[8] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, p. 11, 1972.
[9] T. M. van Veen and F. C. A. Groen, "Discretization errors in the Hough transform," *Pattern Recogn.*, vol. 14, p. 137, 1981.
[10] A. Cornish-Bowden and R. Eisenthal, "Statistical considerations in the estimation of enzyme kinetic parameters by the direct linear plot and other methods," *Biochem. J.*, vol. 139, p. 721, 1974.
[11] G. L. Atkins and I. A. Nimmo, "Current trends in the estimation of Michaelis-Menten parameters," *Analytic. Biochem.*, vol. 104, p. 1, 1980.
[12] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. P. Vetterling, *Numerical Recipes*. Cambridge, MA: Cambridge University Press, 1986, p. 459.
[13] C. A. R. Hoare, "Proof of a program: FIND," *Commun. ACM*, vol. 13, p. 39, 1970.
[14] G. W. Brown and A. M. Mood, "On median tests for linear hypotheses," in *Proc. Second Berkeley Symp. Math. Stat. Prob.*, J. Neyman, Ed., Univ. California Press, Berkeley, 1951.
[15] N. S. Netanyahu, B. Kamgar-Parsi, and A. Rosenfeld, "Application of direction-based pairwise line fitting estimators to noisy edge data," Univ. Maryland Cen. Automation Res. Tech. Rep. (in preparation).

Computing Minimal Distances on Polyhedral Surfaces

ESTAROSE WOLFSON AND ERIC L. SCHWARTZ

Abstract—**We have implemented an algorithm that finds minimal (geodesic) distances on a three-dimensional polyhedral surface. The algorithm is intrinsically parallel, inasmuch as it deals with all nodes simultaneously, and is simple to implement. Although exponential in complexity, it may be used with a companion gradient-descent surface flattening algorithm which produces an optimal flattening of a polyhedral surface [5]. Together, these two algorithms have allowed us to obtain accurate flattening of biological (monkey visual cortex) surfaces consisting of several thousand triangular faces, by providing a characterization of the distance geometry of these surfaces.**

We propose this approach as a pragmatic solution to characterizing the surface geometry of the complex polyhedral surfaces which are encountered in the cortex of vertebrates. Because of its simplicity, this approach may be applicable where the complexity of implementation of computational geometry approaches presents an obstacle.

Index Terms—**Computational geometry, geodesic distance, polyhedron, shortest path.**

I. Introduction

Several algorithms have been described recently that find minimal distances in polyhedral surfaces. An algorithm restricted to convex polyhedra is described in [6], and algorithms to find the shortest path between two points on an arbitrary polyhedral surface have been proposed by [3], [2], and [1].

These algorithms seem difficult to implement. We do not know of the implementation of any of them, including the simpler convex case. Although the above cited solutions are both elegant and subtle, there is a need for actual implementation. In this correspondence, we describe an algorithm whose computational complexity per se is not favorable, but which has yielded good results on highly complex surfaces (monkey visual cortex) with a moderate number of nodes (~1000). Since the algorithm is relatively easy to implement, and has provided good performance in a real-world problem domain of significant complexity, we propose that it may be useful to a variety of other similar applications.

We developed this algorithm for the following specific application: to characterize the metric structure of the cortical surface of the brain in primates and humans, and to "unfold" and flatten these brain surfaces. Apart from artificially created fractal surfaces, the folded, highly convoluted surface of the brain is one of the most complex surfaces encountered in computer graphics. Therefore, we had to develop an algorithm that was simple and practical, and that would not fail in the presence of highly complex data.

One of the reasons for computing the distances in such surfaces is to be able to "flatten" them. It is a common procedure in neuroanatomy to press a cortical surface between glass slides, in order to "flatten" it. The errors and distortions caused by this process are not as bad, perhaps, as one might expect [7], [4]. Nevertheless, there are quantitative studies in this area which require a "flattening" which has minimal errors, consistent with the existing Gaussian curvature of the surface, and for which these errors are well understood. Thus, we use the term "flattening" in this paper to describe the construction of a best least-squares mapping of a polyhedral surface into the plane. Elsewhere [5], we describe an algorithm that does this flattening by using a steepest-descent approach to minimize the difference between the "distance matrix" of the original surface and a planar model of that surface. Obviously, we cannot apply the flattening algorithm until we obtain the original "distance matrix" since this provides the data which implicitly characterizes the surface geometry. The algorithm we describe in this correspondence performs this task.

A. Outline of the algorithm

We start with a polyhedron formed by convex planar polygons. To simplify the problem we use triangular faces. Immediately, we obtain first-neighbor distances: these are the given lengths of the edges of the triangles that form the polyhedron. We then apply the law of cosines[1] to obtain the second-neighbor distances and any lengths and angles of the edges of the given triangles that we do not already know. Any two neighboring triangles have a three-dimensional angle between their planes (the dihedral angle). To find the distance between any two nodes, we could rotate the triangles around the dihedral angle so that the two triangles lie in the same plane, and then apply the law of cosines.

However, we need not actually perform this rotation of the triangles about the dihedral angle. The given angles and edge lengths, together with the law of cosines, suffice to let us calculate the desired distances. Although it is helpful to think of the two triangles as lying in the same plane, it is not necessary to actually perform any computations to cause them to do so. This observation lets us avoid much unnecessary computation.

At this point, we can specify an iterative use of the law of cosines, together with a growing list of angles and distances between nodes in the polyhedron. Some of these angles and distances are the angles and edge lengths of the original triangular faces. Others,

[1]Given the angles and edge lengths of two triangles joined along a common edge, the law of cosines provides the (diagonal) distance between the two nodes not joined by an edge.

Manuscript received February 23, 1987; revised March 28, 1988. Recommended for acceptance by J. O'Rourke. This work was supported by the Air Force Office of Scientific Research under Contract 85-0341, System Development Foundation, and the Nathan S. Kline Psychiatric Research Institute.

E. Wolfson is with the Computational Neuroscience Laboratories, Department of Psychiatry, New York University School of Medicine, 550 First Avenue, New York, NY 10016.

E. L. Schwartz is with Robotics Research, Department of Computer Science, Courant Institute of Mathematical Sciences, 715 Broadway, New York, NY 10003, and the Computational Neuroscience Laboratories, Department of Psychiatry, New York University School of Medicine, 550 First Avenue, New York, NY 10016.

IEEE Log Number 8928496.

which are the result of computations performed during iteration, represent the angles and edges of new objects that we call "p-triangles" and "p-quadrilaterals." The "p" stands for "pseudo," because although all of the nodes of these figures do not really lie in the same plane (that is, dihedral angles exist between the triangles), for the purposes of this algorithm we can imagine that they do.

The first two steps in the iterative application of the law of cosines, as described above, are simple. However, as the level of iteration increases, it is necessary to consider special cases and possible complications. The following sections of this report define the new geometric objects that are essential to a discussion of the algorithm, and offer a proof of the algorithm's validity.

II. Definitions of the Geometric Objects Used in the Algorithm

A. Polyhedron

In R^3, a polyhedron is a figure defined to be a finite set of plane polygons such that every edge of a polygon is shared by at most one other polygon (adjacent polygons) and no subset of polygons has the same property [9]. The nodes and edges of the polygons are the nodes and edges of the polyhedron. This definition allows our polyhedra to be nonconvex, either open or closed, and to have holes; all these cases can be handled by our algorithm.

In the following discussion, we will limit our consideration to polyhedra that consist of triangular faces. The term "triangle" refers to one of these faces.

B. n-Chain of Triangles

An n-chain of triangles is an ordered list of n different triangles such that adjacent triangles in the list share a common edge, such that triangles are adjacent to at most two others in the chain, and such that the n-chain can be unrolled into a planar series of triangles with no overlapping edges in the plane. One major element of our algorithm is the construction of n-chains of triangles. In Fig. 1(a), the set of triangles form a 4-chain. Fig. 1(b) shows a 5-chain and Fig. 1(c) shows a 9-chain. In the following discussion, we use the expression "chains of length n" to mean n-chains.

C. n-Path of Distances

An n-path between two nodes is an ordered list of edges along one or more n-chains, such that adjacent edges in the list share a common node. An alternative term for n-path is edge sequence [1], [6].

D. Diagonal

A diagonal is a straight line between the end points of a particular n-chain. This means that straight lines between the same two points of the polyhedron over different n-chains are different diagonals. The terminology "a new diagonal" indicates the first instance of a straight line path between two particular points over a particular n-chain.

E. p-Triangle and p-Quadrilateral

p-triangles and p-quadrilaterals are convex figures on an n-chain bounded by the diagonals (e.g., see Fig. 1). The following steps provide an inductive definition and clarification of these terms.

Step 1: A 1-chain—A 1-chain and a p-triangle is a triangle of the original polyhedral surface.

Step 2: 2-chains—A 2-chain of triangles (equivalently 2 p-triangles that share a common edge) form a p-quadrilateral [see Fig. 2(a)], provided the quadrilateral formed by the two triangles is convex. (Fig. 2(b) shows a nonconvex example, which therefore is not a p-quadrilateral.) We find the unknown diagonal of the p-quadrilateral by applying the law of cosines. (We already know the other diagonal: it is the common edge of the p-triangles. We use the terms "distance" and "diagonal" interchangeably in the following discussion, because one of the sources of the distances in the computation is the set of diagonals of p-quadrilaterals.)

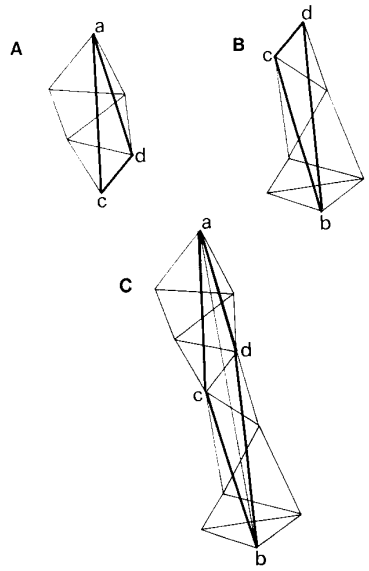


Fig. 1. p-quadrilateral $ACBD$ over a 9-chain (c) is formed from p-triangle ADC with sides AD over a 3-chain and AC over a 4-chain (a) and p-triangle CDB with sides BC over a 4-chain and BD over a 5-chain. This p-quadrilateral generates diagonal AB.

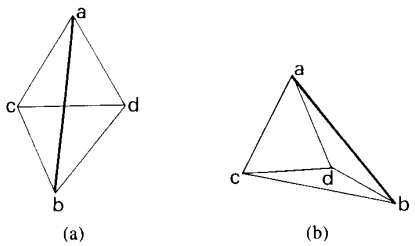


Fig. 2. p-quadrilateral $ACBD$ formed from p-triangles ACD and BCD with common edge CD. The p-triangles are opened along edge CD so that they lie in the same plane. (a) The angles ACB and ADB are both < 180 degrees. Therefore, diagonal AB lies within the p-quadrilateral, and a new diagonal or distance is found between nodes A and B. (b) Angle ADB is > 180 degrees. Therefore, diagonal AB lies outside the n-chain. Thus, the quadrilateral is not a p-quadrilateral, and no distance is found here.

Note that this diagonal lies within the 2-chain of its p-quadrilateral (because the p-quadrilateral is convex). We now define the p-triangles over the 2-chain to be all the p-triangles that can be formed by this new diagonal of the p-quadrilateral and the previously known edges of the p-triangles that lie within the 2-chain. In Fig. 2(a), both ADB and ACB are p-triangles over a 2-chain.

Step 3 (Induction): So far, we have built all the possible p-triangles over chains of length 1 to n. It should be noted that one edge of a p-triangle lies over its entire n-chain and the other two edges lie over subchains of the n-chain. One can join all pairs of these p-triangles which have a common edge and which lie on opposite sides of this edge. When the sum of the angles at the edges of the common edge are each less than 180 degrees we obtain p-quadrilaterals. Those whose combined path is of length $n + 1$ are p-quadrilaterals over $(n + 1)$ chains. A p-quadrilateral over an $(n + 1)$-chain will necessarily be composed of two p-triangles each over a chain less than length $n + 1$ [see Fig. 1(a)–(c)] and since we have all these required p-triangles, we can find all possible p-quadrilaterals over an $(n + 1)$-chain and subsequently their unknown diagonals. These diagonals lie over $(n + 1)$-chains. Now, using our previous set of diagonals over chains of length less than

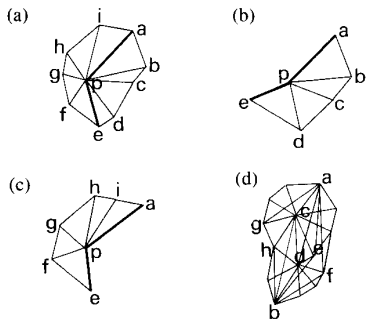


Fig. 3. "Broken distances" between nodes. (a) A three-dimensional view of a node P where the sum of the angles is >360 degrees. Thus, certain flattenings of the triangles around this node might produce angles that are >180 degrees in both directions. Therefore, no new diagonals can be found at this level, and the distance is a "broken diagonal": *APE* in (b) and (c). All the different possible "broken distances" must be examined in order to determine the shortest distance between nodes (for example, *ACB*, *ADB*, *AEB*, *AFB*, *AGHB* for distances between nodes A and B, as shown in (d).

or equal to n, together with these new diagonals, we can form p-triangles over $(n+1)$-chains.

F. m-Broken-Diagonals on n-Chains

A path between two points of the polyhedron along the surface of it which is not a straight line is called a "broken diagonal." An m-broken-diagonal is a broken diagonal with m internal nodes. If the quadrilateral formed by two p-triangles over a 2-chain is not convex, then one of its diagonals must lie outside the 2-chain. The distance between the nodes connected by this diagonal can be determined in either of two ways. One possibility is that it will be found later on another n-chain and p-quadrilateral. The other possibility is that the shortest distance between these two nodes is one of the 2-paths on the 2-chain.[2] This is a 1-broken-diagonal. In either case, we place this diagonal on a list of possible "1-broken-diagonals." A later section of this correspondence explains how the algorithm handles this list.

Then, as we build larger n-chains, some of the quadrilaterals formed by the p-triangles will be nonconvex. If two nodes of a quadrilateral are connected by a diagonal that lies outside the n-chain, then the shortest of the n-paths on the n-chain connecting these two nodes are placed on a list of possible "1-broken-diagonals."

Fig. 3(a) is an example of a three-dimensional view of a node P where the sum of the angles is greater than 360 degrees. Both paths from nodes A to E around P possibly yield 1-broken-diagonals [Fig. 3(b) and (c)]. Fig. 3(d) shows examples of broken diagonals on longer paths.

III. The Algorithm

A. Statement of Algorithm

The essential idea of this algorithm is exhaustive search of the space of distances on the surface of the polyhedron. However, the statement and implementation of this search are not entirely trivial, as the preceding definitions suggest. Given these definitions, we can state the algorithm as follows.

We iteratively build up the table of distances between nodes by building p-triangles and p-quadrilaterals over increasingly larger n-chains. The first iteration considers all 1-chains. The second iteration considers all of the p-quadrilaterals over 2-chains. The third iteration considers all of the p-quadrilaterals over 3-chains, as well as the p-quadrilaterals over 4-chains which, depending on the data, can be built from pairs of 2-chains. It should be noted that if a p-quadrilateral over a 4-chain can be built only from a p-triangle over a 3-chain and a p-triangle over a 1-chain, then it will be obtained only at the next iteration. At the nth iteration, all p-quadrilaterals over n-chains are processed, as are some p-quadrilaterals over chains of length $n+1$ to 2^{n-1}, depending on the data.

The iteration consists of building up new diagonals and combining them with existing diagonals to form new p-triangles. Then, combining these with other p-triangles we form new p-quadrilaterals. Each of these lies over a new n-chain or path of triangular faces and in turn generates a new diagonal along this n-chain. "1-broken diagonals" are generated when quadrilaterals formed by two p-triangles fail to be p-quadrilaterals (i.e., they are nonconvex).

After i iterations, we know all straight line diagonal lengths on chains of length 1 to i. We also know most 1-broken-diagonals[3] on chains of length 2 to i.

For each pair of nodes, we retain the minimum of these straight line diagonal distances and 1-broken-diagonal distances. Note that for some pairs of nodes, we may not yet have any distances. Also, shorter minima may be found later, from construction of m-broken-diagonals.

Now, we construct m-broken-diagonals for m greater than 1. For example, we construct 3-broken-diagonals between a given pair of nodes by joining the 1-broken-diagonals which link these two nodes; we construct the 2-broken-diagonals by linking straight-line-diagonals with a 2-broken-diagonal. When we have constructed all m-broken-diagonals up to some length $m=k$, then we can recombine these to reach lengths up to $m=2\cdot k$ in the next step. Thus, after $\log(i)$ iterations, we have exhaustively constructed all m-broken-diagonals for $m \le i$.

The algorithm terminates when $i=N$, the total number of triangles of the polyhedron. In our application, we take i to be much smaller than N, since we need only find "patchwise" shortest distance solutions. This is because our flattening algorithm [5] only requires overlapping short-range patches of minimal distances in order to successfully flatten the original polyhedron.[4] However, the above procedure is exhaustive, and if run to completion, will find the shortest distance between any two nodes on the polyhedron.

Finally, note that if distances between nonnodal points (i.e., which lie on the faces of triangles) are desired, they may be added to the data structure used to describe the polyhedron. This is trivial, since it amounts to triangulating a triangle with an added internal point.

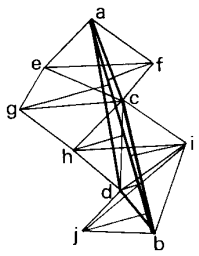
B. Complexity

Since we can build four different $(n+1)$-chains from an n-chain and each of these four $(n+1)$-chains is generated from two different n-chains there are two times as many $(n+1)$-chains as there are n-chains. Thus the number of n-chains after i iterations starting with N triangles is $O(N\cdot 2^{(i+1)})$. Considering that we must maintain all these n-chains since we are building up distances from all nodes simultaneously (thus any n-chain may be a subchain of a larger chain which yields a minimal distance between two other nodes) and the actions taken to generate these n-chains between nodes (building up p-triangles and p-quadrilaterals etc.), the algorithm is exponential in both space and time. Therefore, to find all minimal distances between nodes, the algorithm terminates after $O(N)$ iterations for straight lines plus $O(\log N)$ iterations for combinations of broken distances. This can require a large amount

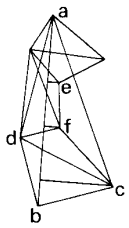
[2]Sharir and Schorr [6] show that for a convex polyhedron, only the first of these two possibilities can occur.

[3]Some nodes may not be connected by broken diagonal paths which are obtained from n-chains. For example, consider some pairs of nodes on triangles which are connected at a single node. The distance between such nodes is found at the same time that m-broken-diagonals of order greater than 1 are determined.

[4]Around each node, we define a patch to consist of all the nodes next-neighbors, second-neighbors, $\cdots$ nth neighbors. Note that for a surface composed of ~ 1000 nodes, its "radius" is ~ 20 nodes, so that a path to 10th neighbors is a considerable fraction of the entire surface. These patches overlap extensively, so that distance relations calculated within a patch allow an accurate flattening to take place.



(a)



(b)

Fig. 4. Given the diagonal AB within an n-chain, we can find a p-quadrilateral that meets our specifications and generates the diagonal. Dropping perpendiculars to the diagonal AB from each of the other nodes C, D, E, F, G, H, I, J, we find node C is closest to AB on one side and node D is closest to AB on the other side. Thus, p-quadrilateral $ACBD$ is formed from the p-triangles ADC and BCD having the common edge DC in (a). In the proof, if edge AC did not lie on the n-chain of diagonal AB, there would be another node (say, E or F) that is nearer to AB and on the same side as node C.

of computation, due to the exponential complexity of the approach. Below, we discuss practical limits, and describe some experiments with complex surfaces (monkey visual cortex) consisting of $\sim 10^3$ nodes.

C. Verification of the Algorithm

Given the preceding definitions and discussions, one can see that we are combinatorially building up all possible paths in all possible ways between two nodes of our polyhedra and finding both the straight line and broken distances between them. It remains to demonstrate the following rather straightforward theorem.

Theorem: Given a p-quadrilateral over an n-chain, there exists a diagonal that lies within the n-chain.

Conversely, given an n-chain, if there exists a straight line (i.e., a line drawn on the flat model of the n-chain), that lies within the n-chain and thus crosses all the triangles making up the n-chain, then there exists a p-quadrilateral whose diagonal is this line.

Proof:

$\Rightarrow$ By definition of p-quadrilateral convexity.

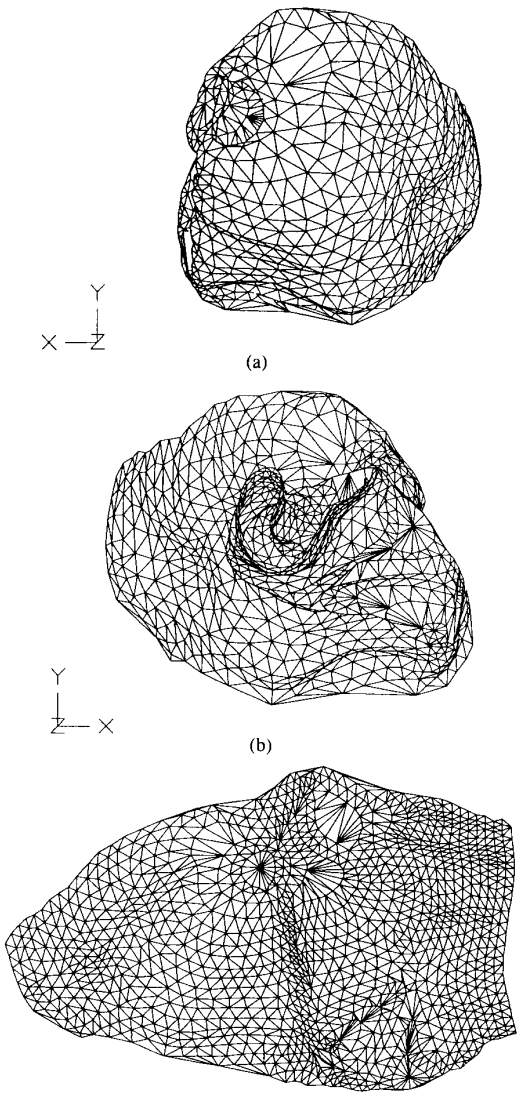
$\Leftarrow$ Choose the closest node of the n-chain on either side of the given straight line. In Fig. 4(a), these are nodes C and D. Thus we have found two triangles, whose common edge is line CD and whose third nodes are the end-points of the given straight line. The union of these two triangles is a quadrilateral.

The angles of the quadrilateral in Fig. 4(a) are less than 180 degrees, because the given diagonal lies within the quadrilateral as constructed.

Both triangles lie within the n-chain. If they did not, then the assumption that their vertices are the closest points to the given straight line would be contradicted [see Fig. 4(b)]. Thus, we have found the required p-triangles and p-quadrilateral.

D. Discussion of the Algorithm

The foregoing theorem proves that if a straight-line path exists between two nodes, exhaustive construction of p-triangles and p-quadrilaterals will reveal it. Any distance that is not found this way must be a "broken diagonal." As all possible "broken diagonals" are found, as described above, the shortest distance between any two nodes will ultimately be determined.



(a)

(b)

(c)

Fig. 5. (a) A three-dimensional polyhedral model of the surface of monkey visual cortex (viewed from top). (b) A three-dimensional polyhedral model of the surface of monkey visual cortex (viewed from bottom). (c) The matrix of interpoint geodesic distances was obtained from the polyhedral surface represented in (a) and (b), and this surface was then "flattened" using a gradient descent method described in [5].

In practice, we run the algorithm far short of all $O(N)$ possible iterations (N typically being on the order of 1000). In our experience with cortical surfaces of several thousand nodes, as shown in Fig. 5, we only need to calculate distances on "patches" containing about 10 nodes in order to obtain a successful flattening from random starting configurations of our flattening algorithm [5]. This indicates that we have correctly captured the metric structure of the entire polyhedral surface with this order of iteration. Naturally, the details of this procedure are application dependent. Note that the "diameter" of a surface of $\sim 10^3$ nodes is about 30. Thus, our "patch size" is roughly $1/3$ the (linear) size of the object. For limited runs such as these (which, given the complexity of the algorithm, are in fact the only feasible runs), the best we can expect is a good approximation to the set of minimal distances in the neighborhood. It is possible that a pathological surface (e.g., one

with many bumps and valleys of just the right size) could have yield a minimal path on a neighborhood size of 10 which was slightly longer than the (true) minimum path on a larger neighborhood. In practice, however, this limitation has not been a problem. Our experience with distance measurements in monkey visual cortex, which is a highly complicated, folded surface, suggest that any errors associated with this algorithm and order of iteration are no more than a few percent.

We use this algorithm as an heuristic approximation to a minimal geodesic-distance algorithm. In our application, running this algorithm to relatively low orders of iteration (i.e., 10) does seem to yield very good approximations to the geodesic distances that we need in order to obtain flattening of the cortical surfaces of the brain.

It is worth pointing out that we have implemented two versions of this algorithm. The first version is the one described above, where at the ith iteration, n-chains up to length 2^{i-1} can possibly be obtained. The other version, at the ith iteration, is limited to n-chains of length i. We have used both runs and merged them for later flattening, thus obtaining a sampling of very long-range distances and also spanning a large neighborhood around each node. Limitations of machine time and storage space make this procedure worthwhile.

IV. Performance

For a polyhedron consisting of about 2500 triangles (about 1200 nodes) representing the surface of monkey visual cortex, we calculated all of the distances over 10-chains on each node. On a Sun-3 workstation, this run took about 2 h and used 12 μbytes of memory; it provided sufficient data for successful flattening of the surface of monkey visual cortex [5].

V. Other Applications

This algorithm finds local shortest-distance patches. Used together with our flattening algorithm, it lets us measure overall shortest distances. This may in fact be the most efficient way to compute long-range shortest distances on the class of polyhedra whose global curvature allows flattening with a relatively small error.[5]

References

[1] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM J. Comput.*, vol. 16, pp. 647-668, Aug. 1987.

[2] D. M. Mount, "Voronoi diagrams on the surface of a polyhedron," Carnegie Mellon Univ., Pittsburgh, PA, Center for Automation Res. Tech. Rep. CAR-TR-121, vol. CS-TR-1496, Dep. Comput. Sci., May 1985.

[3] J. O'Rourke, S. Suri, and H. Booth, "Shortest paths on polyhedral surfaces," in *Proc. Second Symp. Theoretical Aspects of Computer Science*, Aug. 1984.

[4] E. L. Schwartz and B. Merker, "Flattening cortex: An optimal computer algorithm and comparisons with physical flattening of the opercular surface of striate cortex," *Soc. Neurosci. Abstracts*, vol. 15, 1985.

[5] E. L. Schwartz, A. Shaw, and E. Wolfson, "A numerical solution to the generalized mapmaker's problem: Flattening nonconvex polyhedral surfaces," *IEEE Trans. Pattern Anal. Machine Intell.*, this issue, pp. 1005-1008.

[6] M. Sharir and A. Schorr, "On shortest paths in polyhedral surfaces," *SIAM J. Comput.*, vol. 15, no. 1, pp. 193-215, 1986.

[7] R. B. Tootell, M. Silverman, E. Switkes, and R. deValois, "Deoxyglucose analysis of retinotopic organization in primate striate cortex," *Science*, vol. 218, pp. 902-904, 1982.

[8] W. K. Kaplow and E. L. Schwartz, "Measuring mean and Gaussian curvature on triangulated brain surfaces: The differential geometry of macaque striate cortex," Computat. Neurosci. Lab., NYU Med. Center, Tech. Rep. CNS-TR-1-86, 1986.

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

[5]In other work we describe ways to measure the mean and Gaussian curvature of polyhedra [8].

A Numerical Solution to the Generalized Mapmaker's Problem: Flattening Nonconvex Polyhedral Surfaces

ERIC L. SCHWARTZ, ALAN SHAW, AND ESTAROSE WOLFSON

***Abstract*—We describe methods to "unfold" and flatten the curved, convoluted surfaces of the brain in order to study the functional architectures and neural maps embedded in them. In order to do this, it is necessary to solve the general mapmaker's problem for representing curved surfaces by planar models. This algorithm has applications in areas other than computer-aided neuroanatomy, such as robotics motion planning and geophysics.**

Our algorithm maximizes the goodness of fit of distances in these surfaces to distances in a planar configuration of points. We illustrate this algorithm with a flattening of monkey visual cortex, which is an extremely complex folded surface. We find distance errors in the range of several percent, with isolated regions of larger error, for the class of cortical surfaces which we have so far studied.

***Index Terms*—Cortex, flattened surface, geodesic distance, map.**

Introduction

The mapmaker's problem is to find a flat representation of a curved surface, for example, the surface of the earth. Classical mapmaking has been restricted to the relatively simple spherical surface of the earth. In the case that the surface of interest is complex, and possibly nonconvex, there are no known methods of finding quasi-isometric planar representations, that is, those that distort distance relationships as little as possible.[1] (An *isometric* map would be one in which the distance between any two points was identical to the corresponding distance in the original surface.)

The solution to this problem is of importance to computer-aided neuroanatomy, since it is often desired to view the surface of various cortical areas in a planar model. Primate cortex is highly convoluted, and provides one of the more complex surfaces encountered in practical applications.

Motion planning in robotics is another area of application for which the finding of shortest distances on polyhedral surfaces is of importance [10], [11]. Other areas of biophysics and geophysics would seem to provide possible areas of application of a generalized mapmaker's algorithm.

Our interest is in obtaining a flat representation of the cortical surfaces of the brain, because the detailed maps of sensory and other neural data embedded in these surfaces are easiest to study, measure, and model when they are presented in planar form.

This correspondence describes the method we use to find an optimal quasi-isometry that maps an arbitrary curved surface into a plane. This mapping is optimal in the sense that it is derived from a variational principle that optimizes the overall fit between the curved and planar surfaces. The mapping is a quasi-isometry because it optimizes the fit of distances over multiple scales, rather than, for example, local angles (in which case it would be quasi-conformal).

Manuscript received February 23, 1987; revised December 9, 1988. Recommended for acceptance by J. O'Rourke. This work was supported by the Air Force Office of Scientific Research under Contract 85-0341, System Development Foundation, and the Nathan S. Kline Psychiatric Research Center.

E. L. Schwartz is with the Computational Neuroscience Laboratories, Department of Psychiatry, New York University School of Medicine, 550 First Avenue, New York, NY 10016, and Robotics Research, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, New York, NY 10003.

A. Shaw and E. Wolfson are with the Computational Neuroscience Laboratories, Department of Psychiatry, New York University School of Medicine, 550 First Avenue, New York, NY 10016.

IEEE Log Number 8928499.

[1]An early version of this work was described by [4]. An alternative approach has been described by [1].