A Framework For Efficient Minimum Distance Computations

By David E. Johnson and Elaine Cohen

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

Abstract

In this paper we present a framework for minimum distance computations that allows efficient solution of minimum distance queries on a variety of surface representations, including sculptured surfaces. The framework depends on geometric reasoning rather than numerical methods and can be implemented straightforwardly. We demonstrate performance that compares favorably to other polygonal methods and is faster than reported results for other methods on sculptured surfaces.

1 Introduction

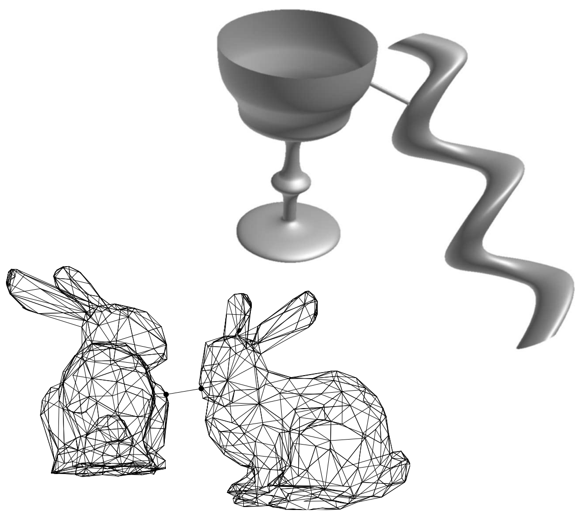

FIGURE 1. The minimum distance between models.

We introduce a framework for minimum distance calculations that applies well to both polygonal and parametric model representations (Figure 1). The resulting methods scale well with problem size, have time-critical properties, and are interactive for large polygonal models and sculptured surfaces.

In robotics, minimum distance queries have been used in path planning [2], path modification [25], and collision avoidance [15]. In computer graphics, minimum distance computations have played roles in physical simulation [1][20] and model prototyping [32]. A haptic prototyping project [13] which uses minimum distance computations to maintain local geometry on the haptic controller [30] motivates our own interest.

2 Background

Much of the early work in this area comes from the robotics and computational geometry communities, although their work has often been focused on convex, polygonal models. In the computer graphics community, the demand for realistic, 3D environments has driven contributions in collision detection and minimum distance computations. We summarize approaches from both groups below.

2.1 Minimum Distance for Polygonal Models

Chin [4] and Edelsbrunner [10] both report on O(log*N*) algorithms for finding the minimum separation between two convex polygons. Dobkin [8] uses a different approach to get the same result and extends his work to arbitrary polyhedra. However, these theoretical algorithms all have unknown and presumably large time coefficients.

2.1.1 Convex Polygonal Models

Lin [17] uses the Voronoi regions of convex polyhedra so that local search methods for minimum distance will always converge. The Voronoi regions structure space so that the closest point can be updated in a constant time step for small movements. The I-COLLIDE system [7] follows this approach and adds a spatial sorting method to reduce the number of object interactions. This method has been extended to handle collisions between concave polyhedral objects by decomposing the object into hierarchies of convex objects [23].

Gilbert [11] employs the Minkowski difference of two convex objects to determine their minimum separation. With slight modification [3] this method can also provide constant time updates for slowly moving polyhedra. Chung [5] added an efficient means of updating the Minkowski difference to create a collision detection method for convex polyhedra.

2.1.2 General Polygonal Models

Quinlan [24] produced an efficient method of finding the minimum separation between general concave polyhedra by exploiting hierarchical, spherical bounds to prune portions of the model. The method prunes by establishing upper bounds on the minimum distance with depth-first descent to the leaf nodes of the models. Allowing approximate results (“relative error”) speeds convergence and demonstrates log(*N*) scalability, where *N* is the number of polygons in the scene. When exact distance is required, however, the log(N) complexity no longer holds. Sato et al [26] created a collision detection method for arbitrary polyhedra by combining Quinlan's sphere-tree method and Gilbert's convex method.

2.2 Minimum Distance for Parametric Models

Existing methods for solving minimum distance queries for sculptured models lose much of the geometric flavor of methods for polygonal models. Instead, root-finding or minimization methods solve systems of equations that describe conditions for the minimum distance.

2.3 Minimum Distance to a Surface

Mortenson [21] derives equations for different types of surface distance measures. For a point P, the closest point on a surface S is the nearest root of

$$(S-P)\times\left(\frac{\partial S}{\partial u}\times\frac{\partial S}{\partial v}\right)=0. \quad \text{(EQ 1)}$$

This high order equation is difficult to solve; a slightly easier approach is to simultaneously satisfy[22]

$$(S-P)\cdot\frac{\partial S}{\partial u}=0 \text{ and } (S-P)\cdot\frac{\partial S}{\partial v}=0. \quad \text{(EQ 2)}$$

Our research group has found this approach to be slow (on the order of one second on an SGI Indigo2) and to exhibit numerical problems in practice.

2.3.1 Minimum Distance between Convex Surfaces

Limaiem [16] has presented methods for finding the minimum distance to convex parametric curves and surfaces as well as the minimum distance between them. His algorithm converges to a local minimum distance by repeatedly finding the closest point on alternating surfaces.

Baraff [1] 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 [27] tracks the closest points between parametric surfaces with local numerical methods initiated by polygonal collision detection.

2.3.2 Minimum Distance between Concave Surfaces

Lin [18][19] also uses a polygonal first pass to initiate numerical methods for concave surface-surface minimum distance finding. Once a bounding polyhedron indicates a potential for collision, resultant methods solve for the minimum distance to the underlying surface. Local methods quickly update the solution in the case of movement. Times range from one second to several minutes for finding the global solutions using an IBM RS/6000.

Snyder has developed a global minimum distance method for parametric surfaces that avoids examining all extrema of the distance and instead finds the global minimum by using interval methods. His method determined the global minimum for an example problem in five seconds on an HP9000 series 835 workstation [28].

2.4 Collision Detection

We relate the collision detection problem to the minimum distance problem by observing that a collision can be detected by when the minimum distance is zero. Below, we describe some collision detection methods that are related to this paper's contribution.

2.4.1 Collision Detection for Polygonal Models

OBBtrees [12] apply to general polygonal models, as does Hubbard's work with sphere trees [14]. Both methods build bounding hierarchies around the original models. The OBBtree work is based on an efficient overlap test for oriented bounding-boxes. Hubbard's work used spheres as bounding primitives and concentrated on generating efficient hierarchies using a medial-axis construction method.

2.4.2 Collision Detection for Parametric Models

Von Herzen [33] developed a collision detection method for time-dependent parametric surfaces which prunes portions of the surface using Lipschitz bounds. Snyder [29] extended the method to better handle manifold contacts and used interval methods for efficiency.

2.5 Summary

Approaches for convex regions typically depend on guaranteed convergence of local methods for efficiency. In the concave case, methods for concave polygonal models have emphasized hierarchical geometric processes, while methods for concave sculptured models treat the problem as root-finding of some distance equation. We would like to find some common ground for polygonal and parametric models.

3 A Minimum Distance Framework

We have developed algorithms to compute the minimum distance to an object and the minimum separation between objects for polygonal and parametric surface representations. These algorithms are based on a common set of geometric operations, allowing us to describe methods for finding minimum distance as part of a general framework.

3.1 An Overview of the *LUB-Tree* Framework

We refer to our minimum distance framework as a lower-upper bound tree (*LUB-tree*) framework. Each surface representation that is part of the framework must provide a set of common operations, including *bounding volume generation, lower bound on distance computation, upper* *bound on model minimum distance computation*, *bounding volume refinement*, and a method of determining *computation termination*. A pruning method based on lower and upper bounds uses these operations to converge to the minimum distance.

The pruning method starts by invoking the *bounding volume generation* operation on each of the two models. We treat these bounding volumes as the top *nodes* of hierarchical bounding trees and connect them as an *active pair* of nodes. *Active pairs* point between nodes that still may be part of the minimum distance solution. We then search and prune the bounding hierarchies using the following procedure.

1. *For each* active pair, *compute lower bounds on the distance between nodes using the nodes'* bounding volumes *and* lower bound *operations.*

2. *Establish an upper bound on the minimum distance between the models using the* upper bound computation *operation.*

3. *Prune the* active pair *list by comparing each lower bound distance to the current upper bound. (A lower bound greater than the upper bound implies that the contained geometry must be farther away than the minimum distance.)*

4. *Split remaining* active pairs *into new* active pairs *by invoking the* refinement *operation.*

5. *Repeat until the* termination of computation method *returns true.*

Essentially, we wish to show that portions of the model cannot be part of the minimum distance solution. Hierarchical bounds allow us to efficiently test portions of the model and potentially remove large portions from consideration without high computational cost.

3.2 Illustrating the Algorithm

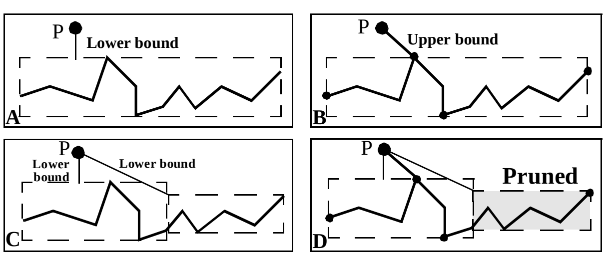

FIGURE 2. A. We compute a lower bound on the active pair. B. An upper bound is found using cached vertices. C. The active pair refines. D. One active pair is pruned.

We illustrate this approach in Figure 2 for the simple case of finding the minimum distance from a point *P* to a polyline. Just as an example, we use an axis-aligned bounding box (*AABB*) as the *bounding volume*, assume the existence of a method that returns the distance to a nearby polyline vertex as the *upper bound* operation, and perform *refinement* by splitting in half the polyline contained within a node. *Computation termination* occurs when the nodes contain single line segments and we cannot refine the *AABB*s any further.

The pruning method starts by creating an *active pair* that points to *P* and to the top-level *node*. The algorithm descends the bounding hierarchy around the polyline segments in a breadth-first manner, while computing lower bounds for each *active pair* and an upper bound at each level in the hierarchy. These bounds determine which *active pairs* refine into new active pairs and which are pruned. When an *active pair* points to a box surrounding a single line segment that segment is accepted for exact distance computation. That exact distance is both a lower bound to that node and a potential upper bound on the minimum distance to the model. The pruning method stops when there are no more active pairs. The upper bound returns as the exact minimum distance.

3.3 Best and Worst-Case Performance

In a pathological case, for a polyline with N segments, there could be $2N$ lower bound and $\log_2 N$ upper bound distance computations, which would make the performance worse than the N distance calculations needed using a simple 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, which is a significant advantage over the linear algorithm.

Now, imagine the case of finding the minimum distance between two models, each with N segments. In the best case, only one active pair survives the pruning at each level, so there would be $\log_2 N$ lower and upper bound calculations. This compares favorably with the N^2 distance calculations a naive algorithm would require. In the worst case, where each segment is within every bounding box and no pruning occurs, there would be $\sum_{i=1}^{N} 2^{2(i-1)}$ lower bound and $\log_2 N$ upper bound distance computations. This exponential lower bound result is fortunately very difficult to produce in anything resembling a useful model. In a more typical bad case of the models being parallel flat surfaces, or concentric spherical regions, the algorithm uses $2N$ lower bound and $\log_2 N$ upper bound distance computations, still better than a naive algorithm.

3.4 Minimum Distance for Concave Polyhedra

The *LUB-tree* framework applies to arbitrary polygonal models (Figure 1). In order to ask minimum distance queries for this surface representation, we must define the *LUB-tree* framework operations. For a *bounding volume*, we precompute a hierarchy of oriented bounding boxes (*OBB*s) using the publicly available *OBBTree* package[12] (Section 2.4.1). Gilbert's convex algorithm (from Section 2.1.1) measures the distance between the *OBB*s to find the *lower bound on distance*. Greater efficiency results from using a two-pass approach to pruning --- a quick sphere-sphere test can compute a rough lower bound on distance and if that distance is less than the current *upper bound* we compute the more expensive, but more precise, *OBB* distance.

Each node points to a small set of vertices from the boundary of the contained geometry. These vertices allow us to compute an *upper bound* on the minimum distance. Since the vertices are part of the model, the distance to a vertex is an upper bound on minimum distance for the model. At each level of the model hierarchy, we save the *active pair* with the smallest lower bound. We find the smallest distance between each set of cached vertices from the nodes of the saved *active pair* to establish an upper bound on the minimum distance between the models.

A*ctive pairs* that remain after pruning split into new active pairs by applying the *refinement* operator. In the case of a precomputed bounding hierarchy, descent to children nodes accomplishes refinement.

The method *terminates* when all the processed *active pairs* point to leaf nodes, which contain single triangles. Gilbert's convex algorithm computes the distances to and between leaf node triangles.

3.5 Minimum Distance for Sculptured Models

Sculptured models, particularly NURBS, are the surface representation of choice in many CAD/CAM packages. We would like to be able to perform minimum distance computations directly on these models without having to resort to conversion to polygonal forms. Since the minimum distance problem can be phrased succinctly for sculptured models [9] (Eq. 2) researchers have focused on methods of solving these high-order equations. A geometric approach, such as the *LUB-tree*, avoids many of the numerical issues that complicate those methods.

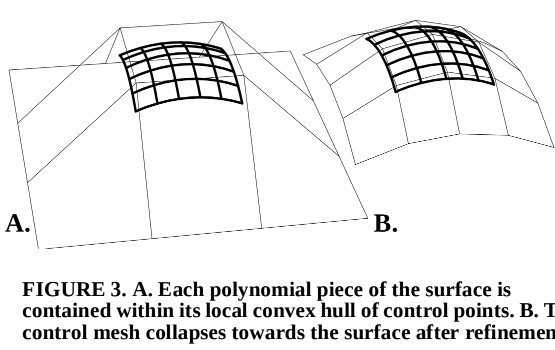

FIGURE 3. A. Each polynomial piece of the surface is contained within its local convex hull of control points. B. The control mesh collapses towards the surface after refinement.

We will quickly review some terminology used for B-spline sculptured surfaces. These surfaces are piecewise-polynomial functions of two parametric variables, commonly u and v, which form the *domain* of the surface. The *control mesh* of the surface provides the vector coefficients, or *control points*, for the basis functions. A local set of control points influences each *polynomial piece* of the surface and completely contains the piece within its convex hull (Figure 3.A). The parametric *nodes* of the surface are an easily computable first-order approximation of the closest points on the surface to the control points [6]. *Refinement* algorithms embed the surface into a new parameter space with more degrees of freedom and compute appropriate additions to the control mesh (Figure 3.B).

We can apply the *LUB-tree* framework to sculptured models by defining the needed operations. A B-spline surface's local convex hull and refinement properties provide the basis for the needed operations.

We form the initial nodes of the bounding hierarchy from the polynomial pieces of the surface. Thus, the *active pair* list is initialized by pairing up the local convex hulls of each surface. Similar to the way we computed lower bounds on polygonal models using *OBB*s, we compute lower bounds for parametric spline models by applying Gilbert's algorithm to the local convex hulls of each *active pair*. Just as performance improved in the polygonal case with the addition of an initial sphere-sphere test, we first prune the *active pairs* by checking the distance between their dynamically updated *AABB*s.

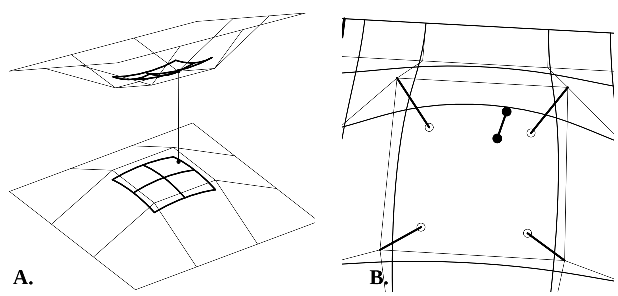

FIGURE 4. A. We find the closest points on the convex hulls of the *active pair* with the smallest lower bound. B. Bilinear interpolation between nodes maps the point onto the surface.

An upper bound on the minimum distance allows us to prune *active pairs*. During each pass through the *active pair* list, we save the *active pair* with the smallest lower bound. We map the closest points between each convex hull from the saved *active pair* onto the underlying surfaces using bilinear interpolation between the parametric nodes associated with the vertices of the control mesh (Figure 4)[27][30]. The distance between the points on the surface forms an upper bound on the minimum distance

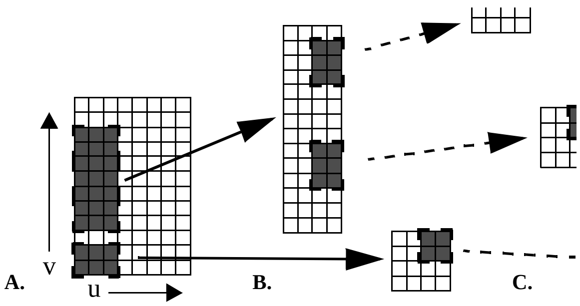

FIGURE 5. A. Polynomial pieces in the parametric domain that remain after pruning are grouped into regions. B. Each region is extracted as a separate surface and refined. C. New pruning and grouping continues the process.

.

The algorithm removes the *active pairs* that have lower bounds greater than the upper bound distance. We wish to refine the remaining polynomial pieces; however, refinement applied repeatedly to the initial surface may produce extraneous refinement in areas that have been already pruned. We group polynomial pieces into active regions formed from contiguous polynomial pieces and extract each active region through refinement at the region's parametric boundaries (Figure 5). We can now refine the extracted surface without extra refinement in the original surface. New *active pairs* form from the refined surfaces.

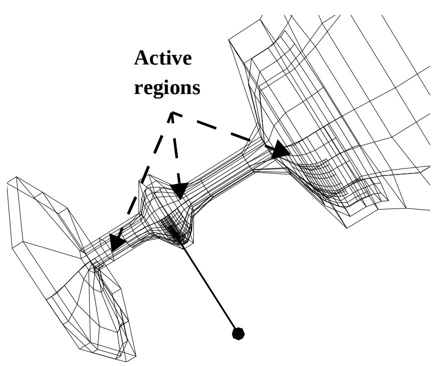

FIGURE 6. Extraction of active regions and surface refinement during pt-surface minimum distance.

Figure 6 demonstrates how the algorithm initially extracts and refines three separate regions on the goblet. As refinement occurs the lower and upper bounds become more accurate and the algorithm prunes the top and bottom portions of the goblet. Finally, the upper bound distance converges to the exact minimum distance on the stem of the goblet, stopping the computation.

3.5.1 Improving Worst-case Convergence

In a tensor-product surface preferred parametric directions exist, namely along isoparametric lines, and active regions that fall along these directions refine most efficiently. Certain spatial configurations of models can produce contiguous *active pairs* that group into an active region that falls

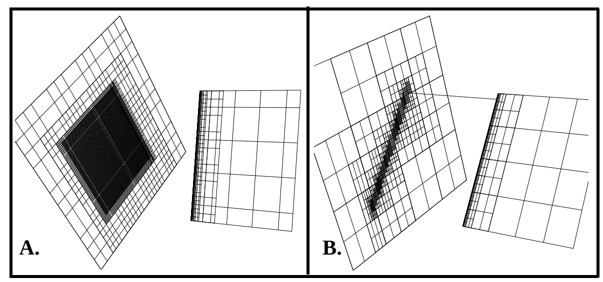

FIGURE 7. A. In this degenerate case, unnecessary refinement occurs. B. Convergence is improved with the notion of *granularity of regions*.

diagonally in parameter space (Figure 7.A). We introduce the notion of *granularity of regions* to reduce unnecessary refinement in these situations. By extracting active regions containing only a limited number of polynomial pieces per region we improve convergence. Using *granularity of regions*, a long diagonal in parameter space is broken into many small surfaces which quickly approximate the solution and avoid refinement in unwanted regions (Figure 7.B).

3.5.2 Performance

Even though the refinement only doubles the number of polynomial pieces at each iteration, for B-splines, the control mesh converges quadratically[6] to the surface. Thus, the upper bound converges in only a few iterations through the *active pair* list.

When running the closest point to surface algorithm we obtain a parametric precision of 10^{-6} at speeds of 10-50Hz. This compares well to the times of around one second we found using a numerical method (Section 2.3). The minimum distance between the cup and spiral (Figure 1) converged at 2-20Hz depending on the distance and their orientations. We contrast these rates to times from one second to several minutes reported in the literature (Section 2.3.2). Our times were measured on an SGI Indigo2.

3.5.3 High-order Surfaces

Earlier, we mentioned our belief that a geometric approach avoids many difficulties associated with more numerical methods. One difficulty that numerical methods have is a lack of stability on high-order surfaces[18]. Using the *LUB-tree* approach we tested a B-spline patch with orders ranging from two to seven in each parametric direction. The algorithm remained stable even for the high-degree surfaces and time to convergence appeared directly proportional to the order in each direction.

3.6 Breadth-first vs. Depth-first Search

We have chosen to search the LUB-tree hierarchy in a breadth-first manner, as opposed to a depth-first search as in Quinlan [24]. In Section 3.6.1, we show that breadth-first search appears to be more efficient than depth-first search. Then we conclude by demonstrating how breadth-first search allows time-critical behavior by establishing upper and lower bounds early in the computation.

3.6.1 Efficiency and Scalability

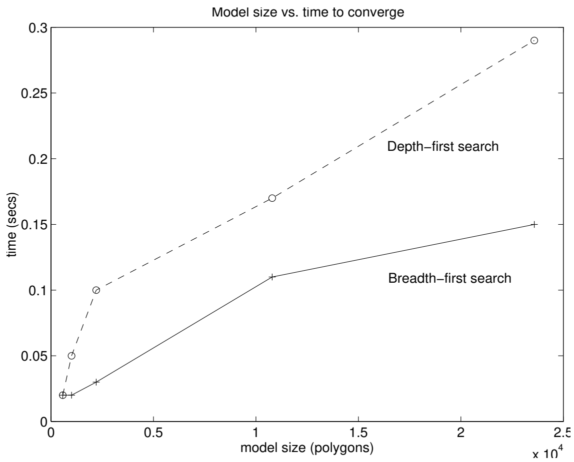

FIGURE 8. Time to convergence for breadth-first and depth-first searches for a range of model sizes.

We tested the time to solution for a range of polygonal model sizes using both the *LUB-tree* approach and a depth-first search similar to Quinlan's. The *LUB-tree* search was approximately twice as fast as the depth-first search and showed sub-linear cost with increases in model complexity. Times for minimum distance between models ranged from 20 ms to 150 ms for models with 575 to 23581 triangles (Figure 8) when running on an SGI Indigo2.

3.6.2 Time-critical Properties

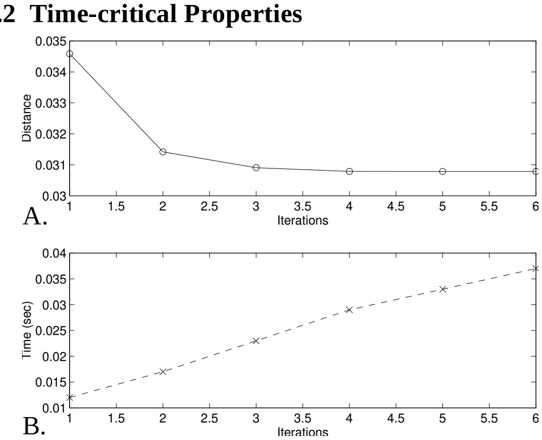

FIGURE 9. A. Minimum distance vs. iterations. B. Total time vs. number of iterations

Our *LUB-tree* methodology has time-critical properties, a useful characteristic in interactive systems. Our method can begin converging immediately to the solution since no large pre-processing step exists and upper and lower bounds get established early in the computation. Figure 9 illustrates this behavior for a parametric surface. The time per iteration is fairly constant, while the error in the distance measure is halved each iteration.

3.6.3 Scheduled Cost

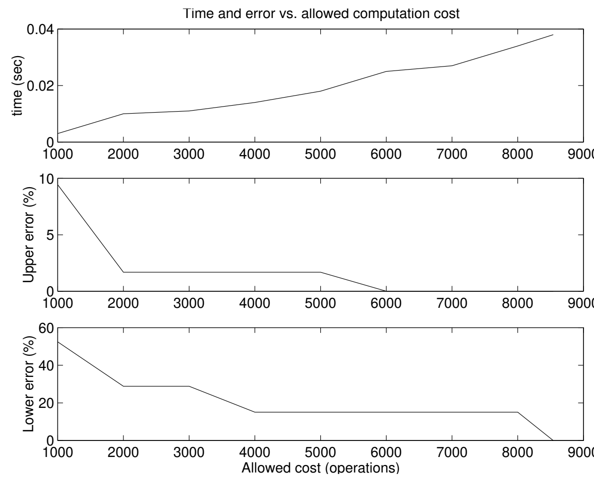

FIGURE 10. LUB-trees allow for scheduled costs with reasonable bounds returned.

We have measured the average time cost for many of the *LUB-tree* operations, such as the time for computing the distance between spheres or the distance between *OBB*s. Using these times, we can schedule an allowed cost for the algorithm and have it return upper and lower bounds on the minimum distance. Figure 10 demonstrates this behavior on a 2000 triangle model. The upper bound distance quickly converges to the correct distance. This suggests that we can use cheaper, approximate solutions with low error. The lower bound distance also converges to within 15% of the correct distance fairly quickly. The error plateaus at the final level of the bounding hierarchy because a lower bound cannot be extracted until the level finishes.

4 Future Work

We have implemented methods for the minimum distance between a point and surface, between surface and surface, and between polyhedral model and polyhedral model, as well for the 2D equivalents. Currently, we are adding operations for finding the distance between models of different surface representations, such as between polygonal and parametric models. In addition, the framework notion supports adding specialized objects with simple distance computations, such as holes made of cylinders. We hope we can improve efficiency for certain classes of models by adding specialized types that occur frequently.

5 Conclusion

We have demonstrated practical computation of distances for non-convex polyhedra and sculptured surfaces within a common framework. The *LUB-tree* method is straightforward to implement and has useful properties such as time-critical behavior and scheduled computation cost.

Acknowledgments

Thanks go to the Alpha_1 group for providing much of the underlying code for this work. Susan Anderson added lots of red ink to preliminary drafts of this paper. Support for this research was provided by NSF Grant MIP-9420352, by DARPA grant F33615-96-C-5621, and by the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219).

References

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

[2]Bobrow, J.E., "Optimal robot path planning using the minimum-time criterion", IEEE Journal of Robotics and Automation, 4(4), pp. 443-450, Aug. 1988.

[3]Cameron, Stephen. "Enhancing GJK: Computing Minimum and Penetration Distances Between Convex Polyhedra", *Int. Conf. Robotics and Automation*, April, 1997.

[4]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.

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

[6]Cohen, E. and Schumaker, L., "Rates Of Convergence Of Control Polygons", in *Computer Aided Geometric Design* 2 (1985), pp. 229-235.

[7]Cohen, J. et al, "I-COLLIDE: An Interactive and exact Collision Detection System For Large-Scaled Environments", *Proceedings of ACM Int. 3D Graphics Conference*, pp. 189-196, 1995.

[8]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

[9]Dokken, Tor, "Finding Intersections of B-spline Represented Geometries Using Recursive Subdivision Techniques", CAGD 2 (1985) pp. 189-195.

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

[11]Gilbert, Elmer, Johnson, Daniel, 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.

[12]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.

[13]Hollerbach, J.M., Cohen, E.C., Thompson, W.B., and Jacobsen, S.C. "Rapid Prototyping of Mechanical Assemblies," *NSF Design and Manufacturing Grantees Conference*, Albuquerque, Jan. 3-5, 1996.

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

[15]Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," The International Journal of Robotics Research, 5(1), pp. 90-98, Spring 1986.

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

[17]Lin, M.C., *Efficient Collision Detection For Animation and Robotics*, Ph.D. thesis, University of California, Berkeley.

[18]Lin, Ming and Manocha, Dinesh, "Fast Contact Determination in Dynamic Environments", to appear in International Journal of Computational Geometry and Applications.

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

[20]Mirtich, Brian, *Impulse-based Dynamic Simulation of Rigid Body Systems*, Ph.D. Thesis, University of California, Berkeley, December, 1996.

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

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

[23]Ponamgi, M., Manocha, D., and Lin, M., "Incremental algorithms for collision detection between solid models", *Proceedings of ACM/SIGGRAPH Symposium on Solid Modeling*, pp. 293-304, 1995.

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

[25]Quinlan, S., *The Real-Time Modification of Collision-Free Paths*, Ph.D. Thesis, Stanford University, 1994.

[26]Sato, Yuichi et al. "Efficient Collision Detection Using Fast Distance-Calculation Algorithms For Convex And Non-Convex Objects", in *Proceedings of the 1996 IEEE International Conference on Robotics And Automation*. Minneapolis, Minn, April, 1996. pp. 771-778.

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

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

[29]Snyder, John et al, "Interval Methods For Multi-Point Collisions Between Time-Dependent Curved Surfaces", Computer Graphics, 27(2). pp. 321-334, Aug. 1993

[30]Thompson II, T.V., Johnson, D.E., and Cohen, E.C. "Direct Haptic Rendering Of Sculptured Models", in *Proc. 1997 Symposium on Interactive 3D Graphics*, (Providence, RI), April 1997, pp. 167-176.

[31]Tornero, J., Hamlin, J., and Kelley, R., "Spherical-Object Representation and Fast Distance Computation For Robotic Applications", *Proceedings of the 1991 IEEE Int. Conf. on Robotics and Automation*, April 1991. pp. 1602-1608.

[32]Stewart, Paul, et al, "CAD Data Representations For Haptic Virtual Prototyping", *Proceedings of DETC'97*, 1997 ASME Design Engineering Technical Conferences, Sept. 14-17, 1997, Sacramento, California.

[33]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.