Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 93 (2005) 83–89
Information Processing Letters
www.elsevier.com/locate/ipl

Optimal bounding cones of vectors in three dimensions ☆

Gill Barequet *, Gershon Elber

Center for Graphics and Geometric Computing, Department of Computer Science, The Technion—Israel Institute of Technology, Haifa 32000, Israel

Received 13 December 2003

Communicated by S.E. Hambrusch

Abstract

The problem of computing the minimum-angle bounding cone of a set of three-dimensional vectors has numerous applications in computer graphics and geometric modeling. One such application is bounding the tangents of space curves or the vectors normal to a surface in the computation of the intersection of two surfaces.

No optimal-time exact solution to this problem has been yet given. This paper presents a roadmap for a few strategies that provide optimal or near-optimal (time-wise) solutions to this problem, which are also simple to implement. Specifically, if a worst-case running time is required, we provide an $O(n \log n)$-time Voronoi-diagram-based algorithm, where n is the number of vectors whose optimum bounding cone is sought. Otherwise, if one is willing to accept an, *in average*, efficient algorithm, we show that the main ingredient of the algorithm of Shirman and Abi-Ezzi [Comput. Graphics Forum 12 (1993) 261–272] can be implemented to run in optimal $\Theta(n)$ expected time. Furthermore, if the vectors (as points on the sphere of directions) are known to occupy no more than a hemisphere, we show how to simplify this ingredient (by reducing the dimension of the problem) without affecting the asymptotic expected running time. Both versions of this algorithm are based on computing (as an LP-type problem) the minimum spanning circle (respectively, ball) of a two-dimensional (respectively, three-dimensional) set of points.

Keywords: Algorithms; Curves and surfaces; Minimum spanning cone; Minimum spanning circle; Maximum empty circle

1. Introduction

A cone $\mathcal{B} = [\vec{v}, \theta]$ in three-space has an axis $\vec{v}$ and angular span θ. A cone whose opening angle is $\pi/2$ is a plane. A cone whose angular span is between $\pi/2$ to π is called a "reflex" cone.

Given a set of spatial vectors, we seek their minimum bounding cone, that is, the cone with minimum angular span that contains all the vectors. Bounding cones are employed in computer graphics and geomet-

☆ Work on this paper by the first author has been supported in part by a grant for Korean–Israeli Research Cooperation. Work of both authors has been supported in part by AIM@SHAPE, a grant of the European Commission 6th Framework.

* Corresponding author.
E-mail addresses: barequet@cs.technion.ac.il (G. Barequet), gershon@cs.technion.ac.il (G. Elber).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2004.09.019

ric modeling for bounding entities in vector spaces, most noticeably tangents and normals of free-form shapes [2]. Other applications of bounding cones are found in the computations of illumination and radiosity [9] and of visibility maps [3].

Two near-optimal iterative algorithms (but with no proven quality) are given by Sederberg and Meyers [7], and by Meenakshisundaram and Krishnan [4]. Given a set of n vectors $\{D_i\}$, both algorithms start with some initial bounding cone $\mathcal{B}_1 = [V_1 = D_1, 0]$ and refine it in each iteration i (for $2 \leqslant i \leqslant n$) by computing a new (but not necessarily the minimum) cone $\mathcal{B}_i = [V_i, \theta_i]$, which contains $\mathcal{B}_{i-1}$ and D_i.

Shirman and Abi-Ezzi [8] use bounding cones for bounding the normal fields of free-form curved patches. They present an exact method that is based on finding the minimum spanning sphere of the vectors represented as points on the unit sphere S^2, and then intersecting this sphere with S^2. The resulting spherical circle defines the bounding cone. This method gives the optimal cone, but is not time-wise optimal (at least as it is stated). The authors say it could use the rather slow algorithm of Lawson [6] for computing the minimum spanning sphere. Instead, to get a practical running time, Shirman and Abi-Ezzi used a bounding-box heuristic, which is very fast in practice but not accurate. As will become apparent later in the paper, the sphere computation could be performed in expected *linear* time, yielding an on-average optimal-time algorithm. Nonetheless, there is no need to solve this problem in three dimensions when the minimum bounding cone is known in advance to be nonreflex. That is, when the corresponding spherical points occupy at most one hemisphere. Finding the minimum spanning spherical circle of such a set of spherical points is a two-dimensional problem, as we also show below.

In the field of computational geometry, finding the minimal spanning circle (MSC) of a given set of points in the plane is considered a classical problem [1]. (The original reference goes back to [10].) The optimal solution (a minimum-radius circle) can be found in $\Theta(n)$ expected time, where n is the number of points. This can be done by representing the problem as an LP-type (linear-programming-like) problem. In fact, the minimum enclosing sphere of n points in *any* dimension can be found in expected $\Theta(n)$ time by using the same method.

In this paper we show that the optimal bounding cone of a set of n vectors can be found in a worst-case $O(n \log n)$-time algorithm by using the spherical Voronoi diagram of a set of n points. We show that when only the average-case time is of interest, the algorithm of Shirman and Abi-Ezzi [8] can be implemented to run in expected $\Theta(n)$ time by computing the minimum bounding sphere of a set of n (spherical) points. Finally, we show that if the vectors are known to span a nonreflex cone, the latter algorithm can be simplified so as to require the computation of the minimum spanning (spherical) circle of n (spherical) points. All these solutions are exact (unlike those of [4,7]) and are far simpler than the solution offered in [8].

2. Algorithm roadmap

Our goal is then to compute the optimum (minimum angle) cone bounding a given set of vectors. Consider first the situation in which the algorithm is expected to run efficiently in the *worst* case. To this aim we apply a Voronoi-diagram approach. Since this technique is quite standard, we provide here only a high-level description of it. We represent all the vectors as points on the unit sphere S^2. In $O(n \log n)$ time we compute the spherical Voronoi diagram of these n points. Then, we compute in $\Theta(n)$ time the maximum empty spherical circle. (That is, the spherical circle whose interior does not contain any of the points.) This can be done in $\Theta(n)$ time by considering sequentially all the vertices of the diagram, whose number is $\Theta(n)$, and picking up the vertex that is the center of the largest empty circle. Irrespective of whether the original vectors span a cone with opening angle which is less or greater than $\pi/2$ (that is, whether or not the minimum spherical disk containing all points is smaller or larger than a hemisphere), the complement of this disk is the intersection of the sought cone and the sphere. Therefore, we can compute in $O(n \log n)$ time the minimum bounding cone of any set of n vectors.

Consider next the situation in which the algorithm is expected to run efficiently in the *average* case. We distinguish here between two cases:

(1) The input vectors are known in advance to span a nonreflex cone. In this case all the corresponding

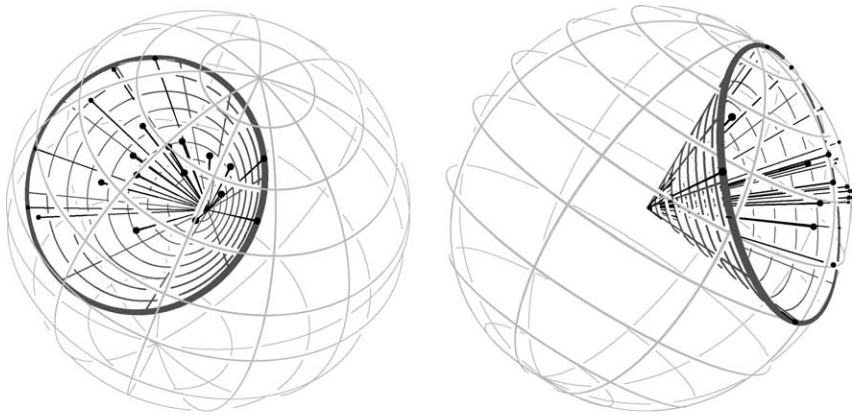

Fig. 1. Two views of an instance of the problem and of its solution.

spherical points lie in a hemisphere, and we apply the algorithm described in Section 3. In a nutshell, this is a randomized algorithm for computing the minimum spherical circle that contains a set of spherical points. The expected running time of the algorithm is linear in the number of points. This computed circle is the intersection of the bounding cone and the sphere, thus it defines uniquely the sought cone.

(2) Otherwise, the input vectors span a reflex cone (or it is unknown a priori whether the minimum cone is reflex or not). In this case the corresponding spherical points may lie in more than a hemisphere, and we apply our implementation of the algorithm of Shirman and Abi-Ezzi [8] described in Section 4. Our implementation replaces the computation of the minimum bounding sphere of a set of points by a randomized algorithm which is essentially identical to the first algorithm but in one higher dimension. The intersection of the latter sphere with the sphere containing the points is the circle that defines uniquely, as before, the sought cone.

Both versions of the randomized algorithm run in expected $\Theta(n)$ time. On one hand, the three-dimensional version (that is used to improve on [8]) solves all instances of the problem. On the other hand, the two-dimensional version (that is restricted to nonreflex cones) runs in one lower dimension, and is thus easier to implement and faster in practice. Therefore, when the vectors are known to span a small (nonreflex) cone, it is worth applying the first version of the randomized algorithm.

3. A nonreflex cone

For ease of exposition, we first describe the algorithm for computing the minimum bounding cone of a set of vectors which are known *a priori* to span a nonreflex cone. As before, we represent all the vectors as points on the unit sphere S^2. (According to the assumption about the vectors, the points span at most a hemisphere of S^2.) Our goal is thus to find the minimum-radius spherical circle c (embedded in S^2) that encloses all the points. Then, the cone whose apex is the origin and whose intersection with S^2 is c is the minimum-angle cone that contains all the original vectors. See Fig. 1 for an illustration of this method.

3.1. The algorithm

In order to find the minimum circle enclosing a set of points on a sphere, we use the algorithm of [1, §4.7, pp. 85–88] for the planar case, with only a few minor modifications.

In the upper level of the algorithm, we iteratively compute the minimum enclosing circle of the first i points, where i goes from 3 to n (see function MinSphericalCircle in Fig. 2). The minimum enclosing circle of the first two points is the circle for which the

ALGORITHM MinSphericalCircle (S)
`Input:` A set S of n points on S^2.
`Output:` A minimum-radius spherical circle on S^2 that fully contains S.
begin
Compute a random permutation $p_1, \ldots, p_n$ of the points in S.
Let c_2 be the smallest spherical circle enclosing $\{p_1, p_2\}$.
for $i = 3, \ldots, n$ **do**
if $p_i \in c_{i-1}$
then $c_i := c_{i-1}$;
else $c_i :=$ MinSphericalCircleWithPoint ($\{p_1, \ldots, p_{i-1}\}$, p_i).
end if
end for
return c_n.
end MinSphericalCircle

FUNCTION MinSphericalCircleWithPoint (S, q)
`Input:` A set S of n points on S^2, and a point q s.t. there exists an enclosing spherical circle of S that passes through q.
`Output:` The minimum-radius spherical circle on S^2 that fully contains S and that passes through q.
begin
Compute a random permutation $p_1, \ldots, p_n$ of the points in S.
Let c_1 be the smallest spherical circle enclosing $\{p_1, q\}$.
for $i = 2, \ldots, n$ **do**
if $p_i \in c_{i-1}$
then $c_i := c_{i-1}$;
else $c_i :=$ MinSphericalCircleWithTwoPoints ($\{p_1, \ldots, p_{i-1}\}$, p_i, q).
end if
end for
return c_n.
end MinSphericalCircleWithPoint

FUNCTION MinSphericalCircleWithTwoPoints (S, q_1, q_2)
`Input:` A set S of n points on S^2, and two points q_1, q_2 s.t. there exists an enclosing spherical circle of S that passes through q_1 and q_2.
`Output:` The minimum-radius spherical circle on S^2 that fully contains S and that passes through q_1 and q_2.
begin
Compute a random permutation $p_1, \ldots, p_n$ of the points in S.
Let c_0 be the smallest spherical circle enclosing $\{q_1, q_2\}$.
for $i = 1, \ldots, n$ **do**
if $p_i \in c_{i-1}$
then $c_i := c_{i-1}$;
else $c_i :=$ the circle passing through q_1, q_2, and p_i.
end if
end for
return c_n.
end MinSphericalCircleWithTwoPoints

Fig. 2. Computing the minimum-radius spherical circle containing a set of spherical points (following closely the algorithm of [1] for the planar case).

two points are the endpoints of its spherical diameter. In the ith step, we check whether the ith point lies inside, on, or outside c_{i-1}, the minimum spherical enclosing circle of the first $i-1$ points. In the first two cases $c_i = c_{i-1}$. Only in the third case we need to recompute c_i, but now it is guaranteed that the ith point is found on c_i. (The proof of this fact is identical to that of the planar case; see [1, §4.7, p. 86].)

Accordingly, we now invoke a secondary function that performs the same task (namely, finding the minimum enclosing circle of a set of points), with the only restriction that one specific point q is known to be on the sought-after circle. (See function MinSphericalCircleWithPoint in Fig. 2.) Again, we add one point at a time, and check whether the newly-added point is inside or on the previously computed circle. If it is in neither place, we need to recompute the circle, but this time we are guaranteed that both q and the newly-added point are on the new circle.

Finally we invoke a tertiary function that performs the same task, this time with the restriction that two specific points q_1, q_2 are known to be on the sought-after circle. (See Function MinSphericalCircleWithTwoPoints in Fig. 2.) Again, we add one point at a time, and check whether the newly-added point is inside or on the previously computed circle. If not, we need to recompute the circle, this time with the knowledge that all of q_1, q_2, and the newly-added point are found on the new circle. We use simple geometry to find the unique circle that fulfills this requirement.

The entire algorithm is shown in Fig. 2. The algorithm was broken into three levels only for clarity of exposition. In fact, the three functions can be implemented as a single function, which also receives an input parameter that specifies how many points are fixed on the spanning circle at the current level of calling to the function.

3.2. Correctness

The correctness of the algorithm is shown in the same way that the correctness of the algorithm for the planar case [1, §4.7, p. 86] is shown. Note that the fact that the points occupy no more than a hemisphere is crucial for the proof. Otherwise, the main inductive claim breaks down and the algorithm is no longer valid.

Specifically, the planar algorithm relies on the fact that if p_i, the point handled in the ith iteration, is outside c_{i-1}, the minimum spanning circle of the first $i-1$ points, then p_i must be on c_i. In the spherical case we should use the terms "inside" and "outside" a circle with care. We follow the straightforward definition that the inside of a spherical circle is the disk that is the smaller out of the two portions of the sphere delimited by the circle. Then, as long as the points occupy less than a hemisphere, the inductive step holds using the same argument as in the planar case. It is rather easy to construct a set S of spherical points that occupy more than a hemisphere and an ordering of S that will result in a nonoptimum bounding cone. The reason for the wrong result is that for such a set S the solution is actually defined by the maximum empty circle, for which the inductive claim (that the new circle contains the current point) is false.[1]

3.3. Complexity analysis

The main algorithm, MinSphericalCircle, performs n iterations, in each of which it either decides in constant time that the minimum-radius circle (so far) does not have to change, or calls the function MinSphericalCircleWithPoint. In the worst case, the main algorithm can call the latter function $\Theta(n)$ times, in case all of the third through the nth points require an update of the enclosing circle. Similarly, the function MinSphericalCircleWithPoint performs $k = \mathrm{O}(n)$ iterations (where k is the size of the point set it receives as a parameter). In each iteration it either decides in constant time not to update the enclosing circle or to call the function MinSphericalCircleWithTwoPoints. As in the main procedure, calling the latter function can occur in $\Theta(n)$ iterations. The running time of the function MinSphericalCircleWithTwoPoints is

[1] Here is a simple two-dimensional example that demonstrates this fact. In two dimensions we seek a minimum-length circular arc that fully contains a set of points on a circle. For clarity we represent such points as the hours on a clock. Consider the sequence $(6, 9, 12, 1{:}30, 3, 4{:}30)$. Processing the points in this order results in the clockwise arc $(6, 4{:}30)$ (whose length is $7\pi/4$), while the optimum is the clockwise arc $(9, 6)$ (whose length is $3\pi/2$). The error occurs while processing the last point 4:30; it is outside the spanning arc at that stage $(6, 3)$, yet it is wrong to look for a new spanning arc with 4:30 as an endpoint. It is quite easy to generalize this example to three dimensions.

$\Theta(k) = \mathrm{O}(n)$, where k is the size of the point set it receives as a parameter. Overall, in the worst case, the entire algorithm requires $\Theta(n^3)$ time.

The average case is much more favorable: we will now show that the expected running time of the algorithm is only $\Theta(n)$, which is optimal. We already know that the expected running time of the lowest-level function MinSphericalCircleWithTwoPoints is $\Theta(k)$, where k is the size of its first parameter (the point set). Let us then estimate the expected running time of the function MinSphericalCircleWithPoint. Assume that its first parameter is also a set of k points. Then it performs $k-1$ steps, in each of which it either spends a constant time on checks and assignments, or calls the function MinSphericalCircleWithTwoPoints. At the ith step (for $2 \leqslant i \leqslant k$) the probability of the latter event occurring is at most $2/i$. This is verified by a simple backward-analysis argument: Let c_i be the circle *after* the ith step. Discard the point p_i and run the algorithm *backward*. The circles c_i and c_{i-1} are different only if p_i was one of the three points defining c_i. One of the three points is known (q), so the probability is at most $2/i$. (Equality would hold if it was known in advance that no four points are cocircular. Otherwise the probability that $c_i \neq c_{i-1}$ is strictly less than $2/i$.) Now, the total expected running time of the function is at most $\sum_{i=2}^{k}((2/i)\mathrm{O}(i)) = \mathrm{O}(k)$.

A similar analysis holds for the expected running time of the main algorithm, MinSphericalCircle. This time the probability of calling the function MinSphericalCircleWithPoint is at most $3/n$, following a similar argument. The total expected running time of the algorithm is, thus, at most $\sum_{i=3}^{n}((3/i)\mathrm{O}(i)) = \mathrm{O}(n)$.

Obviously the running time of the algorithm is also $\Omega(n)$ (that much time is required for just reading the input). Therefore, this algorithm runs in expected optimal $\Theta(n)$ time.

4. A general cone

For a general set of vectors, for which the minimum bounding cone can be reflex, we use the algorithm of Shirman and Abi-Ezzi [8] but modify its main ingredient to use the same idea as in the previous section.

Shirman and Abi-Ezzi's algorithm also represents the vectors as points on the unit sphere S^2, and proceeds by computing S', the minimum bounding sphere of these spherical points. This avoids the problematic step of computing the minimum spherical circle that contains a set of points. It is always true that the intersection of S^2 and S' is the circle that defines the minimum bounding cone of the original set of vectors.

Shirman and Abi-Ezzi suggested an inefficient method for computing S'. However, using the algorithm described in Section 3 (but in three dimensions), one can perform this step in time which is still expected to be linear in the number of points. Consider again the algorithm in Fig. 2. It is easy to modify the algorithm to have four instead of three levels. The running-time analysis remains unaltered. It can be shown, using identical arguments, that all of the four levels still run in $\mathrm{O}(n)$ time on average, where n is the number of points.

Since both versions of the randomized algorithm run in time which is asymptotically linear in the number of vectors, it is the user's choice (or trade-off) to apply the version that fits the *a priori* knowledge about the vectors.

5. Experimental results

We have implemented the simple algorithm described in this paper (see Fig. 2) for computing the minimum-angle cone that encloses a set of vectors. The software was implemented in IRIT [5]. It consists of about 100 lines of code. The running times for all our experiments (up to 10,000 vectors) were negligible (below one second) on a modern Windows-based system and are thus omitted here.

To assess the quality of our optimal-cone solution, we also implemented the simplest averaging heuristic, in which the axis of the bounding cone is set to the average of the (normalized) input vectors, and its angular span is the largest angle between the axis and any of the given vectors. Clearly, this heuristic always runs in $\Theta(n)$ time, but it is biased by large clusters of vectors.

Here are two representative examples of the performance of the optimal algorithm compared to the simple averaging heuristic. Fig. 3 shows two sets of vectors and their bounding cones. The smaller cone (shown in darker grey) is the optimum (minimum angle) enclosing cone, while the larger cone (shown in lighter grey) is the solution obtained by the heuristic method. The difference between the solutions is clearly visible in both examples.

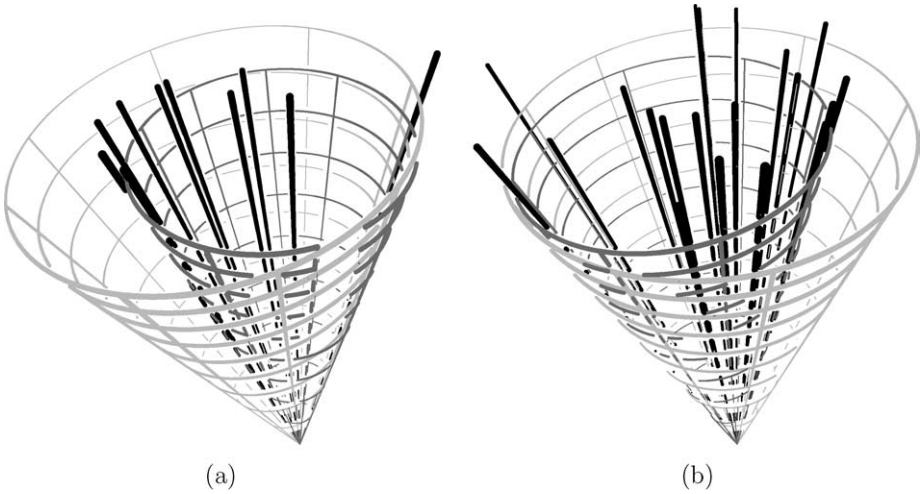

Fig. 3. Two instances of the minimum-enclosing-cone problem.

6. Conclusion

In this paper we present an optimal-time (expected) solution for the problem of computing the minimum-angle bounding cone of a set of vectors in three dimensions. The solution is inherently the same as for computing the minimum spanning circle (or sphere) of a set of points. The expected running time of the algorithm is linear in the number of input vectors. We provide two versions of a randomized algorithm, one of which is simpler but fits only sets of vectors whose minimum bounding cone is nonreflex.

For users who require that the worst-case instance of the problem be solved efficiently, we provide a completely different solution based on a spherical Voronoi diagram of a set of spherical points. This algorithm runs (for all inputs) in $O(n \log n)$ time, where n is the number of input vectors.

Acknowledgement

The authors wish to thank Matthew T. Dickerson for helpful discussions on the maximum empty circle problem.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry, Algorithms, and Applications, second ed., Springer-Verlag, Berlin, 2000.
[2] D.S. Kim, P.Y. Papalambros, T.C. Woo, Tangent, normal, and visibility cones on Bézier surfaces, Comput. Aided Geom. Design 12 (1995) 305–320.
[3] G. Elber, E. Zussman, Cone visibility decomposition of freeform surfaces, Comput. Aided Design 30 (1998) 315–320.
[4] G. Meenakshisundaram, S. Krishnan, A fast and efficient projection-based approach for surface reconstruction, Int. J. High Performance Comput. Graphics, Multimedia, Visualisation 1 (2000) 1–12.
[5] IRIT 8.0 User's Manual, The Technion—IIT, Haifa, Israel, 2000. Available at http://www.cs.technion.ac.il/~irit.
[6] C. Lawson, The smallest covering cone or sphere, SIAM Rev. 7 (1965) 415–417.
[7] T.W. Sederberg, R.J. Meyers, Loop detection in surface patch intersections, Comput. Aided Geom. Design 5 (1998) 161–171.
[8] L.A. Shirman, S.S. Abi-Ezzi, The cone of normals technique for fast processing of curved patches, Comput. Graphics Forum 12 (1993) 261–272.
[9] M. Stamminger, P. Slusallek, H.-P. Seidel, Bounded radiosity—Illumination on general surfaces and clusters, Comput. Graphics Forum 16 (1997) 309–318.
[10] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: H. Maurer (Ed.), New Results and New Trends in Computer Science, in: Lecture Notes in Computer Science, vol. 555, Springer-Verlag, Berlin, 1991, pp. 359–370.