V. Akman

Geometry and graphics applied to robotics

Computer Science/Department of Interactive Systems Report CS-R8727 May

69F21, 69J11, 69K21, 69K34, 69K35

Geometry and Graphics Applied to Robotics

Varol Akman

Department of Interactive Systems
Centre for Mathematics and Computer Science (CWI)
Kruislaan 413, 1098 SJ Amsterdam, the Netherlands

Abstract In the first part of this paper, I review the recent efforts on integrating robotics and computer science ideas. Specifically, I advocate the view that applying results from areas of computer science such as concrete complexity, symbolic computation, and computational geometry will simplify the work of robot programmers. There is now a wealth of interesting results and I only cover a small portion of it.

In the second part, the discussion takes place in the context of model-based robotics. I argue that time has come to build a "Geometer's Workbench," a system integrating geometric knowhow with algorithm animation techniques and interactive graphics to visualize complex situations as encountered in robotics. Such a system is expected to broaden the way geometry is practiced in the style Macsyma has accomplished for algebra.

Keywords and Phrases Minimal paths, Voronoi diagram, continuum method, visibility, polyhedra, computational geometry, concrete complexity, geometer's workbench, algorithm animation, workstation, prototyping, Macsyma, Smalltalk, Model-View-Controller, Lisp.

CR Subject Classification (1987) F.2.2, G.1.5-6, I.1.2, I.2.9, I.3.4-5.

Note This paper will appear in *Theoretical Foundations of Computer Graphics and CAD*, R.A. Earnshaw (ed.), Springer-Verlag, Berlin (1987).

INTRODUCTION

This paper is a personal statement on (i) how robotics may benefit from the advances made in computer science — especially those parts of it dealing with geometric and algebraic objects — and (ii) how this process may be accelerated by building a workbench for experimentation with geometric algorithms.

Part I of the paper is mainly aimed towards robotics practitioners and is slightly of a propaganda nature. I tend to believe that computer science ideas are underused in practical applications and offer a quick review of some of the recent results which I find interesting and deserving merit. The second half of the paper puts the blame for the aforementioned neglect partly on the theoreticians. I am persuaded that if computational geometry is maturing (as many people say) then it is crucial to use its techniques in the professional environment where robotics is a natural candidate. This however is difficult since computational geometers are traditionally most concerned with asymptotic analysis, and in search of elegance, underplay the special cases which are the bugbear of real problems (cf. Forrest [1] for an excellent account of the latter).

My proposal, admittedly the first one that comes to mind, is then a software system, a so-called "Geometer's Workbench," which incorporates geometric

Report CS-R8727
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

knowledge and interactive graphics to assist in experimenting with geometric algorithms, in this case tailored for robotics problems. I study some important functionalities desired in such a workbench in Part II.

When I started working in the area of geometric algorithms for robotics tasks seven years ago, I eagerly looked for references. I remember being able to find a dozen or so relevant papers. It is a remarkable fact that computational geometry has advanced so much since then. There are now literally hundreds of references (including several books) on computational geometry and a substantial number of these papers deal with robotics problems. I can't possibly hope to review fully even the robotics-related work here, let alone mention titles[§]. Hence, in Part I of this paper I review only a certain fraction of the *recent* works. The choice reflects my personal taste and shouldn't be regarded as a list of the most important publications. Finally, a word of caution. What I write in Part II has overlaps with what other people are doing today. If my remarks do not explicitly bear my mark, I wouldn't like to lay claim to them as my property.

PART I

Robotics is crucial for the reindustrialization of the world. It contributes to the safety and flexibility of the manufacturing process and allows productivity to be multiplied many times. However, numerous robotics problems intricately linked with algorithmics remain to be solved satisfactorily. Concrete complexity has much to say about these problems, especially in the calibration their difficulty[#]. It is profitable therefore for a robotics practitioner to be aware of the existing theoretical results.

In robotics, *motion planning* refers to the determination of the route of a robot carrying a workpiece from one location (source) to another (goal) while avoiding clashes with other obstacles. (This is also called the *Findpath problem* after Terry Winograd's SHRDLU.) In a *model-based* robot environment, the robot software has access to a geometric model of the workspace and does motion planning by

[§] The interested reader may contact me for a bibliography that is rather complete. He is also referred to Edelsbrunner et al [2] and van Leeuwen [3] who present fundamental ideas on the relationship between computational geometry and computer graphics.

[#] A fitting example taken from the area of computer vision is the following. Given a straight-edge planar graph, we want to decide if it is the projection of the visible part of a set of opaque polyhedra. Although there exist an extensive artificial intelligence literature on this problem and many practically successful systems, Lefteris Kirousis [CWI, Amsterdam (Jan. 1987)] and Christos Papadimitriou proved that the general problem is NP-complete. Their work was further able to explain the success of those practical works.

referring to it. It is possible to define some variants of Findpath:

- *Warehouseman's problem:* The obstacles are no more required to be fixed; they may be moved around to make space (cf. Winograd's Makespace and Findspace).
- *Generalized motion planning:* The robot under consideration has links, viz. a manipulator arm with several joints. While this is only natural to assume if we are ever going to deal with real robots, one is occasionally tempted to regard robots as points to make life easier.
- *Coordinated motion planning:* There are several objects to be moved in a coordinated fashion so as not to clash with each other (and naturally, with the obstacles).
- *Asteroid avoidance:* What happens when we are traveling in a space ship and have to avoid asteroids? Here, the geometric model of the workspace is constantly changing. There is little work in literature dealing with this problem. It is considered to be difficult [4].
- *Minimal path planning:* Find a path which is not only obstacle-free but is also optimal* with respect to some specified criterion, say the Euclidean (L_2) metric. A variant is the so-called *weighted region problem* where the metrics associated with the several subregions of the workspace are different.

In addition to this list, let us agree to call the version of the minimal path problem for a single polyhedron Boundary (respectively Exterior) Findpath depending on the fact that the source and goal points are on (respectively outside) the given *convex* polyhedron. The single-source many-goals version of these problems will be denoted by the suffix "locus."

In the rest of this section, I review several papers covering problems of the sort listed above. The reader is referred to three recent dissertations — Akman [5], Buckley [6], and Mitchell [7] — for a detailed view of the area. A recent review by Schwartz and Sharir [8] may also be useful although it is rather terse.

* It is not altogether obvious why one should need the minimal paths in practice since these are dangerous paths — they touch the obstacles. Probably what one needs is some kind of safe, middle-of-the-road path. Remember however our remark about the calibration of the difficulty of a problem. It is simply useful to know the complexity of this problem.

1. Ridge points of a polyhedron

Among other works on minimal paths in 3-space, Sharir and Schorr's [9] is one of the most detailed. They mainly consider the case of Boundary Findpath (locus) and present an algorithm which works in time $O(n)$ per query where n is the number of vertices of the polyhedron. Their algorithm is based on the "ridge" points of the polyhedron. A *ridge* point on the polyhedron has the property that there exists at least two minimal paths to it from the source. It turns out that the set of ridge points is a union of line segments and that the union of the vertices and the ridge points is a closed connected set. Defining the union of the latter with the minimal paths from the source to every vertex, one partitions the polyhedron's boundary into disjoint connected regions called "peels" whose interiors are free of vertices and ridge points. The peels can be constructed in time $O(n^3 \log n)$ (preprocessing) using techniques whose implementation may prove difficult. The size of the data structure created by the preprocessing step is $O(n^2)$. David Mount lately improved the time bound by a factor of n.

2. Continuous Dijkstra on an arbitrary surface

Mitchell, Mount, and Papadimitriou [7] give an algorithm to solve Boundary Findpath (locus) for any surface. (It is noted that they are computing geodesics, not true minimal paths.) Theirs is an $O(n^2 \log n)$ time and linear space algorithm for subdividing the surface of an arbitrary polyhedron so that the length of the minimal path from a given source to any goal on the surface is obtainable by simple point-location. This method has striking similarities to Dijkstra's method for minimal paths in graphs. Point-location is achieved in time $O(\log n)$ after which the actual minimal path is backtracked in time proportional to the number of faces that it traverses on the boundary.

Mitchell [7] and Papadimitriou pose a realistic version of the minimal path problem, namely, that of weighted regions. This is motivated by certain problems arising in terrain navigation where a robot must move at different speeds in different types of terrain. In such a problem, one is given a polygonal subdivision of a 2-dimensional manifold in 3-space. Each region (or triangle, without loss of generality) has an associated integer weight. The length of a path is found as the weighted sum of the subpaths through each triangle. The special case of all weights belonging to $\{1,\infty\}$ is equivalent to the usual minimal path problem. To solve the problem they apply a *continuous Dijkstra* technique. They imagine an "expanding wavefront" moving out from the source and record the effects of the discrete events imposing critical changes on it. For n vertices, the time complexity is $O(n^3 L \log n)$ where L is the number of bits required to specify the largest number among the weights and the vertex coordinates.

3. Islands in a workspace

Reif and Storer [10] introduce the notion of an *island* in minimal path computation. For 2-dimensions, an island is defined to be a connected component of the obstacle space. For 3-space, an island is defined to be a maximal convex obstacle surface such that for any two points contained in the island, the minimal path between them is also within the island. Thus, k convex and disjoint polyhedra constitute k islands; a single nonconvex polyhedron constitutes *more than one* island. Let n be the total number of vertices and k be the number of islands. For 3-space they give an algorithm that runs in $O(n^{k^{O(1)}})$ time and another one that consumes $O(n^{\log k})$ space. Both algorithms are based on applications of George Collins' theorem which states that a formula in the theory of real closed fields can be checked for satisfiability in time $O(d\, m^{2^{O(r)}})$ if it has length m, degree d, and r quantifiers. Reif and Storer also give another algorithm for 2-space which runs in preprocessing time $O(n\ (k + \log n))$ and query time $O(\log n)$.

4. Fully polynomial approximation of minimal paths

Papadimitriou [11] gave a *fully polynomial approximation* scheme for the general 3-dimensional minimal path problem, i.e. given an instance of the problem and a real $\varepsilon > 0$, his method returns a path whose length is less than $(1+\varepsilon)$ times the minimal path length, in time polynomial in the size of the instance (say n, the number of vertices) and ε^{-1}. The basic idea in Papadimitriou's method is to break the edges in the scene into a number of short segments (calculated in a tricky way) so that the resulting relative error in the calculation of minimal distance is at most ε. After the segmentation, Dijkstra's algorithm is used. Specifically, let $\psi = L + \log n \varepsilon^{-1}$ be the precision of an instance. Here L is largest integer (coordinate) used in the instance. Then Papadimitriou gave two methods one of which is simpler and takes time $O(n^4 \psi^2 \varepsilon^{-2})$ whereas the other is rather messy yet improves the preceding bound by a factor of $n \varepsilon^{-1}$.

5. A property of L_1-metric Delaunay triangulations

Chew [12] discovered a nice property of Delaunay triangulations which is useful for planning approximate paths in the plane. Consider a set of n points in the plane. Then Chew showed that there is a graph on this set in which L_2 distances between point pairs are at most $\sqrt{10}$ times the minimal L_2 distances. The graph, being planar, has only $\Theta(n)$ edges and accordingly can be searched for an approximate path between the source and the goal points in $O(n \log n)$ time using Dijkstra's algorithm. It is constructed as follows. Consider the L_1-metric Voronoi diagram of the given set. The Delaunay triangulation of the set is the straight-line dual of the diagram and can clearly be obtained from scratch in $O(n \log n)$ time. Now define the *circle graph* derived from this triangulation. The vertex set of the circle graph is equal to the vertices of the triangulation; there are three edges for each triangle of the triangulation, corresponding to three arcs of a circle (which is a square tipped at 45^o due to the L_1-metric!) circumscribed about the triangle. The circle graph is the required graph.

6. Continuum method for path planning

Buckley [6] started his work on continuum method for path planning under the hypothesis that it may sometimes be preferable to combinatorial search in the configuration space. *Continuum method* works directly in the continuous problem space in contrast to say a visibility graph-based combinatorial method. It has an infinite number of possible states. Since there are an infinite number of choices within a bounded region those choices constitute a densely packed field, or continuum. Obviously, continuum method *still* has to replace its continuous search space with a combinatorial search space equivalent according to some criterion. The transformation makes a difficult problem more tractable but usually at the expense of a degree of nonequivalence between the problems.

Buckley introduces a real-valued continuous function which is defined for all relative positions of two convex bodies and characterizes their collision boundary. He then gives algorithms for the computation of this function. His path planning algorithm is based on the use of numeric methods both to detect and eliminate inadmissible trajectory states. He compares it experimentally with Rodney Brooks' combinatorial method for solving planar free-body path planning problems. His experiments seem to indicate a very interesting tradeoff. For simple problems Brooks' program works more efficiently while Buckley's algorithm is able to handle more difficult problems more quickly.

7. From multiple object motion planning to searching

Hopcroft and Wilfong [13] studied the motion planning problem for multiple objects. An object is a 2-dimensional region whose edges are rectilinearly oriented line segments; only translations are allowed with them. Consider the configuration space of n such objects. It is clearly equal to R^{2n}. In particular consider Γ, the set of all points in configuration space that correspond to configurations of objects where they form a single connected component. They prove that Γ consists of facets of various dimensions such that if there is a path in Γ between two vertices then there is path between them along the edges of Γ. In an earlier work [14] the authors proved that if there is a motion between two configurations of Γ then there is a path in Γ between the configurations. It follows therefore that a motion between two vertices of Γ implies a motion along the edges of Γ. Thus the motion planning problem has been reduced from searching the high-dimensional space to a graph searching problem.

8. Minimal paths in 3-space for a set of convex obstacles

Sharir and Baltsan [15] studied the minimal path problem for k convex polyhedra with a total of n vertices. They first describe an algorithm which calculates the minimal path amidst two obstacles (i.e. $k=2$) in time almost equal to $O(n^3 \log n)$. (The exact bound is a bit too technical.) They also show that if $k>2$ then the minimal path can be computed in time polynomial in n but exponential in k. This follows from a result of theirs regarding the number of *minimal path edge sequences,* i.e. sequences of edges on a convex polyhedron for which there exist two points on the boundary such that the minimal path between these points crosses them. Sharir and Baltsan proved that a convex polyhedron with n vertices has $O(n^7)$ minimal path edge sequences.

PART II

One of the most exciting developments in computer graphics is the emergence of powerful workstations. Access to the computing power embedded in these machines is revolutionizing the way research is being done. In this section I shall discuss the geometric and graphical functionalities expected from such workstations to make them geometric workbenches. Although I do not have a bias for using a geometric workbench in any specific area (i.e. it should be a general system), robotics is a good subject area to profit from the existence of such an advanced geometric and graphic aid.

1. Macsyma[†] as a role model

My preliminary thoughts about a geometer's workbench owe to Macsyma [16], a sophisticated computer algebra system built to assist researchers in solving mathematical problems. A user enters symbolic input to Macsyma which in return yields symbolic output. A great deal of knowledge has been stored into Macsyma's knowledge base. It is thought that 100 man-years went into the development of the system which comprises about 300,000 lines of code. In Macsyma the user has access to mathematical techniques which he may not even fully understand but can easily employ to solve his problem. Conjectures can be tested easily and fast with Macsyma. The system is simple to use but not at the

[†] Macsyma is a registered trademark of Symbolics, Inc.

expense of being simplistic; several problems may require serious programming in the Macsyma command language and mastering the "insides" of the system. In short, Macsyma gives the user room to study problems from a more intellectual viewpoint, i.e. leaving the low-level, uninteresting computational details to the computer. It also offers an extensible and "exploratory" programming environment. As a matter of fact, there is already a large amount of specialized Macsyma code in the so-called Share libraries.

Doing geometry, like algebra and many other research endeavours, is an iterative process. We define problems, draw figures, pose conjectures, redraw things, revise our ideas, etc. This exploratory process must be equipped with effective aids to graph data, to draw 2- and 3-dimensional figures to convey as much as possible, to discover properties, and to store all this information in a meaningful and easily retrievable format. These tools must not require a large amount of initial training but have to be powerful. Generally, they should only make their functionality apparent to a user, but when required the internals of the system should allow reprogrammability and editability. These requirements are satisfied in one way or another by Macsyma. The Lisp programming environment that has evolved during the past two decades of artificial intelligence research also delivers these. Assorted Lisp machines, for instance, combine the Lisp programming environment with powerful graphics. Since Macsyma's base language is Lisp, these machines naturally support Macsyma. The top level of a Lisp system is a read-evaluate-print loop that reads expressions from the input stream, evaluates them, and prints the outcome on the output stream. Flexible structure editors, debuggers, and execution tracers provide a rich environment for *rapid prototyping,* an emerging pragmatic philosophy in software development. Windows enhance the interaction and menus and use of a pointing device such as a mouse frees the user from being keyboard-bound. All of the above features must be present in the envisaged geometer's workbench.

2. Special cases mustn't be so special

What kind of geometric knowhow would one expect from a workbench built upon such a workstation? First and foremost, it must be possible to perform conceptually trivial operations such as Voronoi partitions, convex hulls, polyhedral boolean operations, and so on without undue emotional trauma. It is known that implementation of even the simplest geometric algorithms is difficult because of numerical problems and the number of special cases that warrant special care [1]. Franklin et al [17] mention the case of intersecting two polygons, a seemingly trivial operation which can result in about 1,000 lines of Fortran code once all the cases, such as polygons with multiple components that may or may not intersect the other polygon and whose edges may coincide with other edges or vertices, are taken into account.

Algorithm and data structure animation techniques [18] are found to be of crucial assistance in this respect. Since a personal workstation has a much friendlier user interface, the user may gain an insight by real-time observation of the outputs from algorithms. A good example for the need for the latter is derived from the weakness of a bare asymptotic analysis of an algorithm. It is not clear how efficient some of the asymptotically optimal e.g. Voronoi and point-location algorithms are when applied to scenes with moderate complexity. A theoretically ingenious algorithm of Richard Lipton and Robert Tarjan for planar point-location had a notice for the reader stating that the authors didn't think of it as suitable for implementation.

3. Improvising with geometric objects

Another key requirement for a geometer's workbench is the availability of good update facilities. This refers to the user's ability to add new geometric objects or delete the existing ones. It also embodies the concept of modifying (operating on) existing objects to obtain new ones. For instance, one should be able to take a cube, slice it in the corners, drill a hole in its middle to obtain a new object, and give it a name. One must be able to take the convex hull of say three polyhedra and create a new polyhedron. I call this kind of liberal approach in dealing with geometric objects "improvisation." In a very elegant early work Baumgart [19] and recently, Fogg and Eades [20] made some efforts in this respect. Pentland's [21, 22] work depicts probably the state-of-art in supporting improvisation.

The preceding operations require that the system has a good understanding of what an object is. For example, if a cube has a hole it means that one can have a sufficiently small object pass through that hole. This would be trivial knowledge had the system possess a pair of eyes but in the lack of that it has to be stored in some way along with the cube. Similarly, it is normally an illicit operation to take the convex hull of two polyhedra, for the convex hull operation is defined for a set of points. However, the operation makes sense and must be allowed once it is understood that one is in fact dealing with the vertices of the polyhedra under consideration.

As another exercise in friendly visual interface, consider the following problem. Construct the Voronoi tessellation of the 3-dimensional space by a given set of points. The question arises: How can one present the output in the most meaningful manner? Color would help, transparency would help, and finally an ability to selectively review regions would help.

4. Visual aids for robotics

How does the Voronoi diagram on the boundary of a convex polyhedron change when the source point moves? Theoretically, this would amount to parametrizing the diagram's edge set with respect to the source coordinates so that the way they change while the source moves on the boundary can be guessed. Note however that the change in the diagram will by no means be continuous, i.e. there will be certain "jump" points at which the diagram on a given face of the polyhedron will gain a new topology. Accounting for this effect seems involved. Randolph Franklin [private communication (1985)] suggested that one can make movies showing the effect of different locations of the source to study this problem experimentally.

Given a boundary description for a polyhedron, one may be required to determine where the holes are. This problem has been completely solved with the well-known classification of 2-manifolds. However I am not aware of any practical program doing this for a given polyhedral description. Also note that, as long as the source and/or the goal is not inside it, a *bounded cavity* cannot contribute to the minimal path computation and thus can be "filled."[%] Curved objects make minimal path computations extremely difficult, e.g. there may be an innumerable number of minimal paths. This is a domain where utilization of the variational calculus techniques may prove useful. Minimal paths on fancy objects such as Möbius bands and Klein bottles are also confusing.

When subdividing the space to compute minimal paths to any goal around polyhedra [23, 24, 25] we are particularly interested in finding the intersection curve of two arbitrary surfaces efficiently and reliably. The latter requirement necessarily dictates a symbolic (rather than numerical) approach to the problem since there may be all kinds of degeneracies. Another relevant problem is to enumerate the regions of 3-space delimited by several surfaces which may intersect each other in all conceivable ways. Although there should be many relevant results on the intersections of algebraic varieties in the area of algebraic geometry, their introduction to the realm of computational geometry has been started only recently by George Collins and his students.

[%] The implicit assumption here is that the "entrance" of the cavity is planar. When this is violated the filling must be done with care and only partially.

5. The Model-View-Controller triad

This is based on work by Cunnigham [26] on the construction of Smalltalk‡ applications and is relevant to the graphical interface that a geometer's workbench should provide. For other insightful views on graphical interfaces and their "power" the reader is referred to Williams [27] and Bier and Stone [28]. (Williams' paper is also very instructive in that it describes a workbench for economists.)

Smalltalk's [29] approach to building an application is three-fold:

- *Model* This consists of problem data and operations to be performed on it.
- *View* This presents information from the Model to the user via the display.
- *Controller* This interprets inputs from the user and modifies the Model or View accordingly.

In fact, it is quite correct to say that the Model represents the application while the View and Controller represent its user interface. An application may have several of the latter. Windows often provide several Views of a single Model, each different and each with a different Controller to deal with the inputs to that window.

Due to the object-oriented philosophy, any kind of object could represent a Model, View, or Controller as long as it obeys the demanded protocols. A View is not really concerned about the nature of a Model; all it cares is that the Model offers it some information to fill the screen. Similarly, a Model is only slightly aware of being viewed. It just provides answers to questions by its View(s). A Controller has the responsibility for receiving user input in the context of its corresponding View. Input may come from mouse or keyboard. The Controller detects the input and makes something happen. Mouse buttons and key strokes take on different meanings in different windows because different Controllers are listening to them, cf. Dialogue Cells [30]. A Model should redraw when its model changes. There is no magic associated with this. Views are dependents of their Models. As a dependent, a View is sent a message "redraw" whenever its model is altered. Either a Model generates this message itself (as part of a modification protocol) or the change is dictated by a Controller following an editing operation.

‡ Smalltalk-80 is a registered trademark of Xerox Corp. In this paper we write Smalltalk for brevity.

6. Summary of requirements

I have only touched upon some key functionalities that a future geometer's workbench will have to provide. Only experience in developing prototypes will demonstrate the validity and completeness of my views. However, I believe that the main philosophy will stay more or less the same: a window- and menu-oriented user interface, a set of geometric functions similar in scope and generality to the algebraic functions of Macsyma, ability to pursue several computational activities in parallel, and algorithm and data structure animation.

7. SP — A program to compute minimal paths

SP consists of a family of programs written in Franz Lisp and Macsyma command language to experiment with minimal paths in 3-dimensional space. The following description is only cursory and the reader is referred to [5] for details of the system.

The program was designed with the following philosophy in mind. Let a workspace including a set of polyhedra be given. SP, using the geometric descriptions of the specified polyhedra, computes minimal paths in this workspace. It has some interactive graphics facilities and can supply the user with the views of the workspace so that he can have an intuitive feeling about the correctness of a particular computation. I believe that in geometric computations visual debugging is very effective. In this sense, SP resembles to Verrilli's [31] Voronoi-based system; it provides the user with facilities to carry out the needed computations, once in a while asking for his intervention here and there. Following the prototyping approach I either simply excluded those computations which I do not currently know how to perform effectively or reformulated them to be controlled by user advice at certain points. Due to its loosely-coupled structure, it is easy to upgrade SP with new algorithms when they become available.

Currently, one can work with a single convex polyhedron using Franz part of SP. There are facilities to solve Boundary Findpath, Exterior Findpath, and Boundary Findpath (locus). It is also possible to implement an approximate Findpath algorithm for a workspace with several convex polyhedra. Using Macsyma part of SP it is feasible to compute minimal paths in a general workspace although this is not fully automated in the light of the combinatorial explosion that known Findpath algorithms have. Nevertheless, if the user specifies the list of edges that the minimal path must touch, then the problem is solvable using Newton-Raphson method. There are also facilities based on Macsyma functions to deal with general Findpath (locus) but this is not automated yet.

Since it was built as a research tool, the prospective user is expected to know the internals of SP. Fortunately, the interactive nature of Lisp comes into play whenever one wants to debug or inspect the current computation and data structures. Working with SP is incremental in the sense that one computes things, stops and studies them (by plotting if necessary), and continues. To make a rough analogy, it is useful to visualize SP as a sophisticated calculator tailored for minimal path computations.

SP has facilities to read and check the consistency of polyhedral objects. It can also give extensive statistical information about an object. Once a polyhedron is read, SP builds the edge, vertex, and face data structures to access it easily. SP has a facility to unfold (develop) a given face sequence onto the xy-plane. In such a development all polygons must have z-coordinates either 0 or within an ε-neighborhood of 0. SP checks whether this constraint holds true. Figures 1 and 2 depict respectively an example path computed from a development and another mapped back to the surface of an object.

For Exterior Findpath, facilities exist to compute visibility relationships and to construct the silhouettes. Then a new object is created and the minimal path computation proceeds routinely. Figure 3 is an example object created in this way from a cube. For approximate path planning, SP has a facility to find the intersections of the given polyhedra with the source-to-goal line segment and to return a list of point pairs for each polyhedron intersecting the segment. Once these tuples are available, Boundary Findpath is performed for each pair and its associated polyhedron. Further path optimization can also be incorporated. For Boundary Findpath (locus), SP uses a naive Voronoi program. Figure 4 was generated by this program. Since the system is essentially graphical, I have not felt a need to implement a point-location routine.

I regard SP as a first-order and very modest approximation to the large and more general workbench I have proposed above.

ACKNOWLEDGMENTS

I thank Jan van Leeuwen and Paul ten Hagen for providing the excellent environments in which this material was prepared. The encouragement of Rae Earnshaw is also greatly appreciated.

Mention of commercial products in this paper does not imply endorsement.

References

1. A.R. Forrest, ''Computational geometry in practice,'' pp. 707-724 in *Fundamental Algorithms for Computer Graphics*, ed. R.A. Earnshaw, Springer-Verlag, Berlin (1985).

2. H. Edelsbrunner, M.H. Overmars, and R. Seidel, ''Some methods of computational geometry applied to computer graphics,'' *Computer Vision, Graph., and Image Proc.* **28**, pp. 92-108 (1984).
3. J. van Leeuwen, ''Graphics and computational geometry,'' Tech. Rep. RUU-CS-81-18, Computer Science Dept., Univ. of Utrecht, Netherlands (Dec. 1981).
4. J.H. Reif and M. Sharir, ''Motion planning in the presence of moving obstacles,'' *Proc. 26th Annual IEEE Symp. Foundations of Computer Science*, pp. 144-154 (1985).
5. V. Akman, *Unobstructed Shortest Paths in Polyhedral Environments,* Lecture Notes in Computer Science, Vol. 251, Springer-Verlag, Heidelberg (1987). (also: PhD thesis, Rensselaer Poly. Institute, New York)
6. C.E. Buckley, ''The application of continuum methods to path planning,'' AI Tech. Rep. No. 48, Schlumberger Palo Alto Research (Aug. 1985). (also: PhD thesis, Dept. of Mechanical Eng., Stanford Univ.)
7. J.S.B. Mitchell, ''Planning shortest paths,'' Rep. No. 561 (AI Series No. 1), Hughes Aircraft Co. Research Labs (Aug. 1986). (also: PhD thesis, Operations Research Dept., Stanford Univ.)
8. J.T. Schwartz and M. Sharir, ''Motion planning and related geometric algorithms in robotics,'' Manuscript, Courant Inst. of Math. Sciences, New York Univ. (1986).
9. M. Sharir and A. Schorr, ''On shortest paths in polyhedral spaces,'' *SIAM J. Computing* **15**(1), pp. 193-215 (Feb. 1986).
10. J.H. Reif and J.A. Storer, ''Shortest paths in Euclidean space with polyhedral obstacles,'' Tech. Rep. CS-85-121, Computer Science Dept., Brandeis Univ., Waltham, Mass. (April 1985).
11. C.H. Papadimitriou, ''An algorithm for shortest-path motion in three dimensions,'' *Info. Proc. Letters* **20**(5), pp. 259-263 (June 1985).
12. L.P. Chew, ''There is planar graph almost as good as the complete graph,'' *Proc. Second Annual ACM Symp. Computational Geometry*, Yorktown Heights, New York, pp. 169-177 (June 1986).
13. J.E. Hopcroft and G.T. Wilfong, ''Reducing multiple object motion planning to graph searching,'' *SIAM J. Computing* **15**(3), pp. 768-785 (Aug. 1986).
14. J.E. Hopcroft and G. Wilfong, ''Motion of objects in contact,'' *Int'l J. Robotics Research* 4(4), pp. 32-46 (Winter 1986).
15. M. Sharir and A. Baltsan, ''On shortest paths amidst convex polyhedra,'' *Proc. Second Annual ACM Symp. Computational Geometry*, Yorktown

Heights, New York, pp. 193-206 (June 1986).

16. R. Pavelle, "MACSYMA: Capabilities and applications to problems in engineering and sciences," pp. 1-61 in *Applications of Computer Algebra*, ed. R. Pavelle, Kluwer Academic Publ., Boston (1985).

17. W.R. Franklin, P.Y.F. Wu, S. Samaddar, and M. Nichols, "Prolog and geometry projects," *IEEE Computer Graph. and Applic.* 6(11), pp. 46-55 (Nov. 1986).

18. M.H. Brown and R. Sedgewick, "Techniques for algorithm animation," *IEEE Software* 2(1), pp. 28-39 (Jan. 1985).

19. B.G. Baumgart, "GEOMED: Geometric editor," Rep. STAN-CS-74-414, Computer Science Dept., Stanford Univ., Stanford, Calif. (May 1974).

20. I. Fogg and P. Eades, "GED — A graphics editor for computational geometers," Rep. No. 68, Computer Science Dept., Univ. of Queensland, Australia (May 1986).

21. A.P. Pentland, "Perceptual organization and the representation of natural form," Rep. TN-357, AI Center, SRI Int'l, Menlo Park, Calif. (July 1985).

22. A.P. Pentland, "SUPERSKETCH^{TM}," User manual, AI Center, SRI Int'l, Menlo Park, Calif. (Dec. 1985).

23. W.R. Franklin and V. Akman, "Locus techniques for shortest path problems in robotics," *Proc. IFAC Symp. Robot Control (SYROCO'85)*, Pergamon Press (1986).

24. W.R. Franklin and V. Akman, "Euclidean shortest paths in 3-space, Voronoi diagrams with barriers, and related complexity and algebraic issues," pp. 895-917 in *Fundamental Algorithms for Computer Graphics*, ed. R.A. Earnshaw, Springer-Verlag (1985).

25. W.R. Franklin and V. Akman, "Partitioning the space to calculate shortest paths to any goal around polyhedral obstacles," *Proc. Instrument Soc. of America Workshop on Robotics and Expert Sys.*, pp. 45-52 (June 1985).

26. W. Cunningham, "The construction of Smalltalk-80 applications," Manuscript, Computer Research Lab, Tektronix, Inc., Beaverton, Oregon (Nov. 1985).

27. T.L. Williams, "A graphical interface to an Economist's Workstation," *IEEE Computer Graph. and Applic.* 4(8), pp. 42-47 (Aug. 1984).

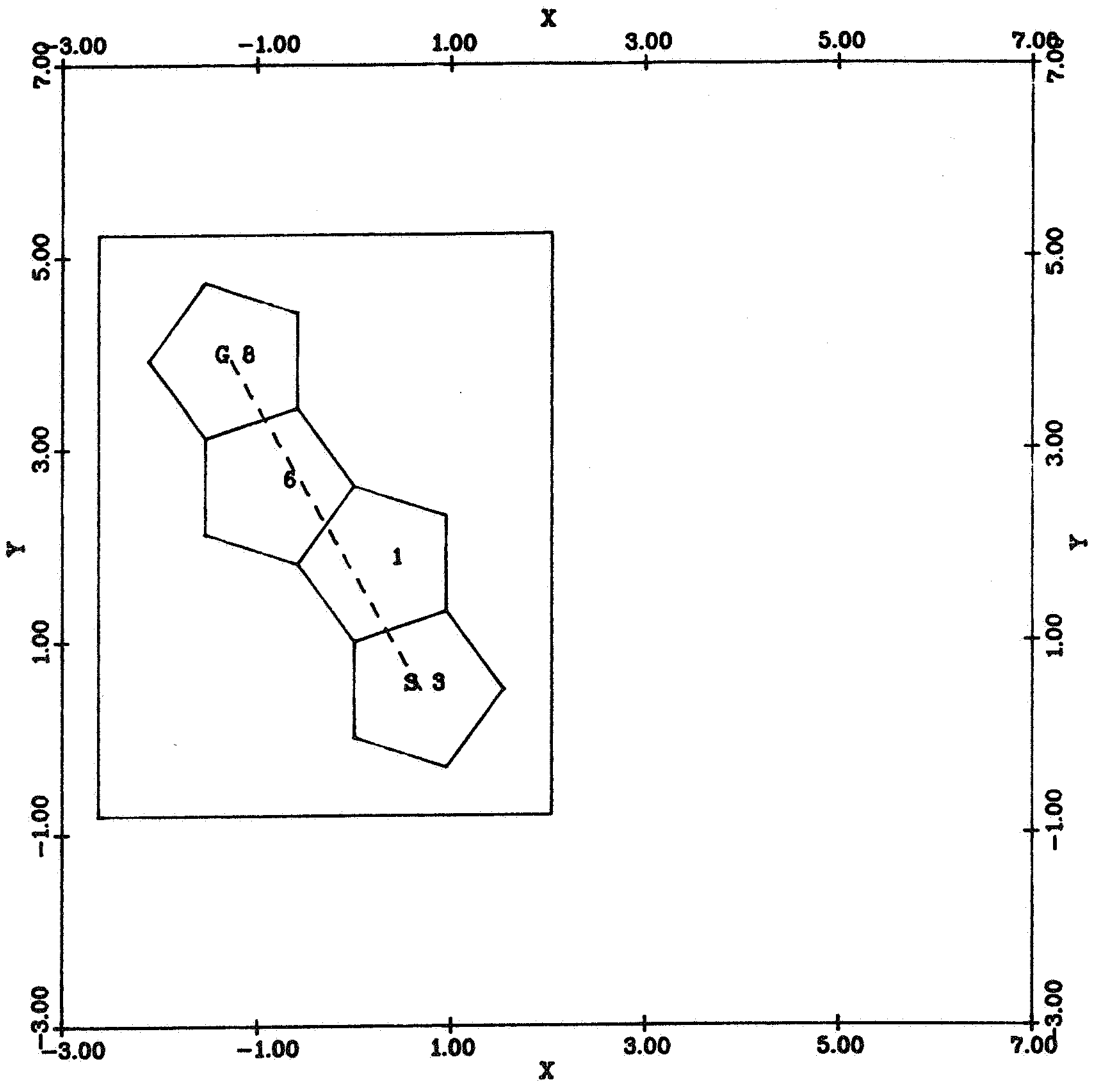
28. E.A. Bier and M.C. Stone, "Snap-dragging," *ACM SIGGRAPH'86 Proc.*, Dallas, Texas, pp. 233-240 (Aug. 1986).

29. A. Goldberg, *Smalltalk-80, The Interactive Programming Environment*, Addison-Wesley, Reading, Mass. (1984).

30. R. van Liere and P. ten Hagen, ''Introduction to dialogue cells,'' Rep. CS-R8703, Centre for Math. and Computer Science, Amsterdam (Jan. 1987).

31. W.R. Franklin, V. Akman, and C. Verrilli, ''Voronoi diagrams with barriers and on polyhedra for minimal path planning,'' *Visual Computer* **1**, pp. 133-150 (1985).

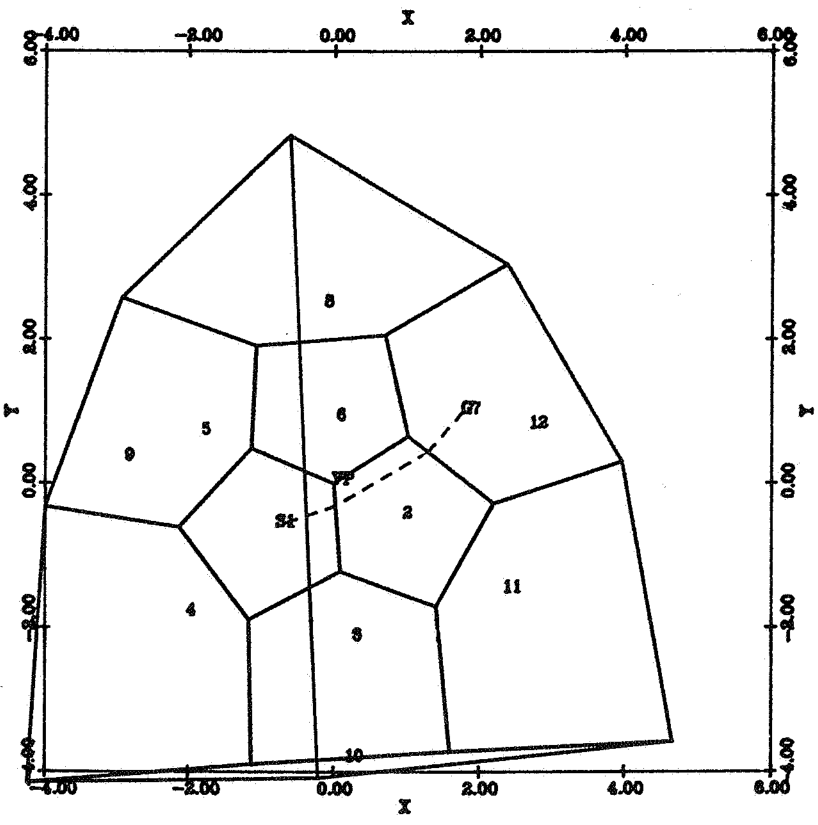
FIGURES



```
Face development sequence:      3   1   6   8
Source face no.:                    3
Goal face no.:                      8
Source point coords.:               0.380     1.447     0.616
Goal point coords.:                 0.996    -0.447     1.612
Source point coords. (plane):       0.688     0.500
Goal point coords. (plane):        -1.276     3.927
Source-to-goal distance:            3.950
Bounding box extents:              -2.627     2.039    -0.809     5.236
```

Fig. 1. A face sequence developed onto xy-plane (cf. Spider-and-Fly)

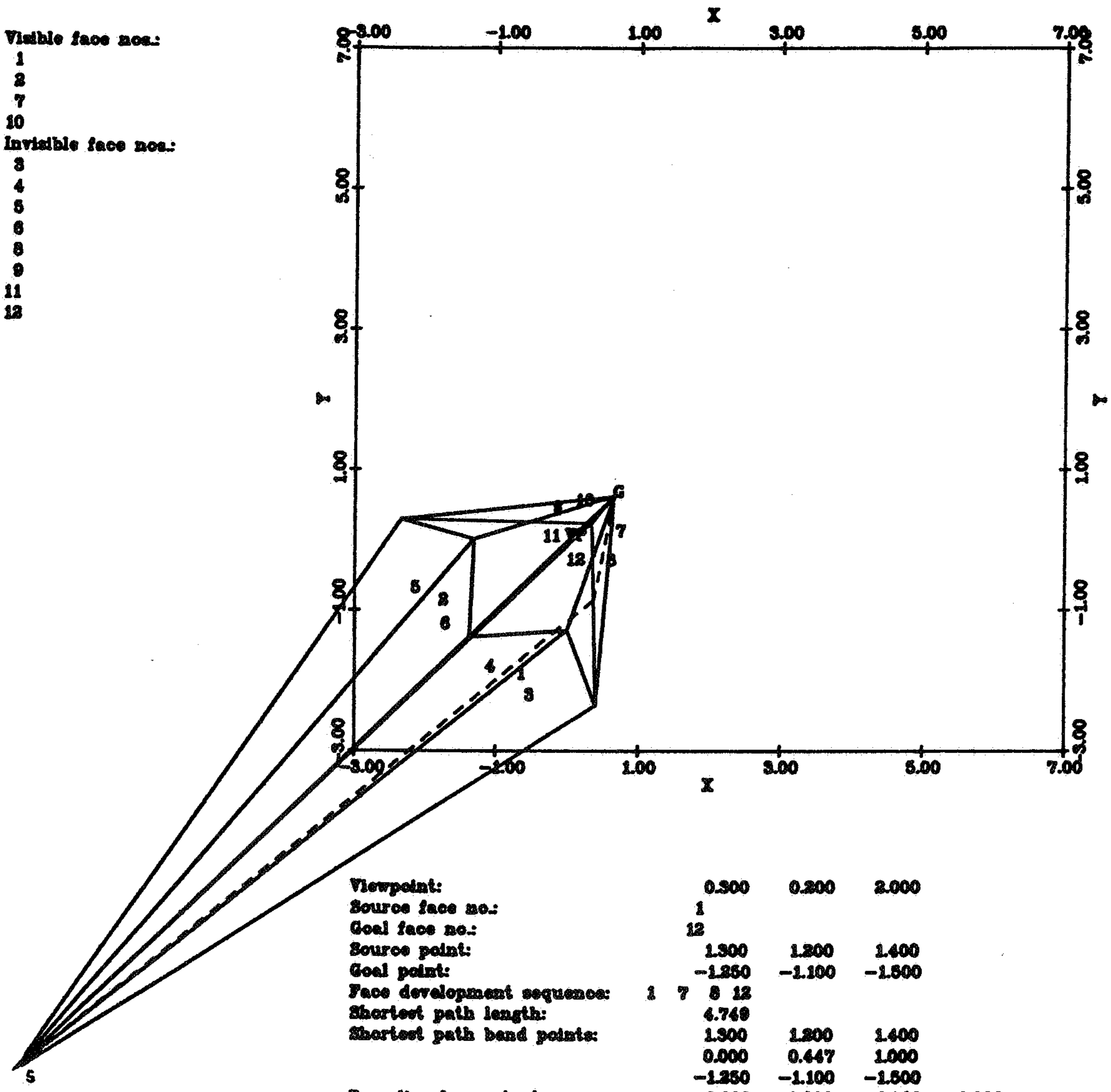
```
Visible face nos.:
 9
 12
Invisible face nos.:
 1
 2
 3
 4
 5
 6
 7
 8
 10
 11
```



```
Viewpoint:                        1.500    0.800    2.500
Source face no.:                  1
Goal face no.:                    7
Source point:                     0.688    0.500    0.000
Goal point:                      -0.118   -0.085    1.611
Face development sequence:    1  2  7
Shortest path length:             2.618
Shortest path bend points:        0.688    0.500    0.000
                                  0.000    0.276    0.000
                                 -0.498   -0.085    0.996
                                 -0.118   -0.085    1.611
Bounding box extents:            -4.000    4.000   -4.000    4.000
```

Fig. 2. A face sequence and its associated minimal path (Boundary Findpath)

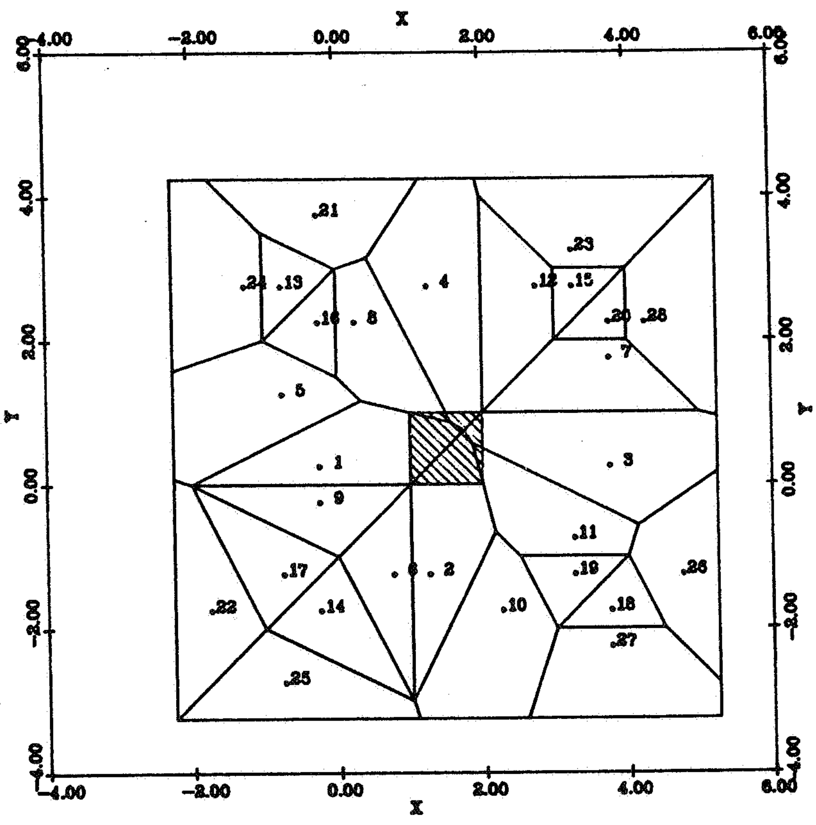
```
Visible face nos.:
1
2
7
10
Invisible face nos.:
3
4
5
6
8
9
11
12
```



```
Viewpoint:                          0.300    0.200    2.000
Source face no.:                    1
Goal face no.:                      12
Source point:                       1.300    1.200    1.400
Goal point:                        -1.250   -1.100   -1.500
Face development sequence:   1  7  8 12
Shortest path length:               4.749
Shortest path bend points:          1.300    1.200    1.400
                                    0.000    0.447    1.000
                                   -1.250   -1.100   -1.500
Bounding box extents:              -3.000    3.000   -3.000    3.000
```

Fig. 3. An example object created by Exterior Findpath

```
Development sequences:
  1:  6  5  1
  2:  6  2  1
  3:  6  3  1
  4:  6  4  1
  5:  6  5  4  1
  6:  6  2  5  1
  7:  6  3  4  1
  8:  6  4  5  1
  9:  6  5  2  1
 10:  6  2  3  1
 11:  6  3  2  1
 12:  6  4  3  1
 13:  6  5  4  3  1
 14:  6  2  5  4  1
 15:  6  3  4  5  1
 16:  6  4  5  2  1
 17:  6  5  2  3  1
 18:  6  2  3  4  1
 19:  6  3  2  5  1
 20:  6  4  3  2  1
 21:  6  5  4  3  2  1
 22:  6  2  5  4  3  1
 23:  6  3  4  5  2  1
 24:  6  4  5  2  3  1
 25:  6  5  2  3  4  1
 26:  6  2  3  4  5  1
 27:  6  3  2  5  4  1
 28:  6  4  3  2  5  1
```



```
Source face no.:               1
Goal face no.:                 6
Source point coords.:           0.000    0.250    0.250
No. of developments:           28
Bounding box extents:          -2.250    5.250   -3.250    4.250
```

Fig. 4. An example of Boundary Findpath (locus) for a cube