Bound Coherence for Minimum Distance Computations

David E. Johnson and Elaine Cohen

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

Abstract

Minimum distance computations on complex geometry commonly employ hierarchies of bounding volumes that are pruned through establishment of upper and lower bounds. In this paper we describe a novel time coherence scheme which utilizes coherence on these bounds, rather than coherence derived from the geometry of the bounding volumes or from the geometry of the underlying model. This method is thus independent of the bounding volume and model representations. In tests, we find using this approach can more than double the speed of a non-coherent approach.

1 Introduction

In this paper we describe a novel time coherence scheme for minimum distance computations which provides a several time speed-up over a previous non-coherent approach [9]. We are interested in minimum distance as a predictor of contact in a haptic assembly project [8]. Others have used minimum distance computations for robotic path planning [2], collision avoidance [10], and collision detection [5][1].

For complex concave geometries a useful approach to compute the minimum distance between models has been to surround the models in the scene with hierarchies of bounding volumes [14][9]. These volumes can be pruned by establishing and refining an upper bound on the minimum distance between two models, then removing bounding volumes that have lower bounds on their distance greater than the current upper bound. As the algorithm descends the bounding volume hierarchies, the lower bounds tighten to the underlying geometry. In the best case, we compute the minimum distance between just a few leaf node geometries.

In a scene composed of moving models, the minimum distance between objects will not change significantly after a small time step. For convex objects the geometric location of the closest points between two models will also not change rapidly. On concave models though, the closest points may jump discontinuously, even if the distance between models changes smoothly. Our approach avoids the problems associated with these discontinuities by using coherence on the distance bounds rather than coherence based on geometric properties. This approach is thus also independent of the model and bounding volume representations.

In interactive applications, the scene must be updated at interactive rates. In haptic applications, the contact algorithm must run in the kHz range. These types of applications provide a great deal of temporal coherence and are well suited for this approach.

2 Background

There is considerable literature on minimum distance computations for convex objects. The temporal coherence in tracking the closest points between two convex objects derives from the geometric coherence of convex objects. These closest point tracking methods are competitive with pure collision detection schemes for efficiency.

Lin showed how the closest points on a convex object could be tracked in constant time [11]. This was later extended into the I-COLLIDE system [5]. Gilbert's algorithm for computing distance between convex objects [6] was later shown to have a similar constant time complexity with minor modification [3]. Turnbull has extended this approach to convex spline patches [17].

Rather than explicitly tracking closest points, an additional way to exploit time coherence for convex objects is through witness planes. A plane that separates two objects can be efficiently computed and offers quick testing of separation on following time steps. Baraff [1] and Chung [4] used this concept in collision algorithms.

Two main approaches exist to compute collision and distance for general concave objects. The first approach draws upon the work done for convex objects by attempting to decompose the concave object into convex partitions [5]. These methods do not usually provide the minimum distance, only the collision status. The second approach is to prune portions of the geometry using hierarchies of bounding volumes. Quinlan introduced this approach using a depth-first search of the bounding hierarchy [14]. Johnson generalized it to other surface representations, including NURBS, by using a breadth-first search [9].

Sato combined convex tracking at a distance with Quinlan's concave algorithm for near contact to create a general collision detection system [15]. He stored the closest pair of triangles for a number of time steps and used the distance between that pair as a way to quickly approximate the minimum distance.

Snyder used time coherence to track penetration depth on spline models in a modeling system [16]. Numerical methods quickly updated previous points to find new penetration depths. In [12] Lin used local updates on global numerical methods to exploit the temporal coherence of moving curved surfaces. These methods depend on the coherence of the underlying geometry for success.

3 The LUB-Tree Framework

In [9], we discussed using a lower-upper bound (LUB) tree for minimum distance computations. We can compute the minimum distance to any surface representation that implements a set of operations: *bounding volume generation, lower bound on distance computation, upper bound on model minimum distance computation, bounding volume refinement*, and *computation termination*. These bounding volumes and bound computations are then used to efficiently prune portions of the geometry and to converge to the minimum distance.

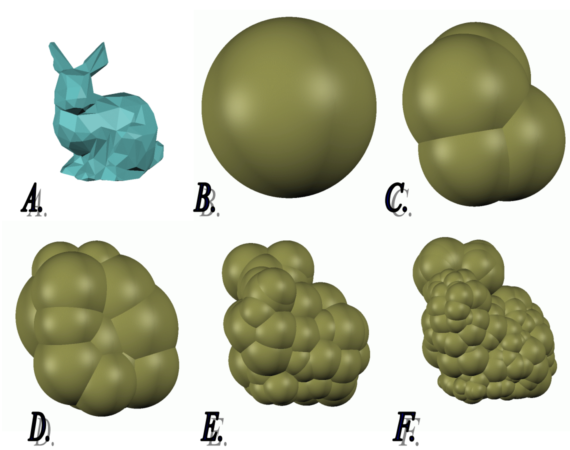

Figure 1: For polygonal models, we pre-compute a bounding hierarchy. Only the spheres of some levels are shown.

We illustrate the action of the LUB-tree framework on general triangular models. We first pre-compute a hierarchy of sphere and oriented bounding box (OBB) bounding volumes for each model using the publicly available OBBTree package [7]. Portions of the sphere hierarchy for an example model are shown in Figure 1. The leaf nodes of the hierarchy surround a single triangle.

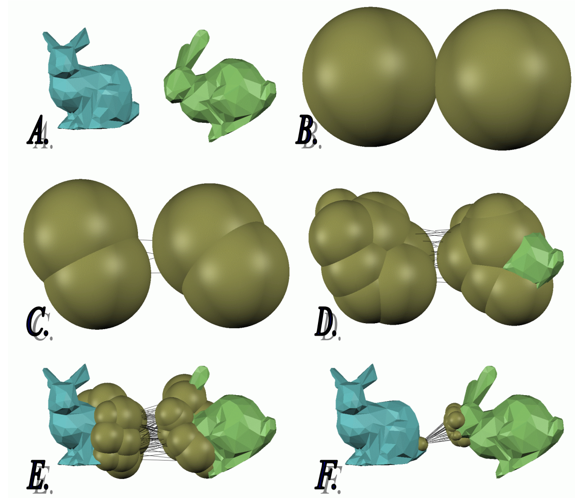

Figure 2: The hierarchies are pruned using bounds on the distance. Surviving portions of the sphere hierarchy are shown at selected levels as well as the active pair *lines.*

The pruning method starts by invoking the *bounding volume generation* operation on each of the two models. In our example, we use the top sphere of each pre-computed bounding hierarchy (see Figure 2B). 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. The lines between the spheres in Figure 2C represent the *active pairs* at the second level in the minimum distance computation. 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 further 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 operation 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 of the model from consideration without high computational cost.

In Figure 2, the bounding spheres are pruned away until only one sphere remains on one model and a few remain on the other. There were twelve levels in the hierarchical computation; five are shown. The closest points between the models must be on the triangles contained by these surviving nodes at the lowest level.

4 Using Bound Coherence to Speed Minimum Distance Computations

These pruning computations change relatively slowly as objects in the scene move. Portions of a model close to portions of another other model will remain roughly in the same configuration from one instant in time to the next. We have identified two areas in which to exploit this coherence: coherence in the upper bound on the minimum distance and coherence in the lower bounds on distance between the bounding volumes.

4.1 Upper Bound Coherence

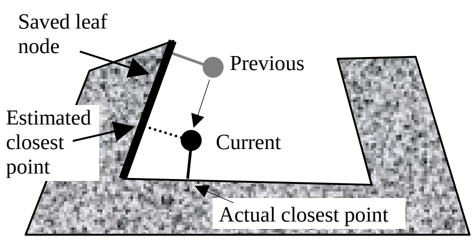

Figure 3: The minimum distance changes smoothly even when the closest point jumps.

The minimum distance between two objects changes smoothly as the objects move, even if the closest points between the models change discontinuously. In Figure 3, the distance from the space point to the concave model changes slowly, but the closest point on the model jumps from the left wall to the bottom.

We save the leaf nodes that produced the closest points from the previous time step and compute the distance between these nodes at the start of the computation at the new time step (the "Estimated closest point" in Figure 3). This distance provides a tight initial upper bound on the minimum distance between the two models. By having a good estimate of the minimum distance, the pruning algorithm may be able to remove some active pairs earlier in the hierarchy than if it were using a loose upper bound estimate.

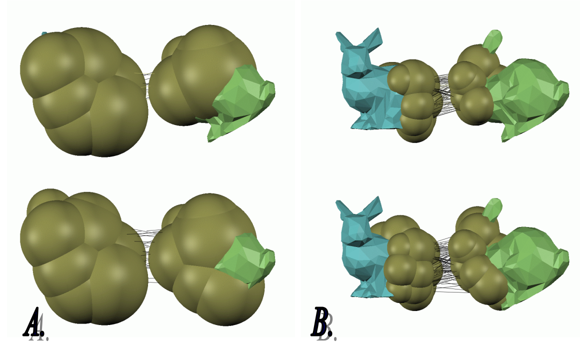

Figure 4: The tops of these images show two different levels of pruning when using upper bound coherence. The bottoms of the images show the same levels with no coherence used. Saving the previous solution allows additional active pairs *to be pruned.*

Figure 4 shows two levels of the pruning process, comparing pruning with upper bound coherence used (upper images) to pruning with no coherence (lower images). In Figure 4A, the fourth level of pruning, there were 14 sphere nodes and 27 *active pairs* in the non-coherent version. The upper bound coherent version reduced that to 10 spheres and 16 *active pairs*.

This upper bound coherence has the greatest impact at the middle of the computation. Initially, the bounding boxes tend to be so large that even an improved upper bound cannot prune them. At the end of the computation, even the non-coherent version would have converged to a reasonable upper bound by then.

4.2 Lower Bound Coherence

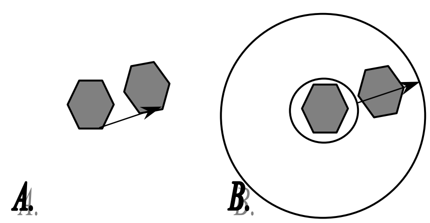

Figure 5: A. We can estimate how far an object has moved. B. The lower bound can expand by that amount and still contain the object (B).

The key to the pruning algorithm is that we may throw away active pairs that have lower bounds on distance greater than the current upper bound on the minimum distance. If the models do not move far during a time step these lower bounds should not change much either.

We can think of this problem as one model being stationary and the other moving without loss of generality. As one model moves, there is a point on the model that moves a maximum distance during the transformation (Figure 5A). Since the purpose of the lower bounds is to show that the contained geometry must be at least the lower bound distance away, if the surface moves, then the bounds become that much more uncertain. The new bound is just the old minus the length of the maximum movement (Figure 5B).

We estimate the maximum movement of a general model by surrounding the model with a bounding box. The maximum movement of the box will be at its vertices. We measure the movement of each correlated vertex from one time to the next and use the largest movement as an overestimate of the model movement and thus, of the possible change in the lower bound distance.

If we have stored the lower bounds for the *active pairs* from the previous time we can loosen these bounds by the movement distance and still be sure the geometry is contained. Using this concept, we can prune saved *active pairs* by retrieving their saved lower bound and subtracting the movement amount, then comparing this value to the upper bound. So we may potentially prune an *active pair* for the cost of retrieval, a subtraction, and a comparison.

Storing the lower bounds for each *active pair* used in a minimum distance computation may require significant storage. Luckily, not every *active pair* holds useful information. If a *pair* survives a level and is refined, then loosening its lower bound the next time is not likely to prune it. So we only save the *active pairs* that result in pruning. The saved *active pair* distances are stored in a hash table, keyed by the ID of the *active pair*. This allows efficient storing and retrieval.

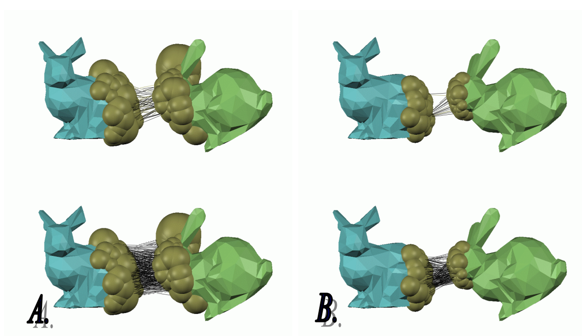

Figure 6: Using lower bound coherence allows us to quickly remove active pairs *without having to compute distances. The top halves of the figures show* the active pairs *computed using coherence; the bottom halves show the* active pairs *without coherence.*

The example in Figure 6 shows many fewer *active pair* lines in the lower bound coherent version than in the non-coherent version. The missing lines were removed from consideration by the quick lower bound coherence test.

5 Testing

We tested the efficacy of using bound coherence in two ways. First, we looked in detail at the pruning costs using the different coherence methods. Second, we made a simple path for one model to follow and tested average computation times for different amounts of movement.

5.1 Pruning Costs

	Non-leaf	**Leaf**	**box**	**Quick**
None	1382	315	371	0
Upper	1167	315	340	0
Lower	521	10	112	1242
Both	458	10	105	1089

Table 1: This table shows the number of pruning tests vs. the type of coherence used in the ideal case (no movement). The table shows the number of non-leaf distance tests, leaf distance tests, OBB tests, and quick tests used in lower bound coherence pruning.

We first measured the number of pruning tests needed while using the different coherence speedups. This test was done under ideal conditions, with no model movement, so these numbers were the best that could occur.

With upper bound coherence turned on, we found that there were moderate gains in efficiency in the number of non-leaf tests. There were no gains in the leaf case. This is to be expected, since the upper bound has typically converged close to the solution by the time the leaf nodes are reached, even with no coherence used.

Using lower bound coherence produced dramatic reductions in the number of both the leaf and non-leaf tests. Since this is the ideal case, we expect that every pruned active pair can be pruned the following time using the quick lower bound coherence test. If the model were moving, we would not expect to see such high gains in efficiency.

Using both types of coherence simultaneously added a modest benefit over the lower bound coherence test.

5.2 Moving Test

This testing under ideal conditions showed the potential value of the coherence methods. We tested the different coherence speedups under more typical conditions by moving one model a number of steps along a space curve and performing a minimum distance computation at each step. The distance traveled at each step is roughly inversely proportional to the number of samples taken along the curve. We hope to show more efficiency with more samples --- and thus more coherence --- along the curve.

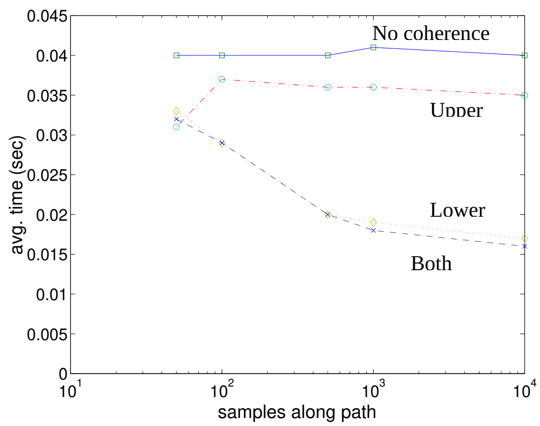

Figure 7: Average time per computation versus number of samples along the space curve. Data was taken with no coherence, with upper bound coherence, with lower bound coherence, and with both upper and lower coherence used simultaneously.

Figure 7 shows the results of moving a 1000 triangle bunny model at different step sizes and with different types of coherence used. The tests were run on an SGI Indigo2 workstation with a 195 MHz MIPS R10000 processor. With no coherence, the average time was essentially constant, at about 0.04 seconds. Using the upper bound coherence gave an almost constant improvement of 0.005 seconds, for an average of 0.035 seconds. We suspect that the constant improvement is related to the way the upper coherence method works. Since we store the leaf nodes that held the minimum distance from the previous time and re-compute their distance at the new time step, the method somewhat adjusts for the size of step taken.

The efficiency of the lower bound method was strongly tied to the size of the step. Initially, the lower bound method was slightly worse than the upper bound method, but improved to be over two times faster than the upper bound method alone. The combination of the two methods added additional benefit.

We repeated the test for a bounding hierarchy made up only of spheres, instead of the sphere and oriented-bounding box combination used in the other tests. The sphere-only hierarchy was slower, almost twice as slow under all conditions, but showed improvement with coherence used.

5.3 Scalability

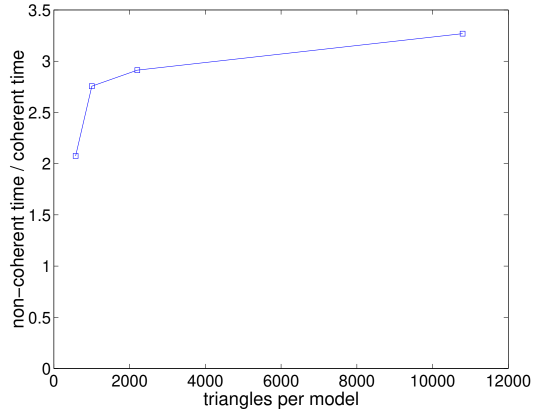

Figure 8: The gain in efficiency increases as the model gets larger. Shown is the ratio non-coherent time vs. upper and lower coherency time for the path sampled 1000 times. Models with 575 to 10793 triangles were tested.

We tested a range of 575 to 10793 triangles per model on the same path. As the models grew larger, the gain in efficiency from using coherency grew as well. Figure 8 graphs the ratio of the non-coherent to coherent times for the path test. With the largest model, the coherent method gained over a three-fold improvement in speed. This demonstrates excellent scalability for using coherency in minimum distance computations.

6 Discussion

Exploiting the bound coherence over time yields significant efficiencies. Additionally, the non-coherent minimum distance computation is easily extended to take advantage of bound coherence.

For the upper bound coherence, we simply had to statically allocate an active pair that stored the leaf nodes containing the closest points. An additional function parameter specified whether to use this stored pair the following time, requiring an additional call to the leaf node distance procedure. In all, this added six lines of C++ code.

The lower bound coherence method used already existing hash table code for storage of the pruned active pairs. We had to add a function parameter for distance moved, check for saved active pairs during the refinement stage, perform the quick prune test based on distance moved, and prep and clean the saved list for a total of ten lines of C++ code plus the hash table.

Using the bound coherence over time opens the possibility of experimenting with tighter, more expensive bounding volumes. If a distance computation is amortized over several time steps with lower bound coherence, then a tighter bounding volume may stay useful longer than a less-expensive, looser bounding volume for an overall gain in efficiency.

7 Conclusion

We have shown that temporal coherence can be efficiently exploited in current minimum distance computations. By utilizing coherence in the bound computations, we can maintain a distance framework suitable for different surface and bounding volume representations.

8 Acknowledgements

We would like to thank the members of Utah's Virtual Prototyping work for inspiring this work. 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).

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Johnson, David E. and Cohen, Elaine, "A framework for efficient minimum distance computations," *Proc. IEEE Intl. Conf. Robotics & Automation*, Leuven, Belgium, May 16-21, 1998, pp. 3678-3684.

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

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

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

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

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

[15] 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.

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

[17] Turnbull, Colin and Cameron, Stephen, "Computing Distances Between NURBS-defined Convex Objects", *Proc. IEEE Intl. Conf. Robotics & Automation*, Leuven, Belgium, May 16-21, 1998.