Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

V. Akman

Steps into a geometer's workbench

Computer Science/Department of Interactive Systems Report CS-R8726 May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Steps into a Geometer's Workbench

Varol Akman

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

Abstract As computational geometry matures, it becomes crucial to use its techniques in the professional environment where graphics and robotics are natural candidates. This however is a nontrivial task since (i) computational geometry concerns itself with asymptotic analysis, and (ii) in search of elegance it ignores the special cases which are the bugbear of practical applications. I see experimentation as a way to resolve these difficulties and propose a software system to act as a "Geometer's Workbench." This entails the integration of geometric knowledge with algorithm animation and object-oriented graphics. The workbench should allow improvisation with geometric objects and is expected to broaden the way geometry is used in the style Macsyma accomplished for algebra.

Keywords and Phrases Minimal paths, Voronoi diagram, polyhedra, computational geometry, 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 *Nieuw Archief (Special Issue: Proc. ICIAM 87)*, A.H.P. van der Burgh and R.M.M. Mattheij (eds.), Amsterdam (1987).

1. Introduction

Like other key areas of mathematics of the old times (most noticeably algebra and number theory) geometry is being revitalized after a long period of dormancy. The counterpart of "classical" geometry in the age of computers is "computational" geometry. In the latter, we are interested in designing efficient algorithms for tasks of geometric nature. For example, we may want to know the inherent complexity of identifying the region in a subdivision of plane (respectively space) by algebraic curves (respectively surfaces), enclosing a given point — a problem popularized as *point-location*. Here classical geometry must be augmented to deal with a new notion, namely, the complexity of computation. In the lack of computers old geometers did not concern themselves with such efficiency problems.

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 computer graphics and robotics are natural candidates. 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 special cases. Don Knuth's detailed analyses

My proposal, admittedly the first one that comes to mind, is then a software system, a so-called "Geometer's Workbench," which incorporates geometric knowhow and interactive graphics to assist in experimenting with geometric algorithms. It is a remarkable fact that computational geometry has advanced so much since its inception a decade ago. There are now literally hundreds of references (including several books) on computational geometry and a substantial number of these works deal with graphics and robotics problems. The reader is referred to Edelsbrunner et al [2] and van Leeuwen [3] who present fundamental ideas on the relationship between computational geometry and computer graphics. As for the ties of robotics and computational geometry, my dissertation [4] may be a good place to start.

2. Macsyma† as an Algebraist's Workbench

My preliminary thoughts about a geometer's workbench owe to Macsyma [5], 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. 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 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 offers an extensible and exploratory programming environment.

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 many relationships 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 visible to a user, but when required the internals of the system should support reprogrammability and editability. These requirements are satisfied in one way or another by Macsyma. The Lisp

---

of algorithms in his classical books run counter to the big-oh trend of today.

† Macsyma is a registered trademark of Symbolics, Inc.

programming environment that has evolved during the past two decades of artificial intelligence research also delivers these. Lisp machines, for instance, combine the Lisp programming environment with powerful graphics. Since Macsyma's base language is Lisp, these machines naturally support Macsyma. At 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 pragmatical 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.

3. Handling Special Cases

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 (cf. Figure 1), 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 [6] 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 [7] 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 say, Voronoi and point-location algorithms are when applied to scenes with moderate complexity. For instance, 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.

4. Improvising with Geometric Objects

Another key requirement for a geometer's workbench is the availability of good update facilities for the underlying geometric model. This refers to the users 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 and 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 [8] and recently, Fogg and Eades [9] made some efforts in this respect. Pentland's [10, 11] Supersketch^{TM} system 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, since 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.

5. Adaptive Grid as an Implementation Tool

Adaptive grid is a data structure invented by Franklin [12] and can be thought of as a sort of hashing for geometric objects (instead of character strings). It is used to alleviate the problem of comparing everything with everything in order to detect the intersections among them. Several implementations using adaptive grid exist; Franklin and Akman [13] deal with hidden line removal via haloed lines, and Franklin and Akman [14, 15] give a hidden surface program for flat-faced polyhedra.

Let G be an integer ≥ 1 and assume that all the polyhedra are projected into the xy-plane. (That is, we fixed our viewpoint and carried out the preliminary transformations.) Without loss of generality, assume that the initial screen is a square of side 1, coordinates limited to $0 \leq x, y < 1$ and real. Essentially adaptive grid is a uniform $G \times G$ grid overlaid on the scene. The fineness ($\frac{1}{G}$) of the grid is some heuristically determined function of the statistics of the scene, e.g. average edge length, average face area, number of edges, number of faces, etc. The idea is to isolate the geometric objects (line segments or faces) into different cells so that

they won't be compared to each other, as much as this is possible. Ways of roughly determining G are imaginable and we won't concern ourselves with it any more. Besides, it is an experimental fact that changing the fineness within a factor of two makes little difference [14]. Now faces in the projection plane are entered into cells of the grid. Thus if a face has a common part with a grid cell, it is added to the list of faces in that cell. Note that this is *not* done by comparing the face under consideration against all the cells; the bounding box of the face will suffice. This way a few extra cells will be included but the algorithm will still perform correctly albeit a bit slower. For integrity reasons, each grid cell is held responsible for its interior, and additionally, its bottom (south) and left (west) sides. Since we excluded the coordinates with x or y equal to 1, the above provision partitions the scene into G^2 squares which are pairwise disjoint.

Obviously, within a cell visibility computation is carried out only with the faces that are in the face list of that cell. Hence we filtered out all those faces in the scene which are far away from this cell — thus the envisioned localization.

It turns out that the adaptive grid is especially perfect in determining which few pairs of a large number of short edges intersect. In this case the average execution time is linear in the expected number of intersections *plus* the number of edges, thus optimal within a multiplicative constant. Furthermore, since practical scenes tend to be resolution-limited[†] and frequently homogeneous, adaptive grid is also powerful even when the above assumption about short edges is relaxed. While one can think of using a hierarchical grid to accommodate regions of the scene where the edges are clustered more and while this would save time in scenes with orders of magnitude variation in edge density, as soon as cells become hierarchical formerly easy tasks such as determining the cells spanned by an edge become more complicated.

Figure 2 shows a set of cubes output by a haloed line program. Haloed lines are used by drafting people in complicated drawings. Briefly we assume that each line has a "halo" that runs along it on both sides. If a more distant line intersects this first line, then part of the farther line that passes through the first line's halo is blotted out. We divided the haloed line computation into two disjoint steps. The first uses the adaptive grid to find all edge crossings fast and writes a set containing all the locations where each edge is intersected in front by another. The second step sorts the intersections along each edge and computes where the visible and hidden transitions take place. Dividing the computation into two steps means that redrawing a plot with a different halo width is quick since only the latter step need

[†] People don't create scenes with enormous variations; they either detail blank expanses or simplify crowded regions with clarifying annotations.

be rerun. Figure 3 shows a hidden line picture of a set of random blocks. Figure 4 was computed by the same program but painted on a raster scene. Figure 9 is from another hidden surface algorithm working with octrees. This algorithm uses another fast technique first to build the octree from a set of parallelepipeds and then to compute the visible voxels in back-to-front order. Since octrees support set operations efficiently by their nature, they are useful in interference detection problems.

6. Support 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 how 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 messy. Randolph Franklin in a 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 [16] we are particularly interested in finding the intersection curve of two arbitrary surfaces efficiently and reliably. The latter requirement necessarily dictates a symbolic approach to the problem since there may be all kinds of degeneracies. Another relevant problem is to enumerate the regions of space separated by several surfaces which may intersect each other in all conceivable ways. Although there are many relevant results on the intersections of algebraic varieties in the area of algebraic geometry, their introduction to the realm of

[#] 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.

computational geometry has been started only recently by George Collins and his students.

7. Interaction and the MVC Triad

The meaning of interaction is just too wide to be employed without some explanation. We accordingly offer a description of what we mean by this term and then offer a more formal viewpoint based on Smalltalk's‡ Model-View-Controller (MVC) paradigm.

Imagine yourself looking at a graphics screen. You normally see a hidden surface picture of say a machine part or a building. There are several regions on the 2-dimensional screen which have different colors, shadows, transparencies, etc. The important thing is that they are all disjoint since the hidden surface remover already handled the overlapping parts suitably. A particularly interesting interaction is then as follows. You point with a mouse to a region on the screen and pick it. There are several alternatives to what happens next. The following lists them in increasing order of sophistication in terms of user-friendliness:

- Picked region is highlighted.
- Boundary of the face which gave rise to this region is highlighted.
- Besides this region all other visible regions which are parts of the face which gave rise to this region are also highlighted.

All alternatives assume that the visible regions are kept not as a set of pixel values (as in ray tracing algorithms) but as geometric data*, e.g. polygons. Performing the second feedback operation is then easy since one keeps an identifier with each visible region. However the last operation may be inefficient; one must go through all the visible regions just to keep the necessary ones. It is my understanding that an ideal system should give the user the last feedback [18].

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 the ability to selectively review regions would help.

As a formal model of interaction, Cunnigham's work [19] on the construction of Smalltalk [20] applications is relevant to the graphical interface that a

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

* This in turn dictates that one is using an object-space hidden surface algorithm in contrast to an image-space algorithm. See Sutherland et al [17] for details.

geometer's workbench should provide. For other insightful views on graphical interfaces and their "power" the reader is referred to Williams [21] and Bier and Stone [22]. (Williams' paper is also very instructive in that it describes a workbench for economists.)

According to Cunnigham the right 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. 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.

8. Summary

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 using MVC-like paradigms, and algorithm and data structure animation.

Appendix

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 the presence of polyhedral obstacles in 3-dimensional space. The following description is only cursory and the reader is referred to [4] 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 [23, 24] 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. To see the effectiveness of Verrilli's system, consider a minimal path following robot (idealized as a point) that must avoid a set of given walls in the plane. The robot starts from a fixed source point each time but goes to a different goal point. Using the *locus method* of computational geometry one can partition the plane into a set of regions (which turn out to be delimited by a collection of edges and hyperbolic portions) such that for every goal in a given region, the sequence of wall corners that must be followed to obtain the minimal path is the same. Thus in Figure 7 taken from Verrilli's thesis, if the goal is inside the shaded region then one knows that the minimal path is via corners 24 followed by 7 followed by 4 followed by 3. The problem is essential in manufacturing where there is a pile of parts in a location and a robot is supposed to carry the parts to many different locations (or in a fast food joint where you have to deliver hamburgers from a fixed location to many windows).

Following the prototyping approach I either simply excluded from SP 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 a Newton-Raphson like 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 the ε-neighborhood of 0. SP checks whether this constraint holds true. Figures 5 and 6 depict respectively an example path computed from a development and another computed similarly and then mapped back to the surface of the 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. For approximate path planning SP has a function 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 a 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 8 was generated by this program. Since the system is graphical, I needn't implement a point-location routine. For this figure the analogy is as follows. Assume that you have a set of construction sites on a mountain and a fixed location where you keep your tools. The idea is to efficiently compute the route of a truck carrying the tools to different sites. In this case the regions of the boundary of the polyhedron under consideration are delimited only by edges. Once the sequence of faces that a minimal path must visit is known, obtaining the path itself is trivial by unfolding the involved faces to the plane. The main cost of computation is then incurred in constructing the diagram itself since querying with different goals is

just point-location which is asymptotically much cheaper.

Acknowledgments

I thank Randolph Franklin, Jan van Leeuwen, and Paul ten Hagen for providing the environments in which the ideas in this paper took shape. Except Figure 7, all the pictures shown herein were computed by software developed by myself in collaboration with Franklin.

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 Processing* 28, pp. 92-108 (1984).
3. J. van Leeuwen, "Graphics and computational geometry," Rep. RUU-CS-81-18, Computer Science Dept., Univ. of Utrecht, Netherlands (1981).
4. 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)
5. 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).
6. 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 (1986).
7. M.H. Brown and R. Sedgewick, "Techniques for algorithm animation," *IEEE Software* 2(1), pp. 28-39 (1985).
8. B.G. Baumgart, "Geomed: Geometric editor," Rep. STAN-CS-74-414, Computer Science Dept., Stanford Univ., Stanford, Calif. (1974).
9. I. Fogg and P. Eades, "Ged — A graphics editor for computational geometers," Rep. No. 68, Computer Science Dept., Univ. of Queensland, Australia (1986).
10. A.P. Pentland, "Perceptual organization and the representation of natural form," Rep. TN-357, AI Center, SRI Int'l, Menlo Park, Calif. (1985).
11. A.P. Pentland, "Supersketch^{TM}," User manual, AI Center, SRI Int'l,

Menlo Park, Calif. (1985).

12. W.R. Franklin, "A linear time exact hidden surface algorithm," *ACM Computer Graph. (Proc. SIGGRAPH'80)* **14**(3), pp. 117-123 (1980).
13. W.R. Franklin and V. Akman, "A simple and efficient haloed line algorithm for hidden line elimination," *Computer Graph. Forum* (1987, to appear).
14. W.R. Franklin and V. Akman, "Adaptive grid for polyhedral visibility in object space: An implementation," *Computer J.* (1987, to appear).
15. W.R. Franklin and V. Akman, "Reconstructing visible regions from visible segments," *BIT* **26**, pp. 430-441 (1986).
16. 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, Berlin (1985).
17. I.E. Sutherland, R.F. Sproull, and R. Schumaker, "A characterization of ten hidden surface algorithms," *ACM Computing Surveys* **6**(1), pp. 1-55 (1974).
18. P.J.W. ten Hagen, A.A.M. Kuijk, and C.G. Trienekens, "Display architecture for VLSI-based graphics workstations," Rep. CS-R8637 , Center for Math. and Computer Science, Amsterdam (1986).
19. W. Cunningham, "The construction of Smalltalk-80 applications," Manuscript, Computer Research Lab, Tektronix, Inc., Beaverton, Oregon (1985).
20. A. Goldberg, *Smalltalk-80, The Interactive Programming Environment,* Addison-Wesley, Reading, Mass. (1984).
21. T.L. Williams, "A graphical interface to an Economist's Workstation," *IEEE Computer Graph. and Applic.* **4**(8), pp. 42-47 (1984).
22. E.A. Bier and M.C. Stone, "Snap-dragging," *ACM SIGGRAPH'86 Proc.*, Dallas, Texas, pp. 233-240 (1986).
23. C. Verrilli, "One-source Voronoi diagrams with barriers — A computer implementation," Rep. IPL-TR-060, Image Processing Lab., Rensselaer Poly. Institute, Troy, New York (1984).
24. 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

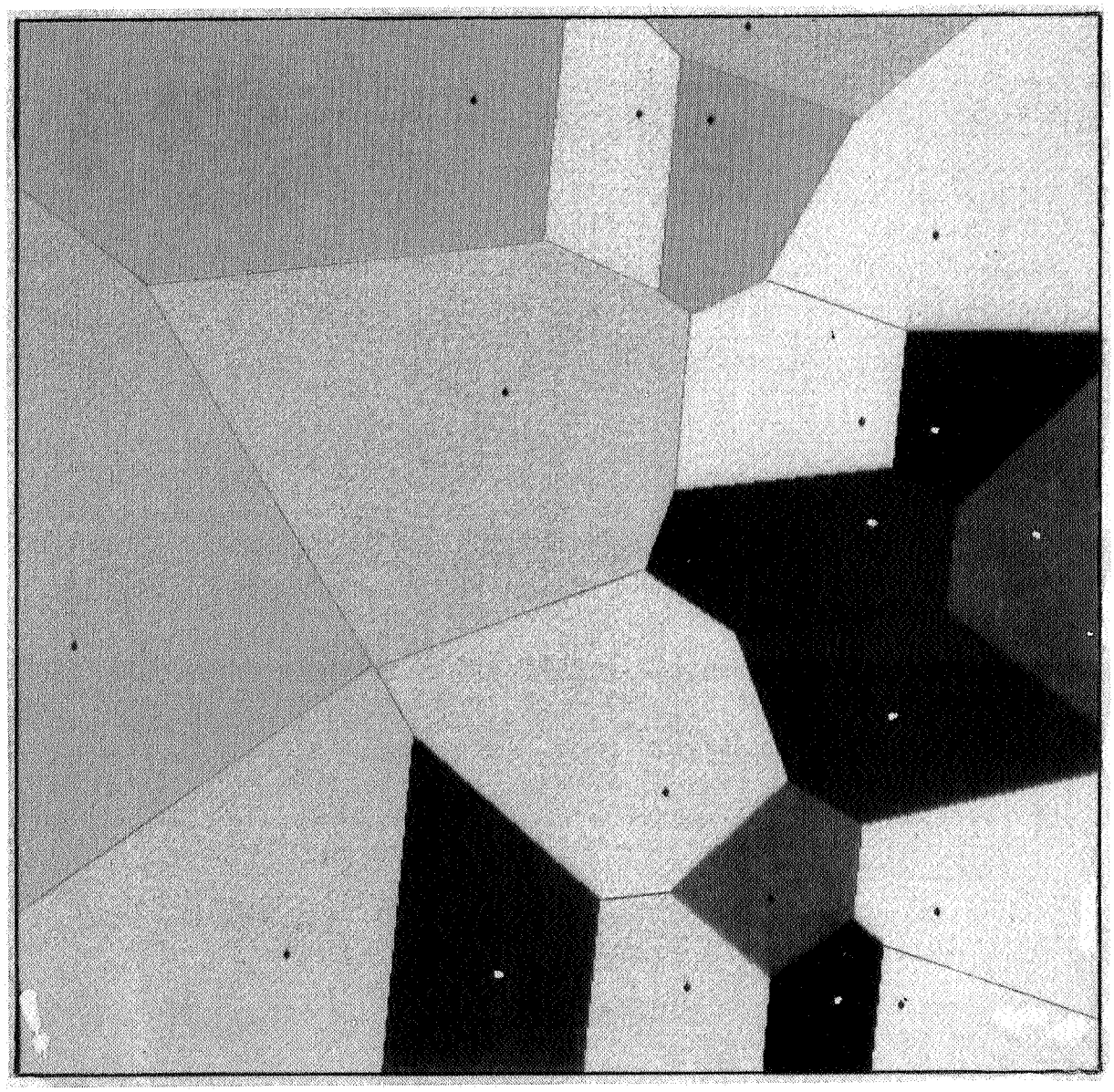


Fig. 1. Voronoi diagram of a set of random points

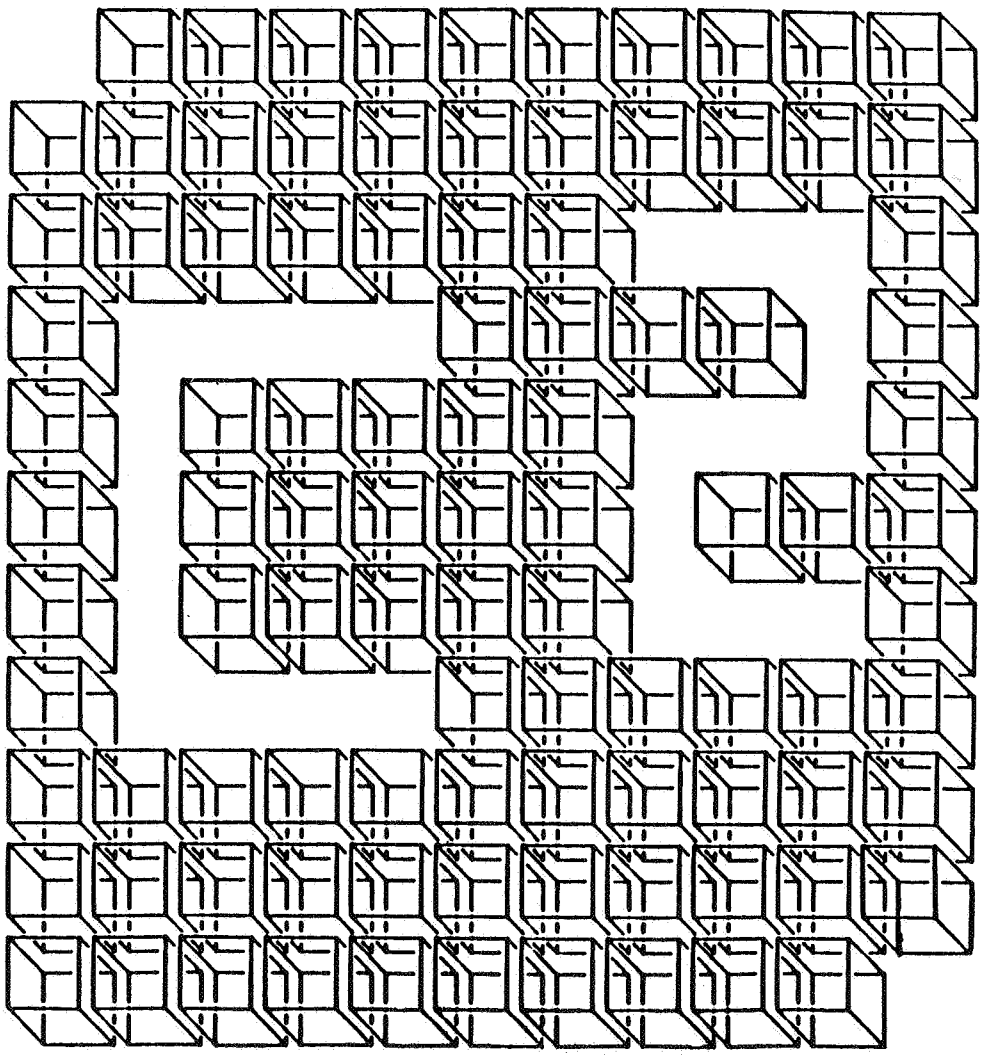


Fig. 2. An arrangement of cubes after haloed line computation

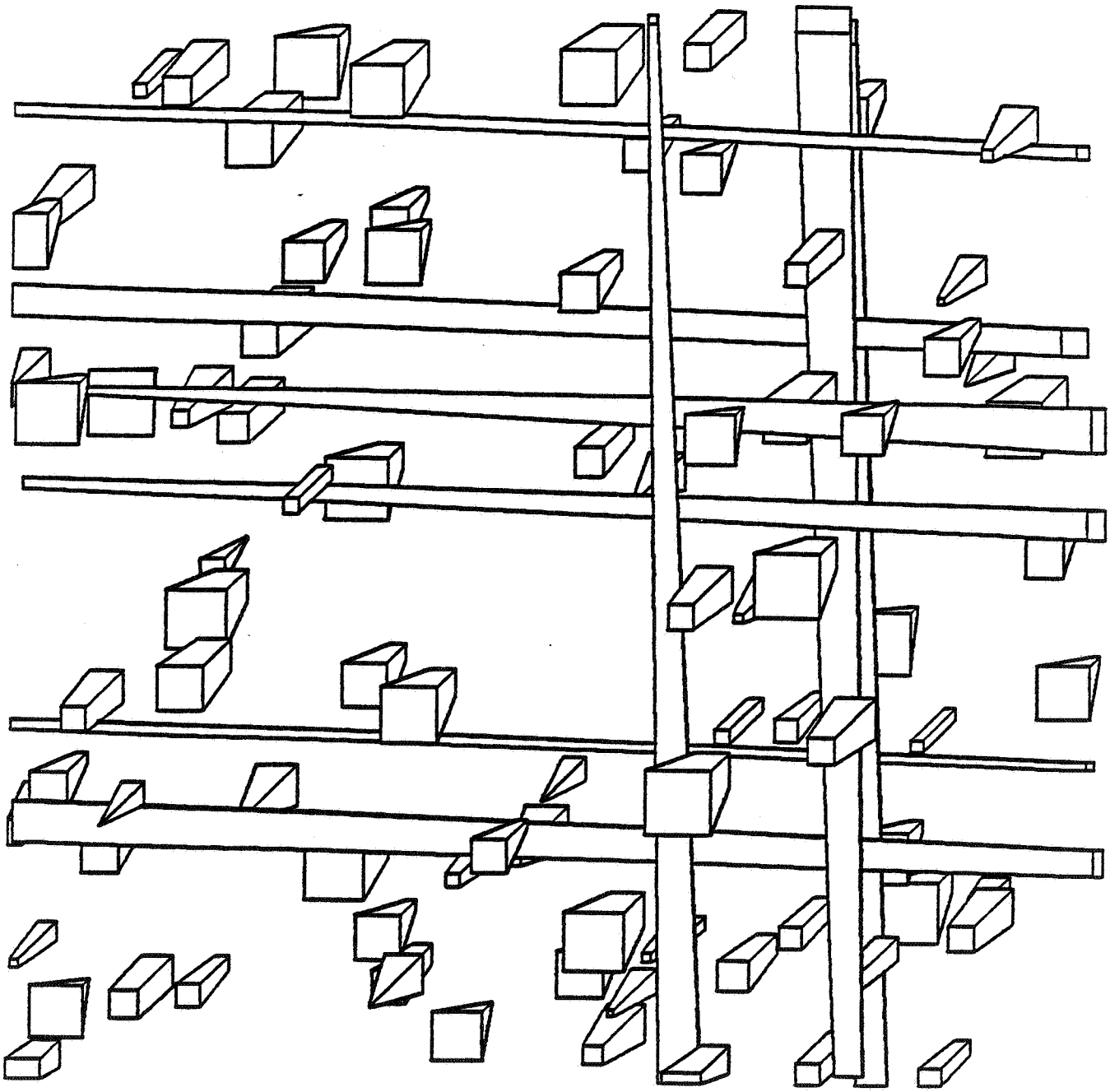


Fig. 3. A random family of blocks after hidden line computation

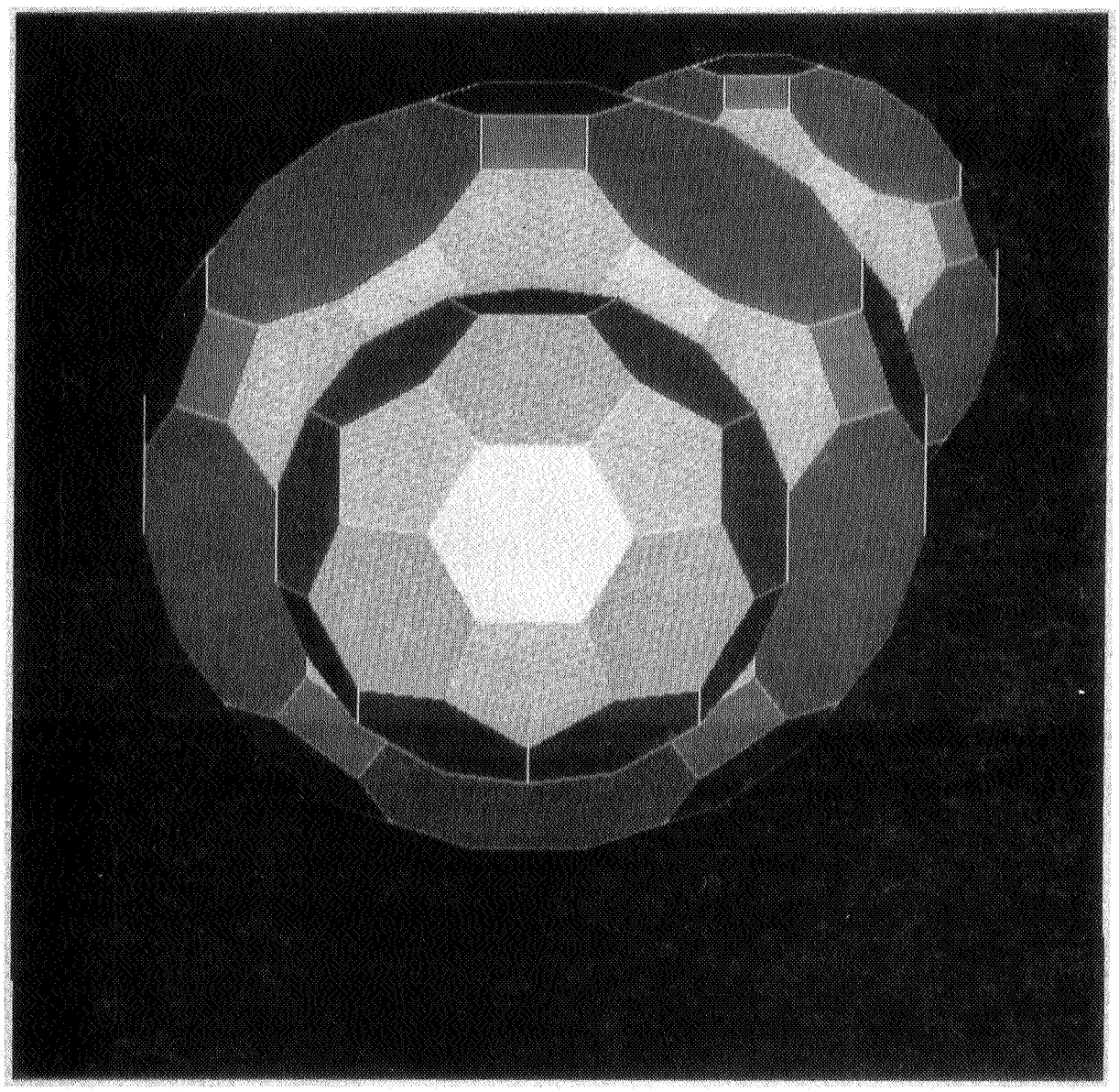


Fig. 4. A family of polyhedra after hidden surface computation

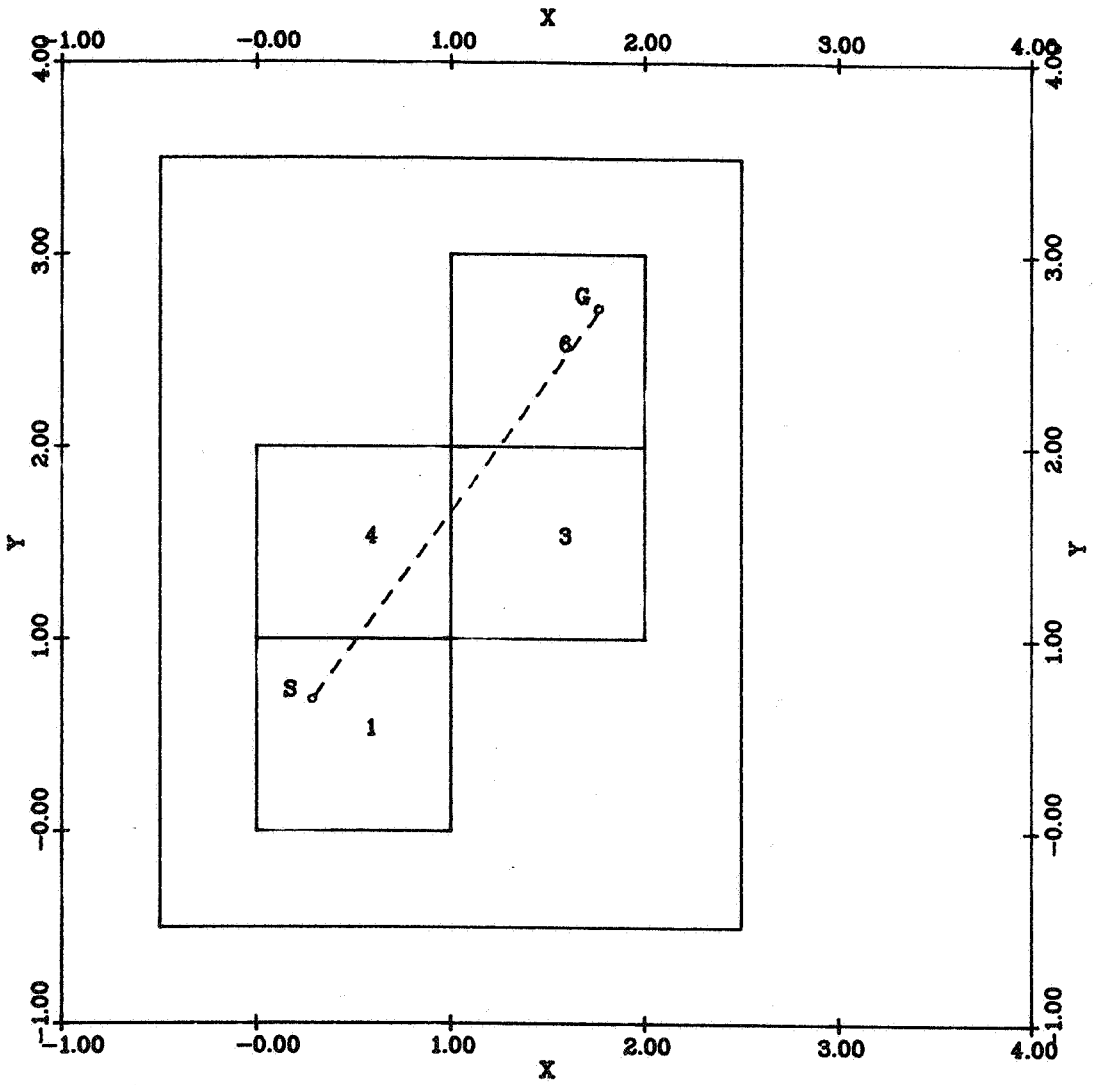
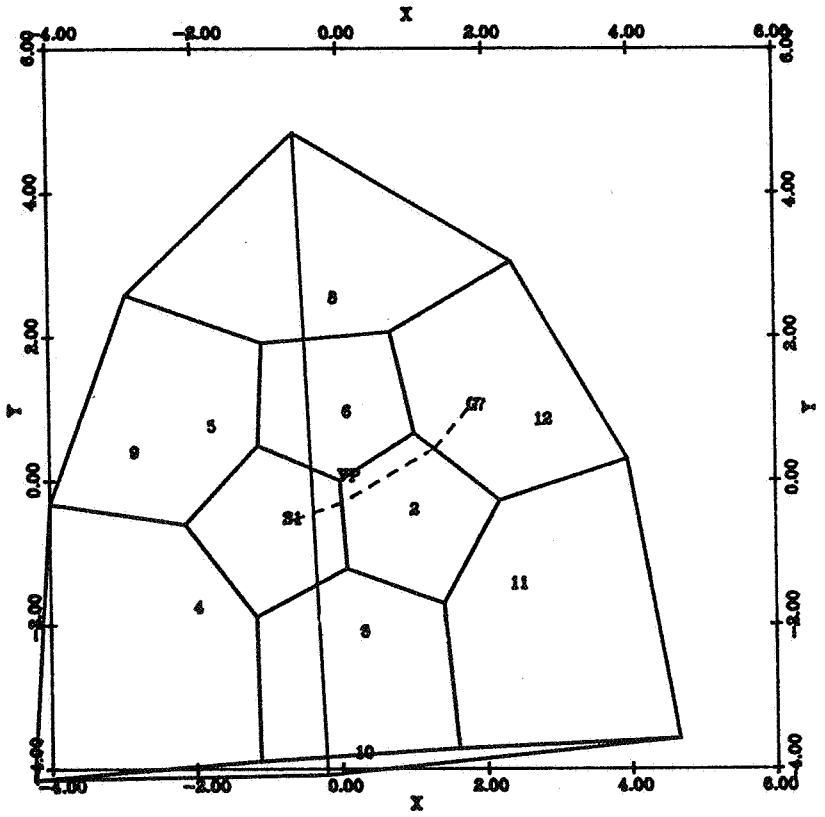


Fig. 5. A planar development for a face sequence of a cube

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. 6. A minimal path mapped to the boundary of an object

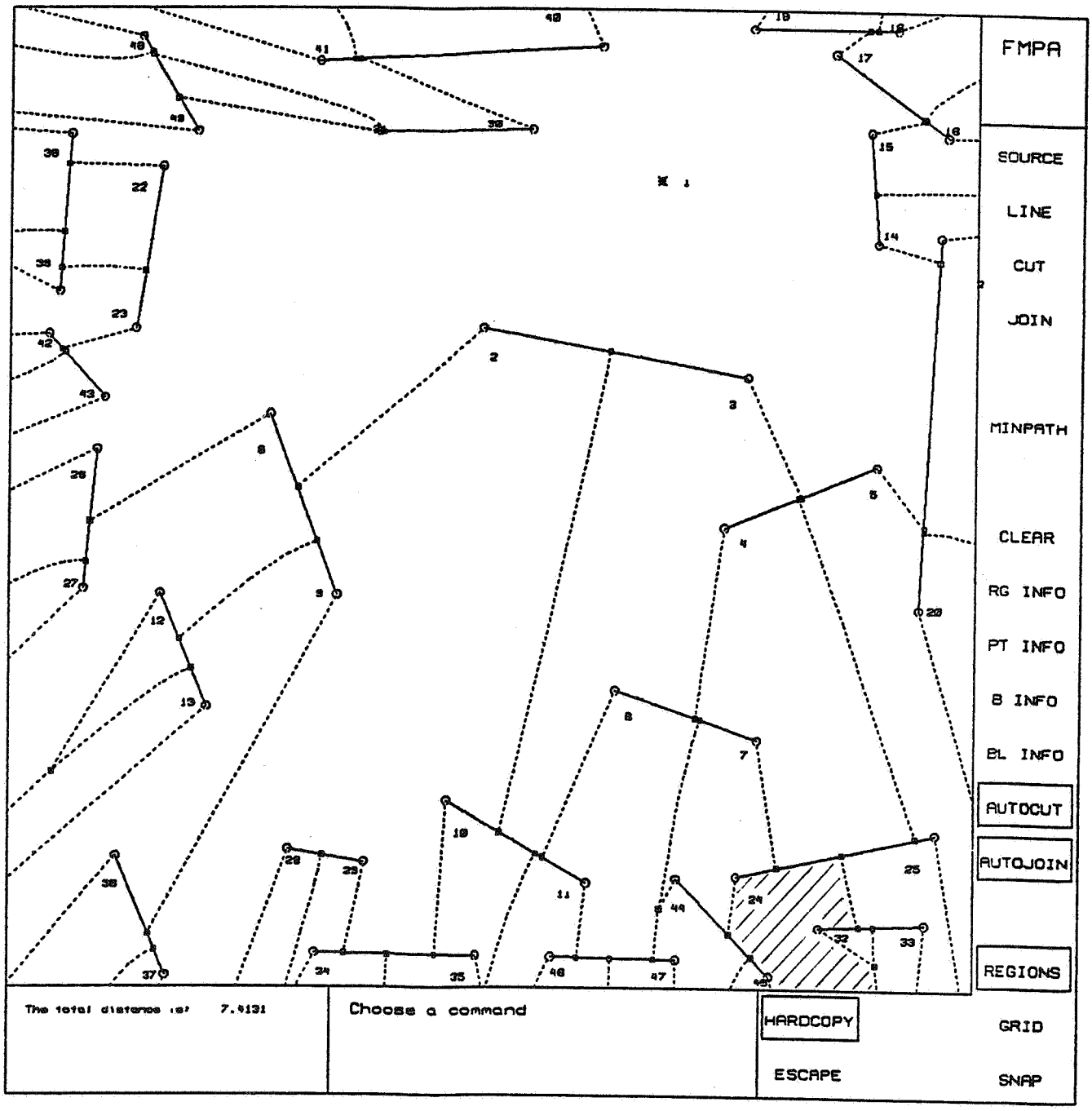


Fig. 7. Verrilli's single-source many-goals Voronoi system

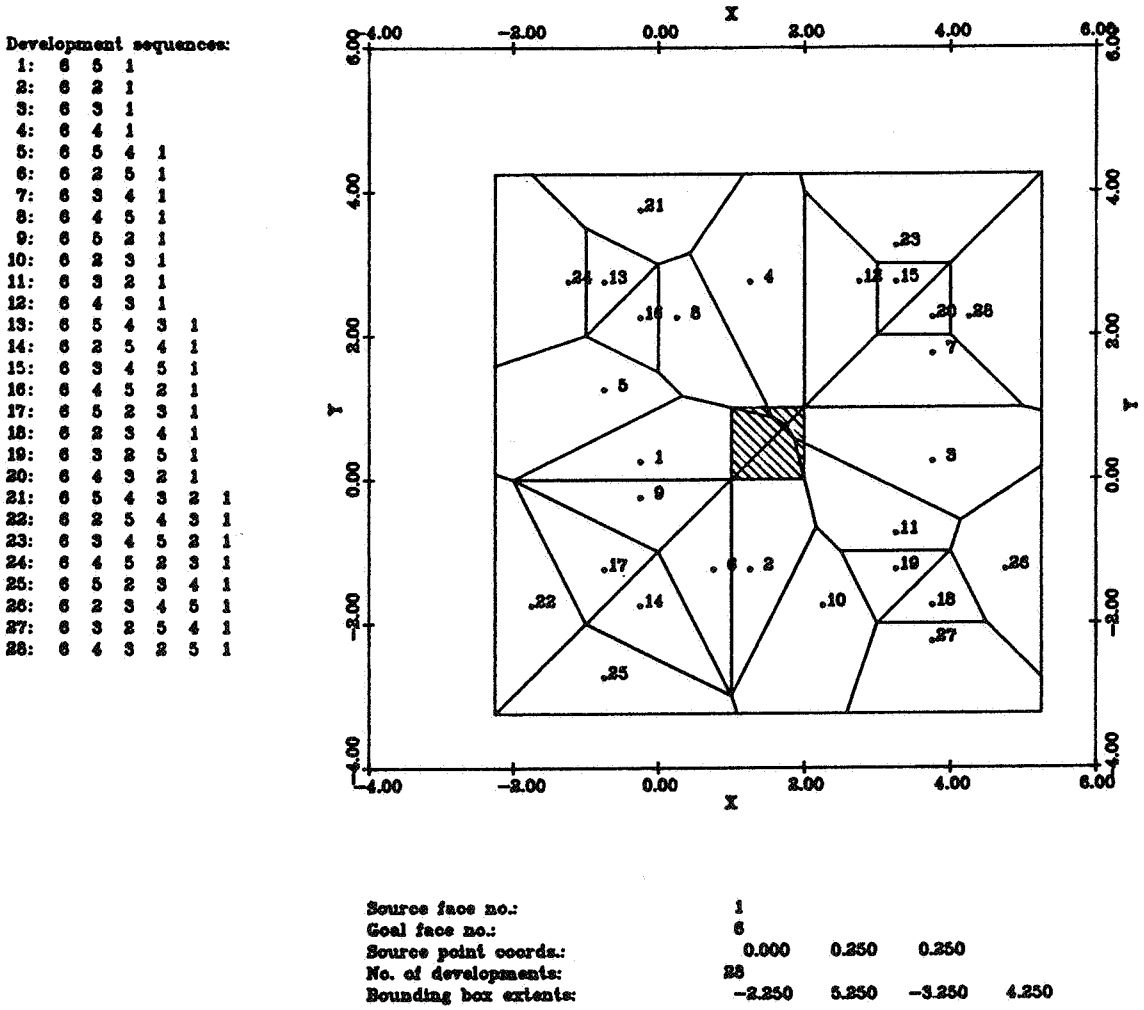


Fig. 8. Partitioning the face of a cube for several goals

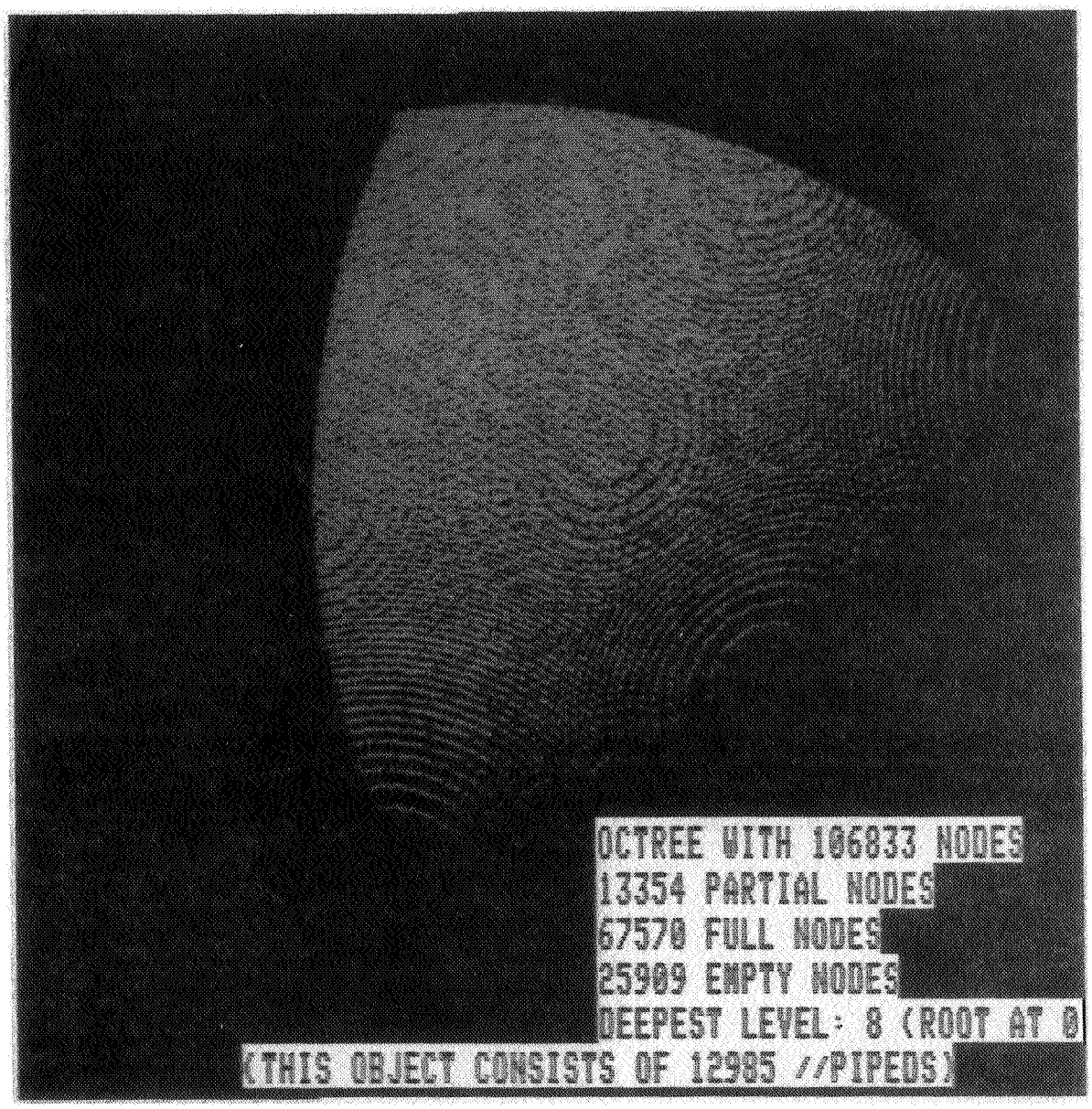


Fig. 9. An octree hidden surface display of a spherical part