Proceedings of DETC2004:
2004 ASME Design Engineering Technical Conferences
September 28-October 2, 2004, Salt Lake City, Utah USA

DETC2004/DAC-57461

UNIFIED DISTANCE QUERIES IN A HETEROGENEOUS MODEL ENVIRONMENT

David E. Johnson
School of Computing, University of Utah
Salt Lake City, USA
dejohnso@cs.utah.edu

Elaine Cohen
School of Computing, University of Utah
Salt Lake City, USA
cohen@cs.utah.edu

ABSTRACT

Computing the minimum distance between two models in a virtual scene is a fundamental operation useful in simulation, path planning, haptics, and modeling. In an environment with heterogeneous model representations, distance functions can be difficult to formulate and may require multiple specialized methods. In this paper, we demonstrate a generalized method for finding the distance between models with different representations and demonstrate it on a variety of models.

INTRODUCTION

Efficient computation of the minimum distance between two virtual models is a needed and useful operation in many applications, such as computer simulation[1], robotic path planning[2], and haptics[10]. Typically, these tasks must occur in a homogeneous model environment, so that the minimum distance function can be specialized for the model representation used. However, one can imagine wanting to do tasks such as path planning on a NURBS model as it moves through a polygonal environment, or even through a point cloud environment taken directly from a 3D laser scan. These types of tasks are restricted in current model environments, yet we believe the need for heterogeneous interactions is growing as new model acquisition technologies become available and new simulation capabilities are developed.

For each type of model representation supported by the environment, minimum distance functions from that model to all the other model types must be written, an approach that quickly leads to a large number of specialized functions. Furthermore, it is not clear how some distance functions should even be formulated, such as between an unstructured triangle model and a NURBS surface.

In this paper, we develop and demonstrate a generalized distance algorithm that works between any two types of models. The models themselves just have to support a single, simpler distance operation, so the number of distance functions becomes linear in the number of model representations. This generalized distance function can handle a wide variety of model types, as shown in Fig. 1, and provides good computational efficiency. This computational capability is important to support realistic tasks in the mixed-model simulation environments emerging today.

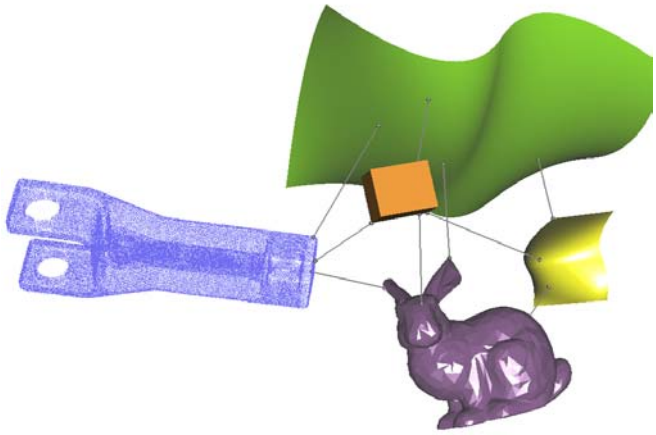

Figure 1: MINIMUM DISTANCES BETWEEN SPLINE, POLYGONAL, CSG PRIMITIVE AND POINT CLOUD MODEL REPRESENTATIONS.

BACKGROUND

Prior research into distance algorithms typically has relied upon different approaches depending on the model types being queried. Distance algorithms for polygonal models have favored geometric pruning methods, while algorithms for parametric surfaces have focused on numerical techniques.

Another way of classifying prior work is by the type of distance query they support. The distance query can return the global minimum between the models, or it can find a local minimum. A local minimum distance query is useful in

temporally coherent applications and usually provides speed advantages over global minimum methods.

The following subsections describe a few prevalent model representations as well as the minimum distance approaches for those representations in more detail.

Distance Queries for Polygonal Models

Polygonal models are typically composed of collections of triangles, and most distance algorithms for polygonal models deal with triangle primitives. The model may or may not contain topological connectivity information. Models without connectivity are known as a triangle cloud, and ones with are properly described as a triangle mesh.

Lin[3] and Gilbert, Johnson, and Keerthi[4] developed fast minimum distance methods for convex polygonal models. Since local gradient search produces a global minimum for convex objects, their algorithms can converge quickly.

Quinlan[5] developed a spherical bounding hierarchy for general triangle clouds. The bounding hierarchy was used to determine an upper bound on minimum distance between the two models, and then to prune away portions of each model with lower bounds on distance larger than the upper bound.

The PQP package, by Larsen et al.[6], followed the successful application of oriented bounding boxes to collision detection[7] by using swept sphere volumes as a bounding hierarchy for triangle clouds. These volumes can control their aspect ratio to more tightly bound contained geometry than sphere bounds, which provided faster distance queries.

More recently, the distance methods for convex model distance queries have been applied to convex decompositions of triangular models[8]. Essentially, this method reduces the number of leaf nodes by replacing triangles with convex collections.

A different approach is taken by [9], which finds local minima between triangular meshes by pruning their hierarchies based on collinearity conditions, rather than distance bounds. This approach has been extended in [10] to use local gradient searches to update local closest point pairs in a haptic virtual prototyping application.

In the methods for general polygonal models, the predominant technique is to create a hierarchy of bounding volumes, and the advancements have come mostly from improving the tightness of the bounding volumes. This approach differs markedly from techniques used for parametric models.

Parametric Models

Parametric models are composed of smooth surface patches, and typical models have fewer primitives than polygonal models. The emphasis in research, then, has not been on efficient means of pruning large numbers of primitives. Instead, methods have explored various numerical methods for quickly and reliably solving systems of equations that describe minimum distance conditions between two parametric models.

The distance between two parametric models $\mathbf{F}(u,v)$ and $\mathbf{G}(s,t)$ is the minimum magnitude of the vector difference

$$D(u,v,s,t) = \left\| \mathbf{F}(u,v) - \mathbf{G}(s,t) \right\|. \tag{1}$$

The magnitude squared function $D^2(u,v,s,t)$ avoids the square root, and extrema of that function occur at coincident zeros from the set of its partial differentials, as in

$$\begin{aligned} (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_u &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{F}_v &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_s &= 0 \\ (\mathbf{F}-\mathbf{G})\cdot\mathbf{G}_t &= 0 \end{aligned}. \tag{2}$$

The system of equations for distance extrema have also been variously defined as sets of cross-products[11] or augmented with explicit normal collinearity conditions[12].

These extrema conditions have been solved by symbolic computation[12], interval methods[13], and Newton-Rapheson iteration[14]. This last technique has the advantage of high speed and rapid convergence, and has been a practical choice for many implementers.

In [15], NURBS surfaces were treated as geometric entities, and a hierarchical pruning approach provided a reasonable tradeoff between speed and robustness. Furthermore, by sharing a common set of operations with polygonal distance finding, it created a framework for unified minimum distance computations. However, this unified approach was not fully developed.

Limaiem[16] showed that alternating orthogonal projection from a point on one surface to another would eventually yield an intersection between the two surfaces. We build on this idea as an approach for a unified minimum distance function.

Other Representations

Other model representations are also used as they provide unique capabilities. Implicit models provide easy intersections and Boolean operations. Since implicit models are the zero set of a scalar function, evaluating that function at a point in space is itself a type of distance function, even though it is not an exact match to Euclidian distance. Ensz et al. has a discussion of implicit functions that do provide a corresponding Euclidean distance in the context of solid modeling[17].

CSG models are typically combinations of primitives, such as spheres, cones, cubes, and tori. Each primitive needs a custom distance function. Most CSG models are not made up of large numbers of primitives, so efficient pruning methods aren't needed.

A representation which recently has become prevalent is the point cloud, derived from laser scanning of physical models or environments. Large number of points make up a detailed model. Although there hasn't been much effort at developing efficient distance methods specifically for point clouds, polygonal approaches are adaptable to this model representation.

Discussion

While there have been approaches for minimum distance methods between two objects of the same representation, or in the same family, there has not been much effort in developing general distance methods. This is probably due to the very distinct approaches taken for the different representations, and the difficulty in merging these styles.

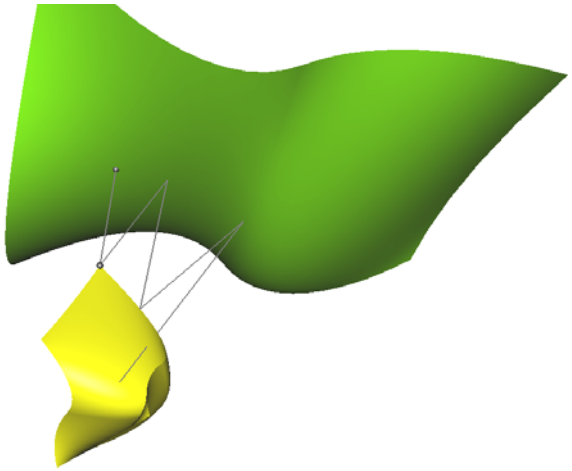

Figure 2: BY ALTERNATING POINT-TO-MODEL MINIMUM DISTANCE QUERIES, THE DISTANCE CONVERGES TO A MODEL-MODEL MINIMUM DISTANCE.

A UNIFIED DISTANCE APPROACH

Our approach is to use algorithms that compute the distance from a point to a model as a basis for heterogeneous model to model distance functions. This permits current simulation and analysis techniques, which now only work on similar representation models, to be applied to models of different representations, such as models derived from different processes. An object-oriented implementation allows these point-model distance algorithms to be combined in a single generic model-model distance function to compute the minimum distance between two models of different representations.

A Generic Model-Model Distance Function

The generic model-model distance function is based on the observation that a local minimum in distance between two models can be found by

1. finding the distance from a query point on the first model to the second model,
2. using that closest point as a new query point and finding the closest point to it back on the first model,
3. repeating this process until the minimum distance converges.

Currently, we initialize this search by finding the closest points between the two oriented bounding boxes containing the models. The search uses the closest point on the first model's bounding box in a point-to-model distance query with the second model. That query returns a point on the second model, which can then be used in the iteration described above. This heuristic for initializing the process works robustly, but there may be other possibilities, such as using multiple initial points, that perform better.

Figure 2 demonstrates this process on two spline models. The alternating closest points are connected by lines showing the convergence of the method. This approach is essentially a point-sampled gradient search in the distance space of the two models, rather than the more common gradient search on the surfaces of the models.

An Object-Oriented Implementation

This approach yields a clean implementation in a heterogeneous model environment. For each type of model representation, a point to model distance function needs to implemented. Then, a generic model-model superclass function can call these specific implementations. Alternately, if the models are not derived from a common class object, template programming can be used to achieve the same result. A pseudo-code implementation of the superclass approach would look like the following.

```
class geometry_object {
   real model_model_distance( geometry_object m1,
                              geometry_object m2)
}
```

Sample code for some model representations would contain

```
class polygon_model : geometry_object {
   point point_model_distance( point query )
}

class spline_model : geometry_object {
   point point_model_distance( point query )
}

class CSG_model : geometry_object {
   point point_model_distance( point query )
}
```

Since all the representations inherit from geometry_object, the geometry_object model_model_distance method calls the consistently named point_model_distance methods for each representation being queried. Each of those functions only has to know about its own representation, so specialized distance functions accounting for all possible model-model interactions aren't needed. C++-style pseudo-code for the generic model-model distance method looks like

```
point
geometry_object::model_model_distance(
                              geometry_object m1,
                              geometry_object m2) {
    point pt2, pt1 =  m1.closest_pts_bbox(m2);    // initialize
    real last_distance = INFINITY,
         distance = -1.0;
    bool first = true;              // model are we querying
    while ( distance != last_distance) {
        if (first)
            pt2 = m2.point_model_distance( pt1 );
        else
            pt1 = m1.point_model_distance( pt2 );
        last_distance = distance;
        distance = pt1.distance( pt2 );
        first = !first;                       // switch models
    }
```

where models m1 and m2 can be any model representation that derives from geometry_object and has a point_model_distance method.

Local Distance

This technique does not guarantee a global minimum, although the result often is a global minimum. The result is

properly characterized as a local minimum distance. Most local methods do a local gradient descent in the regions around the initial estimated minimum distance points, which constrains the result to be in those regions as well. However, unlike these local minimum distance functions, our method can "jump" regions on the models. This because the gradient search is not along the surfaces of the models, but instead in the distance space of the two models.

Convergence

The method converges to a local minimum because each alternating step must yield a point-to-model distance that is closer or the same as the previous step. The prior step provides an upper bound on the minimum distance between the two models, with proof by existence from the query point on the one model and its closest point on the other. The method stops because there is a minimum to distance between models, so eventually the new step must be the same distance as the last and the points are locally closest points. Of course, this argument depends on the convergence of each point to model minimum distance step.

RESULTS

We have implemented the unified distance function approach within our CAD research testbed, Alpha_1. This system supports a number of model representations, for which we implemented point to model minimum distance methods. These point to model methods use a variety of approaches, described in Table 1.

The method reliably converges to a minimum distance between models, even in difficult model configurations. In Fig. 3, the top graph shows slowed convergence as the method traverses a concave portion of one model (see the models in Fig. 2). The lower graph of Fig. 3 shows rapid initial convergence, followed by slower convergence caused from nearly parallel model geometries (Fig. 4). However, in each case, the method still converged in relatively few steps. In the example configuration of Fig. 1, the worst case took twelve steps to converge and the best took three.

We tested the characteristics of this type of search on two polygonal bunny models translated apart from each other. The test randomly locally rotated each model before calling the heterogeneous distance query and a comparison globally convergent distance query. After 1,000 queries, the heterogeneous distance query had the same result 95% of the time. The other tests converged, but to a local minimum rather than the global solution. The results that failed to reach a global minimum were evenly split between results that were a small fraction of the distance off from a global minimum, indicating the iteration became trapped in a small bump near the correct

TABLE I. POINT TO MODEL METHODS

Model Representation	Point to Model Method
NURBS surface	Nodal mapping followed by Newton-Rapheson
Triangle cloud	Linear search
Point cloud	Linear search
CSG primitive	Specialized function

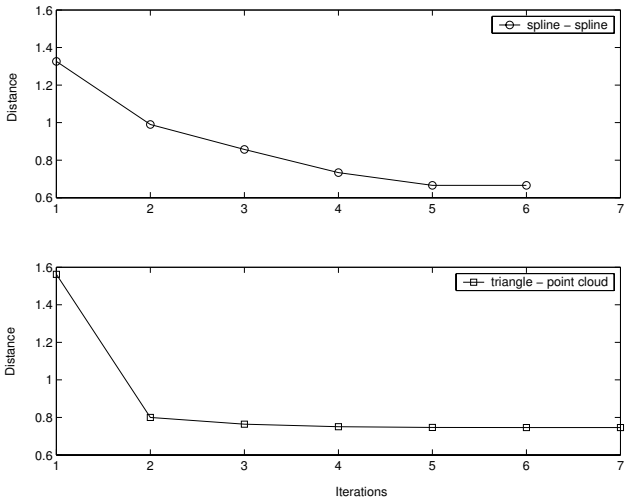

Figure 3: CONVERGENCE RATES FOR TWO DIFFERENT MODEL-MODEL INTERACTIONS. IN THE TOP GRAPH, CONVERGENCE IS SLOWED BY HAVING TO CROSS A CONCAVE PORTION OF ONE MODEL. IN THE SECOND GRAPH, DISTANCE INITIALLY DROPS QUICKLY, BUT THEN SLOWS IN NEARLY COLPLANAR PORTIONS OF THE MODELS. HOWEVER, IN BOTH CASES, ONLY A FEW ITERATIONS ARE NEEDED.

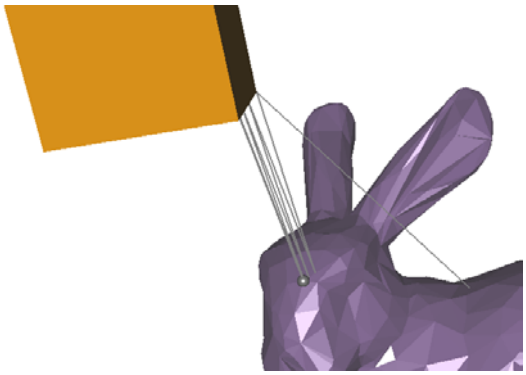

Figure 4: NEARLY PARALLEL GEOMETRIES CAUSES SLOWED CONVERGENCE.

solution, and results that were further off, indicating an iteration that converged on the wrong large-scale feature.

These results compare reasonably to convergence rates of common numerical methods for parametric models. In addition, greater global reliability could be obtained by sampling more initial points, similar to current practice for numerical methods. Additionally, higher rates of global convergence should be obtainable in simulations with coherence, where the last solution initializes the new iteration. We are exploring these issues in further work.

DISCUSSION

We have focused on iteration steps over computation time, because the computation time is highly dependent on the individual point-to-model methods used. The linear search implemented for the triangle cloud and point cloud models means that the times in our system are not competitive with specialized packages such as PQP. However, we can easily replace any of the point-to-model packages with an accelerated technique, and all distance queries to that representation will

get an equivalent boost in speed. This code modularity makes experimentation and piecewise acceleration much easier.

CONCLUSION

Current modeling and VR systems typically use homogeneous model types, or only allow limited interaction between supported types. The approach demonstrated here is very flexibly applied to different model representations. This type of computation is a critical component of heterogeneous model environments that can support new technologies for model acquisition and the need for specialized representations during analysis. Our hope is that it can facilitate new modes of interaction between different types of models without having to resort to model conversion. The unified distance approach can be very competitive with specialized distance functions, as long as the underlying point-to-model functions are optimized.

ACKNOWLEDGMENT

The authors would like to acknowledge support in part from the following grants: NSF DMI9978603 and ARO DAAD 19-01-1-0013.

REFERENCES

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

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

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

[4] E. Gilbert, D. Johnson, S. Keerthi, “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.

[5] S. Quinlan, “Efficient Distance Computation between Non-Convex Objects,” *IEEE Int. Conference on Robotics and Automation*, pp. 3324-3329, 1994.

[6] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha., “Fast Distance Queries using Swept-Sphere Volumes,” *Proc. IEEE International Conference on Robotics and Automation* , 2000.

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

[8] S. Ehmann and M. Lin. “Accurate and Fast Proximkity Queries between Polyhedra using Surface Decomposition,” Computer Graphics Forum (Proc. Eurographics), 2001.

[9] D.E. Johnson and E. Cohen, "Spatialized Normal Cone Hierarchies," in 2001 ACM Symposium on Interactive 3D Graphics, ACM SIGGRAPH, March 2001.

[10] D.E. Johnson, and P. Willemsen, "Six Degree-of-Freedom Haptic Rendering of Complex Polygonal Models," in Proc. *2003 Haptics Symposium* , 2003.

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

[12] M.C. Lin and D. Manocha, “Fast Interference Detection Between Geometric Models,” The Visual Computer, pp. 542-561, 1995.

[13] J. Snyder, “Interval Methods For Multi-Point Collisions Between Time-Dependent Curved Surfaces”, Computer Graphics, 27(2). pp. 321-334, Aug. 1993.

[14] D. Nelson, D.E. Johnson, and E. Cohen, “Haptic Rendering of Surface-to-Surface Sculpted Model Interaction,” in *Proc. 8th Annual Symp. on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, (Nashville, TN), ASME, November 1999.

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

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

[17] M. Ensz, D. Storti, and M. Ganter, “Implicit Methods for Geometry Creation”, Int. Journal of Computational Geometry & applications, Vol. 8, Nos. 5-6 (1998), pp. 509-536.