JOURNAL OF COMPUTATIONAL PHYSICS 59, 177–192 (1985)

On the Construction of the Voronoi Mesh on a Sphere

JEFFREY M. AUGENBAUM

NASA–Goddard Space Flight Center, Code 611, Greenbelt, Maryland 20771

AND

CHARLES S. PESKIN

Courant Institute of Mathematical Sciences, 251 Mercer Street, New York, New York 10012

Received September 16, 1983; revised May 31, 1984

A new construction of the Voronoi mesh on the sphere is presented.The main feature is that the algorithm adds points one at a time until the final Voronoi mesh is built up. By adding one point to an existing Voronoi mesh of K points, only local changes are needed to construct a Voronoi mesh of $K+1$ points. This construction is particularly well suited to time-dependent problems since using information from the Voronoi mesh at the previous time step allows us to reduce the construction to $O(N)$ operations when the two configurations are close, while the algorithm does not break down when they are far apart. Numerical experiments are presented to substantiate the $O(N)$ operation count for a "typical" case.

1. INTRODUCTION

Ever since the developement of modern high speed computers, most of the work on large scale hydrodynamic codes and particularly on meteorological codes have centered around those of Eulerian type. This is due to their ease in programming and higher order of accuracy. However, one of the biggest drawbacks of Eulerian methods is the presence of the nonlinear convective terms $\bar{u} \cdot \nabla \bar{u}$ which leads to inaccurate representations of advection and discontinuities.

An alternative, and conceptually simpler, approach is to use a Lagrangian method. In such a method the fluid particles themselves are tracked and equations are derived based on local spatial interactions. In this formulation the nonlinear convective terms do not appear. The main drawback with the Lagrangian formulation, however, is that these local spatial interactions are time dependent. That is, at each time level, one must know the current neighbors of a given particle to compute accurately the forces acting on the particle. Thus a grid which is continuously deforming and always linking nearest neighbors is needed. Such a grid,

0021-9991/85 $3.00

the "Voronoi diagram" or "Voronoi mesh" has been around for a long time [17], but has only recently been introduced into hydrodynamics codes. (See Peskin [13], Augenbaum [1–3], and Trease [16]. An alternate approach using the dual Delaunay triangulation has been introduced by Crowley [6], Fritts and Boris [9], and Dukowicz [7].)

While there have been several succesful Lagrangian or quasi-Lagrangian codes used for gas dynamics and incompressible flows (e.g., FLAG [6], PIC [10], and ALE [11]), there has been very little application of such methods to meteorological problems. A first attempt was made by Mesinger [12], using a set of floating points to solve the shallow water equations on a sphere. In this method all points within a fixed radius from a given point are used in computing the derivatives at that point.

Mesinger compares his method to existing Eulerian schemes and reports very encouraging results. However, no mention is made of an operation count for finding neighbors particularly when large deformations occur after a number of time steps, nor is there any attempt to optimize the grid algorithm.

In [1, 3], Augenbaum introduced a Lagrangian scheme for the shallow water equations on a sphere based on the use of the Voronoi diagram. In the present paper we describe, in detail, this Voronoi construction which is a generalization of a planar algorithm introduced by Peskin [13]. (References [13 and 19] also discuss the use of the Voronoi diagram for the solution of the incompressible Navier–Stokes equations.) Other constructions for the Voronoi diagram can be found in [5, 8, 15].

2. Construction of the Voronoi Mesh

Partition of the Sphere

Suppose we are given a collection of N points $\{\bar{X}_1, \ldots, \bar{X}_n\}$ on the unit sphere S in R^3. These points can be used to generate a partition of S into convex spherical polygons which overlap at most by having an edge in common. Let

$$H_{kl} = \{\bar{x}: |\bar{x} - \bar{X}_k| \leqslant |\bar{x} - \bar{X}_l| \text{ and } |\bar{x}|^2 = 1\}, \tag{2.1}$$

$$P_k = \bigcap_{\substack{l=1 \\ l \neq k}}^{N} H_{kl}. \tag{2.2}$$

The borders of P_k are made up of points $\bar{x}$ (on S) such that $|\bar{x} - \bar{X}_k| = |\bar{x} - \bar{X}_l|$ for some l. Accordingly the borders are subsets of great circles. These circular arcs are the perpendicular bisectors of the arcs connecting $\bar{X}_k$ and $\bar{X}_l$. Thus the P_k are spherical polygons. They are convex polygons because they are constructed as intersections of convex sets. (Convexity is usually a property of subsets of a *linear* space. The unit sphere is not a linear space, but it is embedded in the linear space R^3. With every subset A on the sphere, we can associate a cone $A' = \{\bar{x} \in R^3:$

$\bar{x} = \alpha\bar{y}$, $\alpha \geq 0$, $\bar{y} \in A\}$. We say that A is "convex" iff A' is convex in the usual sense. It follows that intersections of convex sets are convex, as usual. In particular, the cone corresponding to a hemisphere is a half-space, which is convex. Therefore all intersections of hemispheres are convex spherical polygons.) A point $\bar{x}$ on S, in the interior of P_k satisfies

$$|\bar{x} - \bar{X}_k| < |\bar{x} - \bar{X}_l| \qquad \text{for all } l \neq k. \tag{2.3}$$

Because of the strict inequality, it is impossible for (2.3) to hold for fixed $\bar{x}$ and more than one value of k, which proves that the interiors of the spherical polygons P_k do not overlap. On the other hand, every point $\bar{x} \in S$ is in at least one of the spherical polygons P_k. To prove this, just pick k to minimize $|\bar{x} - \bar{X}_k|$. These considerations justify the statement that we have generated a partition of S into convex spherical polygons which overlap at most by having an edge in common. This configuration depends on the generating points $\{\bar{X}_1, ..., \bar{X}_n\}$ and it is a natural partition in the sense that the kth spherical polygon contains in its interior precisely those points of S which are closer to $\bar{X}_k$ than to any other generating point.

Main Algorithm

The required polygonal network is called a Voronoi diagram [17] (see Fig. 1). An optimal algorithm for the construction of the polygons (in the plane) is known with an operation count $O(N \log N)$ [14]. This operation count is optimal for the situation in which the positions of the generating points are arbitrary. Our purpose is not to extend this algorithm to the sphere. Rather we shall describe a new algorithm that can be used in either the spherical or planar case and that is particularly well suited to time-dependent problems. In such problems, the changes in configuration of the generating points at any given time step result in few, if any, changes in the topological structure of the polygons. When changes in structure do occur, they are local in character.

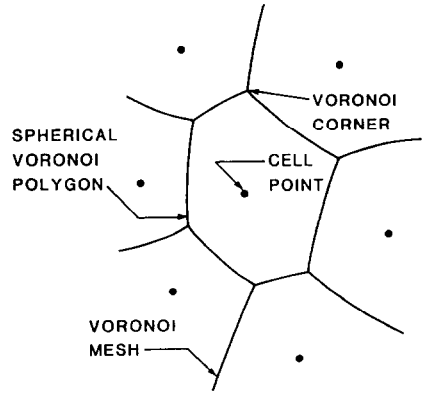

FIG. 1. Portion of a Voronoi diagram or mesh.

These considerations suggest that we aim at an algorithm in which the number of operations depends on how close the current configuration is to the previous configuration. The operation count should be $O(N)$ when the two configurations are close, but the algorithm should not break down no matter how far apart the two configurations are. We shall describe such an algorithm in the spherical case. The planar case is very similar, and we omit the details.

The algorithm is based on certain observations concerning the corners of the polygons. Since the edges are arcs equidistant between pairs of generating points, their intersections (the corners) are equidistant from three of the generating points. The three generating points $\bar{X}_i$, $\bar{X}_j$, and $\bar{X}_k$ define a plane that intersects the sphere dividing it into two spherical caps (see Fig. 2). We shall determine the center of one of these spherical caps: the one that lies to the left of its circular boundary when that boundary is traversed in the direction $\bar{X}_i \to \bar{X}_j \to \bar{X}_k$. The formula for the central point is

$$\bar{X}^c(i,j,k) = \frac{(\bar{X}_j - \bar{X}_i) \times (\bar{X}_k - \bar{X}_i)}{|(\bar{X}_j - \bar{X}_i) \times (\bar{X}_k - \bar{X}_i)|} = \frac{\bar{X}_i \times \bar{X}_j + \bar{X}_j \times \bar{X}_k + \bar{X}_k \times \bar{X}_i}{|\bar{X}_i \times \bar{X}_j + \bar{X}_j \times \bar{X}_k + \bar{X}_k \times \bar{X}_i|}. \tag{2.4}$$

Clearly, the point $\bar{X}^c(i,j,k)$ lies on the unit sphere, remains invariant under an even

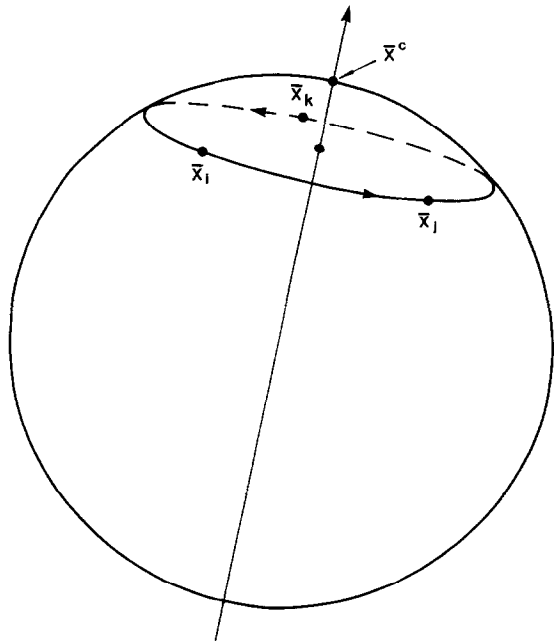

FIG. 2. Three points $(\bar{X}_i, \bar{X}_j, \bar{X}_k)$ on sphere taken in counterclockwise order determine center of spherical cap cut by plane through sphere.

permutation of (i, j, k), and changes sign under an odd permutation of (i, j, k). To check that $X^c(i, j, k)$ is equidistant from $\bar{X}_i$, $\bar{X}_j$, and $\bar{X}_k$, let $l = i, j,$ or k and evaluate

$$|\bar{X}^c(i, j, k) - \bar{X}_l|^2 = |\bar{X}^c(i, j, k)|^2 - 2\bar{X}^c(i, j, k) \cdot \bar{X}_l + |\bar{X}_l|^2$$
$$= 2(1 - \bar{X}^c(i, j, k) \cdot \bar{X}_l).$$

Therefore it is sufficient to show that $\bar{X}^c(i, j, k) \cdot \bar{X}_l$ has the same value for $l = i, j,$ or k. Using the triple-scalar-product identity $(\bar{a} \times \bar{b} \cdot \bar{c} = \bar{a} \cdot \bar{b} \times \bar{c})$, it is easy to see that in all three cases

$$\bar{X}^c(i, j, k) \cdot \bar{X}_l = \frac{\bar{X}_i \cdot (\bar{X}_j \times \bar{X}_k)}{|\bar{X}_i \times \bar{X}_j + \bar{X}_j \times \bar{X}_k + \bar{X}_k \times \bar{X}_i|}, \qquad l = i, j, k.$$

This completes the proof that $\bar{X}^c(i, j, k)$ is the common corner of the spherical polygons P_i, P_j, P_k if the following criteria are both met:

(i) $\bar{X}^c(i, j, k) \in S$,

(ii) for every l, $|\bar{X}_l - \bar{X}^c(i, j, k)| \geqslant r(i, j, k)$,
where $r(i, j, k) = |\bar{X}_i - \bar{X}^c(i, j, k)| = |\bar{X}_j - \bar{X}^c(i, j, k)| = |\bar{X}_k - \bar{X}^c(i, j, k)|$.

If there is some l that violates (ii), we say that $\bar{X}^c(i, j, k)$ is a broken corner and that it has been broken by the point $\bar{X}_l$. In the foregoing test for a broken corner we use the Euclidean (chord) distance which is monotonic function of the spherical distance.

All of the corners can therefore be found by the following simple (but expensive) procedure, which we state for conceptual purposes only: Look at all triples of generating points and test to see whether conditions (i) and (ii) are satisfied. The number of triples is $O(N^3)$, and the number of tests per triple is $O(N)$, so the operation count for this algorithm is $O(N^4)$.

We can do better, however, by adding the generating points one at a time. Suppose the polygonal structure generated by the points $\bar{X}_1, \ldots, \bar{X}_{k-1}$ is already known. The changes in the structure that result when the kth point is added have been described by D. Goldfarb (unpublished) as follows (see Fig. 3).

Each edge of the new spherical polygon P_k lies entirely within one of the old spherical polygons $P_{k'}$ and terminates on one of the old edges of $P_{k'}$. To prove this, consider an arbitrary point $\bar{x}$ on the common edge of the new spherical polygon P_k and the old spherical polygon $P_{k'}$. We have

$$|\bar{x} - \bar{X}_j| = |\bar{x} - \bar{X}_{k'}| \leqslant |\bar{x} - \bar{X}_l| \qquad \text{all } l$$

In this statement, the equality follows from the fact that edges are equidistant between pairs of generating points, and the inequality holds because otherwise $\bar{x}$ would be in the interior of P_l (for some l different from k and k') and not on $P_k \cap P_{k'}$. On the othe hand, the inequality also shows that $\bar{x}$ was in $P_{k'}$ before the

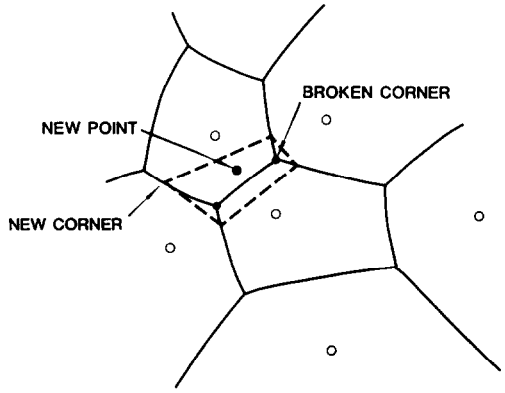

FIG. 3. When a new point is added (free solid circle), some of the old corners are swallowed up in the interior of the new polygon (dashed lines). These broken corners form a connected net in the old graph of corners (solid lines). There is exactly one new corner on each edge joining a broken and an unbroken corner.

point $\bar{X}_k$ was added. Similarly, let $\bar{x}$ be a terminal point of the new edge $P_k \cap P_{k'}$. That is, let $\bar{x}$ be a new corner, equidistant (say) from the three points $\bar{X}_k$, $\bar{X}_{k'}$, and $\bar{X}_{k''}$. Then by the same argument as above,

$$|\bar{x} - \bar{X}_k| = |\bar{x} - \bar{X}_{k'}| = |\bar{x} - \bar{X}_{k''}| \leqslant |\bar{x} - \bar{X}_l| \qquad \text{all } l;$$

and it follows as above that $\bar{x}$ lies on the old edge common to $P_{k'}$ and $P_{k''}$.

Now, let B_k be the set of all spherical polygons that are cut by the addition of the point $\bar{X}_k$. Clearly, B_k is a connected set, since a walk through all of the old spherical polygons in B_k is constructed simply by following the (new) edges of P_k. This walk can be constructed without the edges of P_k being given in advance. Let one of the spherical polygons $P_{k'} \in B_k$ be given. Simply construct the arc equidistant between $\bar{X}_k$ and $\bar{X}_{k'}$ in $P_{k'}$. Where this segment of arc encounters the border of $P_{k'}$, it points to another element of B_k. The procedure can therefore be continued until the entire spherical polygon P_k is constructed and the walk through B_k is complete (see Fig. 3).

It follows from this construction of Goldfarb's that the set of corners broken by the addition of the point $\bar{X}_k$ is also a connected set. To prove this, we first note that the set of broken corners in any one spherical polygon is connected, since all of these corners lie on one side of a great-circle arc and since the spherical polygons are convex. Then, to construct a walk through all of the broken corners, we just follow the walk outlined above covering all of the broken corners of each spherical polygon as that polygon is encountered. Successive polygons on this walk always have exactly two corners in common, exactly one of which is broken, so there is no difficulty in stepping from one spherical polygon to the next.

These observations allow us to use the following algorithm. To update the polygonal structure when the Kth point is added:

(i) Search for a broken corner.

(ii) Search for all of the broken corners (a connected set).

(iii) Find all edges that join broken corners and unbroken corners. Every new corner is located on such an edge, and every edge of this kind has exactly one new corner.

(iv) Construct the edges of the spherical polygon P_k by joining the appropriate pairs of new corners.

To provide an operation count for this algorithm we need to recall certain facts concerning the numbers of corners and edges in a spherical Voronoi diagram generated by N points. We derive these facts from Euler's formula for an arbitrary graph on the surface of a sphere,

$$f - e + v = 2, \tag{2.5}$$

where f is the number of faces (in our case $f = N$), e is the number of edges, and v is the number of vertices (corners). We consider only the generic case, in which every corner is a junction of exactly 3 edges. (If this is false, it can be made true by an arbitrarily small perturbation of the generating points.) Then, since it is also true that each edge has exactly 2 vertices (corners), we have

$$2e = 3v. \tag{2.6}$$

Combining this with Euler's formula and substituting N for f, we obtain

$$v = 2N - 4, \qquad e = 3N - 6. \tag{2.7}$$

Thus the total number of corners and edges in the Voronoi diagram is known in advance and grows only linearly with N. These facts are very convenient from the standpoint of allocating storage. Moreover, the average number of sides (or vertices) *per polygon* is given by

$$n = \frac{2e}{N} = \frac{3v}{N} = 6 - \frac{12}{N} \tag{2.8}$$

so the typical polygon (for large N) is a hexagon.

We shall give a "typical-case" (as opposed to a worst-case) operation count for the algorithm outlined above. Step (i), the search for a broken corner, may require $O(N)$ steps (this is actually a worst-case estimate since the number of corners is $2N - 4$). Once a single broken corner is found, the search for all broken corners (step (ii)) requires a numbr of operations that is typically independent of N (again, in the worst case the search for all broken corners would require $O(N)$ steps since the *total* number of corners is $2N - 4$). The reason for this is that, in the typical case, the new polygon has 6 corners, so only 4 old corners were broken (the net increase in the number of corners must be exactly 2). Moreover, we have proved

that these 4 broken corners form a connected set in the old Voronoi diagram, so they are easily found without extensive searching. Similarly, once the broken corners have been found, we know immediately where to locate the (roughly 6) new corners (step (iii)) and how to connect them to generate the new polygon (step (v)). In short, steps (ii)–(iv) are *local* and require a number of operations which is related to the number of sides of a single polygon and which is therefore independent of N.

In summary, the number of operations in step (i) is $O(N)$ (in the typical case and in the worse case), and the number of operations in steps (ii)–(iv) is independent of N is the typical case and $O(N)$ in the worst case. Since the procedure is used for each point as it is added, the number of operations required to construct the entire polygonal structure is $O(N^2)$ both in the typical case and in the worst case. An important difference between the typical case and the worst case will appear below, however.

Now consider the situation in which the generating points are all moving simultaneously and we are faced with the task of constructing the Voronoi diagram at successive times $t = n\ \Delta t$, $n = 1, 2,\ldots$. This situation arises, for example, in Lagrangian fluid dynamics computations in which the generating points move as fluid markers. In such a case, we should be able to obtain considerable savings in computer time by taking advantage of information obtained when the Voronoi diagram was constructed at the previous time step. The algorithm described above does not seem applicable to this situation, since it builds up the polygonal structure from scratch by adding the generating points one at a time. Despite this, the following simple device makes it possible to take advantage of the previous time step and to reduce the (typical-case) operation count from $O(N^2)$ to $O(N)$:

When the point K is added and a broken corner is found, we store the index of one of the three generating points involved in that corner. At the next time step, when the point K is added, the search for a broken corner starts from one of the corners of the polygon P_k. When the configurations of the generating points on successive time steps are close, this procedure has the effect of making the number of operations in step (i) independent of N. The operation count for the construction of all N polygons then becomes $O(N)$. In general the number of operations varies smoothly (roughly speaking) with the distance between the configurations of the generating points on successive time steps. This is a desirable situation, because it means that the amount of work depends on the essential difficulty of the task.

The algorithm outlined above for adding point K is facilitated by the following data structure, which is based on the fact that each corner has 3 neighboring corners and three generating points associated with it. To store this information, we use two arrays, ICR(L, I) and IPT(L, I), where $L = 1, 2, 3$ and $I = 1, 2,\ldots,$ NCRS. Thus NCRS is the number of corners, ICR(L, I) is the Lth generating point of corner I. In these lists, the three corners and the three generating points of corner I are stored in counterclockwise order. Moreover, the lists are correlated with each other according to the following rule: Among the three generating points of corner I, IPT(L, I) is the only one that is *not* involved in the corner ICR(L, I) (see Fig. 4).

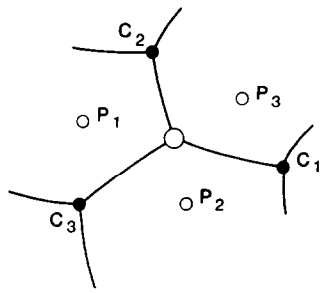

FIG. 4. For a given corner (large hollow circle) the three generating points (P_1, P_2, P_3) are stored in counterclockwise order as are its three neighboring corners (C_1, C_2, C_3). The two lists are correlated so that P_i is opposite C_i.

A subroutine SPOTCR(I) is provided that locates corner I on the sphere. This routine simply finds the center of the spherical cap defined by the three generating points taken in counterclockwise order. It stores the coordinates of the center in the arrays XCR(I), YCR(I), ZCR(I) and the square of the radius of the cap in an array RAD2(I). We again note that we use the Euclidean radius $|\bar{X}^c - \bar{X}_i|$ instead of the spherical radius $|\bar{X}^c - \bar{X}_i|_s$.

The arrays initialized by SPOTCR are used in the test that determines whether corner I is broken by the generating point K with coordinates $X(K)$, $Y(K)$, $Z(K)$. The FORTRAN test is simply

$$\text{IF}((X(K)-\text{XCR}(I))**2+(Y(K)-\text{YCR}(I))**2$$
$$+(Z(K)-\text{ZCR}(I))**2-\text{RAD2}(I)).$$

A negative result indicates that the corner is broken.

We can now describe how this data structure is used to implement steps (i)–(iv) of the algorithm for adding the point K. Step (i) starts from a given corner and looks at the neighbors of this corner, at the neighbors of the neighbors, and so on, until a broken corner is found. The required neighbors are found in the list ICR. To avoid testing any corner twice, a logical array FLAGB is used. When FLAGB(I) = .TRUE., the corner I has already been tested. A list of the tested corners is also generated during the search. This list can be used at the end to reset FLAGB = .FALSE. without performing $O(N)$ operations.

Step (ii), the search for all of the broken corners, starts from the broken corner found in step (i). The search has the same structure as before, except that only the neighbors of broken corners are tested, since the set of broken corners is connected. During this search, the array FLAGB is used in a slightly different way: FLAGB(I) = .TRUE. now indicates that corner I is broken. A list of broken corners, LISTB, is generated during the search in the following way. Initially LISTB contains only the single broken corner that was found in step (i). The entries in LISTB are examined in sequence, and their neighboring corners are tested unless

FLAGB indicates that they are known to be broken. When new broken corners are found, they are added to the end of LISTSB and flagged in FLAGB. Only one sweep through LISTB is required. When the end of LISTB is encountered, all of the broken corners have been found.

In step (iii) the new corners are found. Each new corner is located on an edge joining a broken corner I1 and an unbroken corner I2 = ICR(LA, I1). The new corner is assigned an index of one of the broken corners until these indices are used up. Then new indices are assigned. Let the index assigned to the new corner be INEW, and let LB, LC be such that (LA, LB, LC) is a cyclic permutation of (1, 2, 3). Then the three points that determine the new corner are, in counterclockwise order,

$$\text{IPT}(1, \text{INEW}) = K$$

$$\text{IPT}(2, \text{INEW}) = \text{IPT}(\text{LB}, \text{I1})$$

$$\text{IPT}(3, \text{INEW}) = \text{IPT}(\text{LC}, \text{I1})$$

We also need to know the three neighboring corners of the new corner INEW. One of these is the unbroken corner I2. The other two will be new corners, which are not yet determined. To facilitate the task of linking together the new corners into a new polygon the following information is stored in an array NEXTCR,

$$\text{NEXTCR}(\text{IPT}(\text{LB}, \text{I1})) = \text{INEW}.$$

Once this information has been stored, step (iv) can be accomplished in a single sweep around the new polygon without any searching. Given a new corner I, the procedure for finding the next new corner I2 around the polygon is simply

$$\text{I2} = \text{NEXTCR}\ (\text{IPT}(3, I)).$$

This completes our description of the algorithm that adds a point (ADP).

3. Initialization

The above-described algorithm for adding a point is dependent on the arrays of the previous structure being stored and correlated correctly. Thus it is essential that the initial configuration be correlated correctly. We now describe a simple procedure to construct an initial Voronoi diagram simply and efficiently so that an arbitrary number of points can be added using the addition algorithm.

Suppose that we have N generating points $\{\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_n\}$ with which to construct the Voronoi diagram. The smallest number of points needed to construct a nondegenerate Voronoi diagram is four. Since three points are needed to generate a corner, by taking $\bar{X}_1, \ldots, \bar{X}_4$ as our initial generating points we have a Voronoi diagram consisting of four distinct spherical polygons (P_1, P_2, P_3, P_4) and four cor-

ners ($\bar{X}C(1)$,..., $\bar{X}C(4)$). Once we set up the initial data structure the rest of the generating points $\bar{X}_j$, $j = 5$,..., N can be added one at a time.

The three points that make up corner IC are stored in IPT(L, IC) $L = 1, 2, 3$, as follows. The number given to the corner made up by any three of the four generating points is the same as the fourth generating point, i.e.,

$$\text{IPT}(L, \text{IC}) = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 3 & 3 & 2 & 2 \\ 4 & 4 & 4 & 3 \end{pmatrix}$$

Note that the three points in IPT(L, IC) $L = 1, 2, 3$ that make up corner IC are not necessarily stored in counterclockwise order yet. In fact the coordinates of the corners have not been determined yet. To determine the corners IC we simply call SPOTCR(IC), which will determine the coordinates of corner IC assuming IPT(L, IC) to be stored in counterclockwise order.

We can now check that the entries in IPT(L, IC) are stored in counterclockwise order. Recall that there are two points on the sphere that are equidistant from three given points. We chose the one, in SPOTCR, by using the right-handed rule, assuming IPT(L, IC) to be stored in counterclockwise order. To check that we have the right corner, we test to see if the distance from corner IC to any of its generating points IPT(L, IC) (RAD2(IC)) is less than the distance from corner IC to the fourth point, (RAD3(IC)) not used to generate IC. If the distance is less, then IPT is left alone. However, if corner IC is closer to the fourth point we want to choose the other possible corner as corner IC. To do this we switch the entries IPT(2, IC) with IPT(3, IC) and call SPOTCR(IC) again. By doing this for all four corners we ensure that IPT is stored in counterclockwise order.

All that remains is to define ICR(L, IC), containing three neighboring corners of corner IC, so that (i) the neighbors of corner IC, ICR (L, IC) $\neq$ IC; (ii) its entries are stored in counterclockwise order; and (iii) its entries are correlated to the entries of IPT as defined before (i.e., among the three generating points of corner IC, IPT(L, IC) is the only one that is not involved in the corner ICR(L, IC). This can all be accomplished by simply initializing

$$\text{ICR}(L, \text{IC}) = \text{IPT}(L, \text{IC}), \qquad \begin{matrix} L = 1, 2, 3, \\ \text{IC} = 1, 2, 3, 4. \end{matrix}$$

The initialization routine consists of:

(i) Define entries for IPT(L, IC), $L = 1, 2, 3$, IC $= 1, 2, 3, 4$.

(ii) Compute corners $\overline{\text{XC}}$(IC).

(iii) Test if IPT is defined correctly.

(iv) Define entries for ICR(L, IC).

The routine that does the initialization is called INADP. By combining INADP

with ADP we now have a complete algorithm for constructing a Voronoi diagram from an arbitrary number of generating points.

Grid Generation

While it is true that the algorithm will work for any random distribution of points, it is desirable for most applications to have a grid which is as uniform as possible. It is known, however, that there is no equipartition of the sphere when the number of points is greater than 20. We have found that the following simple procedure, suggested by the Voronoi construction, yields large grids with more or less equal area polygons and some nice symetries.

We start with one of the regular polyhedrons which can be inscribed in the sphere (e.g., tetrahedron). We then take its vertices as our first group of N_R points. We initialize the construction with points $\bar{X}(1),..., \bar{X}(4)$, and then add points (if any) $\bar{X}(5),..., \bar{X}(N_R)$ using ADP. We now have N_R spherical polygons with $2N_R-4$ corners. Since each corner is equidistant from its three generating points it is logical to use the set of $2N_R-4$ corners as our next set of points to add. We then add points $\bar{X}(N_R+1),..., \bar{X}(3N_R-4)$ one by one using ADP. After adding the $2N_R-4$ additional points we now have $3N_R-4$ polygons with $2(3N_R-4)-4$ corners. This procedure can be repeated automatically until a grid that is fine enough for the particular application has been generated.

In Figs. 5 and 6 we show the grids generated by 5 and 6 applications of this procedure starting with an inscribed tetrahedron with vertices at

$$(0, 0, 1), \quad \left(\sin\frac{2\Pi}{3}, 0, \frac{1}{2}\right),$$

$$\left(\frac{1}{2}\sin\frac{2\Pi}{3}, \sin^2\frac{2\Pi}{3}, -\frac{1}{2}\right), \quad \left(-\frac{1}{2}\sin\frac{2\Pi}{3}, -\sin^2\frac{2\Pi}{3}, -\frac{1}{2}\right).$$

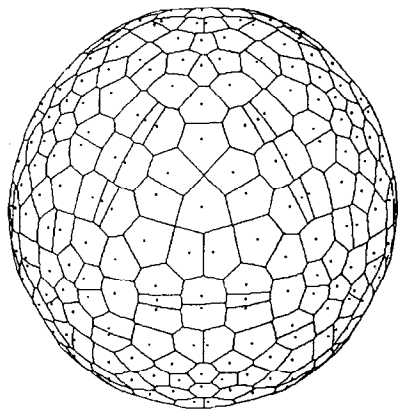

FIG. 5. Stereographic projection onto equatorial plane of Northern Hemisphere of Voronoi diagram consisting of 488 spherical polygons based on an iterative procedure for adding points and starting from an initial configuration derived from an inscribed tetrahedron.

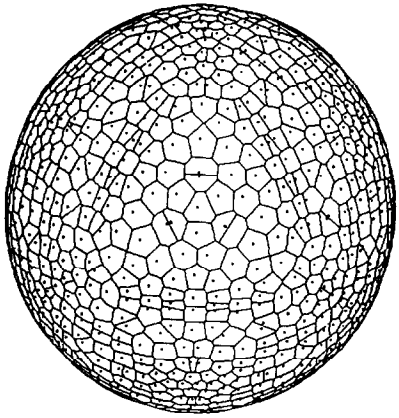

FIG. 6. Stereographic projection onto equatorial plane of Northern Hemisphere of Voronoi diagram consisting of 1460 spherical polygons. This configuration was built up from an initial configuration of an inscribed tetrahedron.

The resulting grids contain 488 points with 5 repetitions of the procedure, and 1460 points with 6 repetitions.

Note that we also have the option to check the points as they are being added and to discard a point if it will be added too close to an existing point.

4. NUMERICAL EXPERIMENTS

We now describe some timing tests which give empirical support to the operation count that was claimed above for both the "typical" case ($O(N)$) and the "worst" case ($O(N^2)$) estimates.

We have placed clock calls into the code so that computing times were printed after the addition of every 100 points. The code was run on an Amdahl V6 serial computer and the results, for three test cases, are summarized in Table I for the grid shown in Fig. 6. The three test cases correspond to a "worst" case, "best" case, and "typical" case. Similar results were also obtained with a completely random grid.

Case I corresponds to a "worst" case. In this case no a priori information was used in the search for the first broken corner. Each point that was added started its search for a broken corner from point number 1. Table I shows that the work in adding 100 points increases linearly with N, and that the amount of work in constructing a Voronoi mesh of N points in $O(N^2)$.

Case II represents the "best" case. In this case perfect a priori information was provided. This is accomplished by using the points from case I and regenerating the Voronoi mesh. Now, however, each point that is added starts its search for the first broken corner from the list of first broken corners (array ICR) that was generated in case I. As each point is added a broken corner will be found on the first guess

TABLE I

Timing Results for Construction of Voronoi Mesh

Number of points	Time (sec)		
	I (Worst)	II (Best)	III (Typical)
100	0.05	0.02	0.03
200	0.19	0.05	0.07
300	0.39	0.08	0.12
400	0.68	0.11	0.17
500	1.17	0.14	0.22
600	1.58	0.17	0.27
700	2.08	0.20	0.33
800	2.89	0.23	0.38
900	3.61	0.27	0.45
1000	4.49	0.30	0.53
1100	5.50	0.33	0.57
1200	6.58	0.36	0.63
1300	7.66	0.39	0.68
1400	9.43	0.43	0.77
1460	10.82	0.44	0.85

Note. Case I: initial construction with no a priori information → "worst" case $O(N^2)$. Case II: initial construction with a priori information → "best" case $O(N)$. Case III: Voronoi mesh is reconstructed after moving the points with a physical flow field. Information from previous construction is used → "typical" case $O(N)$.

and no extra searching is required. Table I shows that the work in adding 100 points in independent of N, and that the total amount of work in constructing a Voronoi mesh of N points is $O(N)$.

Case III represents a "typical" case. In this case we simulate a typical Lagrangian fluid dynamics calculation [1, 3] by moving the points $\bar{X}_j$ off the sphere by approximating the equation $d\bar{X}_j/dt = \bar{U}_j$ with the discrete equation

$$\bar{X}_j^* = \bar{X}_j^n + \Delta t\, \bar{U}_j^n, \qquad j = 1, 2, \ldots, N, \tag{1}$$

where $\bar{X}_j^n = (X_j^n, Y_j^n, Z_j^n)$. $\bar{X}_j^*$ is then projected back onto the sphere by

$$\bar{X}_j^{n+1} = \frac{\bar{X}_j^*}{|\bar{X}_j^*|}. \tag{2}$$

The velocity $\bar{U}_j^{n+1}$ is then evaluated at the new points $\bar{X}_j^{n+1}$. The new Voronoi mesh is then constructed. This completes one iteration. In our experiment we have chosen $\bar{U}_j$ to correspond with the Rossby–Haurwitz wave used in meteorology and given by the stream function [18],

$$\psi(\theta, \lambda) = -\omega \sin\theta + \omega \cos^4\theta \sin\theta \cos 4\lambda, \tag{3}$$

where λ = longitude, θ = latitude, ω = rotation rate of earth. A time step of 2 h (which in large by meterological standards) was used. After six iterations this procedure simulates a 12-h calculation. The results are displayed in Table I. As in case II, information from the previous Voronoi construction is used to start the search for the first broken corner as each point is added. This drastically reduces the searching involved in a "typical" case. The results in Table I show that the work in adding 100 points is independent of N and that the total amount of work in constructing the Voronoi mesh is $O(N)$.

5. Conclusion

In this paper we have presented an efficient construction of the Voronoi diagram on the sphere. Our algorithm differs from other constructions of the Voronoi diagram in that it is based on adding one point at a time to build up the complete polygonal structure. The algorithm is particularly well suited for use in time-dependent numerical codes where information from constructing the Voronoi diagram at the previous time step can be used to reduce the number of operations to $O(N)$ when the changes from one time step to the next are small. The construction does not break down, however, even when the changes are very large.

Another advantage of adding the points one at a time is that it is quite simple to add more points to grid sparse regions of the flow during a calculation without recomputing the whole grid. This should prove useful with the new adaptive grid codes being developed [4].

The fortran program SPHVOR is available, from the first author, upon request.

Acknowledgments

It is a pleasure to acknowledge many helpful discussions with Donald Goldfarb during the initial development of the algorithm described in this paper. The computations were carried out at the Courant Mathematics and Computing Laboratory of New York University under Contract DE-AC02-76ER03077 with the U. S. Department of Energy, and at the NASA–Goddard Space Flight Center where the first author is a National Research Council Resident Research Associate.

References

1. J. Augenbaum, "A New Lagrangian Method for the Shallow Water Equations," Ph. D. thesis, New York University, New York, 1982.
2. J. Augenbaum, *J. Comput. Phys.* **53** (1984), 240.
3. J. Augenbaum, A Lagrangian method for the shallow water equation based on Voronoi mesh-flows on a rotating sphere, *in* "Proceedings of Free Lagrangian Methods Conference," Springer-Verlag, New York/Berlin, in press.
4. M. Berger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," Ph. D. thesis, Stanford University Stanford, Calif., 1982.
5. W. Bristow, J. P. Dussault, and B. L. Fox, *J. Comput. Phys.* **29** (1978), 81.

6. W. P. CROWLEY, "Proceedings of the Second International Conference on Numerical Methods in Fluid Mechanics", Springer-Verlag, New York/Berlin, 1971.
7. J. DUKOWICZ, Lagrangian fluid dynamics using the Voronoi–Delaunay mesh *in* "Numerical Methods for Coupled Problems," Pineridge Press, Swansea, U. K., 1981.
8. J. L. FINNEY, *J. Comput. Phys.* **32** (1979), 137.
9. M. FRITTS AND J. BORIS, *J. Comput. Phys.* **31** (1979), 173.
10. F. A. HARLOW, *Proc. Symp. Appl. Meth.* **15** (1963), 289.
11. C. HIRT, A. A. AMSDEN, AND J. L. COOK, *J. Comput. Phys.* **14** (1974), 277.
12. F. MESINGER, *Mon. Weather Rev.* **99** (1971), 15.
13. C. S. PESKIN, A Lagrangian method for the Navier Stokes equations with large deformations, preprint.
14. M. I. SHAMUS AND D. HOEY, "Proceedings, 16th Ann. IEEE Symposium on Foundations of Algorithms, Berkeley, Calif., 1975."
15. M. TANEMURA, T. OGAWA, AND N. OGITA, *J. Comput. Phys.* **51** (1983), 191.
16. H. E. TREASE, "A Two Dimensional Free Lagrangian Hydrodynamics Model," Ph. D. thesis, University of Illinois at Urbana–Champaign, 1981.
17. G. VORONOI, *J. Reine Angew Math.* **134** (1908), 198.
18. D. L. WILLIAMSON AND G. L. BROWNING, *J. Appl. Meteorol.* **9** (1973), 272.
19. C. BORGERS AND C. S. PESKIN, A Lagrangian method based on the Voronoi diagram for the incompressible Navier–Stokes equation on a periodic domain, *in* "Proceeding of the Free Lagrangian Methods Conference," Springer-Verlag, New York/Berlin, in press.