DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes

Young J. Kim Ming C. Lin Dinesh Manocha
Department of Computer Science
University of North Carolina in Chapel Hill
{youngkim,lin,dm}@cs.unc.edu

Abstract— **We present an incremental algorithm to estimate the penetration depth between convex polytopes in 3D. The algorithm incrementally seeks a "locally optimal solution" by walking on the surface of the Minkowski sums. The surface of the Minkowski sums is computed *implicitly* by constructing a local Gauss map. In practice, the algorithm works well when there is high motion coherence in the environment and is able to compute the optimal solution in most cases.**

Keywords— **Penetration Depth, Minkowski Sums, Incremental Algorithm**

I. Introduction

Computing a distance measure between geometric objects is an important problem in robotics, virtual environments and interactive computer games. When objects are disjoint, the minimum Euclidean distance between them is one of the commonly used distance measure. However when objects are overlapping, the Euclidean distance does not give any useful information related to the extent of intersection or penetration. Therefore a different distance measure is needed to determine the amount of penetration [1].

A number of algorithms have been proposed for computing the Euclidean (or separation) distance between two objects. However, many applications like robot motion planning, dynamic simulation or haptic rendering need to know the extent of penetration between overlapping objects. These include robot motion planning in environments consisting of narrow passages [2], 6 degree of freedom haptic rendering involving object-object interactions [3], [4], [5], contact force computation for dynamic simulation [6], [7].

The natural extension of Euclidean separation distance to overlapping objects is the *penetration depth* (PD) or *intersection depth*. The PD of two interpenetrating objects A and B is defined as the minimum translation distance to make the interiors of A and B disjoint. One of the commonly used metrics for representing and computing PDs is in terms of the Minkowski sum of two objects. In particular, PD corresponds to the minimum distance from the origin of the Minkowski sum of $A \oplus -B$ to the surface of this sum [8]. As a result, commonly used algorithms for PD computation involve computing the Minkowski sum and computing the closest point on its surface from the origin. The worst case complexity of the overall PD algorithm is governed by the complexity of computing Minkowski sums, which can be $O(n^2)$ for convex polytopes and $O(n^6)$ for general (or non-convex) polyhedral models [9].

The exact computation of Minkowski sums can be very expensive for interactive applications, like haptic rendering or real-time simulation. As a result, the recent trend has been on computing a good approximation (or estimate) for penetration depth.

A. Main Results

We present an incremental algorithm to compute PD for convex polytopes in 3D. The algorithm incrementally marches towards a "locally optimal" solution by walking on the surface of the Minkowski sum. We define the locally optimal PD using the features on the *Configuration Space Obstacle* (CSO), defined as $A \oplus -B$, as follows. Let f be a feature on the CSO which corresponds to the locally optimal PD. Then, the distance from the origin to f is always smaller than the distance from the origin to any neighboring feature to f (on the CSO).

The surface of the CSO is implicitly computed by constructing a local Gauss map (or normal diagram) by performing a local walk on the polytopes. It performs incremental computations and exploits spatial and temporal coherence between successive frames. Our algorithm for locally computing the Gauss map follows the similar strategy used by width computation algorithms between convex polytopes.

The resulting algorithm has been implemented and we have tested its performance on a number of benchmarks. In practice, the running time is observed to be "almost constant" – a fraction of a millisecond on a 1 GHz PC, when there is high motion coherence present in the environment. Furthermore, it is able to compute the optimum PD between the underlying polytopes in most cases, whereas the earlier algorithm for estimating PD [8] merely provides the upper and lower bounds. It

This research is supported in part by ARO Contract DAAG55-98-1-0322, DOE ASCII Grant, NSF Grants NSG-9876914, NSF DMI-9900157 and NSF IIS-982167, ONR Contracts N00014-01-1-0067 and N00014-01-1-0496, and Intel

also outperforms a more recent algorithm [10] in terms of accuracy and consistency of the result and running time. For relatively complicated objects, our algorithm runs approximately *six* times faster than [10].

B. Organization

The rest of paper is organized as follows. In Section II, we briefly review the previous work related to penetration depth computation. In Section III, we describe our incremental algorithm in great detail. In section IV, we provide the experimental results of our incremental algorithm, and compare its performance with other algorithms.

II. Previous Work

In this section, we give a brief overview of previous work on collision detection and distance computation, and penetration depth.

A. Collision and Distance Computations

The problems of collision detection and distance computations are well studied in computational geometry, robotics, simulated environments and haptics. Check out [11] for a recent survey.

B. Penetration Depth Computation

Several algorithms have been proposed to compute or estimate the PD. A straight-forward algorithm to compute the PD can be devised by explicitly computing the Minkowski sums in $O(n^2)$ time [1]. Dobkin et al.'s [9] have presented a hierarchical algorithm that computes the directional PD (i.e. the penetration direction is given) in $O(\log n \log m)$ time for two convex polytopes with n and m vertices [9]. Agarwal et al. [12] have presented a randomized approach to compute the PD values in $O(m^{\frac{3}{4}+\epsilon}n^{\frac{3}{4}+\epsilon} + m^{1+\epsilon} + n^{1+\epsilon})$ expected time for any positive constant ϵ.

Given the complexity of optimal penetration depth computation, a few approximate algorithms have been proposed for quick estimation. Cameron [8] has presented an extension to the GJK algorithm for separation distance computation [13] to compute upper and lower bounds on the PD between convex polytopes. Bergen further elaborated this idea in his expanding polytope algorithm [10]. The algorithm iteratively improves the result of the PD computation by expanding a polyhedral approximation of the Minkowski sums of two polytopes.

Some general algorithms for penetration depth estimation are based on discretization of the object space containing the objects. Fisher and Lin [14] have presented a PD estimation algorithm based on the distance field computation and the fast marching level-set method. Hoff et al. [15], [16] have proposed an approach based on graphics hardware and multi-pass rendering for different proximity queries between general rigid and deformable models, including penetration depth estimation.

Other metrics to characterize the intersection between two objects include the *growth distance* defined by Gilbert and Ong [17].

III. Incremental Penetration Depth Computation

In this section, we present our incremental PD computation algorithm for convex polytopes. We also explain how the algorithm avoids a local minimum problem.

A. Width Computation and Penetration Depth

Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in 3D, the *width* of P, $\mathrm{W}(P)$, is defined as the minimum distance between parallel planes supporting P. The width $\mathrm{W}(P)$ of convex polytopes A and B is closely related to the penetration depth $\mathrm{PD}(A, B)$, since it is easy to show that $\mathrm{W}(P) = \mathrm{PD}(P, P)$. Therefore, once a width computation algorithm is readily available, it can be modified to compute the PD.

Our incremental PD algorithm is quite similar to Houle and Toussaint's width computation algorithm [18]. Their main idea is based on the following lemma:

LEMMA III.1 *The width of a set of points P in 3D is the minimum distance between supporting planes, and the plane having the minimum inter-distance (i.e. width) is realized only either by antipodal VF pair or by antipodal EE pair.*

Both Houle and Toussaint's algorithm and ours only search VF[1] and EE antipodal pairs, and seek a witness feature pair respectively, for the width and the PD value. Hence, the main issue in both algorithms becomes finding such VF and EE antipodal pairs. Houle and Toussaint's width algorithm accomplishes it by using the standard dual mapping on the Gauss map (or normal diagram). The mapping is defined from object space to the surface of a unit sphere $\mathbb{S}^2$ as: a vertex is mapped to a region, a face to a point, and an edge to a great arc. Then, the algorithm finds the antipodal pairs by overlaying the upper hemisphere of the Gauss map on the lower hemisphere and computing the intersections between them.

B. Algorithm Overview

In our incremental PD computation algorithm, we do not compute the entire Gauss map for each polytope or their entire Minkowski sum. Rather we compute

[1]Note that we use a regular upper-case letter to denote a general feature (e.g. V, E, F) and use a italic lower-case letter to denote an instantiated particular feature (e.g. v_1, e_1, f_1).

them in a lazy manner based on local walking and optimization. Starting from some feature on the surface of the Minkowski sum, the algorithm finds the direction in which it can decrease the PD value and proceeds towards that direction by extending the surface of the Minkowski sum.

At each iteration of the algorithm a vertex is chosen from each polytope to form a pair. We label it as a *vertex hub pair* and use it as a hub of the expansion of the local Minkowski sum. The vertex hub pair is chosen in such a way that there exists a plane supporting each polytope, and it is incident on each vertex. It turns out that the the vertex hub pair corresponds to two intersected convex regions on a Gauss map, which later become intersecting convex polygons on the plane after *central projection*. The intersection of convex polygons correspond to the VF or EE antipodal pairs from which one can reconstruct the local surface of the Minkowski sum around the vertex hub pair. Given these pairs, we choose the one that corresponds to the shortest distance from the origin of the Minkowski sum to their surface. If this pair decreases the estimated PD value, we update the current vertex hub pair to an appropriate one which is adjacent to the chosen antipodal pair. We iterate this procedure until we can not decrease the current PD value and converge to a local minima. The details of the algorithm are given below.

C. Initialization

The algorithm starts with an initial guess on the vertex hub pair[2] (**VV**). The initial guess is important for the performance of our incremental algorithm. A good initial guess can lead to empirically "almost constant" running time, whereas a bad one can lead to $O(n^2)$ running time in the worst case, where n is the number of features in each polytope. There are many plausible strategies to pick a good initial guess and some of them are application dependent. The goal is to estimate the optimal penetration direction. Once the estimated direction is computed, we take the extremal vertex of each polytope along that direction and use it to form the vertex hub pair.

A good estimate to the penetration direction can be obtained by taking the centroid difference between objects, and computing an extremal vertex pair for the difference direction. For instance, in Figure 1-(a), c_1 and c_2 are the centroids of each object. The extremal vertex pair along the directions of $c_2 - c_1$ and $c_1 - c_2$ is chosen from each object, and is assigned as an initial vertex hub pair. This technique is known to work well for initial guess on closest features for Voronoi Marching based separation distance computation algorithm [19],

[2]We distinguish the vertex hub pair by using bold-faced letters on each polytope (a region/region pair on the Gauss map)

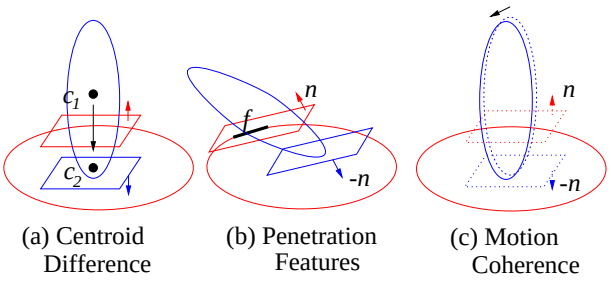


Fig. 1. Various Initial Guessing Strategies

and also works well for estimating the penetration direction.

In other cases, the penetrating features can also suggest a good initial guess. When the objects penetrate, many of the proximity query algorithms report a witness feature pair for it [19], [20]. From this feature pair, one might be able to estimate the direction or compute the actual feature pair that corresponds to the optimal PD. One possible way is to consider the plane normal of a penetration feature as penetration direction. For instance, in Figure 1-(b), the face f is identified as a penetration witness, and its associated plane normal n is used for the extremal vertex query.

Many applications exhibit high spatial or temporal coherence between successive frames. In such environments with high motion coherence, the PD computation result from the previous time frame can provide a good guess for the next time frame. In Figure 1-(c), for example, the previous PD features[3] provide a direction for the extremal vertex query.

D. Iterative Optimization

After the algorithm obtains a initial guess for a **VV** pair, it iteratively seeks to improve the PD estimate by jumping from one **VV** pair to an adjacent **VV** pair. This is accomplished by looking around the neighborhood of the current **VV** pair and walking to a pair which provides a greatest improvement in the PD value. In more detail, let the current vertex hub pair be $\boldsymbol{v_1 v_1'}$. The next vertex hub pair $\boldsymbol{v_2 v_2'}$ is computed as follows:

1. Construct a local Gauss map each for v_1 and v_1',
2. Project the Gauss maps onto $z = 1$ plane, and call them G and G' respectively. G and G' correspond to convex polygons in 2D.
3. Compute the intersection between G and G' using a linear time algorithm such as [21]. The result is a convex polygon and call each vertex comprising the intersection u_i. If u_i is an original vertex of G or G', it corresponds to the VF antipodal pair in object space. Otherwise, it corresponds the EE antipodal pair.

[3]By the PD feature, we mean a pair of features on both polytopes whose supporting planes realize the locally optimal PD value.

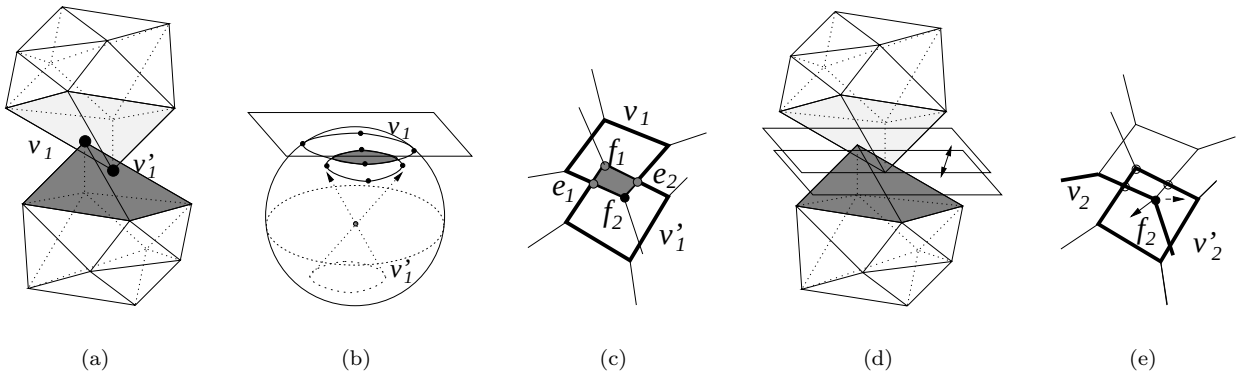


Fig. 2. Iterative Optimization: (a) The current **VV** pair is $\boldsymbol{v_1 v_1'}$ and a shaded region represents edges and faces incident to $\boldsymbol{v_1 v_1'}$. (b) shows local Gauss maps and their overlay for $\boldsymbol{v_1 v_1'}$. (c) shows the result of the overlay after central projection onto a plane. Here, f_1, e_1, f_2 and e_2 comprise vertices (candidate PD features) of the overlay. (d) illustrates how to compute the PD for the candidate PD features in object space. (e) f_2 is chosen as the next PD feature, thus $\boldsymbol{v_2 v_2'}$ is determined as the next vertex hub pair.

4. In object space, determine which u_i corresponds to the best local improvement in PD, and set an adjacent vertex pair (adjacent to u_i) to $\boldsymbol{v_2 v_2'}$.

This iteration is repeated until either there is no more improvement in the PD value or number of iterations reach some maximum value. At step 4 of the iteration, the next vertex hub pair is selected in the following manner. If u_i corresponds to VF, then we must choose one of the two vertices adjacent to F assuming that the model is triangulated. The same reasoning is also applied to when u_i corresponds to EE. As a result, we need one more iteration in order to actually decide which vertex hub pair we want to select. However, we cache the results of this extra iteration and use it for future computations. A snapshot of a typical step during the iteration is illustrated in Figure 2.

E. Local vs Global Minimization

Since our incremental algorithm is a local minimization process, it can get stuck in a local minimum. This can happen based on the choice of the initial pair of features.

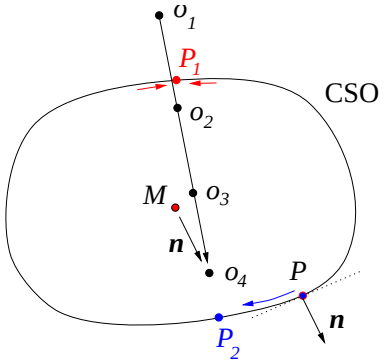


Fig. 3. Escaping from a local minimum: As the origin of the CSO moves from o_1 to o_4, the PD algorithm always reports P_1 as a PD feature. However, using the "centroid difference" vector n, the algorithm starts the search from P, and marches toward P_2.

For example, in Figure 3, as the origin of the CSO moves from o_1 to o_4, the PD algorithm uses a PD feature from previous computation as a starting point of the walk. As a result, the algorithm reports P_1 as a locally optimum PD feature at each frame. In this case, the initial estimate at o_4, P_1, is not a good choice as the true optimal PD feature is P_2. In practice, we can avoid such cases by employing different heuristics suggested in section III-C. In this particular example, the "centroid difference" heuristic solves the problem. In Figure 3, M is the center of mass of the CSO, and $\boldsymbol{n}$ is the centroid difference vector when the origin of the CSO is at o_4. Then, if the algorithm is provided with an additional guess at P, it finds that P is a better estimate as compared to P_1 and starts towards P_2.

A more generic way to escape from the local minimum problem is to employ a global search mechanism. A simple way is to use a discretization approach. As suggested in [12], one might sample the search space (e.g. the Gaussian space in our case), and for each sampled search direction compute a minimum. Another way of the global search is to approximate the whole CSO and launch a search on the approximation. Bergen [10] presents such an approach. The major drawback of this approximation scheme is its numerical instability.

IV. Implementation and Performance

In this section, we describe our implementation and highlight the performance of DEEP on different models and environments. Moreover, we compared its performance with a globally optimal PD computation algorithm and another most recent estimation algorithm presented by [10].

A. Implementation Issues

There are a couple of issues in implementing our PD algorithm presented above. The first one is related to

the collision detection algorithm or library used by our PD algorithm. The second issue is related to handling degeneracies.

A collision detection algorithm is used to check whether two objects overlap. Our prototype implementation, DEEP, takes advantage of our in-house collision library SWIFT [19].

Geometric algorithms are prone to degeneracies. In the case for DEEP, there are two major degeneracies that arise in terms of implementation and application. The first one relates to coplanar faces in any of the polytopes. Coplanar faces result in two main problems. They lead to degenerate configurations for the convex/convex polygon intersection routine and the need to expand the search (walk) to adjacent vertex hubs.

The other degeneracy problem arises from the central projection used during the construction of the Gauss map. The central projection maps the equator of the Gauss map to infinity and it also splits the edge crossing the equator. We avoid this occasion by constructing the Gauss map locally in the following manner. For a given vertex hub pair $\boldsymbol{v_1 v_2}$, take one vertex region, say v_1. Let us denote the set of normals that contribute to v_1, as n_i's. Our goal is to find a normal N such that $n_i \cdot N > 0$ for all i, and N will be the direction of the central projection. This reduces to linear programming. However, it is still possible that the other vertex region v_2 can have an infinite boundary after the central projection. We avoid this by cutting the vertex region v_2 by the equator of v_1.

B. Performance

All the timings presented in this section were obtained on a Linux PC with 1 GHz Pentium III CPU, 256 MB memory, and g++ compiler. We used different models with varying combinatorial complexity as well as aspect ratios to test the performance. The model characteristics used in our benchmarking include:

- The number of faces in the testing models vary from 400 to 4000.
- Three basic primitives, a sphere, an ellipsoid, and a cylinder, are used to generate random models of different aspect ratios. A penetrating object was selected out of the three basic types of objects, whereas only a sphere was chosen as the other object.
- During each time step of the simulation, a penetrating object revolves around a penetrated object, while rotating around its center of mass.

Figure 4 shows the runtime performance of our incremental algorithm (DEEP), and also compares it with the result of the expanding polytope algorithm (EPA) [10]. Roughly, the expected operation count of DEEP is about was $186n + 154$ per iteration including the cost of transformation, where n is the number of adjacent faces to a vertex hub pair. Among them, $166n + 6$ operations are part of the polygon intersection routine, and $20n + 115$ are taken by the Gauss map construction. We also selected benchmarks with high motion coherence. This figure highlights the empirically observed *almost constant time* performance of DEEP on these benchmarks, especially when there is a high motion coherence present between successive frames in the simulation environment. Moreover, as Figure 4-(b) suggests, the performance is not affected by the depth of penetration. However, the running time of GJK-based expanding polytope algorithm [10] grows linearly with the number of faces. This is because of global search in the underlying algorithm and it is also a function of the amount of penetration.

We also evaluated the performance on different scenarios under varying motion coherence. As the motion coherence is reduced, the performance degrades almost linearly. In the environment with low motion coherence, the running time is no longer constant.

C. Comparisons with an optimal solution

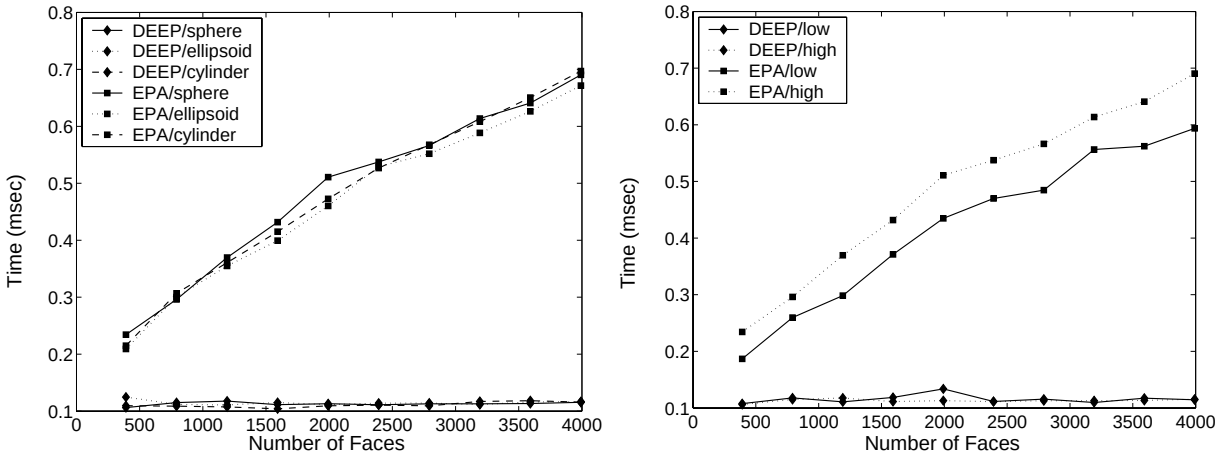
We compared the results of DEEP with an implementation that computes the globally optimal solution. In this case, the rotational motion was not considered. The globally optimal solution is obtained by explicitly computing the Minkowski sums of two convex polytopes. A simple brute-force algorithm to compute the Minkowski sum is based on computing the pairwise Minkowski sum for every vertex pair from each polytope, followed by convex hull of resulting $O(n^2)$ combinations.

For convex polytopes, a penetration depth value simply depends on a penetration direction vector. In our case, DEEP rarely misses the optimal penetration direction. Thus, DEEP was able to compute the global optimum solution in most cases. In terms of comparison to the brute-force implementation, the mean of the error in the PD value was 3.4711^{-6} and its standard deviation was 7.345^{-5}.

V. Conclusions

We present an incremental PD algorithm to compute a locally optimal PD. It runs in almost constant time when there is high motion coherency in the environment, and it finds an optimal solution most of times. The algorithm is based on the implicit, local construction of the Minkowski sums and the iterative optimization on the PD value.

There are some pending research issues in the PD computation. One immediate problem is to devise a more generic algorithm to handle the local minimum problem. We are considering the use of multiresolution techniques or randomized algorithms. Computing the PD between non-convex objects is a very challenging



(a) Fixed PD Amount

(b) Variable PD Amount

Fig. 4. Performance Results of Incremental Algorithm: Figure (a) shows the results of DEEP and EPA for the sphere, ellipsoid and pen model with a fixed PD amount. Figure (b) shows the results of DEEP and EPA for the sphere model by changing the PD value. In both figures, upper lines are the results by EPA, and lower lines are by DEEP. Note that DEEP is invariant of the complexity of a model and the PD amount.

problem. Therefore, any estimation technique or solving the PD for a limited class of non-convex objects (e.g. a convex object vs. a non-convex object) can still be highly desirable. Recently, an approach using a combination of object-space and image-space algorithms for general polyhedral models has been presented in [22].

References

[1] S. A. Cameron and R. K. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Proc. of IEEE Inter. Conf. on Robotics and Automation*, 1986, pp. 591–596.

[2] D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin, "On finding narrow passages with probabilistic roadmap planners," *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, 1998.

[3] W. McNeely, K. Puterbaugh, and J. Troy, "Six degree-of-freedom haptic rendering using voxel sampling," *Proc. of ACM SIGGRAPH*, pp. 401–408, 1999.

[4] A. Gregory, A. Mascarenhas, S. Ehmann, M. C. Lin, and D. Manocha, "6-dof haptic display of polygonal models," *Proc. of IEEE Visualization Conference*, 2000.

[5] Y. J. Kim, M. Otaduy, M. C. Lin, and D. Manocha, "Six-degree-of-freedom haptic display using localized contact computations," in *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (to appear)*, March 2002.

[6] Michael McKenna and David Zeltzer, "Dynamic simulation of autonomous legged locomotion," in *Computer Graphics (SIGGRAPH '90 Proceedings)*, Forest Baskett, Ed., Aug. 1990, vol. 24, pp. 29–38.

[7] Matthew Moore and Jane Wilhelms, "Collision detection and response for computer animation," in *Computer Graphics (SIGGRAPH '88 Proceedings)*, John Dill, Ed., Aug. 1988, vol. 22, pp. 289–298.

[8] S. Cameron, "Enhancing gjk: Computing minimum and penetration distance between convex polyhedra," *Proceedings of International Conference on Robotics and Automation*, pp. 3112–3117, 1997.

[9] D. Dobkin, J. Hershberger, D. Kirkpatrick, and Subhash Suri, "Computing the intersection-depth of polyhedra," *Algorithmica*, vol. 9, pp. 518–533, 1993.

[10] G. Bergen, "Proximity queries and penetration depth computation on 3d game objects," *Game Developers Conference*, 2001.

[11] M. Lin and S. Gottschalk, "Collision detection between geometric models: A survey," in *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.

[12] P. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, and M. Sharir, "Penetration depth of two convex polytopes in 3d," *Nordic J. Computing*, vol. 7, pp. 227–240, 2000.

[13] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between objects in three-dimensional space," *IEEE J. Robotics and Automation*, vol. vol RA-4, pp. 193–203, 1988.

[14] S. Fisher and M. C. Lin, "Fast penetration depth estimation for elastic bodies using deformed distance fields," in *Proc. International Conf. on Intelligent Robots and Systems*, 2001.

[15] K. Hoff, A. Zaferakis, M. Lin, and D. Manocha, "Fast and simple geometric proximity queries using graphics hardware," *Proc. of ACM Symposium on Interactive 3D Graphics*, 2001.

[16] K. Hoff, A Zaferakis, M. C. Lin, and D. Manocha, "Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration," Tech. Rep., UNC-Chapel Hill, 2002.

[17] E.G. Gilbert and C.J. Ong, "New distances for the separation and penetration of objects," in *Proceedings of International Conference on Robotics and Automation*, 1994, pp. 579–586.

[18] M. E. Houle and G. T. Toussaint, "Computing the width of a set," in *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, 1985, pp. 1–7.

[19] S. Ehmann and M. C. Lin, "Accelerated proximity queries between convex polyhedra using multi-level voronoi marching," *Proc. of IROS*, 2000.

[20] M.C. Lin and John F. Canny, "Efficient algorithms for incremental distance computation," in *IEEE Conference on Robotics and Automation*, 1991, pp. 1008–1014.

[21] J. O'Rourke, C.-B. Chien, T. Olson, and D. Naddor, "A new linear algorithm for intersecting convex polygons," *Comput. Graph. Image Process.*, vol. 19, pp. 384–391, 1982.

[22] Y. J. Kim, M. Otaduy, M. C. Lin, and D. Manocha, "Fast penetration depth computation using rasterization hardware and hierarchical refinement," Tech. Rep., UNC-CH, 2002.