational geometry" was not that long ago a field which studied algebraic computations on curves and surfaces. That field has since come to be called "geometric modeling", and it has maintained a close ties with practitioners with a shared interest in geometric CAD systems. If CG is to maintain its claim to the geometric computing turf, it should broaden its definition to include at least the original one.

In addition to path planning, robotics suggests a rich variety of geometric problems. A workshop was initiated in 1994 to encourage interaction between researchers from robotics and computational geometry [59]; a second workshop is currently being planned.

Industrial Robots. To cope with shrinking product life cycles, computational methods are needed to reduce the time required to configure assembly lines. One idea is to formally model a small "vocabulary" of parameterized modular components (e.g., robot arms, conveyor belts, flexible part feeders, modular fixtures, light beams, 2D vision systems) and develop CG-based algorithms to efficiently compile them into an assembly system for a given CAD product model [27]. Here are a few relevant problem areas:

1. Convex Decomposition: For motion planning, collision checking is currently most efficient for convex polyhedra, so we need better methods for convex decomposition of polyhedra. Algorithms should produce a near-optimal number of convex components. As for other research areas, fast, robust code is needed. Convex decomposition has many applications and is currently a bottleneck. See [33] for a survey of current methods and [32] for a practical approach to the problem.

2. Grasping and Fixturing: The goal here is to place contacts – fingers or fixture elements – so as to constrain object motion. This is one of the central problems in

robotics. A number of basic questions have been answered, such as the minimum number of contacts sufficient[2] to hold any nonrotational part in form closure (4 in 2D, 7 for polyhedra) [96, 84] but efficient algorithms are still needed. In modular fixturing, where fixture elements are constrained to a regular lattice, recent results suggest a number of open questions about the existence of solutions for classes of fixtures and parts [154, 142, 115].

When there is uncertainty in part pose or applied forces, minimizing the number of grasp points can be posed as a convex set covering problem. Recently, CG researchers have described efficient and probably practical algorithms for near-optimal grasps. This goes beyond the previous works which either do not consider optimality, or do not allow search over a large space of possible finger placements. If the CG methods pan out in practice, this would be a strong vindication of CG in robotics.

3. Approximation Algorithms for Grasp Circuit Planning: Consider picking up parts as they arrive on a conveyor belt: the gripper may have multiple suction cups that must be "loaded" during each pass: Given a robot gripper with k suction cups and a collection of n parts, rapidly approximate an optimal path for loading the suction cups and depositing the parts in a pallet. Although this problem can be reduced to TSP, recent approximation results were successfully adapted to this and variations where the points to be visited are moving with known velocity [29]. Approximation algorithms have rarely been applied to problems in industrial robotics but hold great promise.

4. Geometric Probing: CAD models are assumed by many algorithms but are often unavailable. Geometric probing can be used to generate such models. Probing hardware includes touch probes, light beams, scanline and raster cameras. Depending on the application and sensor, the probing strategy may compute convex hull, line hull or ray hull. See [134] for a review. More work is needed on online probing strategies that include models of probe and control uncertainty. The problem of probing to minimize error turns out to be dual to grasping, so the algorithms mentioned there are directly relevant.

5. Part Orienting: Algorithmic approaches to orienting ("feeding") parts for industrial assembly are needed. To be practical and cost effective, feeders must be able to rapidly and repeatedly orient parts at subsecond rates. Current feeders are designed by human trial and error. Complete algorithms are needed to take part geometry as input and generate feeding strategies or feeder hardware specifications as output. A complete algorithm for feeding 2D parts with algebraic boundaries is described in [124]. More work is needed to design feeders for 3d parts and to provide accurate mechanical simulation of collisions and statistical behavior of parts arriving in bulk.

6. Assembly Sequence Planning: The problem of disassembling a collection of parts has a strong geometric component. For local motions, a disassembly sequence can be found in polynomial time. The worst case running time is $O(n^4)$, but it is fast in practice [144]. More recently, good worst-case bounds have been given for an important special case: where the set of possible disassembly directions is fixed a priori (true for many robotic workcells). The latter methods use lower envelopes and Davenport-Schinzel sequences to efficient find collisions along the disassembly direction. The main limitation

[2]The number necessary was demonstrated in 1900.

so far is the inability to deal with complex motion during disassembly. e.g. a translation and twist motion would not be possible. Extensions of this type, plus the addition of other constraints on the sequence, will make this a rich area for future research.

Autonomous Robots. Autonomous robots have a far more difficult time than industrial robots. They must deal with time-varying environments containing unmodeled entities (humans) and with partial, inaccurate or outdated knowledge of their environment. While some researchers have nevertheless tackled the model-building problem for mobile robots (which leaves the robot with a classical motion planning problem on the map), others have sought simpler routes to navigation. These algorithms are typically online algorithms with limited state. In [95], robots navigating in the plane used local decision rules to provably reach the goal. These methods used performance measures derived from path length and were early examples of competitive ratios in path planning, [116].

Geometric models of uncertainty have been used, based on the existence of landmarks, which are uniquely recognizable features in the environment. In [90], a nearsighted robot with a bad compass wandered between "islands" of certainty near landmarks. In [77], a robot with better vision moved along corridors in space formed by aligning landmarks.

The argument is made below that the chief goals of autonomous robotics require a further understanding of geometric complexity. A central theme in this line of work is to determine what information is required to solve a task and how to direct a robot's actions to acquire that information. Another key question is to assess the capabilities of a robot in a given environment or class of environments.

These questions can be difficult. Structured environments, such as those found around industrial robots, contribute towards simplifying the robot's task because a great amount of information is implicitly encoded into both the environment and the robot's control program. One goal is to quantify the information encoded under the assumption that, say, the mechanics are quasi-static or the environment is not dynamic. Conversely, one might ask how much information must the control system or planner compute? Successful manipulation strategies often exploit properties of the (external) physical world (e.g., compliance) to reduce uncertainty and hence gain information. Often, such strategies exploit mechanical computation, in which the mechanics of the task circumscribes the possible outcomes of an action by dint of physical laws. Executing such strategies may require little or no computation; in contrast, planning or simulating these strategies may be computationally expensive. Since during execution one might witness very little "computation" in the conventional sense, traditional techniques from computer science have been difficult to apply in obtaining meaningful upper and lower bounds on the true task complexity. It is hoped that a theory can be developed to measure the sensitivity of plans to particular assumptions about the world, and to minimize those assumptions wherever possible.

Because geometry is the natural language for characterizing sensors, tasks, and the complexity of robotics operations, such an undertaking is inherently computational-geometric. More and more, robotics researchers are abandoning off-line models where a priori knowledge of the whole environment is assumed. For example, in the stereotypical off-line model, one might assume that the robot, on booting, reads a geometric model of the world from a disk and proceeds to plan. A preferred alternative now is to consider on-line paradigms where the robot investigates the world and incrementally builds data structures to the external environment. As time evolves, the task effectively forces the agent to move, sense, and update its model of the world. Certainly, from an on-line viewpoint, off-line questions such as "what is the complexity of plan construction for a known environment, given an *a priori* world model?" might often appear secondary, if not downright artificial.

While it is profitable to explore on-line paradigms for autonomous agents and sensorimotor systems, the framework remains to be extended in certain crucial directions. In particular, sensing has never been carefully considered or modeled in the on-line paradigm. The chief difficulty is the absence of tools for measuring the complexity of a robot. Just as complexity classes are characterized by the amount of resources available, one would hope to have relations between the computational power of a robot and its hardware. How much power is gained by adding an extra sensor?

Such questions have been overlooked. As a result, robot builders are able to make unverifiable claims about robot performance and resource consumption. What is missing from autonomous systems is both an accounting method to assess the capabilities of a robot and tools for arguing correctness and completeness for on-line autonomous robot algorithms. The inability to compare systems leads to redundant work and inhibits progress. Computational geometry has established a clear progress metric, while emphasizing completeness and correctness for control algorithms. But the systems for which it has explanatory power are far from real embedded systems. Yet such systems are probably best analyzed and characterized by using geometric methods. This is a direction in which computational geometry could have a significant impact.

Here is a concrete example of an on-line navigation. Some robots are purely reactive and plan their moves without retaining knowledge of their past interactions with the environment. Others build a map, ie, a geometric model of the world it encounters. The former are too weak, the latter are too slow. What are good intermediate strategies? Navigating in a graph with bounded storage is a well-studied problem in computer science. Interesting work in on-line navigation with uncertainty has already been done. To pursue this further and experiment with physical robots would be extremely useful.

The "minimalist" rationale of complexity theory (ie, using the minimum resources to achieve a given goal) has parallels in robotics. Resources are more diverse: they are measured by the number of sensors, sensor probes, actuators, agents, as well as by the amount of computation time and space and communication. In robotics, minimizing the use of resources has become an increasing preoccupation. It has been shown that walking/running machines could be built without static stability, that dextrous manipulation could be performed without sensing. Biped, kneed walker have been built without sensors, computers, or actuators.

Computational geometry is uniquely poised to demonstrate and prove minimalist solutions to robotics problems. Key questions include the following:

- What are minimal configurations (resource-wise) for on-line navigation, parts-feeding, grasping, singulation, recognition, or manipulation.

- Are there sensorless geometric filters[3] for classes of objects? What on-line filters uses a minimum amount of sensing? For example, with a distributed actuator system such as a MEMS array device, one can create a programmable vector-field for parts manipulation: First one centers a part using a radial, inward-squeezing pattern, which centers the part at the origin. Next, one makes the array push in the left direction within a narrow band around the x-axis. The rest of the array pushes in the opposite. This strategy filters small parts to the left and big parts to the right.

[3]A geometric filter is a device which filters parts, ie, sends them to different spatial destinations, based on some geometric characteristic: for example, circular parts to the left, square parts to the right.

Micro-Electromechanical Systems. A wide variety of micromechanical structures (devices typically in the μm range) have been built recently by using processing techniques known from VLSI industry. Various microsensors and microactuators have been shown to perform successfully. For example, a single-chip air-bag sensor is commercially available; video projections using an integrated, monolithic mirror array have been demonstrated recently. More difficult is the fabrication of devices that can interact and actively change their environment. Problems arise from (i) unknown material properties and the lack of adequate models for mechanisms at very small scales, (ii) the limited range of motion and force that can be generated with microactuators, (iii) the lack of sufficient sensor information with regard to manipulation tasks, and (iv) design limitations and geometric tolerances due to the fabrication process.

The design, analysis, and control of micro-electromechanical systems (MEMS) are inherently geometric in nature. While much is known about building MEMS, little is understood about how to use them to manipulate objects. More precisely, manipulation of solid objects subject to hard or intermittent contact is entirely open. The state of the art is somewhat similar to that of robotics before geometric planning algorithms. Development of control strategies for manipulation by MEMS is a key bottleneck. MEMS actuators are tiny; hence one is forced to consider parallel manipulation strategies by teams or arrays consisting of a large aggregate of micro-actuators. It is believed that based on work on sensorless and near-sensorless manipulation, one can develop geometric theories of manipulation and control for microactuator arrays. Can one develop sensorless (open-loop) manipulation algorithms for arrays of MEM actuators? Can these strategies be time-invariant, or is a clock necessary? Can MEMS implement geometric filters? Can MEMS arrays orient or pose parts uniquely, without sensing? Can MEMS arrays implement assembly algorithms?

9 Manufacturing and Product Design

Computational geometry arises at all levels of manufacturing, from design and simulation, to process planning, product inspection and testing. A number of recent resources have emphasized the prominence of geometry in manufacturing [72, 82, 140, 141, 145, 151]. A forthcoming special issue of *Algorithmica* (J. Mitchell, guest editor) focuses on applications of computational geometry to manufacturing.

Solid Modeling. A first issue to tackle is solid modeling, by which is meant the computer representation and manipulation of 3D shapes. About 20 years ago, the "geometric coverage" was limited to solids bounded by surfaces from natural quadrics and planes, ie, built with cones, spheres, cylinders, even with tori. For many years this was considered adequate for mechanical design, notwithstanding the persistent pressure from the aerospace and ship building industry to allow for more complex doubly-curved surfaces. Today, it is common to allow as surface elements subsets of tensor product nonuniform rational B-splines (NURBS), with some systems going further to consider triangular patches and more general types of parametric surfaces.

In the research arena there was an effort at the University of Bath a few years ago that resulted in a modeler with general algebraic surface elements. Algebraic surfaces have many mathematical and computational advantages compared to the parametric elements used now, and are emerging as an active, forefront research area. The commercial project generally cited as being the most advanced is Boeing's design of the 777 aircraft, which was done entirely using geometric and solid modeling (plus engineering analysis, costing analysis, and other nongeometric tasks). The Boeing 777 was modeled in CATIA, a commercial system from

Dassault in France that is marketed in the US by IBM. Interestingly, CATIA is based on 20-year-old technology, which is an indication of the lag between research and production in manufacturing.

There are many technical issues that arise from the type of operations one would like to perform on these surfaces, be it as part of a larger conceptual operation, or in its own right. They are addressed by a community that variously hails from applied mathematics and approximation theory to — on the more experimental side — mechanical engineering. A wide variety of classical computational geometry problems arise in the process; their effective solutions would have considerable impact. This has already happened in some cases.

An issue somewhat less visible to academics, but important in practice, is the simple question: How does one express a complex shape? Or more precisely, how can a human define unambiguously a complex shape to an automaton? This is "the user interface problem" in CAD. Current practice has resulted in user interfaces that require, as an integral part, geometric constraint solvers. The issue is partly technical and partly conceptual.

In the technical category one finds such crisply stated questions as, "Given four spheres, how many lines are tangent to all of them?" (Estimates are known, but the exact answer is not.)

In the conceptual category one will occasionally witness discussions among mathematicians on the "fairness" or the "eye-pleasing quality" of a curve in space. While such subjective discussions might tread on slippery grounds, they are nevertheless essential to the design of commercial products where aesthetics matters (as in the automobile industry). Although computational geometry is an unlikely participant in such discussions, it can learn from this that not all design criteria are quantifiable by optimization functions, and that flexibility and experimentation are vital components of design tools.

There are deep and pressing problems that CG'ers, as computer scientists, may wish to ponder. Specifically, there are two "input-and-interaction" modalities for CAD systems: (1) programming interfaces, and (2) "pick-and-click" graphics user interfaces. Media such as the PADL-2 system for solids bring much of the usual power to the job: naming, parameter passing, conditionals, etc. Language interfaces are nicely suited to defining parametric families of parts, but lack the visual feedback that most engineers consider essential. Graphics user interfaces provide plenty of feedback, but almost no abstractive power (names, parameters, etc.); they deal in instances rather than generics. An important challenge is to devise a user interface with the visualization power of modern GUIs and some of the abstractive power of a programming system.

In its maturing stage, computational geometry has focused its attention mostly on linear objects. The motivation was sound: Why deal with curved shapes if we do not even understand polyhedral objects? In manufacturing, such a limitation is simply unacceptable. The time has come to bridge this gap.

At present, the frontiers in solid modeling for manufacturing include how to define surfaces subject to constraints that require variational approaches and differential equations. Another major push is an attempt to integrate shape design with other aspects of engineering design, such as analysis and costing [153].

One particular challenge for CG, worthy serious attention, is *anisotropy.* Loosely speaking, anisotropy refers to the situation in which material properties vary over the interior of an object. (Wood is a simple example of an anisotropic material.) Some new methods of rapid prototyping processes, of which stereolithography is the best known (although not necessarily the best), are capable of producing parts having anisotropic material properties. The currently popular schemes for representing solids – boundary representations and constructive solid geometry (CSG) – appear to be incapable of supporting anisotropy; thus, new means must be found, since customizable anisotropy is one of the most attractive properties of layered manufacturing processes.

Also, it is still the case that boundary-representation modelers lack robustness: Finite-precision arithmetic can lead to incorrect results (see Section 10). This is an important area of research, together with the design of better visualization tools to facilitate – and ultimately to automate – geometric reasoning.

Solid modeling is the archetypical example of an area of computer science in which object representations and algorithms are inseparable. Boundary representations and triangulated solid representations can be used for describing the same objects, but the algorithms that deal with them are fundamentally different. Representations are often dictated by particular needs. (For instance, boundary representations are natural for graphics display; triangulated boundary representations are required by fast display hardware.) Thus, manufacturing applications must be identified first. Next, the appropriate representations must be chosen. Only then can CG begin to think about efficient algorithms to operate on them. This is a different mode of operation from traditional CG: in classical problems such as polygonal triangulation or 3D convex hulls or 2D Voronoi diagrams, standard representations are more or less equivalent (via linear-time reductions) so it makes sense to think mostly of an object as a geometric entity rather than as its particular representation. Things are quite different in solid modeling.

Intersections of Parametric Surfaces. Spline representations are used extensively in computer graphics and in modeling "sculptured surfaces" found in automobile bodies, airplane airframes, and ship hulls. Several classical methods have been developed; for instance, Coons Patches, Bézier Surfaces, B-splines. The object has been to allow designers (particularly ones using CAD workstations) to produce the desired surfaces with minimal effort and maximum of control of the surface shape. Algorithms for rendering curves and surfaces on graphics screens are well understood. However, computing intersections of lines, curves, or planes with these surfaces can be rather complex and inefficient. Intersecting two such surfaces is usually very messy. Usually the parametric mapping cannot be analytically inverted, so even the "inverse point solution" of finding the parametric coordinates for a point in 3-space that lies on the surface requires an iterative procedure, and intersections must be approximated by using numerical methods.

Repeated computations of intersections with parametric surfaces tend to be very slow and error-prone. If one wishes to intersect many rays with a parametric surface, or to find if a tool envelope penetrates a parametric surface, it may be desirable to approximate the parametric surface by a polygonal surface. The operations can then be done on this approximated surface. The question naturally arises, "What is a good approximation for this particular application?" Much work has been done in the area of approximating surfaces by triangles, particularly with respect to finite element methods. However, for machining applications the question is not finding the fat triangles that are good for finite-element methods or triangles that render nicely on a graphics screen, but how to get a polygonal surface with a small number of triangles that is at all points within a fixed tolerance of the given surface. A number of heuristics have been used, but it is still an area of active research.

Intelligent CAD/CAM: Manufacturing Processes. The manufacturing industry has at its disposal a wide variety of processes for constructing objects, including gravity casting, injection molding, layered manufacturing, material removal via conventional (or chemical or electrical) machining, deformation (forging, rolling, extrusion, bending), composition (as in composite materials, sintered ceramics, and the like), spray deposition, etc.

Each manufacturing process imposes certain restrictions on the types of objects that can be built and the manner in which they are built. For example, a sphere is difficult to build using layered manufacturing (e.g., stereolithography) but not using injection molding. Also, a cube can be manufactured using stereolithography without the use of supporting pillars only when placed on one of its facets.

In all of these manufacturing contexts, CAD/CAM systems of growing sophistication are presently being introduced. Geometric computation has become ubiquitous in the manufacturing industry, as more and more real-world objects begin their design life as geometric objects modeled within a computer.

Given an object to be manufactured, two fundamental questions arise, having to do with the *feasibility* of a process and its *optimization*:

- Can it be manufactured using a particular process?
- For a given process, what is the best way to manufacture the object?

The latter question gives rise to many different problems depending on the meaning of "best." The geometry of the object, coupled with the restrictions imposed by the particular manufacturing process under consideration, play a vital role in determining the answer to these questions. Their importance is quite evident. For example, when designing an object to be built by a certain type of manufacturing process, an engineer must always keep in mind the process that is to be used to manufacture the object. This limits the creativity of the engineer, as the question of design feasibility is ever-present. In fact, the engineer is rarely sure whether the object can be built in the first place, since no formal methods exist to determine the feasibility of an object for most manufacturing processes.

At present, many aspects of the manufacturing processes and their ensuing geometric problems are tackled by relying on heuristics in trial-and-error fashion, which necessitates a great deal of human intervention. To quote from an injection molding book [121].

> "During injection, the mold is tilted into a favorable position that will eliminate surface defects such as bubbles and insure a complete fill. Mold orientation during fill is a cut-and-try process to find the most favorable position."

It is an open problem to decide whether a given object can be built by a particular process: This is a question into which computational geometry should be able to make inroads. Practical solutions especially are highly sought. This form of "automatic design verification" would liberate the designer by informing him or her of the full range of possible processes that can build an object. To make these ideas more concrete, we consider two examples: gravity casting and stereolithography.

- Gravity Casting: In the case of gravity casting, the main motivation for focusing on the geometry of the object to be molded is to remove the "cut-and-try" phase from the process of finding a favorable mold position[4]

[4] A mold refers to the whole assembly of parts that make up a cavity into which liquid (e.g., molten metal) is poured to give the shape of the required component when the liquid hardens.

Given a mold, establishing whether there exists an orientation that allows the filling of the mold using only one pin gate[5] as well as determining an orientation that allows the most complete fill are two major problems in the field of injection molding.

These problems are particularly difficult when one takes into account the fluid dynamics and physics of the molding process; typically, this is handled by costly simulations. To date, only heuristics have been proposed to solve the two problems (feasibility and optimization) above. Efficient solutions from computational geometry might provide a handle on the geometric component of the problem.

- STEREOLITHOGRAPHY: Stereolithography, a form of "layered manufacturing", is one of a growing set of processes used in rapid prototyping; for recent surveys of this and related technologies, see [136, 147].

  The components of the stereolithography manufacturing process (e.g., in the system patented by 3D Systems of Sylmar, CA) consist of a vat of liquid photocurable monomer, a computer-controlled table on a stand that can be moved up and down in the vat and a laser above the vat that can shine on the surface of the liquid and can move in a horizontal plane. The system works as follows:

  In the first stage the table is positioned right below the surface of the plastic and the laser is controlled to move about so that the light shines on the plastic surface and draws the bottom-most cross-section of the object being built. Upon contact the laser light solidifies the plastic, and the first cross-section of the object is formed and rests on the table. In the second stage, the table is lowered by a small amount to allow liquid to cover the hardened layer and the laser then draws the next cross-section of the object. The light from the laser penetrates the liquid just deep enough so that the cross-section is welded to the lower cross-section produced in the previous stage. This process is repeated until the entire object is formed.

  The direction given by a normal to the table pointing to the laser is called the direction of formation for the object. There are objects that can be formed only if the direction of formation is chosen correctly. In practice, supporting structures have to be built as part of the model and be removed in a postprocessing step. Naturally, there are also shapes (say, a sphere) that cannot be formed using stereolithography regardless of the direction of formation chosen.

  Computational geometry has recently made valuable contributions by designing algorithms for determining whether a given polyhedron can be built using various geometric models of the stereolithography process. However, more work is needed to make these models conform to manufacturing reality.

It is unlikely that computational geometry alone will ever supplant the complicated fluid-dynamics simulations performed today by engineers. But to reject or ignore the physics of these problems is also to ignore their very essence. Computational geometers should make the effort of addressing some of these physical problems because they hold rich opportunities for CG. Meanwhile, the hope is that CG can help to reduce the number of costly simulations required. The ultimate usefulness of such algorithms must be determined by collaborative efforts between computational geometers and manufacturing engineers.

[5] A pin gate is the point at which the liquid is injected into the mold.

Verification of Numerical Control Machine Tool Programs. A manufacturing process of particular prominence, with a long history, is milling. A Numerical Control (NC) milling machine cuts a part out of a block of metal or other material (the "stock") by moving a cutting tool through space. For 3-axis machining, a rotating cutting tool, with a vertical axis of rotation, does the cutting. An "NC program" specifies the movement. Typically, the program consists of a "cutter location file," which is a long list of (x, y, z) coordinates for successive tool locations. The tool is moved to the first spatial location, then moves along a straight line (or, say, a circular arc) to the next, then along a straight line from the second location to the third, and so on, until the list of locations is completed. (Actually, the file also contains other information, such as tool sizes, and instructions for when tools must be changed.) For 5-axis machines, two additional coordinates giving the angular orientation of the tool axis are also specified and the tool linearly interpolates both spatial and angular coordinates.

Typically these NC programs are created by tool programmers. The programmers begin with a geometric representation of the part expressed in Constructive Solid Geometry (CSG), or some sort of boundary representation using splines, and generate series of tool movements that are supposed to mill the part from the stock.

As with computer programs, NC programs are error-prone. Bugs can produce cuts that are too deep or leave too much material. Thus, an important practical problem is: "Given a surface of known equation and a file of NC tool movements, does the shape that the tool cuts match the mathematical shape to within a given tolerance?" It has been claimed [53] that "current methods of verifying NC part programs result in one of the highest non-recurring cost factors in producing NC machined parts within the aerospace industry." It is also a significant cost within the aircraft and automotive industries, among others.

The early approach to finding bugs was to use "proofing runs" on wood or foam; this is still a popular technique. Later systems would move a graphical display of a cutter over a display of the part. The user could visually check for errors as the tool moved on the graphics display. Only gross errors could be detected in this way, however. In the 70's, work began on programs that actually detected errors in NC programs or verified that none existed. A good overview of this problem, including a precise mathematical formulation, history, and summary of its current state is found in [100].

Some of these methods use "complete" representations of the current state of the partially cut workpiece. Researchers investigated the feasibility of using CSG systems or solid modelers for the simulation of NC programs [76]. Others developed systems based on octree [111] or boundary representation [135] of the workpiece. For each method, every tool movement would update the workpiece, and at the end the workpiece was compared to the desired piece. Unfortunately, all of these approaches were slow. Each tool movement subtracts a fairly complex shape from the current workpiece representation and requires significant updating. For example, for a simple 3-axis movement with a ball-end cutter the swept volume is bounded by parts of two spheres, three cylinders, and two planes; the number of movements is typically on the order of 10,000.

To avoid this problem, a number of people took advantage of the fact that it is not necessary to represent the entire state of the workpiece to verify the correctness of an NC program. The desired shape is known in advance, and this information can be used to simplify the process. Several researchers used an image-space approach, where an extended Z-buffer is provided for each pixel in a given graphical view of the part, and each tool movement updates the pixel information for each pixel that it passes over [11, 138, 143]. CGTech's Vericut simulator uses such an approach. These methods are faster than the complete methods but they are view-dependent. Because of this, errors might not be visible from a certain view direction and would therefore be missed. (Consider a nearly flat surface viewed edge-on. A large surface would be mapped into a thin line represented by only a few pixels, and errors that occurred where there were no pixels would be missed.)

Another approach is to grow vectors normal to the surface of the part [30]. These vectors are "mowed down" by the passage of the cutting tool, and their lengths are checked at the end of the run. Vectors that stick out above the surface indicate excess material and vectors cut below the surface point to gouges. One might use pixels to select these vectors (one vector for each pixel) [113]. Or one might use properties of the tool size and the surface curvature to pick fewer vectors while still guaranteeing that they will find all errors greater than a user-set tolerance [43].

A recent modification of the extended Z-buffer, called ray representations (ray-reps), has been proposed [99]. A ray-rep is the intersection of a grid of parallel rays in some direction with the part. (Note that the intersection of a ray and a part consists of zero, one, or several line segments.) Given such a representation, all of the operations of CSG can be computed quickly by using a specialized parallel computer for processing ray-reps, called a raycasting engine. All of the approaches using modified Z-buffers or vectors introduce error. Even if the intersection with the tool path movement is calculated exactly (more on this below), in effect the part surface is being replaced by a set of discrete points. What can we say about the correctness of the surface at points where there are no vectors? All of these approaches use enough vectors to be fairly accurate in practice, but with a few exceptions none of them attempt to analyze errors introduced by the approach to check whether they fall within tolerance.

The ray-rep representation seems an ideal candidate for such analysis. Even with the special-purpose parallel computer one wishes not to use more rays that are needed. But how to analyze the errors introduced remains an open question, as is the determination of the optimal number of rays to use. It is also interesting to look at other possible applications of this representation and to analyze the relationship between the spacing of the rays and the error introduced.

A second area that needs more study is the one of intersecting rays with surfaces. Intersecting rays with part surfaces is reasonably well understood. However, when the surfaces are represented parametrically using B-splines or similar methods, computing intersections generally requires a slow interactive technique. One must also compute the intersection of rays with the envelopes swept out by tool movements. This is not too difficult for 3-axis movements of ball-end cutters. But the same cannot be said of the intersection of a ray with a 3-axis toroidal-end cutter movement. Most existing codes use a polygonal approximation to the envelope or a large number of discrete tool locations. Unfortunately, no error analysis is available.

For 5-axis cutter movement, an efficient method has been given for computing the intersection within a given error bound by using 3-axis tool movement as an approximation and relying on tree-based localization technique to quickly decide which of the 3-axis movements is the one intersected first [122]. See also [92]. What remains open is to find a method for generating a polygonal approximation for the 5-axis envelope with a provable error bound that generates a reasonable number of triangles. Finding such a provably good approximation technique is an interesting and important open problem.

Generation of Numerical Control Machine Tool Programs. An ultimate goal, of course, is to automatically produce NC programs that are error-free and perform the machining quickly. The generation of error-free NC programs directly from computer-based geometric models would significantly shorten the product development cycle. While some limited attempts at automatic NC program generation have been reported in the literature [7], a practical system has yet to be achieved. Producing such a system would be a very valuable step, and is quite likely to use a mix of computational geometry and classical optimization.

The special case of pocket machining has received recent attention and has had some success both in the algorithmic understanding of the problem, and in the automatic generation of NC programs. Held's thesis [69] brought to the forefront some of the issues of interest to computational geometers. Offshoots from Held's thesis, including software for offsetting polygonal boundaries, have made it into commercial products, such as *LARK* (available from MTA SZTAKI, a subdivision of the Hungarian Academy of Sciences, and the spin-off company, CADMUS). Provably good approximation algorithms for specific (simplified) classes of problems have been devised by CG'ers for minimizing the total tool motion [5] and the number of retractions necessary in "zigzag" pocket machining [6]. But many more problems must be addressed, particularly those that deal with realistic models of machining.

So far, the discussion has been limited to "nominal spatial effects." It has been assumed that the tool moves exactly as specified and removes all material within its swept volume. In real systems, other considerations for verification or generation must be taken into account. One concern is variations from nominal geometry in terms of allowed tolerances and positioning uncertainties. Other considerations are dynamic effects, such as dealing with deformation of the part and tool due to pressure in cutting, tool breakage if the pressure is too great, and tool chatter and wear.

Layout: Optimal Use of Material. Various layout and cutting problems are of immediate relevance to manufacturing, as they involve the optimal use of valuable raw materials. For example, in the parts nesting problem, we must place a given set of polygonal shapes (markers, templates, or patterns) on a sheet of material (sheet metal, cloth, cardboard, etc.), without allowing overlaps, in order to minimize the waste when the shapes are cut out from the material. Motivated by applications in the apparel industry, CG'ers have been investigating these layout problems, with some recent successes [103, 41, 101, 105].

In an application at Boeing, where very expensive sheet metal stock can be wasted in inefficient layouts, a software package ("2NA: 2-Stage Nesting Algorithm") is in current use on the factory floor, built on principles of optimization and basic computational geometry primitives. However, much more work is necessary before automated nesting algorithms will be able to replace human experts in many aspects of the material usage problem.

10 Robustness

Geometric computation must preserve numerical accuracy and topological consistency: on digital computers this is often akin to squaring the circle. Geometric algorithms are usually described assuming that input data is in general position and that exact real arithmetic provides reliable geometric primitives. Often an implementer substitutes floating-point arithmetic for real arithmetic and uses real-world data, which might be degenerate by accident or design. Hence, the correctness proof of the mathematical algorithm does not extend to the program, and the program can fail on seemingly appropriate input data. This is the well-known problem of "nonrobustness" in geometric computing [71].

The failure rate of a floating-point implementation can be reduced with various ad-hoc methods, such as tolerancing (taking a nearly-zero expression value as exactly zero has the effect of treating a nearly-degenerate configuration as exactly degenerate) or input perturbation. With such methods a usable, though not completely reliable, implementation can often be obtained. This hardly qualifies as a rigorous approach, however: the effort involved is unpredictable, the required expertise is hard to codify, and there is no guarantee that the implementation will succeed on the next problem instance.

In the past few years some research has addressed robustness issues. Proposed approaches include error analysis, sophisticated tolerancing, symbolic reasoning, or software that provides exact arithmetic on integers, rationals, or even algebraic numbers. There is yet no agreement on the best approach. Indeed, all proposed methods have shortcomings, typically limited applicability or inefficiency. A sophisticated programmer implementing a standard algorithm may well find some of these methods useful and relevant; the situation is more problematic for an unsophisticated programmer with a variant problem.

When addressing these issues, it is important to distinguish between degeneracies and numerical precision.

10.1 Degeneracies

Degeneracies arise from the special position of two geometric objects. For example, two segments in general position either do not intersect or intersect at a point interior to both segments. Two intersecting segments in special position may overlap, may share a common endpoint with or without being collinear, may have one segment endpoint interior to the other segment, etc. Real-world data is likely to be degenerate. For example segment endpoints may be explicitly chosen from a coarse grid, to facilitate interactive design.

The effect of degeneracy is to vastly increase the number of special cases. While a sorting algorithm must deal only with the possibility of two keys being equal, a typical geometric algorithm faces the possibility of dozens or hundreds of different special cases [47, 49, 71, 148]. The presence of numerical data, added to the inherent complexity of geometric data types, makes geometric algorithms much harder to implement correctly than combinatorial (say, graph-theoretical) ones. They are also much harder than just purely numerical algorithms (such as those addressed by numerical analysis) many of which consist of large chunks of straight-line code. Since the overall utility of an implementation may depend upon the correct treatment of special cases, the handling of special cases can permeate the implementation. This raises the obvious reliability concern that all cases have been considered and correctly handled.

Symbolic perturbation schemes allow degeneracies to be resolved automatically. Conceptually, each geometric coordinate c_i is replaced with a symbolically perturbed coordinate $c_i + f_i(\epsilon)$, where $\epsilon > 0$ is unknown but very small and the perturbation function f_i is simple, say a polynomial. Substitution of the symbolically perturbed coordinates in a predicate expression results in a polynomial in ϵ with coefficients determined by the original geometric coordinates. The sign of the expression is given by the sign of the first nonzero coefficient, with coefficients taken in order of increasing powers of ϵ. For many classes of predicates, the f_i can be chosen to resolve all degeneracies.

While symbolic perturbation is certainly a useful tool in the implementation of geometric algorithms, existing schemes are not as applicable as might be desired. First, symbolic perturbation requires exact arithmetic, since the correctness of the perturbation depends upon exact evaluation of arithmetic expressions. Second, symbolic perturbation has been worked out in detail for only a small class of predicates. For example, constructed objects are often disallowed, since the perturbation function for a constructed object depends upon how the object was constructed and is much more complicated than the perturbation function for a primitive object. Perhaps most fundamentally, in a degenerate situation an algorithm implemented using symbolic perturbation does not solve the problem instance, but an arbitrarily-chosen nearby problem instance. This might be inappropriate in some applications. For example, a highly degenerate polytope may see its combinatorial complexity blow up exponentially as a result of a small perturbation.

Although much work remains to make symbolic perturbation generally useful, no other approach to the issue of degeneracy has as much promise.

10.2 Numerical precision

A geometric predicate is determined by the sign of an arithmetic expression. Sign-evaluation is exact in the conceptual model of the real numbers. However, a computer implementation cannot use real numbers. There are two scientifically plausible substitutions for exact real arithmetic. One is to use floating-point arithmetic, and somehow deal with the resulting rounding error. The other is to substitute exact arithmetic on a subset of the reals, such as the integers or the rationals. There is now a growing community of researchers who have interest in "real arithmetic computation", and some of these issues are relevant to the exact computation paradigm in CG. One such conference, "Real Numbers and Computers", will be held for the second time in April 1996.

Floating-point arithmetic. Floating-point arithmetic is widely used because it has many practical advantages. It provides a familiar approximation to the real numbers, with useful properties like automatic scaling. It is widely available on different computers and is well supported by programming languages. Current workstations have highly optimized native floating-point arithmetic, sometimes faster than native integer arithmetic. Floating-point arithmetic is sufficiently widespread in scientific computing that programmers rarely consider other options.

However, some geometric predicates cannot be resolved using floating-point arithmetic. If an instance of a predicate is nearly degenerate, then the value of the corresponding expression can be very small, less than the rounding error in the floating-point evaluation of the expression. Hence the sign of the computed expression may well be erroneous. Usually it is possible to argue that the computed expression value is the true value for slightly perturbed coordinate data. Since coordinate data may well be imprecise originally, the erroneous sign may appear to be innocuous. The difficulty arises with multiple predicate evaluations; there is no guarantee that any single global perturbation produces all the computed predicate values. Indeed, the computed predicate values may be geometrically inconsistent. Catastrophic implementation failure can easily result.

There are two broad categories of methods to deal consistently with floating-point rounding error. One category formalizes the notion of tolerances. A typical strategy might associate inner and outer tolerances with an object, say a point. If the inner tolerances of two points intersect, they are deemed coincident and merged; if the outer tolerances are disjoint, they are deemed separate; if neither case holds, the situation is ambiguous. As long as no ambiguity results during a computation, the result is correct. The hope is that ambiguities arise infrequently; an obvious drawback of this strategy is that is not clear what to do if an ambiguity does arise. Another drawback is that the generalization to complex geometric objects is not straightforward.

Since arithmetic operations have non-constant costs, it makes sense to judge the performance of an algorithm in terms of the precision K needed in the output of the algorithm. This gives rise to the notion of "precision-sensitive algorithms" where the running time is a function of K as well as a function of the input size. Since K can be exponential in the input size, exploiting this new parameter can be quite significant. Notice that precision-sensitivity is the bit-complexity analogue of the very fruitful idea of "output-sensitivity" invented by CG. The paper [35] first applied this concept to the NP-hard problem of Euclidean shortest paths.

A second floating-point approach is error analysis. This approach is modeled on the error analysis of numerical methods, particularly linear algebra. The goal is to show that a suitably implemented algorithm provides an answer that is in some precise sense near the mathematically correct answer. Error analysis of geometric algorithms requires consideration of both combinatorial and numeric structure. Often it is easy to argue that an algorithm produces combinatorially valid output, at least with suitably relaxed requirements. It has turned out to be much more difficult to argue that the numeric error associated with combinatorial structure is small. Full error analysis has been carried only for a few simple algorithms.

Exact computation Exact geometric computation [152] requires that every predicate evaluation be correct. This can be achieved either by computing every numeric value exactly, or by some symbolic or implicit numeric representation that allows predicate values to be computed exactly. Exact computation is theoretically possible whenever all the numeric values are algebraic, which is the case for most current problems in computational geometry.

With exact geometric computation, it is no longer reasonable to assume that each arithmetic operation takes constant time, as is the case with floating-point arithmetic. Rather, the cost of an arithmetic operation depends upon its context. Simple geometric predicates can be expressed as the sign-evaluation of an integer polynomial. The required arithmetic bit-length can be estimated from the degree of the polynomial and the bit-length of coordinates. For many predicates involving linear objects (such as orientation predicates) the degree is small, and the required bit-length is relatively minimal. However, even in simple cases software arithmetic is required, say over the integers or rationals, with a resulting increase in performance cost.

The cost of arithmetic also increases because of geometric constructions. A geometric constructor produces a new geometric object from old objects. Typically the coordinates of the new object can be expressed as polynomials in the coordinates of the old objects, and hence bit-length can be estimated from polynomial degree. For example, the coordinates of the intersection point of three planes have bit-length about three times the plane coefficient bit-length; the coefficients of a plane through three such points have bit-length about nine times the original plane coefficient bit-lengths. Hence an algorithm that uses geometric objects constructed to arbitrary depth can require arithmetic with prodigious bit-length, even if the algorithmic predicates are relatively simple.

Complex primitives on linear objects, or simple primitives on curved objects, apparently require that arithmetic be augmented with square roots, and, more generally, arbitrary polynomial roots. Such arithmetic operations can be implemented symbolically using general algebraic techniques such as resultants, Sturm sequences, and root separation bounds. Whether such techniques can be implemented with adequate efficiency is a question of considerable interest.

Because of the potential expense of exact computation, it is often appropriate to avoid it if possible. Any significant application may well have a core requiring exact computation and outer layers that are less demanding. For example, the boolean operations of a solid modeler might require exact computation to guarantee topological consistency, while the computer-graphics rendering of a resulting solid could tolerate some imprecision. Other applications, for example, a geometric theorem-prover, might require exact computation throughout.

To Summarize. For algorithms with modest arithmetic bit-length requirements, exact arithmetic is appealing. Exact arithmetic appears not to be used widely, perhaps because of the performance cost of the required software arithmetic. However, computer hardware continues to increase in speed, which may mean that the performance cost becomes less significant. Furthermore, there is evidence that software exact arithmetic can be better designed for computational geometry applications, decreasing its effective cost.

Exact-arithmetic implementation of geometric algorithms would be much more attractive with the development of software arithmetic packages appropriate for computational geometry[52, 110]. There are many issues to be explored: for example, the use of adaptive-precision arithmetic, the granularity of evaluation, algorithms for primitive evaluation, required arithmetic operations. For example, beyond speeding up basic arithmetic operations, more effective optimization techniques could be used at the expression level. Geometric computations invariably involves larger units of numerical computation, such as expressions like determinants or Euclidean lengths. This opens up an exciting new area of software construction in which many designs are possible.

The utility of exact arithmetic would be increased with the development of exact-arithmetic rounding algorithms. Geometric algorithms often produce objects that have both combinatorial and numeric data. For example, the vertices, edges, and faces of a polyhedron in three dimensions have both numeric coordinates and combinatorial incidence structure. The result of an exact computation may specify the numeric coordinates exactly, with coordinates of large bit-length. However, arithmetic on large coordinates is expensive, and an application may be satisfied with a short bit-length approximation to the numeric coordinates, as long as the combinatorial and numeric data are consistent. Rounding the coordinates of a complicated object such as a polyhedron is not straightforward, since its combinatorial structure may be invalidated by small perturbations of its faces or vertices. However, many applications are insensitive to changes in the combinatorial structure. If the structure is permitted to change, there are methods to round polygons or other planar objects made up of line segments to the integer grid [70, 66] or any nonuniform grid [102, 104]. In general, rounding algorithms, particularly for curved or higher dimensional structures, are as yet inadequately developed.

There are important applications, such as operations on algebraic curves and surfaces, where bit-length estimates appear to rule out the use of exact arithmetic. It is possible that such bit-length estimates are too pessimistic, either in theory because the underlying algebraic machinery is not developed enough to give sharp estimates, or in practice because instances requiring long bit-length are infrequent. Many predicates that are well-understood in the linear domain (incircle, orientation) become much more complex in the curved domain, and their best evaluation using exact arithmetic is not well understood.

A Case-Study: Dimensional Tolerancing and Metrology. We close this section by discussing one application area that seems ideally suited to the techniques of computational geometry and the requirements of exact computation [149, 150]. Dimensional tolerancing theory and practice form a key cornerstone of modern precision engineering, allowing designers to specify tolerances in a specific way that also, in principle, allows machinists to verify them. It has been observed that the precision requirement on manufacturing is essentially increasing linearly with time. Increased precision in manufacturing has strong positive economic implications, from more efficient use of material to greater reliability in parts. But such benefits are partly wasted if the metrological theory and practice do not keep up with increasing manufacturing standards. This gap is partially responsible for the so-called "CMM crisis" identified by Walker in 1988.

Briefly, the crisis concerns the inability of current Coordinate Measuring Machine (CMM) software to reliably compute mathematically-precise outputs, even assuming the input data are reliable. In fact, the reliability of the input data in existing systems are orders of magnitude better than the reliability of the computed values. The goal, then, is to completely remove the unreliability introduced by software. The computational geometry attack on this problem is two-pronged: first, we base algorithms on mathematically correct methods (this is sometimes called "CG methods" in the literature). Second, we implement these algorithms using exact computation in a way that characterizes the input imprecision.

Beyond the basic algorithms in computational metrology, there are some larger system issues. One is the "reference software" project that several national standards agencies (say, in Germany, UK, USA) are attempting to construct. The idea is that CMM vendor software will continue to be proprietary (black boxes) and for contractual purposes, standards will be guaranteed for such software. The National Institute for Standards and Technology has instituted an Algorithms Testing and Evaluation Program for certifying vendor software. The heart of this program is the so-called "reference software." It is clear that the exact computation technology is ideally suited for this work. Note that algorithms in the reference software need not achieve "real-time speeds" to be useful. Another example concerns packages for tolerancing analysis that allow a designer to check the effects of tolerance specifications. Such packages are commercially available but again they lack reliability. An exact-computation version of such a system would be a major advancement.

11 Molecular Biology

With the human genome project underway and the promise of a revolutionary approach to understanding diseases, molecular biology is emerging as one of the most critical scientific disciplines today. It is an area likely to be heavily funded for many years, and its connections to computer science are widely recognized. Besides applied and numerical mathematics, the three major areas of computer science most likely to have an impact on research and development in molecular biology are information retrieval, string and pattern matching, and computational geometry. Several basic problems in molecular biology have a strong geometric flavor. These include problems dealing with the spatial structure of proteins and other macromolecules. Three such problems are briefly described below.

Spatial structure of a protein. The difficulty and reliability in determining the geometry or spatial structure of a molecule depends greatly on the type of available data. X-ray crystallography is currently the primary source of structural information, followed next by magnetic resonance imaging (MRI). To derive molecular structure from MRI data, one must resolve atomic positions given a set of approximate pairwise distances. The traditional tools for this problem come from distance geometry [40].

Lacking X-ray and MRI data, one can only resort to structural analysis of unknown proteins by looking for patterns in the amino acid residue sequence that match those of known proteins. For evolutionary reasons, the many proteins occurring in nature share a limited number of common internal structures and folds. Recognizing such patterns and threading the unknown protein onto it greatly simplifies structure determination when X-ray or MRI data are available. Modeling based on such analysis may also be valuable.

The most difficult version of the problem is also the one with the largest potential benefit. It is the ab initio determination of structure from the amino acid residue sequence [127]. While one can in principle use Newtonian mechanics to simulate the natural folding of the molecule, the sheer scale of the calculation is daunting. A modest-size protein folding simulation with current algorithms would require in the neighborhood of 10^{19} floating-point operations. A variety of heuristic methods are used to find the minimum energy configuration, including simulated annealing, Monte Carlo, and search with reduced degrees of freedom. As yet, none of these methods have come close to a general solution. Practical methods fail in most cases because the target function (say, the sum of energy potentials) has a large number of local maxima at any level of detail.

Ligand-protein docking. Consider an active site on a usually large receptor molecule, and a ligand molecule, which could be small or large. The general question is how snugly the ligand fits into the active site. This recognition mechanism is ubiquitous in biology and is basic to communication, immune response, and control of metabolism. The quality of fit has a geometric and a chemical component. The geometric component measures how well the surface shapes complement each other as a hand in glove. The chemical component measures how well the secondary forces between ligand and receptor atoms hold the two together. In the simplest version of the problem, both ligand and receptor are assumed to be rigid bodies, and the objective is to find a best alignment of the surfaces describing their boundaries. Very often the rigidity assumption is not realistic and flexible models need to be considered. Abstractly, the docking problem is related to the computer vision problem of matching a 3-dimensional geometric model to range data. The widely used schemes for representing the molecular surface and receptor sites in docking algorithms may not be the best choices. Other descriptions, such as alpha-shapes [48], and other matching criteria, such as Hausdorff distance rather than least squares, may be more appropriate.

A problem related to docking is the recognition of motifs in proteins, which are substructures of the proteins with similar geometry. Matching motifs may indicate genetic links or active sites with similar properties.

Drug design. Given a specific receptor, the problem is to find or design a binding small ligand molecule. If the receptor site geometry is known, the problem is to find a molecule that satisfies some geometric constraints and is also a good chemical match. After finding good candidates according to those criteria, a docking step with energy minimization can be used to predict binding strength.

If the site geometry is not known, as is often the case, the designer must base the design on other ligand molecules that bind well to the site. If those other molecules are rigid, the problem becomes one of identifying the substructures or active groups that contribute to the fit. By joining the groups with alternative molecular scaffolding, one can build molecules with good or better affinity for the site. The joining process requires solutions of kinematics and distance geometry problems. If the receptor site geometry is unknown and the ligands are flexible, then the designer must first posit the configurations of the ligands in their bound state by assuming that their active groups are in similar places. New drugs are enumerated which can be folded into a similar appearance.

The Role of Computational Geometry. One of the greatest challenges for computational geometry is to contribute to the understanding of nature in terms of geometry and algorithms through the connection offered by molecular biology. The most difficult of the specific problems mentioned above is the simulation of the folding process that would allow the determination of protein structure from the amino acid residue sequence, and a solution may be out of reach at the moment. The other problems seem tractable, and solutions already exist for easy cases and also for some of the hard ones. The only certainty is that a considerable amount of research is still called for.

Computational geometry can help in several ways. The first is to participate in the design of geometric models. The sphere model for atoms is the fundamental idea connecting particle physics with geometry. There are still open questions as to what extent this simplification is sufficiently accurate and how physical questions can be approached through studying large conglomerates of spheres. Based on various extensions of the spherical model [91, 126], the biology community has developed its own geometric software [38]. The geometry is enhanced by graphics and numerical software visualizing and utilizing the geometric information. New software will have to compete with the available packages, which are already widely used, and it will need to follow data and calling standards so components can be plugged together to solve large problems.

12 Astrophysics

Large ongoing projects, such as the Sloan Digital Sky Survey (SDSS) [83, 22, 123], aim at creating a comprehensive digital map of the northern sky. The project SDSS will result in a complex archive of about 20 terabytes containing exquisitely calibrated digital images of the accessible half of the northern sky, in 5 wavelength bands, from the ultraviolet to the infrared. A processing pipeline will identify and derive parameters (fluxes, sizes, colors) of over 100 million objects, classify them by type (star, galaxy, quasar) and obtain high resolution spectral information for the brightest one million galaxies and 100,000 quasars in the survey area. All this information will be placed into the archive for subsequent access and analysis.

The huge amount of information involved makes storage and access a formidable computational challenge. The data is organized as points in $\mathbf{R}^d$. Typically the dimension d is fairly low, e.g., 8: right ascension, declination, five colors, redshift.

Astrophysical Database. The information should be stored in a database that can support geometric queries of the kind:

- Range Query [31, 97]: Given a d-dimensional rectangular box defined by intervals $\{(a_i, b_i) : 1 \leq i \leq d\}$, where $a_i, b_i \in R_i$ (R_i is the range of the attribute i), find all the objects which lie within the specified box. For example, one may be asked to find all objects within a given right ascension and declination.

- Linear Combination Query: Given a d-dimensional slab of the form $l \leq \sum a_{i_j} x_{i_j} \leq u$, where $i_j \in J$ (with J the index set defined on all attributes), find all the objects that lie in the specified range. This query involves searching whether linear combinations of given attributes have a value within the range given by l and u. For example, we may have a query of the form: find all objects over the whole sky with a given color, ie, objects which satisfy $u - g \leq 1$, where u and g are color bands.

- Convex Polyhedron Query: Given a d-dimensional convex polyhedron, specified as intersections of halfspaces (linear combinations of attributes), find all the objects which lie inside the polyhedron.

- Nearest Neighbor Query: Given a query object q, find the closest (in the Euclidean sense) object in the database. This is a difficult problem to solve efficiently (see [8] for a good history of the problem). In practice, one can implement a simpler version of the problem called *approximate nearest neighbor* [8] which takes a parameter ε, and finds an object in the database whose distance to q is at most $(1 + \varepsilon)$ times the distance to the actual nearest neighbor.

- k-Nearest Neighbors Query [8]: Given a query object q, find k-nearest objects (approximate) to q in the database.

- Proximity Query: Given two regions in d-space and a parameter δ, find all pairs of objects, one in each region, that lie within a distance δ of each other. For example, one may have a query of the form: find all blue galaxies within 2 arcmin of quasars.

One would also be able to compute topological information (genus, fractal dimension) of cell complexes formed by subsets of the stored objects. Traditionally, the relational format has been the favored model for astronomical databases. Such models tend to be slow and unable to accommodate sophisticated geometric query retrieval. In that regard, space partitioning schemes, such as k-D trees, appear much more promising. The appropriate choice of splitting hyperplanes is also a matter of debate. Statistic-based strategies, such as splitting normally to the direction with maximum variance (covariance matrix method), are potential candidates, but many other choices look promising, too. Actually, the current advances in space partitioning raises the hope for yet more efficient algorithms.

For the nearest neighbor problem, Voronoi diagram-based methods are doomed because of the exponential blow in the dimension. Approximation methods have considered [8]. A full taxonomy should be developed so that the best algorithm for any given dimension can be identified. When the dimension is very large, projection methods might work well: pick a random k-flat for small k, and project the points normally to it. Solve the problem in the projection flat. The hope is that a random flat will more or less preserve distance relationships. Current knowledge does not allow us to assess how good such methods are. More sophisticated strategies should also be sought. The size of the answer to the queries could be huge, and so one would like to develop faster (yet accurate) estimation strategies to help the user determine what he is up against. This information can be very useful in practice. To perform accurate estimations, statistical analysis of the distribution of the data in each dimension is required for performing good partitioning. The resulting geometric data structures from the partitioning typically satisfy some nice properties which can be used for performing the estimation efficiently.

The massive amount of data necessitates the use of hierarchical memory organization. External memory models have been proposed which let researchers design I/O efficient algorithms. Recently several I/O efficient algorithms for specifically geometric problems have been proposed which make use of novel techniques [4, 24]. Chiang [34] gives experimental evidence of the importance of redesigning geometric algorithms to minimize page faults and to optimize the number of I/O accesses.

Computational Topology. An important feature of a galaxy is its topology. Does the galaxy distribution consist of isolated high-density clumps (meatball topology), does it have isolated voids surrounded by walls (bubble topology), or does it resemble a sponge, with interlocked high- and low-density regions?

The standard first step is to turn galaxies, which are represented by finite set of points, into smooth surfaces in 3-space. To do so choose a weighting function (a Gaussian) convolving with each point. This induces a continuous density function throughout the sky. Isodensity contours form the surfaces whose topology we wish to compute. The Gauss-Bonnet formula allows us to express the genus of the surface in terms of the integral of the Gaussian curvature. The integral is computed by sampling the surface by a fine enough grid. These techniques work fairly well and have been used to determine the topology of many galaxies from satellite observations [106].

Alternatively, the points can be connected to form a simplicial complex. Instead of a weighting function we choose a spherical ball around each point and consider the boundary of the union of balls. To compute the components of this surface and the genus of each component, we convert the union of balls into a complex as follows. First, clip the balls to within their Voronoi cells. The resulting cells overlap along their boundaries. The alpha complex represents pairwise, triplewise, and quadruplewise overlap by edges, triangles, and tetrahedra. The complex is homotopy equivalent to the union of balls [46], and there is a fast linear space algorithm that computes the three betti numbers. The number of surfaces, their genuses, and how they are nested is readily derived from this information.

If other parameters are taken into account, the dimension of the ambient space shoots up and to reveal the full topology is more difficult. Of all the topological invariants, the homology groups are the most popular, mostly because of their computational tractability. The homology groups describe the cycle structure of a topological space. It would be extremely useful to be able to compute homology groups efficiently for low-dimensional geometric complexes. Outside astrophysics, there are many important applications in pattern recognition and classification in biology and chemistry as well as in robotics and scene analysis. Comparing homologies is a good (if not foolproof) test to rule out topological equivalence: two spaces with different homologies cannot be homeomorphic; of course, the converse is not true.

There are also applications to the classification of time series and the analysis of dynamical systems. Spectral analysis has long been the standard tool for classifying time series. In the absence of sharp harmonics, the signal is dismissed as noise. This might be a mistake. Analysis for nonlinear dynamics can be done by computing topological invariants of the attractor set for the system being measured. To do so one can map the time series into $\mathbf{R}^d$ ($d \approx 10$) by scanning a "comb" of length d across the series and thinking of each d-tuple of entries at the teeth as a point in $\mathbf{R}^d$. By grouping all $(k+1)$-tuples of points at a distance less than ε from one another, one can define k-simplices. Finally, by introducing the obvious boundary operators, this yields a complex whose (Čech) homology is called the ε-homology of the set. Because Čech homology commutes with inverse limits, this is an appropriate method for computing the homology of the attractor. Moreover the use of ε-homology (where simplices are defined only in terms of pairwise distances) should have computational advantages. This is similar to the idea of alpha complex alluded to in the earlier paragraph.

Computing homology groups, or their ranks referred to as betti numbers, can be done by algebraic methods, but it is often inefficient. Exploiting the specific geometry of the manifolds can lead to much faster algorithms. There is a compelling analogy with graph algorithms. Many problems on graphs can be phrased algebraically and solved by inverting linear systems. That is not the way it is done, however. Graph algorithms seek to exploit the inner combinatorial structure of graphs to avoid generic algebraic treatments. Similarly, computational geometry can be used to bypass linear algebra in the computation of homologies. An example is the algorithm for simplicial complexes embedded in $\mathbf{R}^3$ described in [42]. In particular, the use of Mayer-Vietoris sequences should be useful in designing efficient divide-and-conquer schemes for computing the homology of higher-dimensional complexes by computational geometric means.

N-Body Problem. One of the most central computational problems in physics is the simulation of a particle system with gravitational or Coulombic forces. The first task is to compute the potential field of the system. The potential at each point is the sum of the various potentials determined by the particles. The potential at x contributed by a particle at p is often of the form $q_p/\|x-p\|_2$, where q_p depends on the strength of the charge (or mass) of the particle at p.

Assuming infinite arithmetic, it is easy to compute the potential in $O(n^2)$ time, where n is the number of particles. A method, called the fast multipole method, based on multipole expansion was developed in [65] and refined by introducing computational-geometric ideas in distance geometry [25].

To apply the fast multipole method, one must first construct a tree decomposition of the set of particles, along with a set of pairs of nodes in the tree. This step can be shown to require $\Theta(n \log n)$ time in the algebraic model. Given such a decomposition, the fast multipole method requires only linear time. However, the constants involved in the fast multipole method are so large that for realistic numbers of particles, this turns out to be the most time-consuming part of the computation.

The large constant in the fast multipole method arises both from geometry and from the need to manipulate series expansions, the size of which grows with the desired output precision. Despite the large constant, the method has proven useful for simulations involving millions of particles, for which the direct approach is not even feasible. Reduction of this constant would have significant practical implications for particle simulation, both by speeding up existing applications, and extending the useful range of the fast multipole method to smaller sets of particles.

There is a great deal of flexibility in the geometrical decomposition needed for the fast multipole method, and some gains could result from investing more effort in this part of the algorithm. Currently, the tree computed is a fairly standard box decomposition that guarantees a small number of pairs of nodes for which the fast multipole method must be applied. However, for any given point set, there may be another tree that results in a somewhat smaller number of pairs, in which case it may be worthwhile to expend the effort needed to find such a tree.

The efficient implementation of the fast multipole method poses non-trivial technical challenges. For example, in current implementations, one typically constructs a tree in which the leaves are subsets of particles rather than single particles, and computes the potential between these subsets directly. The size of these subsets has a strong effect on the efficiency of an implementation, and must be chosen carefully in order for the method to be of any benefit.

Efficient parallel implementations would be particularly useful, because a particle simulation can require many potential computations, each of which carries the simulation through one small time step. The more steps that can be performed, the more can be learned about the dynamics of large particle systems, so any speed increases will have a direct impact on computational physics.

13 Resources for Computational Geometers

13.1 Journals

Three journals are devoted to the subject of computational geometry: *Discrete and Computational Geometry*, *International J. Computational Geometry and Applications*, and *Computational Geometry: Theory and Applications*.

Computational geometry papers also often appear in algorithm journals such as *Algorithmica* and *J. Algorithms*, and occasionally are published in general theoretical computer science and discrete mathematics journals including *Discrete Mathematics*, *Information Processing Letters*, *J. ACM*, *J. Combinatorial Theory, Ser. A*, *J. Computer and System Sciences*, *SIAM Journal on Computing*, and *Theoretical Computer Science*.

In addition, geometric papers with a strong applications content may be published in specialized journals aimed at the particular application of the paper. *ACM Trans. on Graphics* publishes papers in computer graphics. *Computer Aided Geometric Design* is largely concerned with topics related to splines and surface approximation, but has also published several papers on triangulation. *Annals of Mathematics and Artificial Intelligence* published in 1991 a special issue on algorithmic robot motion planning; the same subject is also covered by *J. Intelligent and Robotic Systems*. Some areas of geometry (especially robotics) have close connections with computational algebra and can be found in *J. Symbolic Computation*.

13.2 Conferences

There are several annual conferences and workshops devoted entirely to computational geometry. The oldest of these is the highly selective ACM Symposium on Computational Geometry. Others include the Canadian Conference on Computational Geometry, the ARO-MSI Workshops on Computational Geometry, and the European Workshop on Computational Geometry.

There is usually a large amount of geometry at the major algorithms conferences, including the ACM-SIAM Symposium on Discrete Algorithms (SODA), the Int. Symp. Algorithms and Computation (ISAAC), the European Symp. on Algorithms (ESA), the Workshop on Algorithms and Data Structures (WADS), and the Scandinavian Workshop on Algorithm Theory (SWAT). The three major annual theory conferences, ie, the ACM Symposium on Theory of Computing (STOC), the IEEE Symposium on Foundations of Computer Science (FOCS), and the International Colloquium on Automata, Languages and Programming (ICALP) also usually contain some amount of computational geometry.

The 2nd Federated Computing Conference to be held in Philadelphia, May 1996, is to include not only the ACM Symp. Comp. Geom., but also a new ACM Workshop on Applied Computational Geometry. DIMACS, Schloss Dagstuhl, INRIA, and Tel Aviv University regularly organize workshops in computational geometry. DREI, an NSF-sponsored program for high school teachers is having a special year in computational geometry, which includes a research component (3 workshops).

Pointers to calls for papers, programs, contents, and proceedings for many of these conferences as well as for conferences relating to various applications of computational geometry can be found online at

http://www.ics.uci.edu/~eppstein/junkyard/jconf.html

13.3 Web Sites

Directly related to the theme of this task force, David Eppstein's web page "Geometry in Action"

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

is devoted to applications and potential applications of computational geometry, and includes pointers to over 100 individual projects and applications in areas such as astronomy, geographic information systems, CAD/CAM, data mining, graph drawing, graphics, medical imaging, metrology, molecular modeling, robotics, signal processing, textile layout, typography, video games, vision, VLSI, and windowing systems. This page also includes pointers to other web sites including the ones discussed here. Eppstein maintains some open problems at

http://www.ics.uci.edu/~eppstein/junkyard/open.html

A large bibliography of computational geometry papers, maintained by Bill Jones and Otfried Schwarzkopf, is available at

http://www.cs.ruu.nl/people/otfried/html/geombib.html

Schwarzkopf also maintains some other web resources including a dictionary of French computational geometry terminology:

http://www.cs.ruu.nl/people/otfried/html/francais.html

Other lists of computational geometry resources are maintained at Carleton Univ.

http://www.scs.carleton.ca/~csgs/resources/cg.html

and by Jeff Erickson at Berkeley

http://www.cs.berkeley.edu/~jeffe/compgeom.html

Various items concerning conferences, seminars, software, bibliography and a list of researchers with web pages can be found in

http://wwwdim.uqac.uquebec.ca/~jmrobert/CG.html

which is maintained by Jean-Marc Robert.

13.4 Mailing Lists

Several computational geometry mailing lists are distributed at AT&T Bell Laboratories: *compgeom-announce* for announcements about professional activities, *compgeom-discuss* for discussion or questions, and *compgeom-tribune*, a newsletter sent out aperiodically by Hervé Brönnimann. Mail to compgeom-xxxx@research.att.com will be forwarded to subscribers to those lists. (Mail sent to compgeom-tribune will be forwarded to its editor.) For more information, send mail to compgeom-request@research.att.com with the message body "send readme" or see an HTML version of the readme file provided at

http://pine.fernuni-hagen.de/GI/compgeom/compgeom.html

The compgeom-request server is also capable of some other services including email searches of the computational geometry bibliography.

The Geometry Forum at Swarthmore College maintains several related newsgroups: *geometry.announcements* for announcements of programs and resources for geometers; *geometry.college* for topics relevant to the study and teaching of college-level geometry; *geometry.forum* for news and discussion of the Forum news service; *geometry.institutes* for proceedings of geometry conferences and institutes; *geometry.pre-college* for K-12 math discussions and online projects; *geometry.puzzles*, for your spare time; *geometry.research*, for advanced geometry investigations and research topics; and *geometry.software.dynamic*, for discussions of work using the Geometer's Sketchpad and/or Cabri Geometre. The contents are also available via mailing lists; to subscribe, send email to `majordomo@forum.swarthmore.edu` with message body "subscribe geometry-xxxx". For more information, see

http://forum.swarthmore.edu/~sarah/topics/about.newsgroups.html

Messages from all these groups are archived and can be searched at

http://forum.swarthmore.edu/~sarah/HTMLthreads/index.html

13.5 Software

Many people have implemented the basic geometric algorithms such as convex hulls and Voronoi diagrams; a few more complicated geometric algorithms are also implemented and available.

Nina Amenta has collected a large number of free computational geometry programs at

http://www.geom.umn.edu:80/software/cglist/

The programs are arranged by subject and include convex hulls, Voronoi diagrams, Delaunay triangulation, low dimensional linear programming, triangulation, point location, shape reconstruction, collision detection, constraint solving, and visualization.

Seth Teller has a collection of C code and SGI executables for linear programming, voronoi diagrams, and manipulation of NURBS surfaces at

http://graphics.lcs.mit.edu/~seth/geomlib/geomlib.html

Joe O'Rourke has made C code corresponding to the material in his textbook, "Computational Geometry in C", available at

ftp://grendel.csc.smith.edu/pub/compgeom/

The code itself is in file "sharfile", and a description of the other files in this directory is in "README".

Various people have provided computational geometry packages for the *Mathematica* symbolic computation package; these routines are available from MathSource at

http://www.wri.com/mathsource/

A keyword search on MathSource for "geometry" turned up roughly 40 packages (not all related to computational geometry); of particular interest are "Computational Geometry" by E. C. Martin, "CirclePack" by O. Goodman, and "Vertex Enumeration", by K. Fukuda and I. Mizukoshi.

LEDA, a Library of Efficient Data Types and Algorithms, is a project at the Max Planck Inst. für Informatik, run by Kurt Mehlhorn, that provides a large collection of data types and algorithms in a form that can be used by non-experts. Along with basic non-geometric data structures such as binary search trees, LEDA includes implementations of several geometric algorithms including convex hulls and line segment intersection. For more information see

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

14 Conclusions

Computational geometry enjoys two unique assets: (1) its diversity and potential to affect most forms of computing; (2) its mature algorithmic foundations. The challenge is now to make this potential come true, and build an effective pipeline connecting theory to practice. We believe that this will revitalize the field and open new vistas for geometric research, both of a practical and a theoretical nature. The solutions of the most exciting open problems might be unknown. But the most exciting open problems themselves might be unknown, too. We are confident that creating the pipeline will unveil many of these problems.

15 Acknowledgments

In June of 1994, I assembled a task force to reflect on the needs to reform computational geometry and initiate a community-wide debate. Its members were: Nina Amenta (Xerox PARC), Tetsuo Asano (Osaka Electro-Comm. U.), Gill Barequet (Tel Aviv U.), Marshall Bern (Xerox PARC), Jean-Daniel Boissonnat (INRIA), John Canny (U.C. Berkeley), Bernard Chazelle (**Chair**, Princeton U.), Ken Clarkson (AT&T Bell Laboratories), David Dobkin (Princeton U.), Bruce Donald (Cornell U.), Scot Drysdale (Dartmouth U.), Herbert Edelsbrunner (U. Illinois at Urbana-Champaign), David Eppstein (U.C. Irvine), A. Robin Forrest (U. East Anglia), Steve Fortune (AT&T Bell Laboratories), Ken Goldberg (U.C. Berkeley), Michael Goodrich (Johns Hopkins U.), Leonidas J. Guibas (Stanford U.), Pat Hanrahan (Stanford U.), Chris M. Hoffmann (Purdue U.), Dan Huttenlocher (Cornell U.), Hiroshi Imai (U. Tokyo), David Kirkpatrick (UBC), D.T. Lee (Northwestern U.), Kurt Mehlhorn (Max Planck Inst.), Victor Milenkovic (U. Miami), Joe Mitchell (SUNY at Stony Brook), Mark Overmars (U. Utrecht), Richard Pollack (Courant Institute, NYU), Raimund Seidel (U. Saarbrücken), Micha Sharir (Tel Aviv U. and NYU), Jack Snoeyink (UBC), Godfried Toussaint (McGill U.), Seth Teller (MIT), Herb Voelcker (Cornell), Emo Welzl (ETH Zürich), and Chee Yap (Courant Institute, NYU).

Although the Task Force never met physically as a whole, a considerable amount of exchanges and discussions ensued. People outside the Task Force also contributed copiously to its deliberations. The present document summarizes some of these reflections. It evolved from bits and pieces submitted by individual members. The final assembly and editing of the document (ie, the final typos) are mine.

I thank the Task Force members for their enthusiastic response and the tremendous amount of work they put into it. I also wish to acknowledge the helpful comments of Alok Aggarwal, Timothy Baker, Paul Callahan, Martin Held, Robert Lupton, Prabhakar Raghavan, Kumar Ramaiyer, Subhash Suri, Roberto Tamassia, and Jovan Zagajac. Thanks to all.

Bernard Chazelle
Princeton, April 1996

References

[1] Agarwal, P.K., Sharir, M., Toledo, S. *Applications of parametric searching in geometric optimization*, Proc. 3rd ACM-SIAM Symp. Disc. Alg. (1992), 72–82.

[2] Airey, J.M., Rohlf, J.H., Brooks, F.P. *Towards image realism with interactive update rates in complex virtual building environments*, ACM Siggraph, Special Issue on 1990 Symp. Interactive 3D Graphics 24 (1990), 41–50.

[3] Alt, H., Behrends, B., Blömer, J. *Measuring the resemblance of polygonal shapes*, Proc. 7th Ann. ACM Symp. Comp. Geom. (1991), 186–193.

[4] Arge, L. *The buffer tree: a new technique for optimal I/O-algorithms*, in *Workshop on Data Structures and Algorithms*, 1995.

[5] Arkin, E.M., Fekete, S., Mitchell, J.S.B. *Optimal tours for lawnmowing and milling*, SUNY Stony Brook, 1995. An earlier paper appears in the 5th Canad. Conf. on Comp. Geom., 1993, 461–466.

[6] Arkin, E.M., Held, M., Smith, C.L. *Optimization problems related to zigzag pocket machining*, Proc. 6th ACM-SIAM Symp. Disc. Alg. (1996), 419–428.

[7] Armstrong, G.T., Carey, G.C., dePennington, A. *Numerical Code Generation From a Geometric Modeling System*, Solid Modeling by Computers: From Theory to Applications, ed. M.S. Pickett and J.W. Boyse, Plenum Press, New York, 1984.

[8] Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A. *An optimal algorithm for approximate nearest neighbor*, Sympos. Disc. Alg., 1994.

[9] Asano, T., Kimura, S. *Contour representation of an image with applications*, Proc. SPIE's International Symposium on Vision Geometry IV, San Diego (July 1995), 14–22.

[10] Asano, T., Chen, D.Z., Katoh, N., Tokuyama, T. *Polynomial-time solutions to image segmentation*, Proc. of the 7th Ann. SIAM-ACM Conference on Discrete Algorithms (Jan. 1996), 104–113.

[11] Atherton, P., Earl, C., Fred, C. *A graphical simulation system for dynamic five-axis NC verification*, Autofact Show of the Society of Manufacturing Engineers, Detroit, Nov. 1987, 2-1 - 2-12.

[12] Baird, H.S. *Mode-Based Image Matching Using Location*, MIT Press, 1985.

[13] Baker, T.J. *Developments and trends in three-dimensional mesh generation*, Appl. Numerical Mathematics 5 (1989) 275–304.

[14] Barequet, G., Sharir, M. *Piecewise-linear interpolation between polygonal slices*, CVGIP: Image Understanding, to appear.

[15] Bern, M., Eppstein, D. *Mesh generation and optimal triangulation*, In "Computing in Euclidean Geometry" (2nd Edition). Eds. D.-Z. Du and F.K. Hwang, World Scientific Press (1995), 47–123.

[16] Boissonnat, J.-D. *Shape reconstruction from planar cross sections*, Computer Vision, Graphics and Image Processing 44 (1988), 1–29.

[17] Boissonnat, J.-D., Faugeras, O.D. *Triangulation of 3-D objects*, Proc. 7th Int. Joint Conf. on Artificial Intelligence (1981), 658–660.

[18] Boissonnat, J.-D., Geiger, B. *Three-dimensional reconstruction of complex shapes based on the Delaunay triangulation*, eds., Acharya, R.S. and Goldgof, D.B. In "Biomedical Image Processing and Biomedical Visualization", 1905 (1993), SPIE, 964–975.

[19] Boissonnat, J.-D., Teillaud, M. *On the randomized construction of the Delaunay tree*, Theoret. Comput. Sci. 112 (1993), 339–354.

[20] Bouma, W., Fudos, I., Hoffmann, C.M., Cai, J., Paige, R. *A geometric constraint solver*, Computer Aided Design 27 (1995), 487–501.

[21] Brost, R., Goldberg, K. *A complete algorithm for synthesizing modular fixtures for polygonal parts.* IEEE Transactions on Robotics and Automation, 1996. To appear.

[22] Brunner, R., Ramaiyer, K., Szalay, A., Connolly, A., Lupton, R. *An object-oriented approach to astronomical databases*, Proc. 4th Ann. Conf. Astronomical Data Analysis Software and Systems, Baltimore, 1994.

[23] Buttenfield, B., McMaster, R., eds. *Map Generalization: Making Rules for Knowledge Representation*, John Wiley & Sons, 1991.

[24] Callahan, P., Goodrich, M., Ramaiyer, K. *Topology B-trees and their applications*, Workshop on Data Structures and Algorithms, 1995.

[25] Callahan, P., Kosaraju, S.R. *A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields,* Proc. 24th Ann. ACM Sympos. Theory Comput. (1992), 546–556.

[26] Canny, J. *The Complexity of Robot Motion Planning*, M.I.T. Press, Cambridge, 1988.

[27] Canny, J., Goldberg, K. *A RISC approach to sensing and manipulation*, Journal of Robotic Systems, 12(6):351–362, June 1995.

[28] Cass, T.A. *Feature matching for object localization in the presence of uncertainty*, Proc. IEEE Conf. Computer Vision and Pattern Recognition (1990), 360–364.

[29] Chalasani, P., Motwani, R., Rao, A. *Approximating tsp variants for robot grasp and delivery*, in preparation, July 1994.

[30] Chappel, I.T. *A new approach to automatic tool path generation for numerically controlled milling machines*, Proc. 4^{th} Int. Conf. on Manufacturing Engineering, Brisbane, Australia, May 1988, 29–32.

[31] Chazelle, B. *Computational geometry: a retrospective*, In "Computing in Euclidean Geometry" (2nd Edition), Eds. D.-Z. Du and F. Hwang, World Scientific Press (1995), 22–46.

[32] Chazelle, B., Dobkin, D.P., Shouraboura, N., Tal, A. *Strategies for polyhedral surface decomposition: an experimental study*, Computational Geometry: Theory and Applications, 1995, to appear.

[33] Chazelle, B., Palios, L. *Decomposition algorithms in geometry*, Algebraic Geometry and its Applications, C. Bajaj, Ed., Chap.27, Springer-Verlag (1994), 419–447.

[34] Chiang, Y.J. *Experiments on the practical I/O efficiency of geometric algorithms: distribution sweep vs. plane sweep*, Workshop on Data Structures and Algorithms, 1995.

[35] Choi, J., Sellen, J., Yap, C. *Precision-sensitive Euclidean shortest path in 3-space*, Proc. 11th Ann. ACM Symp. Comput. Geom. (1995), 350–359.

[36] Clarkson, K.L. *Randomized geometric algorithms,* In "Computing in Euclidean Geometry", Eds. D.-Z. Du and F. Hwang, World Scientific Press (1992), 117–162.

[37] Cohen, M.F., Wallace, J.R. *Radiosity and Realistic Image Synthesis*, Academic Press Professional, San Diego, CA (1993).

[38] Connolly, T.H. *Analytical molecular surface calculation*, J. Appl. Cryst. 16 (1983), 548–558.

[39] Cook, L.T., Cook, P.N., Lee, K.R., Batnitzky, S., Wong, B.Y.S., Fritz, S.L. Ophir, J., Dwyer III, S.J., Bigongiari, L.R., Tempelton, A.W. *An algorithm for volume estimation based on polyhedral approximation*, IEEE Trans. Biomedical Engineering 27 (1980), 493–500.

[40] Crippen, G.M., Havel, T.F. *Distance Geometry and Molecular Conformation*, Wiley, New York, 1988.

[41] Daniels, K., Milenkovic, V.J. *Multiple translational containment. Part I: An approximation algorithm*, Algorithmica, special issue on CG in manufacturing, to appear.

[42] Delfinado, C. J. A. and Edelsbrunner, H. *An incremental algorithm for betti numbers of simplicial complexes*, Proc. 9th Ann. Symp. Comput. Geom., 1993, 232–239.

[43] Drysdale, R.L., Jerard, R.B., Schaudt, B., Hauck, K. *Discrete simulation of NC machining*, Algorithmica 4 (1989), 33–60.

[44] Dobkin, D., and Teller, S. *Computer Graphics*, To appear in "CRC Handbook of Discrete and Computational Geometry" XXX publisher, YYY city (1996).

[45] Edelsbrunner, H. *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[46] Edelsbrunner, H. *The union of balls and its dual shape*, Disc. Comput. Geom. 13 (1995), 415–440.

[47] Edelsbrunner, H., Mücke, E.P. *Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms*, ACM Trans. Graphics, 9 (1990), 66–104.

[48] Edelsbrunner, H., Mücke, E.P. *Three-dimensional alpha shapes*, ACM Trans. Graphics 13 (1994), 43–72.

[49] Emiris, I., Canny, J. *A general approach to removing degeneracies*, SIAM J. Comput. 24 (1995), 650–664.

[50] ESRI White Paper Series. Environmental Systems Research Institute, Inc. *ARC/INFO: GIS Today and Tomorrow*, Mar. 1992.

[51] Federal Geomatics Bulletin, 4(1), 1992. GIS Division, Energy, Mines and Resources. Ottawa, Canada.

[52] Fortune, S., Van Wyk, C. *Efficient exact arithmetic for computational geometry*, Proc. 9th ACM Symp. Comp. Geometry (1993), 163-172.

[53] Fridshal, R., Cheng, K.P., Duncan, D., Zucker, W. *Numerical control part program verification system*, Proc. Conf. CAD/CAM Technology in Mechanical Engineering, MIT (March 1982), MIT Press, 236-254.

[54] Funkhouser, T., Séquin, C., and Teller, S., *Management of Large Amounts of Data in Interactive Building Walkthroughs*, Proc. 1992 Workshop on Interactive 3D Graphics, 11-20.

[55] Funkhouser, T., Séquin, C. *Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments*, Computer Graphics 27, Proc. Siggraph (1993), 247–254.

[56] Gigus, Z., Canny, J.F., Seidel, R. *Efficiently computing and representing aspect graphs of polyhedral objects*, IEEE Trans. Pat. Anal. Mach. Intel. 13 (1991), 542–551.

[57] Gold, C. M. *Problems with handling spatial data—the Voronoi approach*, CISM Journal 45 (1991), 65–80.

[58] Gold, C. M. *An object-based dynamic spatial model, and its application in the development of a user-friendly digitizing system*, Proc. 5th Intl. Symp. Spatial Data Handling, IGU Commission on GIS (1992), 495–504.

[59] Goldberg, K., Latombe, J.-C., Halperin, D., Wilson, R. editors. *The Algorithmic Foundations of Robotics: First Workshop.* A. K. Peters, Boston, MA, 1995.

[60] Goodchild, M. F. *A spatial analytical perspective on geographical information systems*, Int. J. GIS 1 (1987), 327–334.

[61] Goodchild, M. F. *Geographic information systems and cartography*, Cartography 19 (1990), 1–13.

[62] Goodchild, M. F. *Issues of quality and uncertainty*, J. C. Muller, editor, Advances in Cartography, Elsevier Applied Science, London (1991), 113–139.

[63] Goodchild, M. F., Kemp, K. K., Poiker, T. K. *NCGIA Core Curriculum*, National Center for Geographic Information Analysis, University of California, Santa Barbara CA 93106–4060, USA, 1989. On WWW at `http://www.env.gov.bc.ca/~tclover/giscourse/ToC.html`.

[64] Goodman, J.E., O'Rourke, J. (eds), *CRC Handbook of Discrete and Computational Geometry*, CRC Press, Boca Raton, to appear.

[65] Greengard, L.F. The Rapid Evolution of Potential Fields in Particle Systems, The MIT Press, 1988.

[66] Guibas, L.J., Marimont, D. H. *Rounding arrangements dynamically*, Proc. 11th Ann. Symp. Comput. Geom., 1995 (190–199).

[67] Hamann, B. and Moorhead, R.J. *A survey of grid generation methodologies, the National Grid Project, and scientific visualization efforts at the NSF Engineering Research Center for Computational Field Simulation*, to appear in: Nielson, G. M. et al., eds., IEEE Computer Society Press, Los Alamitos, California.

[68] Heckbert, P.S., Garland, M. *Fast polygonal approximation of terrains and height fields*, Technical Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[69] Held, M. *On the computational geometry of pocket machining*, Vol. 500 of *Lecture Notes in Computer Science.* Springer-Verlag, June 1991.

[70] Hobby, J. *Practical segment intersection with finite precision arithmetic*, Manuscript, AT&T Bell Labs, October 1993.

[71] Hoffmann, C.M. *The problems of accuracy and robustness in geometric computation*, IEEE Computer 22 (1989), 31–42.

[72] Hoffmann, C.M. *Geometric and Solid Modeling*, Morgan Kaufmann, 1989.

[73] Hoffmann, C.M. *Geometric approaches to mesh generation*, in "Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations", eds. I. Babuska, J. Flaherty, W. Henshaw, J. Hopcroft, J. Oliger, T. Tezduyar, Springer Verlag, IMA Volumes in Mathematics and its Applications 75 (1995).

[74] Hoffmann, C.M., Vermeer, P.J. *Geometric constraint solving in R^2 and R^3*, Computing in Euclidean Geometry, eds. D. Z. Du and F. Hwang, World Scientific Publishing, 2nd ed., 1994.

[75] Ho-Le, K. *Finite element mesh generation methods: a review and classification*, Computer-Aided Design 20 (1988) 27–38.

[76] Hunt, W.A., Voelcker, H.B. *An exploratory study of Automatic verification of programs for numerically controlled machine tools*, Production Automation Project Tech Memo No. 34, University of Rochester, Jan. 1982.

[77] Hutchinson, S. *Exploiting visual constraints in robot motion planning*, IEEE Conference on Robotics and Automation (1991), 1722–1727.

[78] Huttenlocher, D.P., Kedem, K., Sharir, M., *The upper envelope of Voronoi surfaces and its applications*, Disc. Comp. Geom. 9 (1993), 267–291.

[79] Huttenlocher, D.P., Klanderman, G.A., Rucklidge, W.J. *Comparing images using the Hausdorff distance*, IEEE Trans. Pat. Anal. Mach. Intel. 15 (1993), 850–863.

[80] Huttenlocher, D.P., Ullman, S. *Recognizing solid objects by alignment with an image*, Int. J. Computer Vision, 5 (1990), 195–212.

[81] Iles, K. *Data considerations – Some principles for accuracy and reliability of data in a GIS system*, GIS'89 Symposium (1989), Vancouver, Canada, 103.

[82] Information Technology for Manufacturing: A Research Agenda, *National Academy Press*, 1995.

[83] Kent, S.M., Stoughton, C., Newberg, H., Loveday, J. Petravick, D., Gurbani, V., Berman, E., Sergey, G. *Sloan digital sky survey*, Proc. 3rd Ann. Conf. Astronomical Data Analysis Software and systems, Victoria, British Colombia, 1993.

[84] Kirkpatrick, D., Mishra, B., Yap, C.K. *Quantitative Steinitz's theorems with applications to multifingered grasping*, Disc. Comput. Geom. 7 (1992).

[85] Klein, H.M, Schneider, W., Alzen, G., Voy, E.D., Günter, R.W. *Pediatric craniofacial surgery: Comparison of milling and stereolithography for 3D model manufacturing*, Pediatric Radiology 22 (1992), 458–460.

[86] Kumler, M. P. *Intensive comparison of triangulated irregular networks (TINS) and digital elevation models (DEMS)*, Cartographica 31 (1994), monograph 45.

[87] Lamdan, Y., Wolfson, H.J. *Geometric hashing: A general and efficient model-based recognition scheme*, International Conference on Computer Vision (1988), 238–249.

[88] Latombe, J.-C. *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[89] Laurini, R., Thompson, D. *Fundamentals of spatial information systems.* Academic Press, London, 1992.

[90] Lazanas, A., Latombe, J.-C. *Landmark-based robot navigation*, American Association for Artificial Intelligence Conference (1992), 816–822.

[91] Lee, B., Richards, F.M. *The interpretation of protein structures: estimation of static accessibility*, J. Mol. Biol. 55 (1971), 379-400.

[92] Li, S.X., Jerard, R.B. *5-Axis Machining of Sculptured Surfaces with a Flat-End Cutter*, CAD, 26 (1994), 165–178.

[93] Löhner, R. *Finite elements in CFD: what lies ahead*, Int. J. Numer. Meth. Eng. 24 (1987), 1741–1756.

[94] Lorensen, W.E., Cline, H.E. *Marching cubes: A high resolution 3D surface construction algorithm*, Computer Graphics 21 (1987), 163–169.

[95] Lumelsky, V. *Algorithmic and complexity issues of robot motion in an uncertain environment*, Journal of Complexity 3 (1987), 146–182.

[96] Markenscoff, X., Ni, L., Papadimitriou, C.H. *The geometry of grasping*, IJRR 9 (1990).

[97] Matoušek, J. *Geometric range searching*, ACM Comput. Surveys 26 (1995), 421–461.

[98] Mehlhorn, K. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, Germany, 1984.

[99] Menon, J.P., Robinson, D.M. *Advanced verification via massively parallel raycasting*, Manufacturing Review 6 (1993), 141–154.

[100] Menon, J.P., Voelcker, H.B., *Towards a comprehensive formulation of NC verification as a mathematical and computational problem*, J. Design and Manufacturing 3 (1993), 263-277.

[101] Milenkovic, V.J. *Multiple translational containment. Part II: Exact algorithms*, Algorithmica, special issue on CG in manufacturing, to appear.

[102] Milenkovic, V.J. *Robust polygon modeling*, Special issue of Computer-Aided Design on Uncertainties in Geometric Computations, 25 (1993), 546–566.

[103] Milenkovic, V.J., Li, Z. *A compaction algorithm for nonconvex polygons and its application*, European J. Operations Research, 84 (1995), 539–560,

[104] Milenkovic, V.J. *Practical methods for set operations on polygons using exact arithmetic*, Proc. 7th Canad. Conf. Comput. Geom., 1995 (55-60).

[105] Milenkovic, V.J. *Translational polygon containment and minimal enclosure using linear programming based restriction*, Proc. 28th Ann. ACM Symp. Theory Comput. (to appear) 1996.

[106] Moore, B., Frenk, C.S., Weinberg, D.H., Saunders, W., Lawrence, A., Ellis, R.S., Kaiser, N., Efstathiou, G., Rowan-Robinson, M. *The topology of the QDOT IRAS redshift survey*, Monthly Notices Royal Astronomical Society, 256 (1992), 477–499.

[107] Müller, H., Klingert, A. *Surface interpolation from cross sections*, in: Focus on Scientific Visualization (H. Hagen, H. Müller, and G.M. Nielson, eds.), Springer Verlag, Berlin, 1993, 139–189.

[108] Mulmuley, K. *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice-Hall, 1994.

[109] Murray, R., Li, Z., Sastry, S. *A Mathematical Introduction to Robotics Manipulation.* CRC Press, 1994.

[110] Näher, S. *The LEDA user manual*, Version 3.1 (Jan. 16, 1995). Available by anonymous ftp from `ftp.mpi-sb.mpg.de` in directory `/pub/LEDA`.

[111] Navazo, I., Ayala, A.D., Brunet, P. *A geometric modeler based on the exact octree representation of polyhedra*, Computer Graphics Forum, 5, 91-104.

[112] Okabe, A., Boots, B., Sugihara, K. *Nearest neighbourhood operations with generalized Voronoi diagrams: A review.* Int. J. GIS 8 (1994), 43–71.

[113] Oliver, J.H., Goodman, E.D., *Color graphic verification of NC milling programs for sculptured surfaces*, 10th Ann. Automotive Computer Graphics Conference and Exposition, Engineering Society of Detroit, Dec. 1985.

[114] O'Rourke, J. *Computational Geometry in C*, Cambridge Univ. Press, 1994.

[115] Overmars, M., Rao, A., Schwarzkopf, O., Wentink, C. *Immobilizing polygons against a wall*, ACM Symp. Comput. Geom. (1995), Vancouver, BC.

[116] Papadimitriou, C.H., Yannakakis, M. *Shortest paths without a map*, Theoretical Computer Science 84 (1991), 127–150.

[117] Pellegrini, M. *Monte Carlo approximation of form factors with error bounded a priori*, Proc. 11th Ann. ACM Symp. Comput. Geom. (1995), 287–296.

[118] Perkal, J. *On the length of empirical curves*, Discussion Paper 10, Michigan Inter-University Community of Mathematical Geographers, University of Michigan, Ann Arbor, 1966.

[119] Peucker, T. K., Fowler, R. J., Little, J. J., Mark, D. M. *The triangulated irregular network*, Amer. Soc. Photogrammetry Proc. Digital Terrain Models Symposium (1978), 516–532.

[120] Preparata, F.P, Shamos, M.I. *Computational Geometry: an Introduction*, Springer-Verlag, New York, 1988.

[121] Pribble, W.I. *Molds for reaction injection, structural foam and expandable styrene molding*, in: Plastics mold engineering handbook, J.H. DuBois and W.I. Pribble (Eds.), Van Nostrand Reinhold Company Inc., New York, 1987.

[122] Quinn, J., *Accurate Verification of Five-Axis Numerically Controlled Machining*, Ph.D. Thesis, Dartmouth College, May 1993.

[123] Ramaiyer, K., Brunner, R., Szalay, A., Connolly, A., Lupton, R. *Prototype astronomical database*, Proc. 2nd ACM Workshop on Advances in Geographic Information Systems, Gaithersburg, Maryland, 1994.

[124] Rao, A., Goldberg, K. *Manipulating algebraic parts in the plane*, IEEE Transactions on Robotics and Automation 11 (1995).

[125] Raper, J., ed. *Three Dimensional Applications in Geographic Information Systems*, Taylor & Francis, 1989.

[126] Richards, F.M. *Areas, volumes, packing, and protein structures*, Ann. Rev. Biophys. Bioeng. 6 (1977), 151–176.

[127] Richards, F.M. *The protein folding problem*, Scientific American 264 (1991), 54–63.

[128] Rucklidge, W.J. *Locating objects using the Hausdorff distance*, Proc. 5th Int. Conf. on Computer Vision (1995), 457-464.

[129] Sack, J.-R., Urrutia, J. (eds), *Handbook of Computational Geometry*, Elsevier Science Publishers, in preparation.

[130] Sapidis, N. and Perucchio, R. *Advanced techniques for automatic finite element meshing from solid models*, Computer-Aided Design 21 (1989) 248–253.

[131] Schumaker, L.L. *Reconstructing 3D objects from cross-sections*, in: Computation of Curves and Surfaces (W. Dahmen, M. Gasca, and C.A. Micchelli, eds.), Kluwer Academic Publishers, 1989, 275–309.

[132] Schwartz, J.T., Sharir, M. *On the piano movers' problem: 2. general techniques for computing topological properties of real algebraic manifolds*, Advances in Applied Mathematics 4 (1983), 298–351.

[133] Sharir, M. *Almost tight upper bounds for lower envelopes in higher dimensions*, Disc. Comput. Geom. 12 (1994), 327–345.

[134] Skiena, S.S. *Problems in geometric probing*, Algorithmica 4 (1989), 599–605.

[135] Speen, R.B. *A dynamic approach to modeling for numerically controlled verification*, Masters Thesis, Purdue University, 1985.

[136] Stucki, P., Bresenham, J., Earnshaw, R., Guest Editors. *Rapid prototyping technology*, Special issue of IEEE Comp. Graphics and Applications 15 (1995), 17–55.

[137] Teller, S. *Visibility Computations in Densely Occluded Polyhedral Environments*, CS Dept., UC Berkeley (1992), TR UCB/CSD 92/708 or http://cs-tr.cs.berkeley.edu:80/TR/UCB:CSD-92-708.

[138] Van Hook, T. *Real-time shaded NC milling display*, ACM SIGGRAPH, 20, 4, (1986), 15–20.

[139] van Oosterom, P. *Reactive Data Structures For Geographic Information Systems*, Oxford University Press, 1993.

[140] Voelcker, H.B., Requicha, A.A.G., Conway, R.W. *Computer applications in manufacturing*, in "Annual Review of Computer Science", Ed. J. F. Traub, Vol. 3. Palo Alto, CA: Annual Reviews Inc. (1988), 349–387.

[141] Voelcker, H.B., Requicha, A.A.G. *Research in solid modeling at the University of Rochester: 1972-1987*, in "Fundamental Developments of Computer-Aided Geometric Modeling", Ed. L. Piegl, Academic Press Ltd., London, England (1993), 203–254.

[142] Wallack, A., Canny, J. *Planning for modular and hybrid fixtures*, IEEE International Conference on Robotics and Automation, May 1994.

[143] Wang, W.P., Wang, K.K. *Real-time verification of multiaxis NC programs with raster graphics*, IEEE Proc. 1986 Int. Conf. Robotics and Automation, San Francisco (April 1986), 166–171.

[144] Wilson, R.H., Kavraki, L., Lozano-Perez, T., Latombe, J.-C. *Two-handed assembly sequencing*, Intl. J. of Robotics Research, 14 (1995), 335–350.

[145] Wozny, M.J., Regli, W.C., Guest Editors. *Computer science in manufacturing*, Special issue of Comm. ACM 39 (1996), 33–85.

[146] Xiang, W.-N. *A GIS method for riparian water quality buffer generation*, Int. J. GIS 7 (1993), 57–70.

[147] Yan, X., Gu, P. *A review of rapid prototyping technologies and systems*, Computer-Aided Design 28 (1996), 307–318.

[148] Yap, C. *Symbolic treatment of geometric degeneracies*, J. Symb. Comput. 10 (1990), 349–370.

[149] Yap, C. *Towards exact geometric computation*, Fifth Canadian Conf. on Computational Geometry (1993), 405–419. To appear, Comp.Geom.Theory and Appl.

[150] Yap, C. *Exact computational geometry and tolerancing metrology*, In "Snapshots of Computational and Discrete Geometry, Vol 3," Eds. by David Avis and Jit Bose, McGill School of Comp.Sci, Tech.Rep. No.SOCS-94.50, 1994. (A special issue dedicated to Godfried Toussaint)

[151] Yap, C. *Report on NSF Workshop on Manufacturing and Computational Geometry*, IEEE Computational Science and Engineering, 2 (1995), 82–84. Full report on web at URL `http://cs.nyu.edu/cs/faculty/yap/index.html`.

[152] Yap, C., Dubé, T. *The exact computation paradigm*, In "Computing in Euclidean Geometry" (2nd Edition). Eds. D.-Z. Du and F.K. Hwang, World Scientific Press (1995), 452–492.

[153] Zagajac, J. *A fast method for estimating discrete field values in early engineering design*, to appear in IEEE CG&A.

[154] Zhuang, Y., Goldberg, K. *On the existence of solutions in modular fixturing*, International Journal of Robotics Research, to appear.