Comput. & Graphics Vol. 9, No. 4, pp. 407–413, 1985
Printed in Great Britain.

0097-8493/85 $3.00 + .00

Technical Section

EFFICIENT ALGORITHMS FOR AUTOMATIC VIEWER ORIENTATION

DAVID P. ANDERSON
Computer Sciences Department, University of Wisconsin–Madison, 1210 West Dayton Street, Madison, WI 53706, U.S.A.

Abstract—The calculation of a perspective image of an object involves the position and orientation of the viewer. In many graphics applications the viewpoint and object are fixed, and an orientation is sought in which the object is "centered" in the field of view. Previous work has proposed that the viewing direction be the axis of the narrowest circular cone emanating from the viewpoint and containing the object, and has shown how this direction can be calculated based on the viewpoint and a set of n object points which includes the object's extreme points. This paper presents algorithms which efficiently accomplish this task, in $O(n)$ average and $O(n \log(n))$ worst-case time. The orientation problem is converted into a problem in spherical geometry, and the proposed algorithms are based on existing algorithms for the analogous plane geometry problems.

1. INTRODUCTION

The orientation of a viewer in three-space is described by unit vectors R, A and U representing the right, ahead and up directions, respectively. These vectors must satisfy

$$A = U \times R, \quad U = R \times A. \tag{1}$$

In the *viewer coordinate system* the viewpoint V is the origin and R, A and U are basis vectors. The viewer coordinates of a point p are (r, a, u), where

$$r = (p - V) \cdot R, \quad a = (p - V) \cdot A, \quad u = (p - V) \cdot U. \tag{2}$$

The *image plane* for perspective projection is the plane normal to A and passing through the point $(V + A)$. Its natural coordinate system has origin $V + A$ and basis vectors R and U. The perspective image of an object point p is the intersection of the line containing p and V with the image plane. If (r, a, u) are the viewer coordinates of p as given by (2), the image plane coordinates of the image of p are (x, y), where

$$x = \frac{r}{a}, \quad y = \frac{u}{a}. \tag{3}$$

If a is nonpositive then p is not visible to a viewer in the given position and orientation, and its projection is not defined. An orientation will be called *feasible* for a particular viewpoint and object if a is positive for all object points, that is, if the object is entirely in front of the viewer.

Given a position V and an object, what is the optimal orientation? Intuitively, we want to look towards the center of the object. One could take this to mean the center of mass of the object, or the center of the smallest box containing the object, but these *ad hoc* choices are inadequate in some cases, particularly when the viewpoint is close to the object. Anderson [1] proposes this criterion: The optimal "ahead" vector A is the axis of the circular cone with vertex P of least vertex angle which encloses the object, if such a cone exists. This choice of A minimizes the maximum local distortion due to perspective projection, and provides a feasible orientation if one exists. Once A is determined, the orientation given by

$$R = \frac{A \times (0, 0, 1)}{|A \times (0, 0, 1)|}, \quad U = R \times A \tag{4}$$

is optimal in the sense that U makes as small as possible an angle with the (0, 0, 1) direction, which we assume is the "up" direction in 3-space.

For our purposes, the object to be drawn will be represented by a finite set X of points in 3-space, whose convex hull includes the object. If the object is polyhedral, X can be its vertex set. Curved or textured objects can be represented by the vertices of an enclosing polyhedron. The orientation based on these points will then approximate the optimal orientation for the original object.

Translating X if necessary, we may assume that V is the origin. It is then clear that multiplying the elements of X by positive scalars does not affect the solution. Let Y be the set of the vectors in X normalized to length one. Then the smallest cone enclosing X is generated by the smallest circle on the sphere that encloses Y (Fig. 1). This circle is determined by the points of Y which are extreme in the geometry of the sphere (defined in Section 2). These spherical extrema are not to be confused with the extreme points of X in $\mathbf{R}^3$; each spherical extremum in Y arises from an extreme point of X, but not conversely.

An algorithm for finding the smallest enclosing circle (and hence the optimal viewing direction) is given in [1]: the h spherical extrema are found in $O(n^2)$ time. An $O(h^4)$ worst-case procedure then finds the smallest circle enclosing these points. Since all the points may be spherical extrema, the worst case time is $O(n^4)$. This is unacceptably large since X may contain thousands of points.

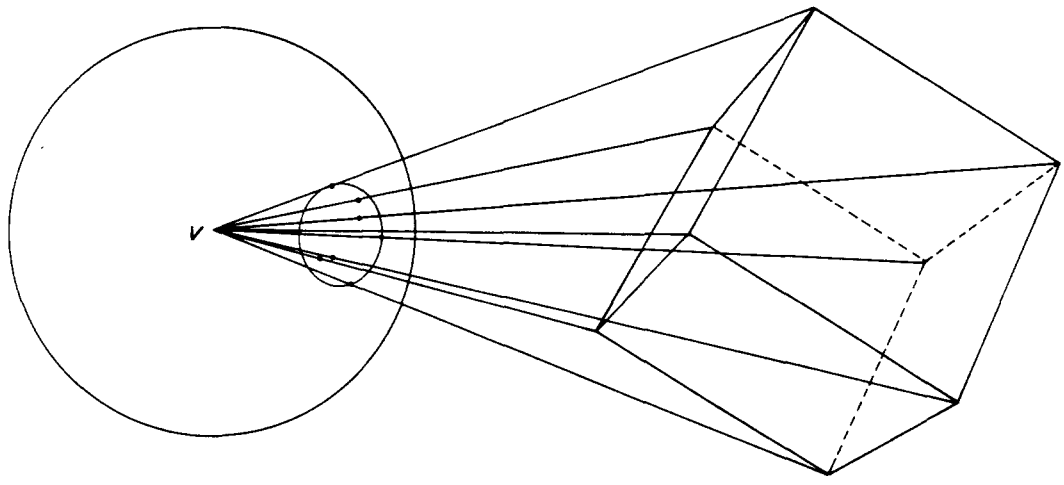

Fig. 1. The smallest cone and smallest circle problems.

Lawson [8], motivated by an application outside computer graphics, points out that the problem can be solved by finding the smallest sphere in 3-space which encloses Y, and intersecting this with the unit sphere. The resulting circle is the smallest enclosing circle. He refers to an iterative algorithm for approximating the smallest enclosing sphere. The convergence rate, however, appears to be poor even for small sets of points.

This paper describes three algorithms:

(1) An $O(n \log(n))$ worst-cast algorithm for finding the convex hull (i.e. an ordered list of the extreme points) of n points on the sphere. For certain input distributions the average runtime is $O(n)$.

(2) An $O(n \log(n))$ algorithm for finding the smallest circle enclosing n points on the sphere, making use of the farthest-point spherical Voronoi diagram of the set.

(3) An algorithm for finding the smallest circle enclosing a set of points on the sphere, given its n-vertex convex hull. This algorithm is simpler than (2) and is slow ($O(n^2)$) in the worst case but may be $O(n)$ on the average.

These algorithms can be combined in several ways to solve the optimal orientation problem:

(A) Use (2) by itself.

(B) Use (1) to find the extreme points, followed by (2).

(C) Use (1) to find the convex hull, followed by (3).

2. SPHERICAL GEOMETRY—DEFINITIONS

In spherical geometry, we consider only points that lie on the surface of the unit sphere. A *great circle* is a circle of radius one on the sphere. Great circles play the role of lines. A *small circle* is a circle on the sphere which is not a great circle. A small circle divides the sphere into two regions; its *interior* is the smaller of these.

The *opposite* of a set of points is its reflection through the origin. The *segment* between two nonopposite points p and q is the shorter arc of the great circle containing them. The distance $d(p, q)$ between p and q is the length of this arc. The *midpoint* of p and q is the point on this arc equidistant from p and q.

A set is *hemispherical* iff it is contained in some open half-sphere. A set X is *convex* iff either it is the entire sphere, or it is hemispherical and whenever $p, q \in X$, the segment between p and q is also in X. The *convex hull* of a set X is the intersection of all convex sets containing X; this set is itself convex. If X is finite and hemispherical its convex hull is polygonal; the vertices of this polygon are in X and are called the *extreme points* of X.

We will sometimes refer to a clockwise or counterclockwise traversal of a hemispherical closed curve. The direction is relative to a viewpoint outside the sphere from which the entire curve is visible.

For computational purposes, a spherical point can be represented as a unit vector in 3-dimensional Cartesian coordinates. The distance $d(p, q)$ is the angle between the vectors p and q. A line (great circle) can be represented by either unit normal vector of the plane containing it. A particular normal vector corresponds to a direction of the line. One can determine whether a point lies to the left or right of a directed line by projecting the point's vector onto the line's normal. The statement "r lies to the right of the segment from p to q" means

$$r \cdot (q \times p) \geq 0. \tag{5}$$

The two intersections of a pair of great circles are the normalized cross-product of the two normal vectors, and its negative.

We will speak of ordering a set X of points according to their angle about a point p. This can be done by constructing the tangent plane to the sphere at p, projecting the vectors from p to points in X onto this plane, and ordering the projections about some direction in the plane. The vertices of a convex spherical polygon, traversed in a particular direction, have monotonic angles about any point in the interior of the polygon.

3. A SPHERICAL CONVEX HULL ALGORITHM

Numerous algorithms have been proposed for finding the convex hull of a set of points in the plane, and

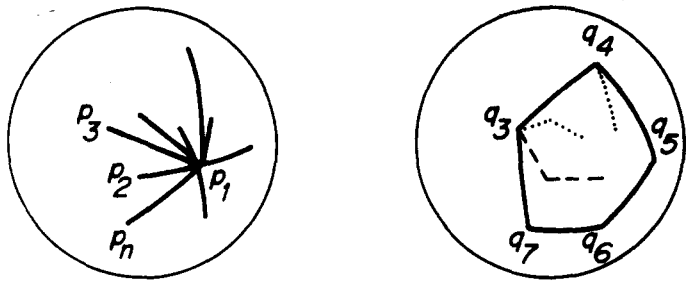

Fig. 2. Graham's convex hull algorithm on the sphere. (a) shows the points sorted in angular order around p_1. (b) shows the convex hull; the dotted segments are those eventually deleted from the tail of the list q, and the dashed segments are those eventually deleted from the head.

many of these can be adapted to spherical geometry. Certain difficulties can arise; for example, in planar geometry the four points with extreme x or y coordinates are always extreme points, whereas in the sphere no coordinate system has a similar property. In addition, a spherical convex hull algorithm must be able to detect nonhemispherical sets.

The following spherical convex hull algorithm is an adaptation of the planar algorithm proposed by Graham [7] (see Fig. 2):

Input: n spherical points $p_1, \ldots, p_n$.
Output: a list $q_j, \ldots, q_k$ of the vertices of the convex hull of $p_1, \ldots, p_n$,
  traversed clockwise, or an indication that the set is not hemispherical.
If one of $p_2, \ldots, p_n$ is opposite p_1, stop (the set is not hemispherical).
Reorder $p_2, \ldots, p_n$ so that they are in clockwise angular order about p_1.
$q_1 \leftarrow p_1$ ($q_j, \ldots, q_k$ are the extreme points found so far)
$q_2 \leftarrow p_2$
$j \leftarrow 1$
$k \leftarrow 2$
for $i \leftarrow 3$ to n
    $temp \leftarrow q_k$
    while p_i is to the left of the segment from q_{k-1} to q_k
        $k \leftarrow k - 1$
        exit while if $j = k$
    end while
    if $j = k$
        if p_i is to the left of the segment from $temp$ to q_j
            stop (the set is not hemispherical)
        else
            $q_{j+1} \leftarrow p_i$
            $q_{j+2} \leftarrow temp$
            $k \leftarrow j + 2$
        end if
    else
        $k \leftarrow k + 1$
        $q_k \leftarrow p_i$
        while p_i is to the left of the segment from q_j to q_{j+1}
            $j \leftarrow j + 1$
        end while
    end if
end for

The differences between this algorithm and Graham's planar algorithm are: (a) in the planar algorithm, p_1 is a known extreme point (e.g. a coordinate extremum), and so deletions need be done only from the tail of the list q, whereas here deletions are done from both ends; (b) in the plane the special case of a nonhemispherical set does not exist; in the sphere (but not in the plane) it is possible for a point to lie on the "wrong side" of all the edges of a convex polygon.

Proof of correctness for the algorithm has two parts:

(1) Proof that it works correctly when the set is hemispherical. This is similar to the planar proof and we omit it.

(2) Proof that it detects nonhemispherical sets correctly. This follows from the following observation: if X is a hemispherical convex polygonal region and p is

a point, then $X \cup \{p\}$ is nonhemispherical iff p lies in the opposite of X, that is, iff p lies on the left of all the clockwise-directed edges of X.

The worst-case time used by both the planar and spherical algorithms is $O(n \log(n))$; the $O(n \log(n))$ portion arises from sorting the points into angular order, and the rest of the calculation is $O(n)$. Under certain statistical assumptions on the original set of points, the distribution of the set of angles will satisfy the requirements of linear-time sorting algorithms such as bucket sort, reducing the expected runtime to $O(n)$. Other convex hull algorithms with $O(n)$ expected time [2, 10] could also be adapted to the sphere.

4. THE SPHERICAL SMALLEST ENCLOSING CIRCLE PROBLEM

The spherical smallest-circle problem is as follows: Given a hemispherical set X of points on the sphere, what is the small circle of least radius such that X is contained in the union of the circle and its interior? Note that if X is not hemispherical, no such circle exists.

The analogous planar problem is: Given a finite X set of points in the plane, what is the smallest circle that encloses X? In both the planar and spherical cases, the smallest circle is determined by the two farthest points in X, or by three points. This suggests a $O(n^4)$ worst-case exhaustive search algorithm. A more clever algorithm, to which we will return later, reduces the worst-case time to $O(n^2)$. An $O(n \log(n))$ worst-case algorithm, based on Voronoi diagrams, is reported by Shamos [11].

4.1. *Voronoi diagrams and smallest enclosing circles*

The closest-point Voronoi diagram ($V_c(X)$ of a planar set $X = \{x_1, \ldots, x_n\}$ is defined as follows: let R_i be the region in the plane consisting of those points which are closer to x_i than to any x_j, $j \neq i$. It is easy to see that the R_i are disjoint, polygonally-bounded regions which, together with their boundaries, cover the plane. $V_c(X)$ is the graph formed by the boundaries of the R_i. The farthest-point Voronoi diagram $V_f(X)$ is defined similarly, with "closest" replaced by "farthest."

If X is a set of points on the sphere, the spherical closest- and farthest-point Voronoi diagrams of X are defined similarly, with "plane" replaced by "sphere." An example of a closest-point spherical Voronoi diagram is shown in Fig. 3. Closest- and farthest-point spherical Voronoi diagrams are related as follows:

Lemma 1.

Let X be a set of spherical points, and let X' be the opposite of X. Then

$$V_f(X) = V_c(X'). \tag{6}$$

In words, the farthest-point Voronoi diagram of a set is the closest-point diagram of its opposite.

Proof. Observe that

$$d(x, p) = \pi - d(x', p), \tag{7}$$

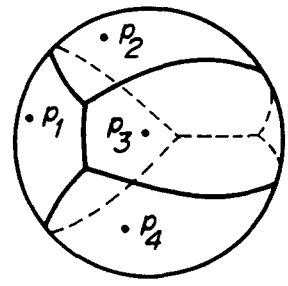

Fig. 3. The closest-point Voronoi diagram for four points on the sphere.

since the distances are complementary arcs of the half-circle through x, p and x'. Now p is in the ith region of the closest-point diagram of X iff

$$d(p, x_i) < d(p, x_j) \quad \text{for all} \quad j \neq i, \tag{8}$$

which, using (7), is equivalent to

$$d(p, x_i') > d(p, x_j') \quad \text{for all} \quad j \neq i, \tag{9}$$

which means p is in the ith region of the farthest-point Voronoi diagram of X'. ■

It is known [11] that the largest possible number of vertices and edges of the closest- or farthest-point planar Voronoi diagram of a set of n points is $O(n)$. This is true in the sphere also.

If X is a set of n planar points for which $V_f(X)$ is known, the smallest circle enclosing X can be found in $O(n)$ time as follows (see [11]):

(1) For each edge S of $V_f(X)$, see if S contains the midpoint of the two points in X which determine it. If so, these two points determine the smallest circle and their midpoint is its center.

(2) Otherwise, for each vertex v of $V_f(X)$, find the distance r_v from v to any of the three (or more) points in X which determine it (v is equidistant from these points). Clearly the circle centered at v with radius r_v encloses X. The vertex for which r_v is minimal is the center of the smallest circle enclosing X.

This procedure works in the spherical case as well, as long as X is hemispherical. The "midpoint" in step (1) must be the one defined in Section 2 (note that the opposite of this point is also equidistant from the two points). In step (2) we only consider those vertices v for which $r_v < \pi/2$.

4.2. *Computing spherical Voronoi diagrams*

For the purpose of finding enclosing circles we are interested in computing the farthest-point spherical Voronoi diagram of a set of points. By Lemma 1 this is equivalent to finding the closest-point diagram of the opposite of a set, and since this is easier to visualize we will describe a closest-point algorithm.

The $O(n \log(n))$ algorithm for computing the closest- or farthest-point Voronoi diagram of a set X of n points in the plane, given by Shamos in [11], can be adapted to the sphere. The planar algorithm uses a divide-and-conquer approach based on a procedure for merging

the Voronoi diagrams of two sets which are separable (i.e. have disjoint convex hulls), in time proportional to the sum of the sizes of the sets. Given a set, the algorithm splits it in half around the median x coordinate (thus ensuring that the subsets are separable), recursively finds the Voronoi diagram of each subset, then merges the diagrams.

First we give a rough description of a procedure for merging the closest-point Voronoi diagrams of two spherical sets X and Y, under the assumptions that X and Y are hemispherical and separable. As in the planar case, this involves constructing a piecewise linear curve P consisting of those points equidistant from X and Y. Whereas in the plane P goes to infinity in two directions, on the sphere it is a simple closed curve. P separates X from Y. $V_c(X \cup Y)$ is formed by removing the portions of $V_c(X)$ which lie on the other side of P from X, removing the portions of $V_c(Y)$ which lie on the other side of P from Y, and joining the remaining portions of $V_c(X)$ and $V_c(Y)$ with P.

P is computed as follows ($B(a, b)$ denotes the perpendicular bisector of a and b, oriented so that it crosses the segment from a to b going from right to left):

1. Find points $x_0 \in X$, $y_0 \in Y$ of minimal distance. This can be done in $O(|X| + |Y|)$ time, using an algorithm analogous to the planar algorithm given in [4]. The idea is to start with an arbitrary point $a_0 \in X$, find the closest point $b_0 \in Y$, then find the point $a_1 \in X$ which is closest to b_0, and so forth until fixed points are found. It is necessary to traverse each convex hull in one direction only.

2. Let p_0 be the midpoint of x_0 and y_0. P will initially be extended from p_0 in the direction $B(x_0, y_0)$. Let $k = 0$ (k is the iteration number). Let A_0 and B_0 be the Voronoi regions (relative to X and Y, respectively) in which x_0 and y_0 lie.

3. In general, we have computed a vertex p_k on P, and are extending P along a ray R, within Voronoi regions A_k and B_k corresponding to points x_k and y_k. If R hits the boundary of A_k before the boundary of B_k, let p_{k+1} be this point of intersection, let A_{k+1} be the Voronoi region on the other side of the boundary, let x_{k+1} be the corresponding point in X, and let $B_{k+1} = B_k$ and $y_{k+1} = y_k$. P will now be extended along $B(x_{k+1}, y_{k+1})$. Proceed analogously if R hits the boundary of B_k first.

4. The procedure stops when $A_k = A_0$ and $B_k = B_0$. This occurs when P has wrapped around to its starting point p_0. Otherwise increment k and go to step 3.

Given this merging procedure, the closest-point spherical Voronoi diagram of a set X can be computed as follows:

1. Choose some point p such that neither p nor its opposite is in X, and order the points in X according to their angle about p.

2. Choose a great circle containing p but disjoint from X. This circle divides X into sets X_0 and X_1.

3. Divide X_0 into pairs of points which are adjacent in the angular order. The Voronoi diagram of each pair is their perpendicular bisector. There may be a left-over single point, whose Voronoi diagram is null.

4. Merge the Voronoi diagrams for adjacent pairs into diagrams for quadruples, and so forth until $V_c(X_0)$ and $V_c(X_1)$ have been obtained.

5. Merge $V_c(X_0)$ and $V_c(X_1)$ to obtain $V_c(X)$. Note that at each stage we are merging the diagrams of sets which are hemispherical and separable, so the previous procedure can be used.

4.3. *A simpler spherical smallest circle algorithm*

The relative complexity of the Voronoi diagram algorithm, together with the availability of a fast, simple convex hull algorithm and the possibility that in many cases the number of extreme points grows as $\sqrt{n}$ or less, suggest that a simple smallest circle algorithm, even of $O(n^2)$ time, may be preferable to the Voronoi diagram approach.

Elzinga and Hearn [6] propose an algorithm which starts with a small circle and iteratively enlarges it until the optimum is found. We will give a slightly simpler algorithm which starts with a circle enclosing all the points, and iteratively shrinks it. The algorithm is based on the following

Lemma 2.

If X is a finite set of points in the sphere, and C is a small circle which encloses X, then C is the minimal circle enclosing X iff every closed semicircle of C intersects X.

Proof. Suppose C encloses X, and D is a closed semicircle of C disjoint from X. Let a and b be the angular extrema of $X \cap C$. Let p be the midpoint of a and b, and let q be $(p - c)$, where c is the center of C. For each $x \in X$, let α_x be the largest α such that $\alpha \leq 1$ and the circle with center $c + \alpha q$ passing through a and b contains x. Note that $\alpha_x > 0$ for all $x \in X$. Let $\bar{\alpha} = \min_{x \in X} \alpha_x$. Then the circle centered at $c + \bar{\alpha} q$ passing through a and b encloses X, and is smaller than C. ■

This motivates an algorithm which starts with an enclosing circle and iteratively shrinks it using the procedure of the proof of Lemma 2, until the condition of Lemma 2 is satisfied. Although this approach will work with an arbitrary set of points, a more efficient version is possible if X is given by the vertices $x_0, \ldots, x_{k-1}, x_k = x_0$ of its convex hull, traversed in clockwise order.

1. Let x_i be an arbitrary point in X. Find the point in X which is farthest from x_i; renumber if necessary so that this point is x_0. Let C_0 be the circle centered at x_i with radius $d(x_0, x_i)$, and let $c_0 = x_i$.

2. Let x_{cw_0} and x_{ccw_0} be the points in $X \cap C_0$ which are closest, in the clockwise and counterclockwise directions, respectively, to the point in C_0 opposite x_0. If x_0 is the only point in $X \cap C_0$, let $cw_0 = k$ and $ccw_0 = 0$. In all cases $cw_0 > ccw_0$. Let $j = 0$ (j is the iteration number).

3. If c_j does not lie strictly to the right of the segment from x_{cw_j} and x_{ccw_j}, then stop; C_j is optimal by Lemma 2.

4. For each x_i, $ccw_j < i < cw_j$, define α_{x_j} as in the proof of Lemma 2. Viewing all points as unit vectors,

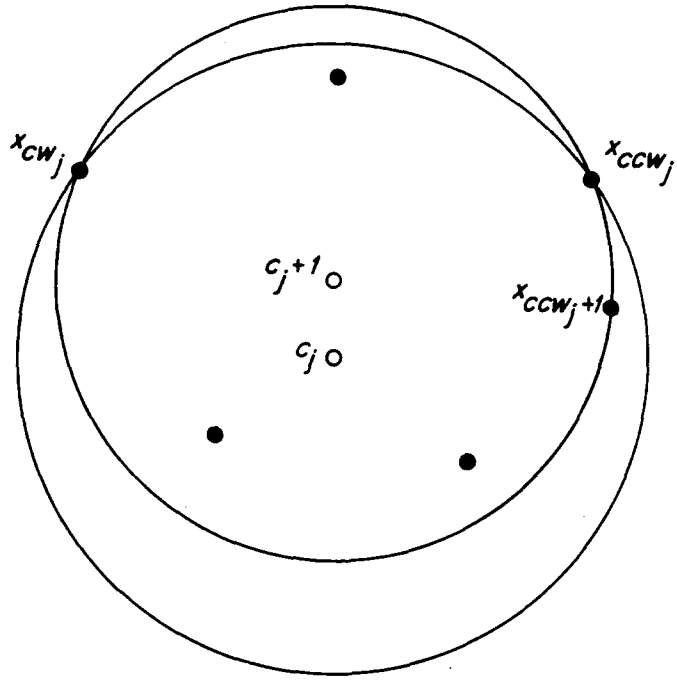

Fig. 4. One iteration of the smallest-circle algorithm of section 4.3.

α_{x_i} can be computed as the maximum of 1 and the root of the linear equation

$$(c_j + \alpha q) \cdot x_{cw_j} = (c_j + \alpha q) \cdot x_i, \tag{10}$$

where

$$q = (x_{cw_j} + x_{ccw_j}) - c_j. \tag{11}$$

Let $\bar{\alpha} = \min_{ccw<i<cw} \alpha_i$. Note that $\alpha_i = 1$ for $0 \le i \le ccw$ and $cw \le i < k$ so these need not be computed.

5. Let c_{j+1}, the new center, be $c_j + \bar{\alpha} q$ normalized to unit length, and let $r_{j+1} = d(c_{j+1}, x_{cw_j})$ be the new radius. Let cw_{j+1} be least such that $d(c_{j+1}, x_{cw_{j+1}}) = r_{j+1}$ and $x_{cw_{j+1}}$ is to the left of q. Let ccw_{j+1} be greatest such that $d(c_{j+1}, x_{ccw_{j+1}}) = r_{j+1}$ and $x_{ccw_{j+1}}$ is to the right of q. Increment j. Go to step 3.

The algorithm generates a sequence of circles, each of which encloses X, is determined by two or three points of X, and is smaller than the previous circle (see Fig. 4). It terminates only when the optimality condition is met. These facts imply its correctness.

The work at each iteration is proportional to $ccw - cw$; this difference is at most n and decreases by at least 1 at each iteration, so the worst-case performance is $O(n^2)$. The average performance in practice will probably be better; it is reasonable to expect that the difference $ccw - cw$ will shrink, on the average, by constant ratio at each iteration, in which case the runtime will be $O(n)$.

5. DISCUSSION

We will now discuss the relative merits of the algorithms A, B and C (see the Introduction), both in terms of their performance and their implementation difficulty. We have implemented algorithm C; the procedure is short (300 lines) and uses only simple data structures. We have not implemented the spherical Voronoi diagram algorithm, but we suspect that it would be longer and more complex.

Algorithms B and C consist of a convex hull phase followed by a smallest-circle phase. The average performance of these algorithms depends on how the distribution of h changes as n increases, where n is the size of the original set X and h is the number of spherical extrema of the normalized set Y. If the object is a n-point approximation to a circle then $h = n$. If the object has a fixed silhouette and detail is added only to the middle, then $h = O(1)$. These are the extreme cases; more useful estimates can be obtained either by looking at large sets of real-world data, or by considering random distributions of input data. We will show that under certain assumptions, the average total runtime may be $O(n)$ even if the smallest-circle calculation takes $O(n \log(n))$ or worse.

For algorithm B, let us assume that the convex hull algorithm runs in $O(n)$ average time for a particular input distribution, and that it is followed by the $O(h \log(h))$ Voronoi diagram algorithm for finding the smallest enclosing circle. Bentley and Shamos show [2] that if

$$E[h] = O(n^p), \quad p < 1, \tag{12}$$

then

$$E[h \log(h)] = O(n^q), \quad q < 1. \tag{13}$$

Hence if (12) holds, the average total time of algorithm B is still $O(n)$.

It is known [5] that when n points are chosen from a uniform distribution on a planar disk, the expected number of extreme points is

$$E[k] = O(n^{1/3}). \tag{14}$$

This suggests that if object points are chosen randomly from a truncated circular cone emanating from the viewpoint (so that their projections on the sphere are uniformly distributed over a circle on the sphere) then eqn (12) will hold if the angle of the cone is sufficiently small. Similarly, if the points are chosen from a bivariate normal distribution in the plane, then

$$E[h] = O(\log^{1/2}(n)), \tag{15}$$

which also satisfies (12). Hence for some input distributions the average time of algorithm B is $O(n)$.

For algorithm C, suppose the $O(n)$ expected-time convex hull algorithm is followed by an $O(h^2)$ expected-time smallest-circle algorithm. Then the expected time for the smallest-circle computation is

$$E[h^2] = \mathrm{Var}(h) + E[h]^2 \tag{16}$$

so the total time for algorithm C is $O(n)$ as long as $\mathrm{Var}(h) = O(n)$ and

$$E[h] = O(n^p), \quad p \le \tfrac{1}{2}. \tag{17}$$

Unfortunately, there apparently no results concerning the variance of h for interesting point distributions. It should also be pointed out that the smallest-circle algorithm of Section 4.3 may be $O(n)$ average for some distributions, in which case algorithm C is also $O(n)$ average regardless of the distribution of h.

6. CONCLUSION

We have shown that finding the optimal orientation reduces to the spherical smallest-circle problem, and have given three algorithms for efficiently solving this problem. The third of these (algorithm C) stands out as being both simple and fast. Such a procedure would be a useful addition to any three-dimensional graphics software system.

We have dealt only with the case of a single object for which a feasible orientation exists. This may be too limited for some applications. For example, if there are several objects X_i one might attach weights w_i to each object, and define the viewing direction $\bar{v}$ to be the solution of

$$\min_{|v|=1} \sum_{i=1}^{n} w_i f(v, X_i), \tag{18}$$

where $f(v, X_i)$ is some monotonic function of the angle between v and the center of the minimal circular cone enclosing X_i. Varying these weights continuously would generate smooth "pans" between different objects. Note that this method requires that there be a feasible orientation for each object, but not necessarily for the union of the objects.

REFERENCES

1. D. P. Anderson, An orientation method for central projection programs. *Comput. Graphics* **6**, 35–37 (1982).
2. J. L. Bentley and M. I. Shamos, Divide and conquer for linear expected time. *Inform. Proc. Lett.* **7**, 87–91 (1978).
3. J. L. Bentley, B. W. Weide and A. C. Yao, Optimal expected time algorithms for closest point problems. *ACM Trans. Math. Software* **16**, 563–580 (1980).
4. F. Chin and C. A. Wong, Minimum vertex distance between separable convex polygons. *Inform. Proc. Lett.* **18**, 41–45 (1984).
5. B. Efron, The convex hull of a random set of points. *Biometrika* **52**, 331–344 (1965).
6. J. Elzinga and D. W. Hearn, Geometrical solutions for some minimax location problems. *Trans. Sci.* **6**, 379–394 (1972).
7. R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Proc. Lett.* **1**, 132–133 (1972).
8. C. L. Lawson, The smallest covering cone or sphere. *SIAM Rev.* **7**, 415–417 (1965).
9. D. T. Lee and R. L. Drysdale, Generalizations of Voronoi diagrams in the plane. *SIAM J. Comput.* **10**, 73–87 (1981).
10. F. P. Preparata and S. J. Hong, Convex hull of finite sets of points in two and three dimensions. *CACM* **20**, 87–93 (1977).
11. M. I. Shamos and D. Hoey, Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science* (1975), pp. 151–162.