Minimum Distance Queries For Polygonal And Parametric Models

David E. Johnson and Elaine Cohen

UUCS-97-003

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

February 26, 1997

Abstract

Calculation of the minimum distance to a geometric object or the minimum separation between objects is a fundamental problem that has application in a variety of arenas. Minimum separation queries for sculptured surfaces are believed particularly difficult, yet are increasingly important as modeling problems grow larger and interactivity demands grow more stringent. We present a set of algorithms based on easy to compute bounds that allows efficient solution of minimum separation queries for many types of surface representations. These algorithms have simple and robust implementations and have average case performance matching the most efficient theoretical algorithms. These algorithms are tested in an interactive application designed to be the front-end of a haptic display system for virtual prototyping of assemblies.

1.0 INTRODUCTION

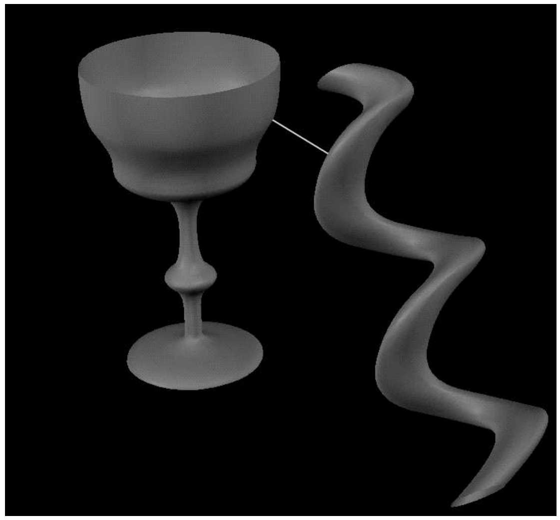

FIGURE 1. Finding the minimum distance between two parametric surfaces.

Calculation of the minimum distance to a geometric object or the minimum separation between objects is a fundamental query for a model representation. Current solutions range from using hierarchies for polytopes to using numerical methods for sculptured surfaces, and tend to be efficient only in the special case of collision detection, where the minimum separation is zero.

We introduce a paradigm for minimal distance calculations that applies well to both polygonal and parametric model representations (Figure 1). The resulting methods are robust, scale well with problem size, show time critical behavior, and are interactive both for large polygonal models and for sculptured surfaces.

This work is motivated by a virtual assembly prototyping project. Modern CAD design systems are no longer content with just capturing the geometry of a model. Design systems that handle larger collections of models, known as assemblies, must also capture the intended relationship between parts. For a designer of assemblies, this need to understand part relationships requires new means of interaction with models.

Our virtual prototyping project attempts to address some of the needs of assembly designers. The virtual prototyping project is built around a force-feedback input device and NURBS-based research modeling environment. The current system runs the modeling and display environment on a workstation and the low latency haptic processes on a real-time microprocessor system.

Minimal distance queries are used to initiate fast local methods on the haptic controller. For example, a minimum distance query is used to check the distance between the haptic probe and the surfaces of the models. When the distance becomes small, the haptic process starts a local tracking method using the parametric value for the closest point returned from the global query.

In addition, global distance queries are used to provide information back to the model designer. Showing selected distances between models helps the designer understand the spatial relationships and to accurately position models. Accurate distance calculations can also be used to show such things as part clearances during an assembly process. We achieve interactive rates for a variety of model types and surface representations.

2.0 BACKGROUND

Minimum distance solving is a fundamental query for model representations.The minimum distance for two objects A and B represented by compact sets K_A and K_B is most naturally defined as

$$d(A, B) = min(\|x - y\| : x \in K_A, y \in K_B) \quad \text{(EQ 1)}$$

Measures of minimum distance for a surface representation have utility in data fitting, robotics, animation, and interactive systems, to name just a few areas. An important special case of finding the minimum separation between two objects is collision detection.

For some surface representations, minimum distance queries are simple to answer. The distance from a point to a plane is found using a single dot product with the unit length plane normal. As soon as the plane becomes bounded, however, the situation becomes more complex, since the point must be checked against the polygon edges. Since useful models contain hundreds or thousands of polygons, naive algorithms that check against all primitives can quickly involve millions of tests.

Implicit primitives, such as spheres, also have simple distance properties. This should not be surprising --- implicit surfaces are defined to be the zero set of a function, so some measure of distance is inherent in the definition. This view of minimum distance queries as a root finding operation holds for piecewise polynomial surface formulations such as NURBS surfaces.

Much of the early work in this area comes from the robotics and computational geometry communities, although much of the work has focused on convex polygonal surface representations. Many of these methods utilize geometric approaches that attempt to classify the space around the model into regions that take advantage of coherence in the model and model movement.

2.1 Minimum Distance Between Polygonal Models

The problem of finding the minimum distance between polygonal models has received a great deal of attention in the computational geometry literature. Chin[1] and Edelsbrunner[2] both report on O(logN) algorithms for finding the minimum separation between two convex polygons. Dobkin[19] uses a different approach to get the same result and extends his work to arbitrary polyhedra. His algorithm produces a hierarchy of simpler and simpler polyhedra with a constant time step to move from the closest point on one level to the closest point on the next more complex level. While these results are useful in showing the asymptotically best algorithms available they are not really practical for real-time applications.

The robotics community has provided the most practical methods for solving repeated minimum distance queries between convex polygonal models. Two main approaches stand out in the literature, those based on Voronoi regions and those based on Minkowski differences. Lin[10] recognized that the Voronoi regions of a convex object have a simple structure such that local methods for minimum distance will always converge. With suitable preprocessing, the closest point on an object can be updated in constant time. This method has been extended to handle collisions between concave polyhedral objects by decomposing the object into hierarchies of convex objects. However, the globally exact minimum distance is no longer tracked.

Gilbert[13] used the Minkowski difference of two convex objects to determine when objects are in contact. Since the Minkowski difference A - B is convex if A and B are convex, a search on the difference will converge. When there is no collision, Gilbert's method returns the minimum separation between objects. Chung[14] used this idea along with efficient means of updating the Minkowski difference to create a collision detection method for convex polyhedra.

Quinlan[15] has an efficient method of finding the minimum separation between general concave polyhedra. His method is based on sphere trees, which have also been used in collision detection[16]. Spheres bound portions of the model such that the distance to a sphere provides a lower bound estimate to the distance to the contained geometry. Spheres are pruned by first finding an upper bound to the minimum distance by doing a depth first search of the sphere tree until some actual distance between parts of the two models is measured. Spheres that have a lower bound distance greater than the distance established between the two models are removed from further consideration. This method works best when the exact distance is not measured, but some relative error allowed, since the depth first search is not as deep and the more expensive leaf-leaf distance tests do not have to be performed. Calculation times vary from 2 milliseconds for inexact distances to 200 milliseconds for exact calculations for models with thousands of polygons.

2.2 Minimum Distance to Parametric Models

Polygonal representations have been the most pervasive surface representation in minimum distance calculations. However, Snyder[9] makes several arguments in favor of parametric representations. Parametric representations allow for exactness in collision, they do not have false stable regions such as a faceted sphere would have, and computation of exact surface properties allows for faster convergence of numerical methods.

Existing methods for solving minimum distance queries for parametric representations lose much of the geometric flavor of polygonal methods. For example, the closest point on a curve Q(t) to a point P, can be determined by solving for t in

$$(Q(t) - P) \cdot Q'(t) = 0. \tag{EQ 2}$$

Eq. 2 expresses that the closest point on a curve is an orthogonal projection ("OP") of P onto Q(t). This relationship is perhaps more intuitively seen by saying that the closest point is one of the extrema on the distance squared curve

$$D^2(t) = (Q(t) - P)^2. \tag{EQ 3}$$

The extrema are found by differentiating Eq. 3 and finding the roots of

$$\frac{d}{dt}\left((Q(t) - P)^2\right) = 0. \tag{EQ 4}$$

Note that Eq. 4 has the same roots as Eq. 2. Finding the extrema for a NURBS curve can be done through refining the derivative curve in regions where its control polygon crosses zero, or through local root finding methods such as Newton-Rapheson iteration, or a combination of the two. When the extrema are found, the extremal points are compared for distance and the extremum with the smallest distance is selected as the parametric value for the closest point.

Plass and Stone[17] use Eq. 4 in a curve data-fitting algorithm. Newton-Rapheson iteration is used to find the local root. A similar method is used to perform point inversion[4]; however, the starting point for root-finding is found through multiple evaluations of the original curve. A data-fitting algorithm for Bezier curves[5] uses the idea of Eq. 2, but solves for the roots through curve refinement. Eq. 2 was again used to solve for the bisector of a point and a planar parametric curve in[3].

Mortenson[6] derives the equations for different types of surface distance measures. The minimum distance to surfaces and between surfaces can all be stated as equations involving points on the surface and the surface normals at the respective points. These symbolic computations are global solutions and difficult to compute. For a point P, the closest point on a surface S is the nearest root of

$$(S-P)\times\left(\frac{\partial S}{\partial u}\times\frac{\partial S}{\partial v}\right)=0. \quad \text{(EQ 5)}$$

Cross product surfaces are difficult to compute, a slightly easier approach is to simultaneously satisfy

$$(S-P)\cdot\frac{\partial S}{\partial u}=0 \quad \text{(EQ 6)}$$

and

$$(S-P)\cdot\frac{\partial S}{\partial v}=0. \quad \text{(EQ 7)}$$

A global solution of this system requires computation of high order dot product surfaces, solving for the zero curves in each parametric direction, and finding the intersections of those curves. Our research system has previously implemented this approach. Solutions are found on the order of one second for surfaces of low order and with a small number of possible minima. There are also numerical issues that make high precision solutions difficult.

Limaiem[7] has presented methods for finding the minimum distance to parametric curves and surfaces as well as the minimum distance between them. The minimum OP (orthogonal projection) to a curve is found by sampling the curve and finding where Eq. 2 changes sign. The parametric points are interpolated between the changed sign points to solve for the zero. The OP onto a surface is found by repeated OP's onto isocurves of the surfaces of alternating direction. The minimum distance between two surfaces is found through successive orthogonal projections from one surface to the other. By repeatedly finding the closest point on alternating surfaces, the algorithm converges to a local minimum distance. The paper restricts the analysis to convex regions to guarantee convergence.

Baraff[8] uses the closest points between strictly convex surfaces to create “witnesses”, simpler geometry that captures the disjointedness of two models. Local numerical methods update the closest points. Snyder[9] also uses local numerical methods to track the closest points between parametric surfaces. Polygonal collision detection on bounding meshes initiates the exact surface methods.

Lin[10][11], also uses a polyhedral first pass to initiate numerical methods for concave surface-surface minimum distance finding. Once a bounding polyhedra indicates a potential for collision, resultant methods solve for the exact minimal distance. Local methods update the solution in the case of movement, and global methods check the local update solution. Times ranging from a second to several minutes are reported to find the global solutions.

2.3 Interval Methods

Interval methods have been used to provide robust calculations on parametric surfaces. They share some characteristics of the minimal distance methods I propose. In particular, interval methods use a geometric approach instead of a numerical approach to problem solving, and make use of bounded regions in parametric space to restrict the solution space. Snyder has developed a global minimum distance method for parametric surfaces with a reported solution time of five seconds between two fairly complicated sculptured surfaces[21]. His approach avoids finding all extrema of the distance curve, instead interval methods are used to find the global minimum. Interval methods provide robustness for the global minimum search.

Interval methods have also been used in sculptured surface collision detection[20], ray-surface intersection, surface tesselation and other operations on parametric surfaces. In a sense, interval methods show the utility and robustness of approaches that emphasize geometry over numerical issues.

2.4 Collision Detection

Collision detection can be considered a special case spatial classification query. The collision detection problem has received a great deal of attention in both the robotics and the computer graphics literature. Many collision detection algorithms are based on model intersection, with the majority of effort going to process the models so intersection can be performed efficiently, as in[12]. The most efficient algorithms follow a simple paradigm:

Detect regions of possible intersection; prune away the remainder; tighten.

The work done with oriented bounding box trees, or OBB trees[18], follows this paradigm as does Hubbard's work with sphere trees[16]. In the OBB tree method, a hierarchy of bounding boxes is used around each polygonal model. If a box on one model intersects a box on the other model, then the children boxes of each are tested against each other. These algorithms are effective because box-box intersection algorithms are very fast, and the hierarchy of boxes allows for efficient coverage of the entire model. These types of collision detection algorithms are not useful for estimating the minimal distance between models. A pair of bounding nodes can be rejected very early, even though the contained geometry may be very close.

3.0 Minimum Distance Queries for Arbitrary Geometry

In order to allow real-time interaction in the virtual prototyping environment, we have developed algorithms to solve the minimum distance to an object and the minimum separation between objects. These algorithms have good average case behavior, and scale well with environment complexity. In addition, they have many time-critical properties, so that their performance will degrade gracefully.

3.1 Spatial Classification using Upper and Lower Bounds

We introduce the use of inexpensive bounding regions that allow fast computation of upper and lower bounds (ULB) to the minimum distance solution. These bounds can be used to prove that the contained geometry cannot be part of the solution. Used in conjunction with hierarchical data structures, this iterative method allows efficient spatial classification queries. This approach can be thought of the dual of the intersection paradigm --- instead of

Detect regions of possible intersection; prune away the remainder; tighten,

we apply the paradigm of

Detect impossible regions, prune those away, tighten remainder.

At each iteration there are two main steps to the process. In the first step, an inexpensive lower bound distance is established for each bounding region. In the second step, an inexpensive upper bound to the minimum distance is established for the entire model. All bounding regions that have a lower bound distance greater than the upper bound to the minimum distance cannot be part of minimum distance solution and can be ignored. The hierarchy is refined for the active nodes and the process repeats.

The efficiency of the algorithm depends on finding bounding regions that inexpensively compute lower bounds, yet that are tight enough to provide reasonable convergence. Efficiency also depends on a reasonable way of finding an upper bound to the minimal distance.

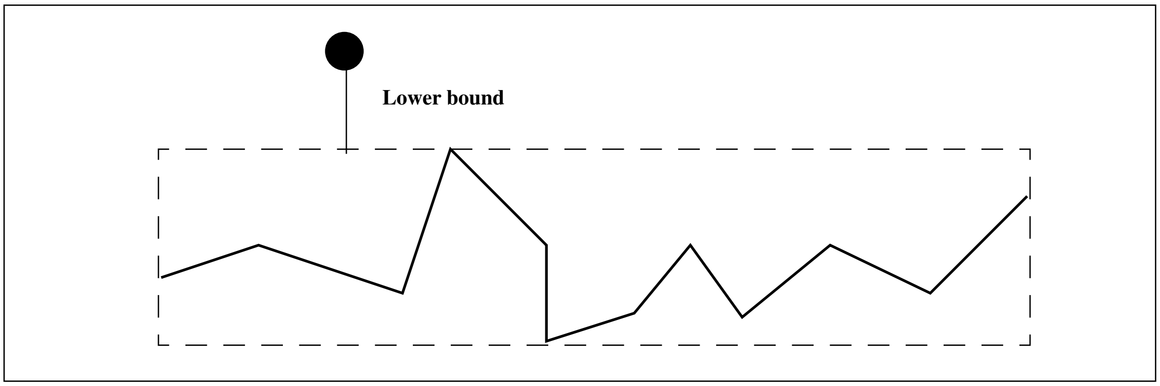

FIGURE 2. Minimum Distance to Polyline.

Let us examine the behavior of this paradigm for the simple case of finding the minimum distance from a point P to a polyline, using simple axis-aligned bounding boxes (AABBs).The algorithm starts by enclosing the entire polyline in a bounding box. A lower bound to the minimum distance to the polyline can be found be computing the distance to the box (Figure 2). In order to apply the new methodology, an upper bound to the minimum distance is needed as well. While a lower bound shows that all the geometry contained by the box is further away than the lower bound, an upper bound shows that the minimum distance must be at least that close.

Several possibilities exist for finding an upper bound for the closest point on the polyline.We desire a minimal upper bound, but also want to minimize computation. The minimum upper bound would be the closest point on the segments contained by the box, which, unfortunately, is just the original problem again.

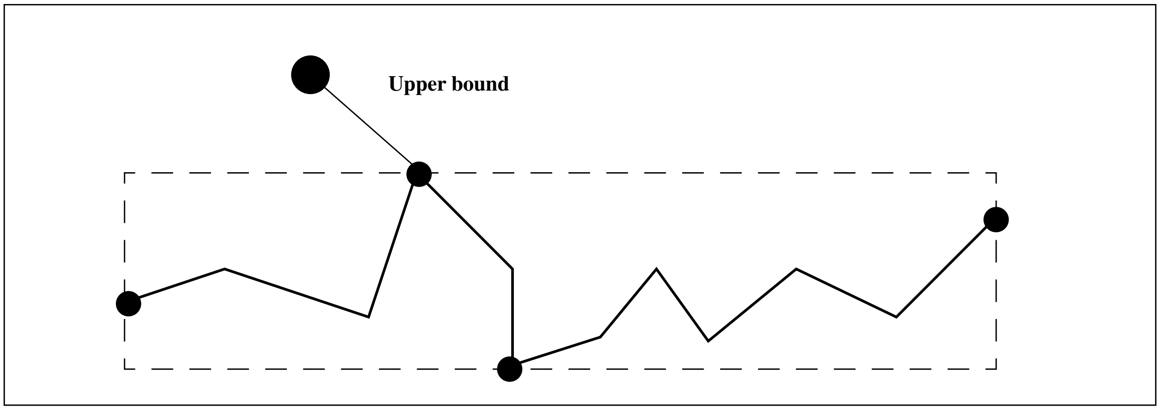

FIGURE 3. Estimating an upper bound.

An inexpensive, yet not too loose upper bound can be found by retaining the polyline vertices that create the boundaries of the closest bounding box (the black dots in Figure 3). A vertex that created the boundary is likely to be reasonably close to the point P.

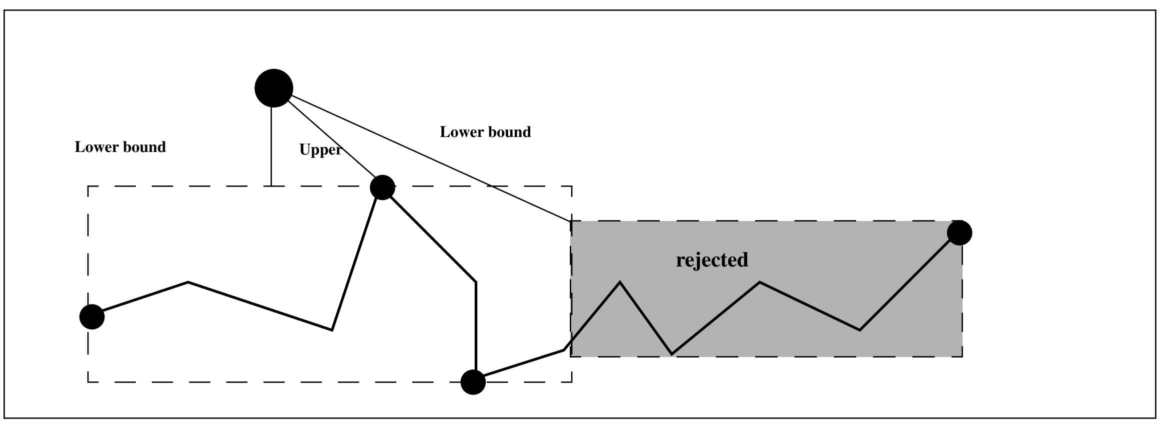

FIGURE 4. Pruning boxes based on bounds.

The algorithm proceeds by subdividing the polyline in half and creating new bounding boxes out of the subdivided portions. If the lower bound to the minimum distance on one of them is larger than an existing upper bound to the minimum distance, then all the polyline segments in that box must be further away than the vertex that defined the current upper bound distance (Figure 4). That box can be ignored in all further computation.

The algorithm traverses a hierarchy of bounding boxes around the polyline segments. At each level, the lower bound distance to each box is computed, and a new upper bound distance for the level is found from the nearest bounding box. If a box has a lower bound distance greater than the level's upper bound to the minimum distance, then it cannot be a candidate for containing the true closest point. At each iteration, the lower bound distances to the boxes tend to get larger and the upper bound to the minimum distance tends to shrink. When boxes surround single line segments the segments are accepted for exact distance computation.

```
### Pseudocode for min dist to polyline
Queue = MakeBBox( Polyline );
While ( Queue )
  ForEach BBox in Queue
    BBox->Dist  = CalcMinBound(BBox);
    if ( BBox->Dist < MinLowerBoundBox->Dist)
      MinLowerBoundBox = BBox;
  UpperBound =
    CalcUpperBound(MinLowerBoundBox);
  ForEach BBox in Queue
    if ( BBox->Dist > UpperBound )
      Prune(BBox);
    else
      BBox1 = Subdivide( BBox, left );
      BBox2 = Subdivide( BBox, right );
      if ( NumSegs( BBox1 ) == 1 )
        SaveQueue->Append( BBox1 );
      else
        NewQueue->Append( BBox1 );
      if ( NumSegs( BBox2 ) == 1 )
        SaveQueue->Append( BBox2 );
      else
        NewQueue->Append( BBox2 );

  Queue = NewQueue;

return ExactDistance( SaveQueue );
```

In the worst case, there would be $N\log_2 N$ lower bound distance computations and $\log_2 N$ upper bound computations, worse than the N distance calculations needed using a naive linear algorithm. In the best case, one of the bounding boxes would always be removed at each level and there would be $\log_2 N$ lower and upper bound calculations, a significant advantage over the linear algorithm.

3.1.1 Minimum Distance Calculations for Concave Polyhedra

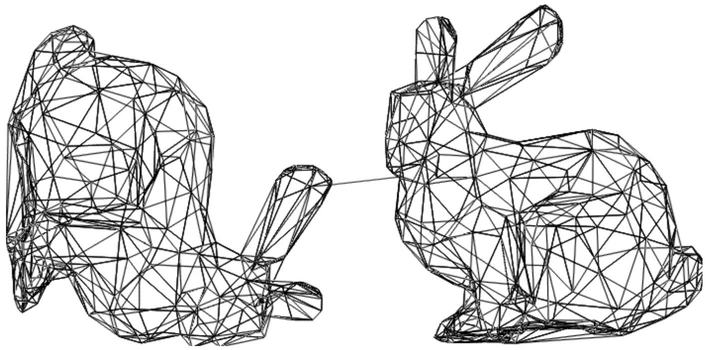

FIGURE 5. The minimum distance between 2 polygonal bunnies (575 triangles each).

We apply the above paradigm to arbitrary polygonal models and extend it to find the minimum separation between pairs of models (Figure 5). In the general case of repeated distance calculations to moving models, AABBs have significant disadvantages over other bounding regions. The tree of boxes must be updated after every rotational movement and they provide poor coverage for polygons with large aspect ratios and for polygons that have a dominant axis that is not aligned with a spatial axis.

Our experience with AABBs shows the utility of the lower-upper bound method, and they remain a good choice for one-time distance calculations. In the general polygonal case with repeated distance measures, we use oriented bounding boxes to provide tighter coverage of the models. Distances to and between OBBs are determined using Gilbert's convex algorithm. This is very efficient for small convex models, and we can compute about 30,000 box-point distances and 15,000 box-box distances per second.

In order to establish a reasonable upper bound distance, a small subset of vertices that maximize spatial distribution within the box are cached, rather than the vertices that make up the faces of the box as in the AABB polyline example. These vertices provide better upper bound calculations on average.

The extension from point-mesh to mesh-mesh minimum distance algorithm is simple. A lower bound distance for each box is still needed, as is an upper bound to the minimum distance at each iteration. The lower bound distances are computed by finding the minimum distance between each active OBB of one model and all the active OBBs of the other. We also use bounding spheres derived from the OBBs as a rejection test before doing the more expensive Gilbert's algorithm.

Since the number of active boxes N on one model and M on the other is small, the O(NM) cost at each level to establish lower bound distances is small. The upper bound estimate to the minimum distance is computed by finding the nearest pair of cached vertices for the closest boxes between models.

The efficiency of the polygonal mesh method depends on creating tight bounding trees for the model. We used the publicly available OBBTree package[18] to generate a hierarchical oriented bounding tree. Very minor modification was required to generate the cached sample points in each box for use by the upper bound distance calculation. The distributed OBB package does not include some of the suggestions in the related paper for improving the fit of the boxes around the model, and thus the boxes produced were not very tight --- a visual estimation on some of the tested models shows the produced boxes were about 30% larger than optimal.

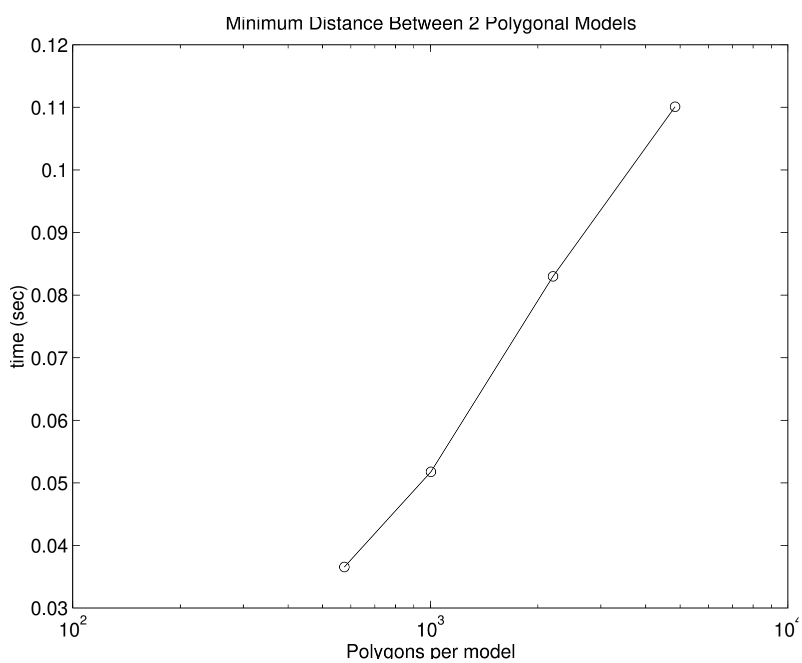

FIGURE 6. Time vs. model size plotted on a semilogx scale for minimum distance between various resolution bunny models.

The scalability of the algorithm was tested by moving different resolutions of the bunny model around the same stationary bunny and repeatedly measuring the distance between them. The algorithm shows log(N) behavior, where N is the number of polygons in each model, over a range of model sizes from 500 triangles to 5000 triangles (Figure 6). Times to measure the exact distance ranged from 36 milliseconds to 110 milliseconds.

3.1.2 Minimum Distance to Arbitrary Sculptured Models

The conciseness of problem statement for the parametric minimum distance problem, as in the minimum distance to curve (Eq. 2), has led people to focus on methods of solving these equations. However, an analysis of these equations shows that they solve for much more information than is needed. Use of these equations makes solution time highly dependent on the number of extrema in the solution. In addition, the system can involve equations of high order, causing problems with robustness and numerical precision.

The same methodology that applies to polygonal models can be applied as well to parametric models. The situation is slightly different, since a hierarchical bounding tree cannot be efficiently pre-computed and stored for a surface. Fortunately, since parametric representations are compact, the necessary bounding information can be quickly dynamically computed.

We take advantage of certain properties of parametric surface representations for efficient solution of the closest point problem. B-splines, NURBS and Bezier patches are among the representations that share a local convex hull property. For a B-spline, each polynomial piece of the surface is bounded by the portion of the control polygon that contributes towards the piece. For a bi-cubic patch, the local convex hull consists of 16 points (Figure 7).

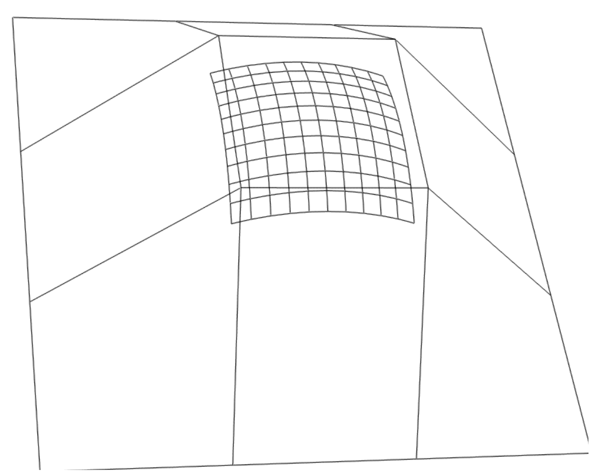

FIGURE 7. A B-spline patch and its local convex hull.

These local convex hulls allow efficient lower bound distance computation. We use Gilbert's algorithm to find the nearest feature on the local convex hulls. Performance is improved by first pruning active pieces with their AABB. Creation of the AABBs is efficient since the local convex hull is compact. An entire parametric piece can be rejected by finding the distance to its surrounding local convex hull.

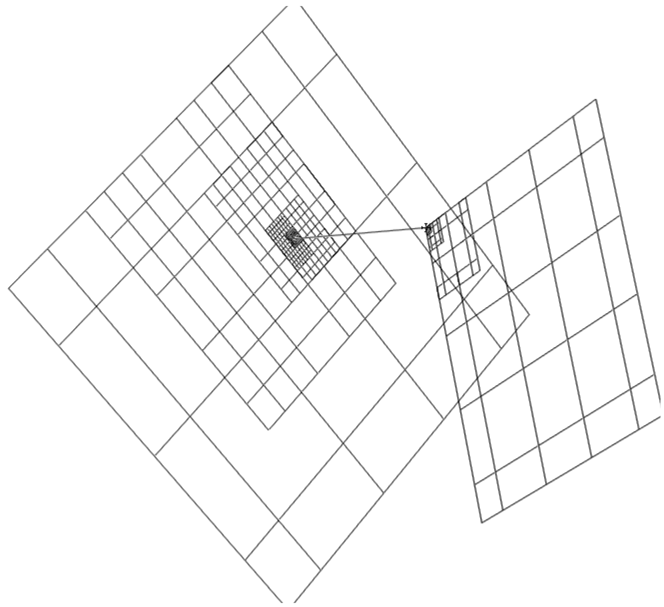

FIGURE 8. Levels of surface refinement created by the minimum distance between two surfaces algorithm.

It is necessary to be able to find an upper bound to the minimum distance for this method to be suitable. A point on the surface is found by saving the closest convex hull and mapping the closest point on the convex hull to the surface with nodal mapping[9]. Patch pieces that are further away than the upper bound distance are removed from further consideration. Patch pieces that remain are grouped into features, and the portion of the surface containing each feature is extracted and refined using surface refinement. The algorithm iterates over the new, smaller surface patches (Figure 8). The extension from point-surface to surface-surface is the same as for the point-mesh to mesh-mesh extension

.

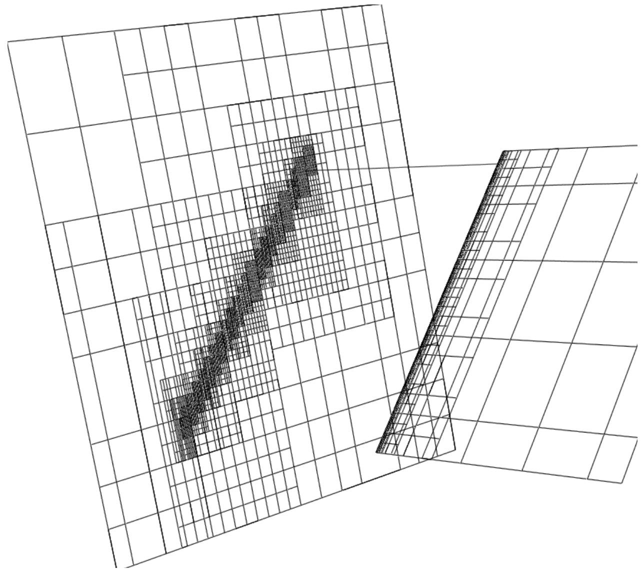

FIGURE 9. Convergence for this manifold solution is improved using granularity of active regions.

In our implementation of the surface-surface distance test, we made a few modifications to improve the worst case behavior. In a tensor product surface, there are preferred parametric directions, namely along isoparametic lines. Active regions that fall along these boundaries are most efficiently aligned. Certain configurations of models can produce long, thin active regions that, in the worst case, fall diagonally in parameter space. We introduce the notion of granularity of regions to reduce unnecessary refinement in these situations. By extracting active regions only up to a certain size, we improve convergence. For example, with a granularity of 2x2, a long diagonal in parameter space would be broken into many small surfaces (Figure 9), each containing mostly the diagonal. In the non-granular case, the entire rectangular region containing the diagonal would be extracted and refined.

We achieve interactive performance for finding the minimum distance to a surface and the minimum distance between surfaces. A parametric precision of 10^{-6} is obtained at speeds of 10-50Hz when running the closest point to surface algorithm. The minimum distance between surfaces runs at 2-20Hz, depending on the surfaces and orientations.

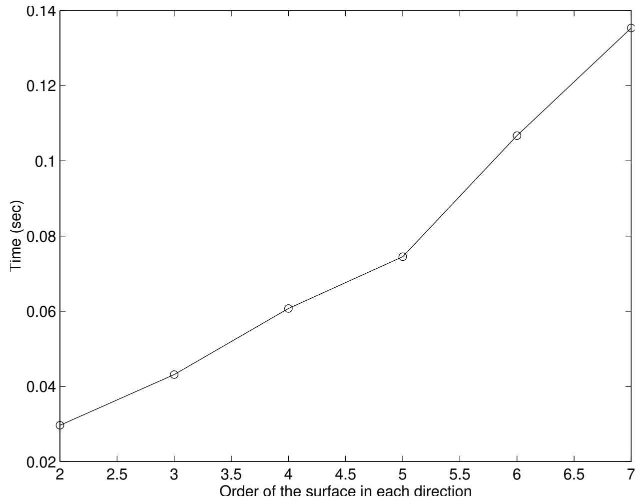

FIGURE 10. Time to convergence vs. order of the surface patch.

The algorithm is stable over a range of surface orders. We tested a B-spline patch with a ten by ten control mesh at different surface orders. Convergence is slower on higher degree patches, since the local convex hull is larger for each patch segment and the segment becomes harder to prune. However, the algorithm remained stable even for high degree surfaces (Figure 10).

3.1.3 Time-critical Properties

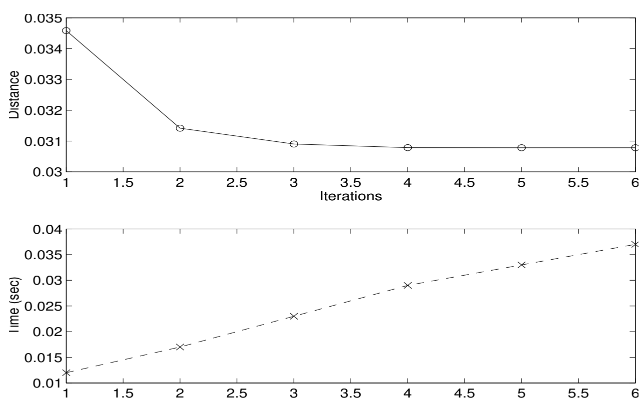

FIGURE 11. A. Minimum distance vs. iterations.
B. Total time vs. number of iterations

Our methodology has strong time-critical properties, a useful characteristic in interactive systems. The algorithm can be interrupted at any iteration for an partially converged solution. Since there is no large pre-processing step for the solution, each iteration takes the same order of computation time, making it possible to schedule for a certain precision result at a known computational cost. Figure 11 shows that time per iteration is fairly constant, while the error in the distance measure is halved each time step.

Detection of solution manifolds is a desirable property of minimal distance and collision algorithms. This algorithm would suffer in the degenerate case of trying to find the closest point to a circle from the center, since no pruning would take place and the entire curve would be refined to high resolution. The time-critical property of the algorithm can help. If the algorithm does not converge in the expected time, the iteration can be aborted and the entire remaining manifold returned. Since we can compute tight bounds on the expected convergence, any result after that time should be a decent approximation to the complete manifold. Alternatively, when a region fails to reduce in size after a number of iterations, it can be removed from computation and returned as a manifold solution.

3.2 Assembly Application

We have integrated and tested these algorithms in a display front-end for a haptic rendering system. The display process is intended to alert the haptic processes when surface contact is possible. The closest point to surface algorithm is used to determine when the haptic probe may contact a surface. A pure collision detection algorithm would not be suitable in this application since information must be sent to the haptic process before contact occurs. The minimum distance between surfaces algorithm provides a similar role in alerting low latency collision processes to possible contact between surfaces.

In addition, we have found that displaying minimal distance lines between a 3D cursor and the objects in a scene helps improve a designer's 3D location sense and depth perception. Similarly, displaying the minimum distance line between two objects helps a designer understand their spatial relationship. When trying to move an object in a scene, the object can be placed more easily relative to other objects with this minimum distance information displayed.

4.0 Future Work

We have implemented methods for minimum distance between a point and surface, between surface and surface, and between polyhedral model and polyhedral model, as well for the 2D equivalents. There is nothing preventing hybrid representation methods from working as well, such as surface to polyhedral model. We would like to extend this work into a "unified" minimum distance system that can handle many different types of models.

Several factors can improve the speed of these algorithms. The two areas to address are improving the bounding boxes and improving the estimate of upper bound. As previously noted, we feel the OBBs used around the polyhedral models could be significantly tightened. Since the tree construction is a preprocess step, more time could be invested in fitting the model. Also, the local convex hulls around the polynomial pieces of the parametric surfaces are rather loose, and we would like to investigate methods for tighter boxes.

Finding better estimates to the upper bound distance would improve pruning of the tree, as well as speed convergence. Improved estimates can also help in situations such as contact, where an early determination of zero distance allows early termination of the iteration.

5.0 Conclusion

This new paradigm for solving minimum distance problems is applicable to a wide variety of graphics primitives. While the worst-case behavior confers no advantage over naive algorithms, the expected average case performance is as good as the best optimal algorithms, and has much better real-time performance. This paradigm allows for the first practical computation of distances between non-convex polyhedra, and spatial classification queries on sculptured models that are two orders of magnitude faster than current methods. We expect these methods to allow whole new classes of applications on sculptured models.

6.0 Acknowledgements

This work was supported in part by DARPA (F33615-96-C-5621) and the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219) and NSF grant MIP-9420352. All opinions, findings, conclusionsor recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies. Thanks to the University of North Carolina, Chapel Hill for providing the OBBTree package and the bunny model. Thanks to the Alpha_1 group for supporting this work.

7.0 REFERENCES

[1]Chin, Francis and Wang, Cao An. "Optimal Algorithms for the Intersection and the Minimum Distance Problems Between Planar Polygons", IEEE Transactions on Computers, Vol. C-32, No. 12, Dec. 1983, pp. 1203-1207.

[2]Edelsbrunner, H. "On Computing the extreme distances between two convex polygons", Tu Graz, Tech. Rep. F96, Aug. 1982.

[3]Farouki, Rida and Johnstone, John. The bisector of a point and a plane parametric curve. *Computer Aided Geometric Design* **11**, pp. 117-151 (1994)

[4]Piegl, Les and Tiller, Wayne. *The NURBS book*. Springer-Verlag, Berlin. p. 230, 1995

[5]Schneider, Philip. Graphics Gems I. Academic Press, San Diego. p. 607, 1990

[6]Mortenson, Michael. *Geometric Modeling*, John Wiley & Sons, New York pp. 305-317, 1985

[7]Limaiem, Anis and Trochu, Francois. "Geometric Algorithms for the Intersection of Curves and Surfaces", Computer & Graphics, Vol. 19, No. 3, pp.391-403, 1995.

[8]Baraff, David. Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulation. Computer Graphics, Vol. 24, No. 4, pp.19-28, 1990

[9]Snyder, John M. An Interactive Tool for Placing Curved Surfaces without Interpenetration. *Proceedings of Computer Graphics*, pp. 209-218, 1995

[10]Lin, Ming, Manocha, Dinesh, and Canny, John. Fast Contact Determination in Dynamic Environments.

[11]Lin, Ming and Manocha, Dinesh. Fast Interference Detection Between Geometric Models. The Visual Computer, pp. 542-561, 1995

[12]Garcia-Alonso, Alejandro, N. Serrano, and J. Flaquer. "Solving the Collision Detection Problem," IEEE Computer Graphics and Applications, pp.36-43, May 1994

[13]Gilbert, Elmer, Johnson, Danial, and Keerthi, S. "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," IEEE Journal of Robotics and Automation, pp. 193-203, April 1988

[14]Chung Tat Leung, Kelvin. *An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments*. M. Phil Thesis, The University of Hong Kong, 1996

[15]Quinlan, Sean. "Efficient Distance Computation between Non-Convex Objects," ASME Conference on Robotics and Automation, pp. 3324-3329, 1994

[16]Hubbard, P.M., Interactive collision detection, Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality, October 25-26, 1993, pp. 24-31

[17]Plass and Stone, "Curve-Fitting with Piecewise Parametric Cubics," Computer Graphics, vol. 17, issue 3, 1983, p.233-239.

[18]Gottschalk, S.,Lin, M.C., and Manocha, D., "OBBTree: A Hierarchical Structure for Rapid Interference Detection," Computer Graphics Proceedings, Annual Conference Series, 1996, pp.171-180.

[19]Dobkin, David and Kirkpatrick, David. "Determining the Separation of Preprocessed Polyhedra - A Unified Approach", in Proc. 17th International Colloq. Automata Lang. Program., in Lecture Notes in Computer Science, Vol. 443, Springer-Verlag, pp.400-413, 1990

[20]Von Herzen, Brian, Barr, Alan, and Zatz, Harold. “Geometric Collisions For Time-Dependent Parametric Surfaces”, in Computer Graphics, Vol. 24, Number. 4, August 1990 (Proceedings SIGGRAPH 1990), pp. 39-48

[21]Snyder, John, “Interval Analysis for Computer Graphics,” Computer Graphics, 26(2), pp.121-130, July 1992.