Approximating Centroids for the Maximum Intersection of Spherical Polygons

Jong S. Ha[1] and Kwan H. Yoo[2]

[1]Woosuk University, jsha@woosuk.ac.kr
[2]Chungbuk National University, khyoo@cbucc.chungbuk.ac.kr

ABSTRACT

This paper considers the problem of investigating the spherical regions owned by the maximum number of spherical polygons. We present a practical $O(n(v+I))$ time algorithm for finding the approximating centroids for the maximum intersection of spherical polygons, where n, v, and I are, respectively, the numbers of polygons, all vertices, and intersection points. In order to elude topological errors and handle geometric degeneracies, our algorithm takes the approach of edge-based partitioning of the sphere. Furthermore, the numerical complexity is avoided since the algorithm is completely spherical.

Keywords: spherical algorithms, visibility maps, maximum intersection, computational geometry

1. INTRODUCTION

We can well establish many problems of orienting models or equipments in manufacturing [3,4,28] with the geometric technique of finding the maximum intersection of visibility maps derived from the surface normals of the models, which are represented as spherical convex polygons on the unit sphere S^2.

For investigating the intersection of spherical convex polygons, there have been two kinds of approaches: *tessellation approach* [21,23] and *partition approach* [3,15,22]. The tessellation approach is a digital-image-oriented method that subdivides S^2 into a set of small triangles, and computes the ownership of each triangle. The partition approach is a polygon-object-oriented method that partitions S^2 with the boundaries of polygons into a set of faces and computes the ownership of each face.

The tessellation approach has been used in practical applications, since it can take robust computation. This approach has trade-off between the accuracy of solutions and the efficiency of computing time. Subdivision of the sphere has a disadvantage in geometric computations for choosing a preferred solution such as finding a centroid of all solutions.

The partition approach is theoretically analyzed to be able to obtain the exact boundaries of all solutions in an efficient time complexity. Unfortunately, however, its implementation may fall into the numerical complexity as pointed out by Suh and Kang [21]. For example, infinite values can be obtained when the central projection is applied into points on S^2 for using planar algorithms. Another serious problem is the topological violation of nearby neighboring intersection points of the polygons, which is caused by the limit of numerical computation. A topological error makes the boundary representation [2,25,27] of the partition approach inconsistent.

In this paper, we present a robust and practical algorithm for computing the *approximating centroids* of maximally intersected faces, which is a new method of the partition approach. In order to achieve the robustness by eluding *topological errors* such as the violation of nearby intersection points and handling *geometric degeneracies* such a point or an edge visibility map, our algorithm updates the ownerships of edges rather than those of faces whenever S^2 is incrementally partitioned by each polygon. The partition method of our algorithm will be called *edge-based partition*, while the previous method [3,15,22] is called *face-based partition*. Our edge-based partition excludes the central projection of spherical entities into S^2 in order to avoid the *numerical complexity*.

2. PRELIMINARIES

2.1 Definitions and Notation

Two points are *visible* to each other if there is a line segment joining them without any interference between them. The *visibility map* of a surface is a set of spherical points, which represents all directions in which any point on the surface is visible from infinity. The visibility map is closely related to the *Gaussian map* that is a set of normal vectors to the surface. If the Gaussian map of a surface S_i is $G_i = \{g_1, \cdots, g_m\}$, the (*positive*) visibility map of S_i is obtained by $\bigcap_{k=1}^{m} HS(g_k)$, while the *negative* visibility map of S_i is $\bigcap_{k=1}^{m} -HS(g_k)$, where $HS(g_k)$ is the hemisphere of $\{p \| p \cdot g_k \geq 0\}$ and g_k is called its *pole*.

Let P_i and V_i be, respectively, the convex hulls of the Gaussian map and the visibility map of S_i. Then, P_i and V_i are *dual* to each other [13]. That is, a vertex v_k of P_i corresponds to an edge e_k of V_i such that e_k is on the great circle defined by $\{x \| x \cdot v_k = 0\}$, and the vertices of P_i are in the same order with the corresponding edges of V_i; and vice versa.

Given a set of Gaussian maps ☝ $= \{G_1, \cdots, G_n\}$, we denote their convex hulls with 🏳$= \{P_1, \cdots, P_n\}$. The duals of 🏳, i.e., the convex hulls of positive visibility maps of ☝, are represented with ✞ $= \{V_1, \cdots, V_n\}$. We represent the convex hulls of negative visibility maps of ☝ with $-$✞ $= \{-V_1, \cdots, -V_n\}$.

2.2 Ownership Vector and its Applications

Assume that S^2 is partitioned with ✞$\bigcup -$✞ and the ownerships of all faces are computed. We assign an *ownership vector* per each face for representing which visibility maps own (contain) the face. The *i*-th bit of the ownership vector of a face is set to $+1$ if the face is owned by V_i, while it is set to -1 if the face is owned by $-V_i$. The *i*-th bit will be set to 0, if the face is owned by neither of the two visibility maps. Let n_k^+ and n_k^- be, respectively, the numbers of $+1$ bits and -1 bits of the ownership vector of a partitioned face F_k. We will call these numbers *ownership numbers*.

Let 🏳 be the set of convex hulls of Gaussian maps derived from pocket surfaces of a designed $3D$ model, which is a set of faces obtained by the difference of the model from the convex hull of the model. Then, we can solve the geometric problems of orienting models or equipments in manufacturing by counting n_k^+ and n_k^- as the follows.

In the mould design forming the seal of the model, the maximum intersection of ✞$\bigcup -$✞ is the set of optimal directions for orienting the model to minimize the number of auxiliary devices. In other words, we have to find the faces such that n_k^+ is maximized. See the reference [4] for the details.

The optimal orientation problem on NC machining has been formulated into five geometric problems [3] on the sphere; find the densest hemispheres containing the maximum subset of 🏳, great circles separating 🏳, great circles bisecting 🏳, and great circles intersecting the minimum or the maximum subset of 🏳. Surprisingly, four problems among them are solved by computing the maximum intersection of ✞ or ✞$\bigcup -$✞. (One problem is solved by computing the minimum intersection of ✞$\bigcup -$✞.) The poles of the densest hemispheres containing the maximum subset of 🏳 are in the spherical region of the *maximum intersection* of ✞. To determine the minimum subset intersection of 🏳 is to find the spherical region of the *maximum intersection* of ✞$\bigcup -$✞. That is, we have to find the faces such that $n_k^+ + n_k^-$ is maximized. Determining the separators of 🏳 is to find the faces such that $n_k^+ + n_k^- = n$, where n is the number of spherical polygons in 🏳. Since $n_k^+ + n_k^- \leq n$, the separators can be determined with the minimum subset intersection of 🏳. The bisectors of 🏳 are the special separators such that $\left| n_k^+ + n_k^- \right| \leq 1$. See the reference [3] for the details.

3. MAXIMUM INTERSECTION OF VISIBILITY MAPS

For solving orientation problems with geometric techniques on S^2, first we have to construct visibility maps ✞ and $-$✞ from a given Gaussian map ☝. Next S^2 is partitioned with ✞ and $-$✞, and the ownership of each partitioned face is computed for finding the faces with the maximum ownership number.

A robust spherical algorithm for constructing ✞ and $-$✞ from ☝ is described in Appendix. In this section, focusing on the methods of partitioning S^2 with ✞$\bigcup -$✞, we briefly review the previous approach partitioning S^2 for finding the faces with the maximum ownership number. In order to make the partition approach practical, we present a robust algorithm that finds approximating centroids for the maximum intersection with efficient computations.

3.1 Limitations of Face-based Partition

In the previous partition approach [3,15,22] which we call the face-based partition, they directly manipulates the ownership vectors of partitioned faces. S^2 is incrementally partitioned into a set of faces by each polygon of ✞∪−✞ , and then the ownership of each face is updated at each partitioning.

This approach is a fundamental solution to be analyzed to get the boundary of all solutions in an efficient time $O(nv \log n)$, where n and v are, respectively, the numbers of polygons and all vertices. Unfortunately, however, its implementation has not been reported yet as far as the authors know, since it has serious problems from the practical viewpoint of implementation.

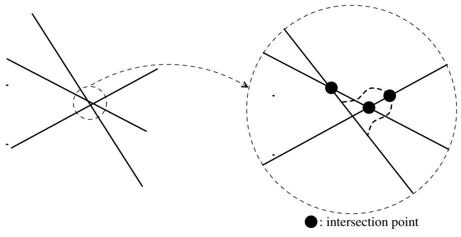

Fig. 1. Topological violation of nearby intersection points

A face can be described with ordered edges or ordered vertices that form the face and their relations such as topology. The typical data structure for representing the face is the boundary representation [2,25,27]. These kinds of data structures representing the face become inconsistent to be useless under the topological violation of nearby neighboring intersection points as illustrated in Fig. 1. The treatment of this ill-condition remains an active topic of research [1,10,12]; however, its implementation needs a heavy computation burden.

There is another limitation of the face-based partition. It stems from the degeneracy cases of the visibility maps such as a point or an edge. Although these cases can be found very often in CAD/CAM models including orthogonal parts, they cannot be dealt with by this approach.

3.2 Edge-based Partition

We present another partition approach called edge-based partition, which manipulates the ownership vector of each partitioned edge rather than those of partitioned faces. It does not employ the data structures for representing the face, but uses a simpler data structure such as circular lists for representing only edges in the face boundary. This approach is freed from the topological violation and also it can treat the degeneracy cases of the visibility maps, since the edges are handled discretely without constructing the face boundary; in other words, their topological relations are not represented. The face boundary representing the whole region within the face can not be obtained directly. Hence, we also develop a method to get a reasonable point within the face by exploiting the discrete edges.

3.2.1 Updating the ownerships of partitioned edges

The most important procedure of edge-based partition is to partition edges and compute ownerships of the partitioned edges. Whenever a pair of polygons in ✞∪−✞ are intersected incrementally one by one, the intersected edges are partitioned and ownership vectors of edges within other polygon are updated. Since the order of incremental intersections is independent of the final result, the order of intersections can be randomized.

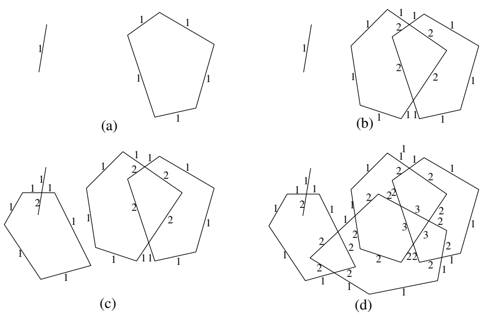

Fig. 2. Incremental intersection and the ownerships of edges

An example of edge-based partition including the degeneracy case of an edge is illustrated in Fig. 2. Conceptually, it starts from an empty space and polygons come out one after another. Whenever a polygon comes out, the polygon is intersected respectively with every polygon in the space.

There are several linear algorithms [18,20,24] for intersecting a pair of convex polygons in the plane. For adapting those planar algorithms for the spherical case, great circles are often mapped into lines in a plane by the central projection. This projection technique is an elegant technique from theoretical point of view, but it results in the serious problem of unbounded numerical values from the view of implementation.

In this paper we employ the algorithm of Ha and Shin [9], inspired by the method in O'Rourke et al [18]. The algorithm is based on an edge-advancing rule that proceeds edge by edge around the polygons for computing the intersection. The main rule is to advance an edge if it chases another edge, but this rule becomes ambiguous in the spherical space since arcs on the sphere always chase each other. The spherical algorithm is based on the *facing* relation between edges and their relative positions. See the reference [9] for the detailed formulation and algorithm of the edge advancing mechanism on the sphere.

3.2.2 Finding centroids for the maximum intersection

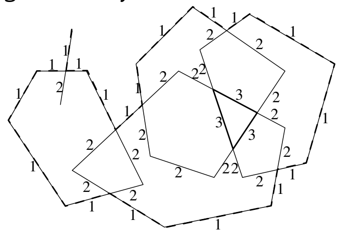

Fig. 3. Counting ownership numbers of partitioned edges

As illustrated in Fig. 3., assume that S^2 is partitioned by $2n$ number of polygons in ✞⋃−✞ and each partitioned edge keeps the number of polygons owning it. A spherical entity such as an edge or a face with k ownership number is said to be *k-intersected*. We denote MIN and MAX, respectively, the minimum and maximum ownership numbers in the partition. The interior and exterior of an edge are defined from the view of the polygon containing the edge.

As shown in Tab. 1., we can observe that *k*-intersected faces are in the interior of *k*-intersected edges or the exterior of *(k+1)*-intersected edges. Theoretically, we can construct the boundary of a *k*-intersected face by gathering edges and then sorting them. However, the boundary obtained by the sorting still has inconsistency under the topological error of intersection points. In this paper, we develop a robust method for finding the approximating centroids of MAX-intersected faces, that is, the maximum intersection.

Case	*Edges forming k-intersected faces*
k=MIN	- the exterior of *k*-intersected edges
MIN<*k* *k* <MAX	- the interior of *k*-intersected edges - the interior of *k*-intersected edges and the exterior of *(k+1)*-intersected edges - the exterior of *(k+1)*-intersected edges
k=MAX	- the interior of *k*- intersected edge

Tab. 1. Edges forming *k*-intersected faces

MAX-intersected faces have nice properties for an efficient computation as the following:

- MAX-intersected faces are *always convex*.
- MAX-intersected faces have *no hole*.
- The edges forming one particular MAX-intersected face have *the same ownership vector*. (In general, the MAX-intersected edges may have ownership vectors differently to each other.)

From this observation, our algorithm first determines the maximum ownership number by examining all edges. All edges are examined again for gathering edges with the determined ownership number, and grouping the edges according to the ownership vector. If we try to construct the exact boundaries of all solution faces by ordering the grouped edges, the topological error can not be avoided. By exploiting their convexity and applying geometric techniques, we find appropriate points within the solution faces, which are more useful in practice.

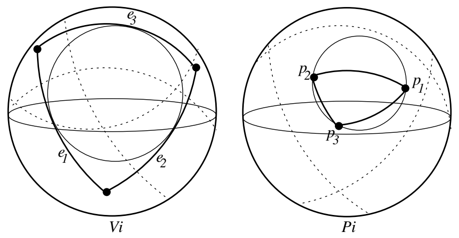

Fig. 4. Inscribing and circumscribing circles of dual polygons

The center of the circle inscribing a convex polygon, which is the largest circle within the polygon, is a good approximation to the centroid of the polygon. The center of the inscribing circle coincides with the center of the circle circumscribing its dual polygon [13], as illustrated in Fig. 3. Finding the inscribing circle is complicated since there are many candidate edges in the polygon. However, the circumscribing circle can be found in linear time by finding the smallest enclosing sphere [16] of the vertices of the polygon.

For each edge e_i of a MAX-intersected face, we find a vertex v_i such that e_i is on the great circle defined by $\{x \| x \cdot v_i = 0\}$. The vertex v_i is the pole of the great circle passing the edges e_i. The set $\{v_i\}$ is just the vertices of the dual polygon of the MAX-intersected face. We can treat the set $\{e_i\}$ discretely, since the smallest enclosing sphere algorithm does not use the order of $\{v_i\}$ in the boundary of the dual polygon. In other words, we get the inscribing circle of the MAX-intersected face without constructing the face boundary.

Finally, we describe the algorithm of edge-based partition for finding centroids of the maximum intersection as follows.

```
Procedure MaximumIntersection ( ✞ ∪ −✞ )
Input: A set of spherical convex visibility maps $\{V_1,\cdots,V_n,V_{n+1},\cdots,V_{2n}\}$ (We replaced $-V_i$ with $V_{n+i}$ for a simpler description.)
Output: Approximating centroids of maximally intersected faces
  Initialize the ownership vectors of all edges ∈ ✞ ∪ −✞
  for $i=1$ to $2n-1$ do
    for $j=i+1$ to $2n$ do
      Compute the intersection points between $V_i$ and $V_j$ [9].
      Partition the edges $\in V_i \cup V_j$ .
      if ( $i \le n$ ) then
        Set the i-th bit of the positive ownership vector of each edge $\in V_j$ , if $V_i$ owns it.
      else
        Set the (i-n)-th bit of the negative ownership vector of each edge $\in V_j$ , if $V_i$ owns it.
      endif
      Update the ownership vector of each edge $\in V_i$ symmetrically.
    endfor
  endfor
  Find the maximum ownership number among all edges ∈ ✞ ∪ −✞ .
  Gather edges with the maximum ownership number, and group them into $\{E_1,\cdots,E_l\}$ according to their ownership vectors.
  for $i=1$ to $l$ do
    if (the number of edges of $E_i < 3$ ) then
      Compute a centroid directly.
    else
      Compute the poles of great circles passing each edge $\in E_i$ .
      Find the smallest sphere enclosing those poles [16].
    endif
  endfor
endProcedure MaximumIntersection
```

3.3 Analysis of Time Complexity

The time complexity of the algorithm **MaximumIntersection** is dominated by the reiterated loop for intersecting all pairs of visibility maps in ✞ ∪ −✞ . We can intersect a pair of V_i and V_j in linear time $O(v_i+v_j)$ [9], where v_i and v_j denote the numbers of vertices of V_i and V_j , respectively. However, we have to carefully analyze the time complexity of our algorithm, since the number of edges of a polygon increases whenever an edge is partitioned.

The total number of scanning all vertices in the reiterated loop is $\sum_{i=1}^{2n-1}\sum_{j=i+1}^{2n}(v_i+v_j)$. Each intersection point generated in the loop is scanned at most $(2n-1)$ times. Let I be the number of intersection points in ✞ ∪ −✞ . Then, the time complexity is described as $\sum_{i=1}^{2n-1}\sum_{j=i+1}^{2n}(v_i+v_j)+(2n-1)\times I = O(n(v+I))$, where $v=\sum_{i=1}^{2n}v_i$ and $I=O(n^2)$.

Finally, we get the following result.

Theorem 1. *The approximating centroids of the maximally intersected faces of spherical convex polygons can be found in $O(n(v+I))$ time, where n , v , and I are, respectively, the numbers of polygons, all vertices, and intersection points.*

3.3 Experimental results

Our algorithms for constructing visibility maps and finding the centroids of the maximum intersection have been implemented as a class of C++ programming language in the environments of PC Windows 98. We used the public codes of Hohmeyer [11], and Erickson and Honda [8] for the two parts of the linear programming and the smallest enclosing sphere. The execution results of our implementation were visualized in the IRIT system [7] as shown in Fig. 5. and Fig. 6.

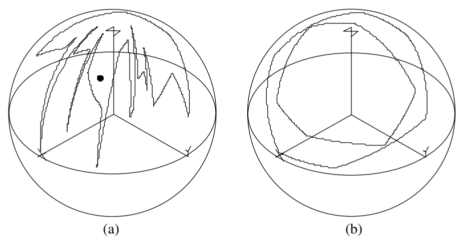

Fig 5. Construction of visibility maps

Fig. 5. illustrates a visibility map that was constructed by our implementation for a given Gaussian map G_i : (a) a simple polygon that was generated by angular sorting around an extreme point, and (b) the convex hull P_i of the polygon versus its dual polygon V_i .

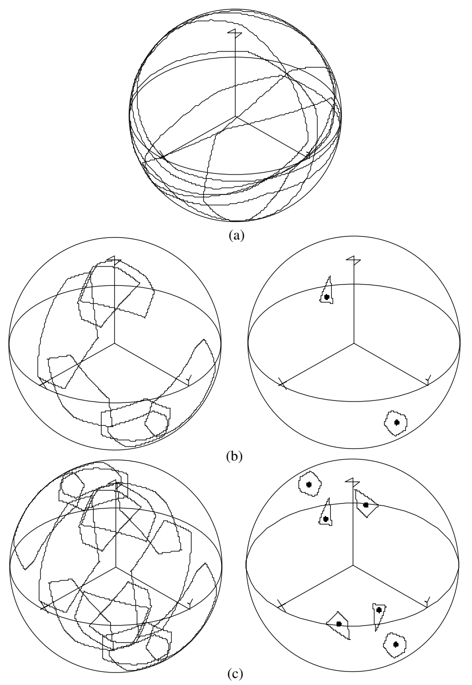
Fig. 6. Finding the centroids of solution faces

Fig. 6. illustrates the maximum intersection that was computed by our implementation: (a) the set ⚐ of convex hulls for a given set ☝ of Gaussian maps, (b) the centroids of the maximum intersection of the set ✞ of visibility maps, and (c) the centroids of the maximum intersection of ✞ $\cup -$ ✞ .

We tested our implementation with randomly generated data for comparing the time complexity $O(n(v+I))$ with the running time in practical cases. The size of I is $O(n^2)$ in the worst case, but the experimental running time was almost proportional to $O(nv)$. This is because $I<v$ in our random data that is colser to practical cases.

4. CONCLUSIONS

We have presented a robust and practical algorithm for partitioning a sphere with spherical convex polygons and finding the maximum intersection of the polygons. To avoid topological errors and numerical complexity, we developed the approach of edge-based partition that treats edges discretely and runs directly on the spherical space. It also can handle geometric degeneracies such as point or edge visibility maps that often appear in CAD/CAM models.

Our algorithm for finding the approximating centroids of all maximally intersected faces has the $O(n(v+I))$ time complexity, while the algorithm of Chen *et al.* [3] for finding the complete boundary of the faces takes $O(nv\log n)$ time, where n , v , and I are, respectively, the numbers of polygons, all vertices, and intersection points. Note that the approximating centroids are preferred to the boundary itself for the maximum intersection in many manufacturing optimizations. We also suggest that our algorithm runs more efficiently by assuming that $I<v$ in practical cases.

5. REFERENCES

[1] Barker, S.-M., Towards a topology for computational geometry, *Computer-Aided Design*, Vol. 27, No. 4, 1995, pp 311-318.

[2] Baumgart, B.-G., A polyhedron representation for computer vision, In *National Computer Conference*, Anaheim, CA, IFIPS, 1975, pp 589-596.

[3] Chen, L.-L., Chou, S.-Y. and Woo, T.-C., Separating and intersecting spherical polygons: computing machinability on three- four- and five-axis numerically controlled machines, *ACM Tr. on Graphics*, Vol. 12, No. 14, 1993, pp 305-326.

[4] Chen, L.-L., Chou, S.-Y. and Woo, T.-C., Paring directions for mould and die design, *Computer-Aided Design*, 25(12), 1993, pp 762-768.

[5] Chen, L.-L. and Woo, T.-C., Computational geometry on the sphere with application to automated machining, *Tr. ASME, J. Mechanical Design*, Vol. 114, 1992, pp 288-295.

[6] Dyer, M.-E., Linear time algorithms for two- and three-variable linear programs, *SIAM J. Computing*, Vol. 13, No. 1, 1984, pp 31-45.

[7] Elber, G., *IRIT User's Manual*, 1996.

[8] Erickson, J. and Honda, H., The implementation of the smallest enclosing sphere, *Source Code*.

[9] Ha, J.-S. and Shin, S.-Y., Edge advancing rules for intersecting spherical convex polygons, *International Journal of Computational Geometry and Applications*, Vol. 12, No. 3, 2002, pp 207-216.

[10] Hoffmann, C.-M., The problems of accuracy and robustness in geometric computation, *Computer*, Vol. 22, No. 3, 1989, pp 31-41.

[11] Hohmeyer, M.-E., The implementation of linear programming, *Source Code*.

[12] Hu, C.-Y., Patikalakis, N.-M. and Ye, X., Robust interval solid modeling part I: representations, *Computer-Aided Design*, Vol. 28, No. 10, 1996, pp 807-817.

[13] Gan, J.-G., Woo, T.-C. and Tang, K., Spherical maps: their construction, properties, and approximation, *ASME J. Mechanical Design*, Vol. 116, 1994, pp 357-363.

[14]Graham, R.-L., An efficient algorithm for determining the convex hull of finite planar set,

Information Processing Letters, Vol. 1, 1972, pp 132-133.

[15] Gupta, P. *et al.*, Efficient geometric algorithms for workpiece orientation in 4- and 5-axis NC machining, *Computer-Aided Design*, Vol. 28, No. 8, 1996, pp 577-587.

[16] Megiddo, N., Linear-time algorithms for linear programming in and related problems, In *Proc. 23rd Annual IEEE Sym. Found. Comput. Sci.*, 1982, pp 329-338.

[17] Megiddo, N., Linear programming in linear time when the dimension is fixed, *J. ACM*, Vol. 31, No. 1, 1984, pp 114-127.

[18] O'Rourke, J., Chien, C.-B., Olson, T. and Naddor, D., A new linear algorithm for intersecting convex polygons, *Computer Graphics and Image Processing*, Vol. 19, 1982, pp 384-391.

[19] Seidel, R., Small-dimensional linear programming and convex hulls made easy, *Discrete and Computational Geometry*, 1991, pp 423-434.

[20] Shamos, M.-I. and Hoey, D., Geometric intersection problems, *Seventeenth Annual IEEE Symposium on Foundation of Computer Science*, 1976, pp 208-215.

[21] Suh, S.-H. and Kang, J.-K., Process planning for multi-axis NC machining of free surfaces, *Int. J. Prod. Res.*, Vol. 33, No. 10, 1995, pp 2723-2738.

[22]Tang, K., Woo, T. and Gan, J., Maximum intersection of spherical polygons and workpiece orientation for 4- and 5-axis machining, *ASME J. Mechanical Design*, Vol. 114, 1992, pp 477-485.

[23]Tangelder, J.-W.-H., Automated fabrication of shape models of free-form objects with a sculpturing robot, Ph. D. Thesis, Industrial Design Engineering of Delft Univ. of Technology, 1998.

[24]Toussaint, G.-T., A simple linear algorithm for intersecting convex polygons, *The Visual Computer*, Vol. 1, 1985, pp 118-123.

[25]Weiler, K., Edge-based data structures for solid modeling in curved-surface environments, *IEEE Computer Graphics and Application*, Vol. 5, No. 1, 1985, pp 21-40.

[26]Welzl, E., Smallest enclosing disks (balls and ellipsoids), *New Results and New Trends in Computer Science, Springer Lecture Notes in Computer Science* ,Vol. 555, 1991, pp 359-370.

[27]Woo, T.-C., A combinatorial analysis of boundary data structure schema, *IEEE Comput. Graph. Appl.*, Vol. 5, No. 3, 1985, pp 19-27.

[28]Woo, T.-C., Visibility maps and spherical algorithms, *Computer Aided Design*, Vol. 26, No. 1, 1994, pp 6-16.

APPENDIX. ROBUST COMPUTATION OF VISIBILITY MAPS

A spherical algorithm using the *bounded values* on S^2 is described for constructing the visibility maps without the central projection causing the nemerical complexity. Prior to computing the visibility map V_i of a given surface S_i, we have to construct the convex hull P_i of its Gaussian map G_i. We test the hemisphericity of G_i before constructing P_i since P_i exists if and only if G_i is hemispherical. After constructing P_i, we test degenerate cases such as a point or an edge. In ordinary cases, V_i is obtained by simple calculation with P_i.

A1. Hemisphericity

G_i is hemispherical if and only if $\bigcap_{k=1}^{m} HS(g_k) \neq \varnothing$. This intersection of hemispheres is represented by a homogeneous system of linear inequalities: $x_k x + y_k y + z_k z \geq 0$ for all $g_k(x_k, y_k, z_k) \in G_i$. If we apply the linear programming (LP) to this system, it may result in a trivial solution, since the LP algorithms [6,16,17] always give an extreme point of the feasible region.

Using the central projection of G_i, the hemisphericity checking has been transformed into the 2D linear separability between two sets of points in a plane [5]. This transformation is very elegant in the geometrical sense, but some points may be projected into infinitely distant place. By exploiting the central projection from another viewpoint, we develop two LP systems that can be robustly implemented. The great circle bounding $HS(g_k)$ is projected into two planes $E^+ : z = 1$ and $E^- : z = -1$, while the point g_k is projected into one plane E^+ in the previous transformation. The part of a great circle in the half-space $z > 0$ corresponds to a line in E^+, and the other part of the great circle corresponds to a line in E^-. Then, the inequality $x_k x + y_k y + z_k z \geq 0$ is represented as $x_k x + y_k y + z_k \geq 0$ in E^+ and $x_k x + y_k y - z_k \geq 0$ in E^-, respectively. Hence, we can get a non-trivial solution by applying LP into the two systems of linear inequalities as:

- Linear inequalities 1:

  $x_k x + y_k y + z_k \geq 0$ for all g_k in E^+
- Linear inequalities 2:

  $x_k x + y_k y - z_k \geq 0$ for all g_k in E^-,

The hemisphericity checking has only two degenerate cases: there are two anti-podal points g_k and g_l such that $g_k = -g_l$, or there are three points in the general position of a great circle gc, that is, the three points are on gc but not on any hemi-circle of gc. In the former case, the visibility map is composed of only one edge. The latter case has only one hemisphere that contains G_i, i.e., the surface is visible from only one direction. Hence, we do not construct P_i, but directly compute V_i in these cases.

In order to find the degeneracy, we exploit the smallest sphere enclosing G_i. Let (x^c, y^c, z^c) denote the center of the smallest sphere. When $(x^c, y^c, z^c) \neq (0,0,0)$, which means that the smallest enclosing sphere s^c is strictly smaller than S^2, the convex hull of G_i can be constructed. However, when $(x^c, y^c, z^c) = (0,0,0)$, which means that s^c is S^2 itself, the convex hull of G_i is not defined. Let gc^c be the great circle bounding G_i; then the two degenerate cases can be determined by testing whether or not there exists any hemi-circle of gc^c containing all points of G_i lying on gc^c. The problem of identifying these subcases is exactly the $2D$ version of the smallest enclosing sphere.

The linear-time algorithms for solving the LP in the fixed dimension have been presented by Megiddo [17] and Dyer [6]. Megiddo [14] has shown that the smallest enclosing sphere can be also obtained by extending the LP algorithm. Thereafter, Seidel [17] and Welzl [26], respectively, reported simpler randomized algorithms for each of them.

A2. Spherical convex hull

Once we know the Gaussian map G_i is hemispherical, we can construct the spherical convex hull P_i. For avoiding numerical complexity, we adapt the planar algorithm of Graham [14] directly to the spherical space, which is mainly composed of 3 steps; find an extreme point from a set of points, sort the set of points by angles around the extreme point, and scan the set of points in the sorted order.

In order to find an extreme point from G_i, we exploit the solution (x^c, y^c, z^c) of the smallest sphere enclosing G_i, which has been computed in the hemisphericity checking. Let (a,b,c) be the farthest point from (x^c, y^c, z^c) among all points in G_i; that is, (a,b,c) is an extreme point.

The angular sort dominating the overall time complexity $O(m \log m)$ needs the most careful computation. For robust and efficient computation of the angular sort, we use the orthogonal projection of G_i into a plane $PL(a,b,c)$ that is tangential to S^2 at the point (a,b,c). If G_i is orthogonally projected into $PL(a,b,c)$, its order of angular sort does not change. Hence, this orthogonal projection makes it possible to use the $2D$ angular sort without any numerical difficulty.

The $2D$ coordinate values of G_i projected into $PL(a,b,c)$ are computed by rotating G_i such that the point (a,b,c) corresponds to $(0,0,0)$ on Z-axis and then taking the values of x and y coordinates. Hence, the compound transformation matrix is obtained as:

$$R_x \cdot R_y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & -\frac{b}{d} & -\frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d & 0 & a & 0 \\ -\frac{ab}{d} & \frac{d}{c} & b & 0 \\ -\frac{ac}{d} & -\frac{b}{d} & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ where } d = \sqrt{b^2 + c^2}.$$

In order to compute the angular sorting robustly, the slope of a line passing through two points (x_1, y_1) and (x_2, y_2) is often replaced by another measure $\frac{dy}{|dx| + |dy|}$, where $dx = x_2 - x_1$ and $dy = y_2 - y_1$. Hence, the value used in the angular sorting for a point $g_k(x_k, y_k, z_k)$ is finally obtained as:

$$\frac{cy_k - bz_k}{\left|(b^2 + c^2)x_k - aby_k - acz_k\right| + \left|cy_k - bz_k\right|}$$

After constructing P_i, we have to check two degenerate cases of only one or two vertices. In these cases, V_i is bounded only by a great circle or two hemi-circles, which means that a boundary edge includes anti-podal points. Hence, we partition the boundary edges into several edges to avoid the degeneracy. In ordinary cases, we can compute V_i simply using the duality [13] with P_i; a vertex v_k of V_i is determined by the cross product of two end vertices p_{i-1} and p_i of the dual edge e_k of P_i.