SWIFT: Accelerated Proximity Queries Using Multi-Level Voronoi Marching

Stephen A. Ehmann **Ming C. Lin**

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{ehmann,lin}@cs.unc.edu
http://www.cs.unc.edu/∼geom/SWIFT/

Abstract

We present an accelerated proximity query algorithm between moving convex polyhedra. The algorithm combines Voronoi-based feature tracking with a multi-level-of-detail representation, in order to adapt to the variation in levels of coherence and speed up the computation. It provides a progressive refinement framework for collision detection and distance queries. We have implemented our algorithm and have observed significant performance improvements in our experiments, especially on scenarios where the motion coherence is low.

Keywords: Collision detection, level-of-details, Voronoi diagrams.

1 Introduction

Proximity queries, i.e. distance[1] computations and the closely related collision detection problems, are ubiquitous in robotics, design automation, manufacturing, assembly and virtual prototyping. The set of tasks include motion planning, sensor-based manipulation, assembly and disassembly, dynamic simulation, maintainability study, simulation-based design, tolerance verification, and ergonomics analysis.

Proximity queries have been extensively studied in robotics and several specialized algorithms have been proposed for convex polyhedra as well as hierarchical approaches for general geometric models. In this paper, we present a novel algorithm that precomputes a hierarchy composed of a series of bounded error levels-of-detail (LODs) and uses them to accelerate proximity queries. Algorithms to generate LODs for polygonal models have been widely used for rendering acceleration, animation and simulation applications [12, 22, 24]. One of our goals is to take advantage of multiresolution representations commonly used in rendering applications and use them for proximity queries as well.

[1]Distance is commonly defined as the Euclidean separation distance by many applications, and we adopt the same definition in this paper.

Given a convex polyhedron, our algorithm precomputes a bounded error level-of-detail (LOD) hierarchy. It constructs a series of bounding volumes that enclose the original polyhedron. Then, it establishes parent-child relationships between the features of adjacent levels. At runtime, the hierarchy is traversed according to the type of query being performed. Within each level, the surface of an object is "marched" across using a modified Lin-Canny [16] closest feature tracking algorithm based on Voronoi regions. We refer to such a technique as "Voronoi marching" in this paper. Spatial locality and temporal coherence is captured by both the Voronoi regions and the hierarchy of LODs.

The algorithm has been implemented and analyzed using various benchmarks. Experiments were performed using various shapes of objects and various multiresolution options. Furthermore, it is compared against a straightforward directional lookup table scheme we devised to determine overall effectiveness. We observe speedups for each type of query over a simple surface traversal.

1.1 Main Results

In this paper, we present an accelerated proximity query algorithm between moving convex polyhedra which exploits multiresolution representations. Our main contributions are:

- An accelerated proximity query algorithm between convex polyhedra using multi-level Voronoi marching. The substantial performance improvements are mainly due to the use of a multiresolution representation and a faster Voronoi marching algorithm.

- A progressive refinement algorithmic framework for proximity computation. For many applications including motion planning and tolerance verification, a relatively inexpensive *approximate* distance computation with bounded error is sufficient. Our framework allows applications to progressively refine the distance estimate to suit their needs, while minimizing overall computation cost.

- A better understanding of the issues involved in designing suitable level-of-detail representations for proximity queries. These issues include level of coherence, object aspect ratios, and contact scenarios.

1.2 Organization

The rest of the paper is organized as follows. Section 2 gives a brief survey of related work. Section 3 presents an overview of our approach. The design and computation of the multi-level-of-detail representation is described in Section 4 along with our lookup table scheme. Next, a proximity query algorithm using the multiresolution representation is described in Section 5 along with other ways to accelerate queries. Section 6 describes our prototype implementation and shows the performance of our system, SWIFT. Finally, we conclude with future research directions in Section 7.

2 Previous Work

Distance computation and intersection (collision) detection problems have been fundamental subjects of study in robotics, computational geometry, simulation, and physical-based modeling. There is a wealth of literature on both analyzing the theoretical complexity of proximity queries and on designing algorithmic solutions to achieve interactive performance. We will limit the scope of the discussion in this paper to rigid convex polyhedra, although some of the techniques may be applicable to other domains and model representation as well.

2.1 Proximity Queries for Convex Polyhedra

Most of the earlier work has focused on algorithms for convex polyhedra. A number of algorithms with good asymptotic performance have been proposed in the computational geometry literature [6]. Using hierarchical representations, an $O(log^2 n)$ algorithm is given in [4] for the convex polyhedral overlap problem, where n is the number of vertices. This elegant approach is difficult to implement robustly in 3D, however.

Good theoretical and practical approaches based on the linear complexity of the linear programming problem are known [18, 23]. Minkowski difference and convex optimization techniques are used in [9] to compute the distance between convex polyhedra.

Erickson, et al [7] recently proposed a new class of kinetic data structures for collision detection between convex polyhedra. This class of hierarchical representations has only been analyzed for the 2D case however.

In applications involving rigid motion, geometric coherence has been exploited to design algorithms for convex polyhedra based on either traversing features using locality or convex optimization [2, 5, 16, 15, 19]. These algorithms exploit the spatial and temporal coherence between successive queries and work well in practice.

2.2 Hierarchical Representations

Bounding volume hierarchies are presently regarded as one of the most general methods for performing proximity queries between general polyhedra. Specifically, sphere trees, cone trees, axis-aligned box trees, oriented box trees, k-d trees and octrees, trees based on S-bounds, and k-dops have been used for fast intersection queries for general polyhedra, as well as polygon soups [13, 20, 10, 21, 1, 14]. For most scenarios, these hierarchies excel at intersection detection but do not do so well when it comes to distance computation.

2.3 Multiresolution Techniques

Multiresolution modeling techniques, such as model simplification, have been proposed to extract the shape of the underlying geometry [25]. A recent survey on polygonal model simplification is available [17]. The main idea behind using a multiresolution hierarchy is to compute and utilize a correspondence between the original model and a simplified one. We will discuss the use of a pair of simplification algorithms due to Dobkin and Kirkpatrick [4] and to Garland and Heckbert [8]. Cohen [3] has presented algorithms based on successive mappings and appearance preserving simplification.

For proximity query, Guibas, et al. [11] proposed an elegant approach that exploits both coherence of motion and hierarchical representation for faster distance computation. Our approach differs from their *H-Walk* algorithm in that our algorithm can easily compute an approximated distance with a guaranteed error tolerance without always descending and/or ascending the entire hierarchy.

3 Algorithm Overview

Our algorithm operates on orientable 2-manifolds that are closed and represented using triangles. The polyhedra must be convex and undergo rigid motion. We will use the terms "polyhedron" and "object" interchangeably.

Multiresolution techniques typically consist of two main components. They involve the precomputation of a hierarchical representation upon which subsequent queries are performed. We wish to compute a multiresolution representation that supports proximity queries.

In the preprocessing stage, we compute a level-of-detail representation for each object. This hierarchy is organized as a sequence of convex polyhedra P_0, P_1, P_2, $\cdots$, P_k where P_0 is the input polyhedron. Each successive polyhedron is composed of fewer features and has the property that it bounds the original object. Furthermore, a correspondence is established between successive levels in each direction of the sequence. More details of the actual construction of this representation are given in Section 4.

A query using this hierarchy is performed in much the same way for each type of proximity query. Basically, as long as certain levels of two objects are intersecting, then the hierarchy is refined. When two levels are found to be disjoint, then it may be possible to end the query at a subsequent point of refinement. Furthermore, for queries other than intersection, a tolerance may be provided which specifies how close objects must be in order to answer the query. In Section 5 the hierarchical query is described in more detail.

4 Hierarchical Representations

The multi-level-of-detail representation of each object that is constructed in the algorithm's preprocessing stage must have certain properties in order to be useful and efficient for performing proximity queries. Next, we describe desirable characteristics for the representation, discuss the steps involved in the creation of the hierarchy, and touch upon various tradeoffs along the way. Then, we discuss two hierarchy creation methods we implemented. We conclude with a description of our lookup table scheme.

4.1 Terminology

Recall that the hierarchy is organized as a sequence of convex polyhedra P_0, P_1, P_2, $\cdots$, P_k where P_0 is the input polyhedron. We will call P_0 the finest level and P_k the coarsest level. We will call the level P_{i-1} the "child" of the level P_i and the level P_{i+1} the "parent" of P_i. We use the convention that the finest object is at the bottom of the hierarchy so the term *moving up* the hierarchy means moving to a coarser level. Moving to a finer level is termed *moving down* the hierarchy.

The maximum deviation of P_i from P_0 is represented by ϵ_i. The computed distance between certain levels of two objects that are found to be disjoint is δ and the maximum error associated with the distance is ϵ. The distance tolerance given by the application is δ_{max} and the error tolerance is ϵ_{max}.

4.2 Desired Features

To design a multi-level representation for accelerating distance computation while preserving locality and coherence, our goal is to produce a hierarchy that provides the following characteristics:

- **Simplified Combinatorial Complexity:** Each level of representation in the hierarchy should have less combinatorial complexity than its child and higher combinatorial complexity than its parent.

Ideally we would like to create $O(\log n)$ levels, where n is the number of vertices in the original polyhedron P_0. This is possible if a constant fraction of features are eliminated for each level. In addition, it is wise to stop the hierarchy construction when a certain number of levels or features has been reached.

- **Bounding Volumes:** If we wish to have a mechanism to compute approximate distances with bounded error tolerances ϵ_{max}, then we can make use of the levels P_i of the hierarchy which bound the original polyhedron P_0 with a global surface deviation $\epsilon_i \leq \epsilon_{max}$. Query performance can be further improved by aiming to create a hierarchy where the bounding volumes are kept as small as possible.

- **Local Correspondence:** For each level of the hierarchy P_i for $i > 0$ that bounds P_0, there must also be spatial coherence between it and its adjacent levels. This implies that a feature on polyhedron P_i will have a corresponding feature on P_{i+1} and on P_{i-1} which are proximate in position and orientation. These are called the parent and child features respectively.

4.3 Construction

The levels of the hierarchy are constructed in order starting with P_1. In particular, when constructing P_i, P_{i-1} provides the topological and geometric information required to reduce the number of features while P_0 provides geometric information to satisfy the bounding criterion. The process of constructing a level in the hierarchy is given by the following steps to create P_i:

1. Create P_i as a high quality simplification of P_{i-1}.
2. Make P_i convex by computing its convex hull.
3. Compute the center of mass of P_i and translate P_i such that its center of mass coincides with P_{i-1}'s center of mass.
4. Scale P_i from its center of mass so that it barely bounds P_0.
5. Compute the maximum deviation ϵ_i of P_i from P_0. This is the same as computing the Hausdorff distance between P_i and P_0.
6. Assign the feature correspondences from P_i to P_{i-1} and from P_{i-1} to P_i.

For the first step, any simplification algorithm may be used as long as the topology is maintained. Any convex hull algorithm may be used for the second step.

The computation of the center of mass is straightforward for a closed polyhedron. To scale P_i in Step 4, the maximum scaling required over all of the faces of P_i is found and applied. The scaling required for a face is computed by finding the extremal vertex on P_0 in the direction of the (outward pointing) face normal and computing the scaling factor required to cause the vertex to coincide with the scaled face's supporting plane.

The Hausdorff distance (ϵ_i) can be computed in this context by computing the maximum deviation over all the vertices of P_i. The deviation of a vertex of P_i is computed by computing its distance from P_0. The vertices are the only points on P_i that have to be checked because they represent local maxima of the deviation function over P_i.

To assign children (features of P_{i-1}) to the features of P_i, the nearest feature of P_{i-1} is found for each vertex of P_i and the nearest vertex is selected from this nearest feature. Each neighbor of each vertex of P_i (including the vertex) is assigned the nearest vertex as its child feature. To assign parents (features of P_i) to the features of P_{i-1}, the extremal vertex of P_i is found for each face of P_{i-1}. Each face as well as its edges and vertices are assigned the extremal vertex as their parent. These feature correspondence schemes were chosen for their spatial coherence. Of course, there are other schemes that may work better but it is unlikely that a sizeable overall improvement would be gained.

4.4 Dobkin-Kirkpatrick Hierarchy

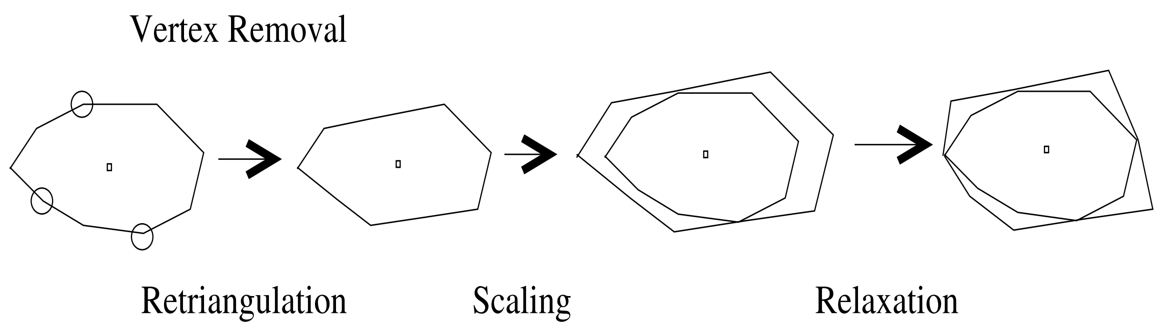

Figure 1. Dobkin-Kirkpatrick Hierarchy Construction

The first method that we implemented is based on the Dobkin-Kirkpatrick hierarchy [4]. It is a specialized algorithm that is the same as the steps given above but with a different scaling procedure and different parent and child feature assignments. An illustration of the process is shown in Figure 1.

4.4.1 Simplifying

The Dobkin-Kirkpatrick simplification does not specify which vertices should be removed but simply that an independent set is be removed at each step. If *any* independent set is removed, very large bounding volumes (Step 4) can result. Therefore, we try to choose the independent set in an intelligent manner. This is done by assigning an importance value to each vertex. A higher importance means that the vertex is more important and should not be removed. It also indicates that the geometry is not very flat near the vertex. Vertex v_{ij} of object P_i is assigned an importance value

$$I_{ij} = \sum_{k=1}^{V}(1 - \cos\theta_{ijk})$$

where V is the valence of the vertex and θ_{ijk} is the dihedral angle of the kth edge of the vertex. Thus, to create the independent set, vertices are added to it in order of increasing importance. As each vertex is added to the set, its neighbors are marked and not allowed to be subsequently added, otherwise the independent set criterion would be violated. These vertices are removed and the holes are re-triangulated in a convex manner. This can be done by taking the outer convex hull of the vertices neighboring the removed vertex. Levels are created until a tetrahedron is reached, until a minimum number of triangles is reached, or until the center of mass of the original object falls outside of the simplified object we are trying to create.

Even with this scheme, we found that excessively large bounding volumes occurred (Step 4). The reason stems from the choice of vertices that are placed in the independent set. We found that if we add as many vertices as possible to the independent set, we still add vertices with high importance because most of the low importance vertices have neighbors that are added. This may cause the volumes to become large when vertices which have sharp peaks are removed because the faces that fill a hole will have to be scaled by a large amount in order to bound a high importance (sharp) vertex. To reduce the effects of this problem we decided to only add vertices to the independent set that meet the criterion $I_{ij} \leq 1$. In other words, we stop the independent set creation when there are no more vertices whose importance values are less than or equal to one. This causes fewer vertices to be removed per level but keeps the sizes of the bounding polyhedra in check.

4.4.2 Bounding

The next step we discuss is Step 4 which involves scaling the sequence of polyhedra so that they bound the finest object. A scaling factor is computed for each P_i for $i > 0$ such that P_i bounds the finest polyhedron. Each level is then scaled by the amount computed for it. The scaling is done from the center of mass of the finest polyhedron.

We found that due to the structure of the simplification, we can add a relaxation process at this point. The relaxation involves trying to move the vertices back toward the center of mass while maintaining convexity and boundedness of P_0. Note that in both the scaling and the relaxation, the vertices move along rays from the center of mass through their original positions. The holes caused by vertex removal form regions which we call *center of mass regions*. By keeping track of the vertices that belong to certain center of mass regions, the scaling and relaxation computations can be performed more efficiently. Then, the maximum surface deviation is computed as ϵ_i.

4.4.3 Assigning Feature Correspondence

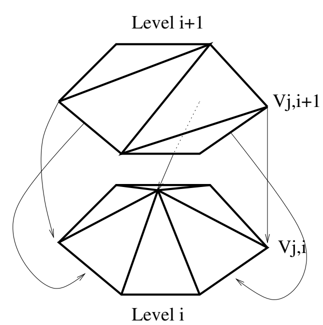

Figure 2. Dobkin-Kirkpatrick Level Correspondence

Each vertex, edge, and face on P_i for $i > 0$ is assigned a child pointer which points to a feature on P_{i-1}. The assignment can be done in a variety of ways. Figure 2 shows some of the pointers based on our assignment scheme. We assign the vertices at the coarser level to have as children their corresponding vertices at the finer level. Edges and faces at the coarser level can be of two varieties: *kept* or *removed*. Kept features are ones that are not destroyed by the vertex removals. They include edges and faces that have none of their vertices in the independent set. Removed edges and faces are ones that have one of their vertices in the independent set. They are replaced with *new* features at the coarser level. The kept features are assigned children that correspond to themselves at the finer level. For simplicity, we choose to assign to the new features, the vertex that was removed from the finer level in order to create them. There are other schemes, which may perform better, that can be used to assign the children in this latter case. This assignment is actually done during the building of the hierarchy.

4.5 QSlim Hierarchy

The drawback of the Dobkin-Kirkpatrick hierarchy is that many levels are created which causes a slow-down for the query algorithm. The problem is that the decimation rate is not high enough. That lead us to consider this type of hierarchy which relies on a more general simplification algorithm which is able to achieve high decimation rates (arbitrary) while at the same time maintaining small bounding volumes.

The second method that we implemented is based on the publicly available QSlim package. We used the QSlim system which is an implementation of Garland and Heckbert's quadric error metric simplification algorithm [8] for the first step of the hierarchy creation process. It allows an arbitrary face target for each step of simplification allowing us to choose the decimation rate. We use the term "decimation rate" to mean the fraction of triangles left when a coarser level is created from a finer one. For example, if a level has 1000 triangles and a decimation rate of 0.25 is applied, then the newly created (coarser) level has 250 triangles. For the second step, we used the QHull convex hull library that is also publicly available. All the other steps remain the same.

There is a triangle count cutoff that determines when to stop building the hierarchy. In addition, there is a constant factor that determines what the decimation rate is across levels. These parameters affect the query performance and are discussed later.

The quadric error metric method employed by QSlim is very desirable for keeping the volumes of the bounding polyhedra small since only flat areas on a fine model are refined. This also has the effect of causing the tesselation density per solid angle of the simplified convex polyhedra to become more or less constant.

4.6 Directional Lookup Table

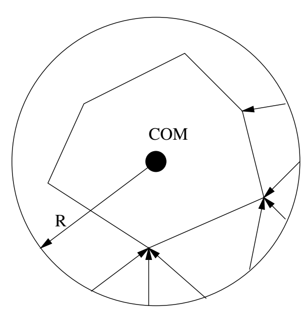

Figure 3. Lookup Table Construction

For performance comparison, we also designed and implemented a directional lookup table scheme for initializing the Voronoi marching. The lookup table is based upon direction from the center of mass (COM) of a polyhedron. Vertices are stored in the table. Figure 3 shows the idea of the construction in two dimensions.

The parameters used to construct the table are resolution and a bounding sphere radius factor. The resolution of the lookup table determines how close together the samples are. The table is based on spherical coordinates and has an entry for each angle increment (determined by the resolution) in the altitude and azimuth directions excluding the north and south poles. There are two additional entries for the poles. The polyhedron for which the table is built has a "radius" which is defined as the maximum distance of any of its vertices from its center of mass. In other words, a sphere centered at the center of mass with radius equal to the polyhedron's radius would bound the polyhedron at any orientation. The bounding sphere radius factor is multiplied by the polyhedron's radius to construct another (larger) bounding sphere. The radius factor must be greater than or equal to 1. Sample points are generated on the new bounding sphere in the directions prescribed by the lookup table. For each of these points, the nearest vertex on the polyhedron to the point is stored at the corresponding location in the table. Using this table for query purposes is discussed in Section 5.3.

5 Proximity Query

Proximity queries can be in the form of intersection detection, error-bounded approximate distance computation, exact distance computation, or contact determination. The hierarchy is queried in much the same way for all four of these query types. The basis is a multiresolution extension on the algorithm proposed by Lin and Canny [16]. First we describe the marching within a level and then show how to answer a query by marching across levels of a multiresolution hierarchy for each of the query types.

5.1 Voronoi Marching Within a Level

We have implemented an algorithm which marches across the surface of a convex polyhedron using Voronoi regions. It is basically the same algorithm as the original *Lin-Canny* [16] algorithm and the improved version, *V-Clip* [19], in the sense that it has the same high level behavior and computes the same results. We do not give a full description here but rather a quick sketch.

The original algorithm is based on traversing the external Voronoi regions induced by the features of each convex polyhedron. The invariant is that at each step, either the inter-feature distance is reduced or the dimensionality of one or both of the features decreases by one, i.e. a move from a face to an edge or from an edge to a vertex. We employ the notion of states with the same definition as in *V-Clip*. The state transitions are a bit different because we sometimes change features on both objects in one step. For example, this happens from the edge-edge state to the vertex-vertex state. Traditionally, the feature whose Voronoi plane is found to be violated *first* is the one that is updated to. There are other methods for updating to a closer feature. For instance, we found that performance improves when we update to the feature whose plane is *most* violated which means that we have to check all the planes. This gives a higher cost to each local decision but yields a global benefit. We are currently investigating benefits from better choices of the next feature and other low level optimizations. A comparison of our low level marching implementation is compared against *V-Clip* in Section 6.

5.2 Multi-Level Voronoi Marching

In the case of intersection detection, the search normally starts at a pair of features related to the closest features from the previous query so that coherence may be exploited. Like in Lin-Canny, the distance between two convex polyhedra is minimized by marching across their surfaces. If an intersection is detected, a finer level of the hierarchy is traversed to. If at *any* level, we reach a minimum in the distance function and the objects are disjoint, then this is reported. With no additional cost, an approximate distance can be provided as δ with the error ϵ, when the objects are disjoint. The distance between the levels of the two objects is δ and $\epsilon = \epsilon_i + \epsilon_j$ is the associated error equal to the sum of the two level deviations. The pair of closest features are used to find the features to be used for the next query. This is done by traversing their parent pointers, using the deviations ϵ_i, and keeping track of the distance δ that is decreased by the differences in deviation. If an intersection has been detected and P_0 has been reached for both objects, then intersection is reported. The pair of intersecting features is stored for the next query.

For error-bounded approximate distance computation, the application can provide a distance tolerance δ_{max} and an error tolerance ϵ_{max}. If the levels of the two objects intersect all the way to the finest levels of both objects, then intersection is reported and the pair of features is stored. Otherwise, two levels were found to be disjoint during the refinement. The disjoint features are used as in the intersection case to determine the initial features to be used for the next query. If at any point $\delta > \delta_{max}$ the query is terminated and nothing is reported. If at any point, $\epsilon \leq \epsilon_{max}$ then the error tolerance has been met and δ and ϵ are reported. This means that the exact distance is in the range $[\delta, \delta + \epsilon]$.

To compute exact distance, the same method is followed but only the distance tolerance δ_{max} is specified. As long as $\delta \leq \delta_{max}$, the distance is refined until the finest levels are reached at which point δ is reported. Finally, contact determination (closest points) queries can be handled the same as exact distance ones.

For the Dobkin-Kirkpatrick hierarchy, we found that there are many levels created. We tried three different refinement methods: refine one level on one object, refine one level on both objects, and refine two levels on both objects. These are called “Single”, “Single-Single”, and “Double-Double” respectively in Section 6 where we compare the performance of each of the methods.

5.3 Directional Lookup Table

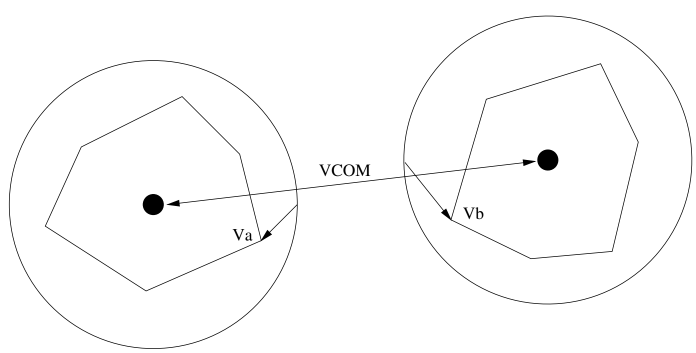

Figure 4. Lookup Table Initialization of a Query

When using the directional lookup table to help answer a query, the features from the previous query are not used. Therefore, the query time should be motion independent. Shown in Figure 4 is a two dimensional representation of the initialization of a query. First the vector defined by the center of mass of the two objects being tested (VCOM) is computed. The vector is transformed to the local coordinate frame of the objects and is used to find the table location that is the nearest to the direction given by the vector. The vertices at the corresponding locations in each object’s table (V_a and V_b) are used as the feature pair to start marching from. When the lookup table is used, there is no hierarchy being used. The performance of the query depends solely on how the tables are built for each object. In the next section we see how various settings affect the performance of the lookup tables.

6 Experimental Results

The models used to test our query algorithm were empirically created by taking the convex hull of randomly sampled points on objects of various aspect ratio. Three different ellipsoidal models were created, each with 2000 triangles. Their aspect ratios are (1,1,1), (1,1,0.1), and (1,0.1,0.1). They are referred to as "fat", "plate", and "long" respectively. Each of the models was normalized to have a bounding sphere of radius r_0.

Following the performance benchmarking algorithm due to Mirtich [19] and used in [11], we created various scenarios. The algorithm has one object remain fixed while another orbits about it in an elliptical trajectory. An orbit radius is defined as the distance between the centers of the models when they are closest in the orbit. Three different pair combinations were used: fat-fat, plate-plate, and long-long. The orbit radius was set to be either $2r_0$ (small) or $3r_0$ (large). Thus, the models either just touch or are always separated by a distance of at least r_0.

Our implementation is called SWIFT (Speedy Walking via Improved Feature Testing). We obtained empirical observations for it by running programs on an SGI Reality Monster using a single 300 MHz MIPS R12000 CPU. Timings were done by using a highly accurate free-running hardware clock.

6.1 Marching Within a Level

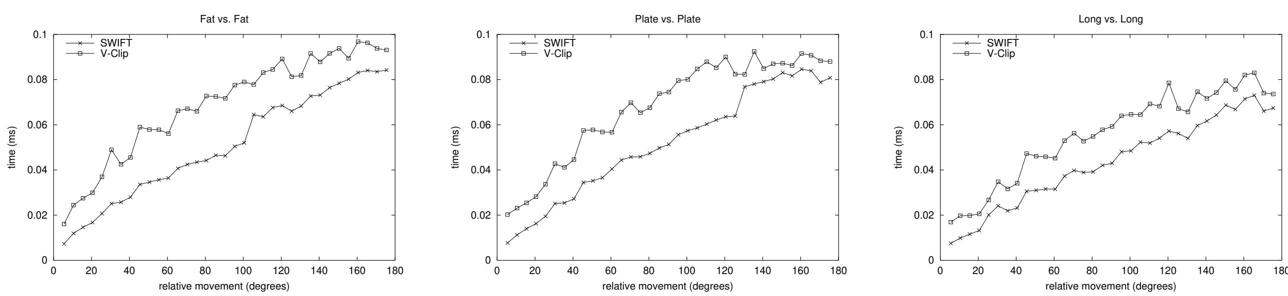

Figure 5. Performance of Marching Within a Level: *SWIFT* vs. *V-Clip*

Our implementation of the algorithm that marches within a level was compared to *V-Clip* [19] and found to be faster[2]. Figure 5 shows timings for different object pairs using the orbiting algorithm with an orbit radius of $2r_0$. No hierarchical or lookup table enhancements were used in the comparison. All future experimental results that are presented are based solely on the SWIFT implementation. Next, we add the directional lookup table and observe its performance.

6.2 Directional Lookup Table

In Figure 6 we show the results of using a directional lookup table (LUT) with an orbit radius of $2r_0$. The first parameter given in the graph legends is the bounding sphere radius factor of the lookup table and the second is the resolution in degrees. For the fat pair of objects, the resolution of the table is the only thing that matters. Furthermore, notice that the performance of the lookup table for the fat pair is much better than for the other two pairs because it (the LUT) is based on a sphere and the fat objects are very

[2] In the preliminary version of this report, available in the Electronic Proceedings of IEEE IROS 2000, we stated that our implementation was nearly twice as fast. However, we have since gathered more data and found that to be an inaccurate assessment.

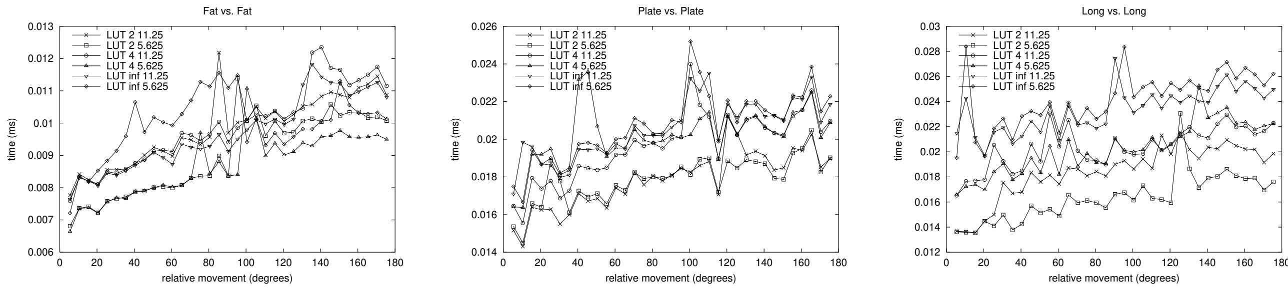

Figure 6. Performance of Various Lookup Tables

nearly spheres. For the long and plate pairs, a radius factor of 2 seems to be best. It should be stated that this may be because of the scenario we are using. The sizes of the lookup tables of various resolutions are: 8 KB for 5.625 degrees and 2 KB for 11.25 degrees. We will use the "LUT 2 5.625" performance data for the overall comparison.

6.3 Multiresolution Hierarchy

The level-of-detail hierarchy we have implemented can also be used to achieve speedups for both intersection detection and exact distance computation. The titles on the graphs in the following figures reflect the object pairs that were used as well as whether the orbit was small or large.

6.3.1 Dobkin-Kirpatrick Hierarchy

Shown in Figure 7 is the performance of various intersection detection queries using the Dobkin-Kirkpatrick hierarchy. Similarly, Figure 8 shows exact distance computation performance. In each entry of the legends of the graphs is the number of triangles that the hierarchy was cutoff at, the number of levels created (in parentheses) and the inter-level marching method used (as introduced in Section 5).

There is not a lot of difference among the various settings for intersection detection since most of the time is spent traversing the coarsest level. However, the settings which produce a coarser coarsest level do the best. Since all the levels are traversed most of the time in the case of exact distance, it makes sense the the methods to do the best are the ones that employ double descent on the levels of both objects. We will use the "DK Hier 100 Double-Double" performance data for the overall comparison.

6.3.2 QSlim Hierarchy

Shown in Figure 9 is the performance of various intersection detection queries using the QSlim hierarchy. Similarly, Figure 10 shows exact distance computation performance. In each entry of the legends of the graphs is the decimation rate (Section 4.5) and the number of levels created (in parentheses).

For the intersection detection query, the 0.125 hierarchy is anomalous because its coarsest level consists of 250 triangles while the other hierarchies have fewer. This results in a slower query because for intersection detection, most of the time is spent traversing the coarsest level. The coarsest hierarchy does the best on the intersection detection tests because of the reason previously stated. For the exact distance tests, it seems that having more levels is not as good as just having two. Even having three is noticeably slower. Also, it still seems like a coarser hierarchy does better for this type of query as well. We will use the "QS Hier 0.03125" performance data for the overall comparison.

6.4 Overall Performance Comparison

Figure 11 gives the overall performance comparison of intersection detection using the best results from each category along with the performance of SWIFT when there is no hierarchy or lookup table (Nothing). Similarly, Figure 12 gives the overall performance comparison for exact distance computation.

For intersection detection, the hierarchies do well because of their simple coarsest levels but they do not perform as well when it comes to exact distance computation because of the overhead associated with marching on the finest level to the true minimum of the distance function. The QSlim hierarchy always outperforms the Dobkin-Kirkpatrick hierarchy because there are too many levels to traverse in the latter.

6.5 Summary

The reason for the difference in the performance of the two query types is due to the fact that in exact distance computation, we cannot stop once the models are found to be disjoint like we can in intersection detection. Approximate distance computation is highly dependent upon the application so we did not evaluate its performance but expect the majority of cases in which it is used to resemble the intersection detection query. It can be shown that its cost lies in between the cost of intersection detection and exact distance computation.

It can be seen from the profiles of our timing curves that our multi-level Voronoi marching algorithm is able to keep the impact of the relative motion of the objects under control. It is also apparent that a simple lookup table scheme like the one we have presented suffices to perform in the same kind of manner. It is possible that if the objects are composed of more triangles (10,000 or 100,000), then a hierarchy of three or four levels may win out over the simple lookup table and the two level hierarchies.

7 Conclusion

We have presented a new algorithm for accelerating proximity queries between convex polyhedra using level-of-detail representations. We described the construction of the required hierarchical data structures and discussed various design issues involved. The runtime algorithm along with its issues were addressed and the performance presented.

Our implementation has been shown to have good performance. The source code is available to the public and provides a complete and entire proximity query library, with the two types of hierarchies presented. It is available at

http://www.cs.unc.edu/~geom/SWIFT/

We have shown that a hierarchical representation integrated with Voronoi marching offers a progressive refinement framework for exact and approximate distance computation, which optimizes performance while meeting the application's need for bounded error computation. We have also shown that a simple directional lookup table can be used very effectively.

This research is the first step toward the design of distance query algorithms using multiresolution representations. There are many potential directions for future research. There is a need to develop a suitable metric for measuring or quantifying the optimality of the hierarchies, coherence levels, and level relationships. On the theoretical side, there is the interesting problem of proving a better bound on the computation of the distance between two convex polyhedra. The extension of this framework to non-convex objects and deformable models remains a very challenging problem.

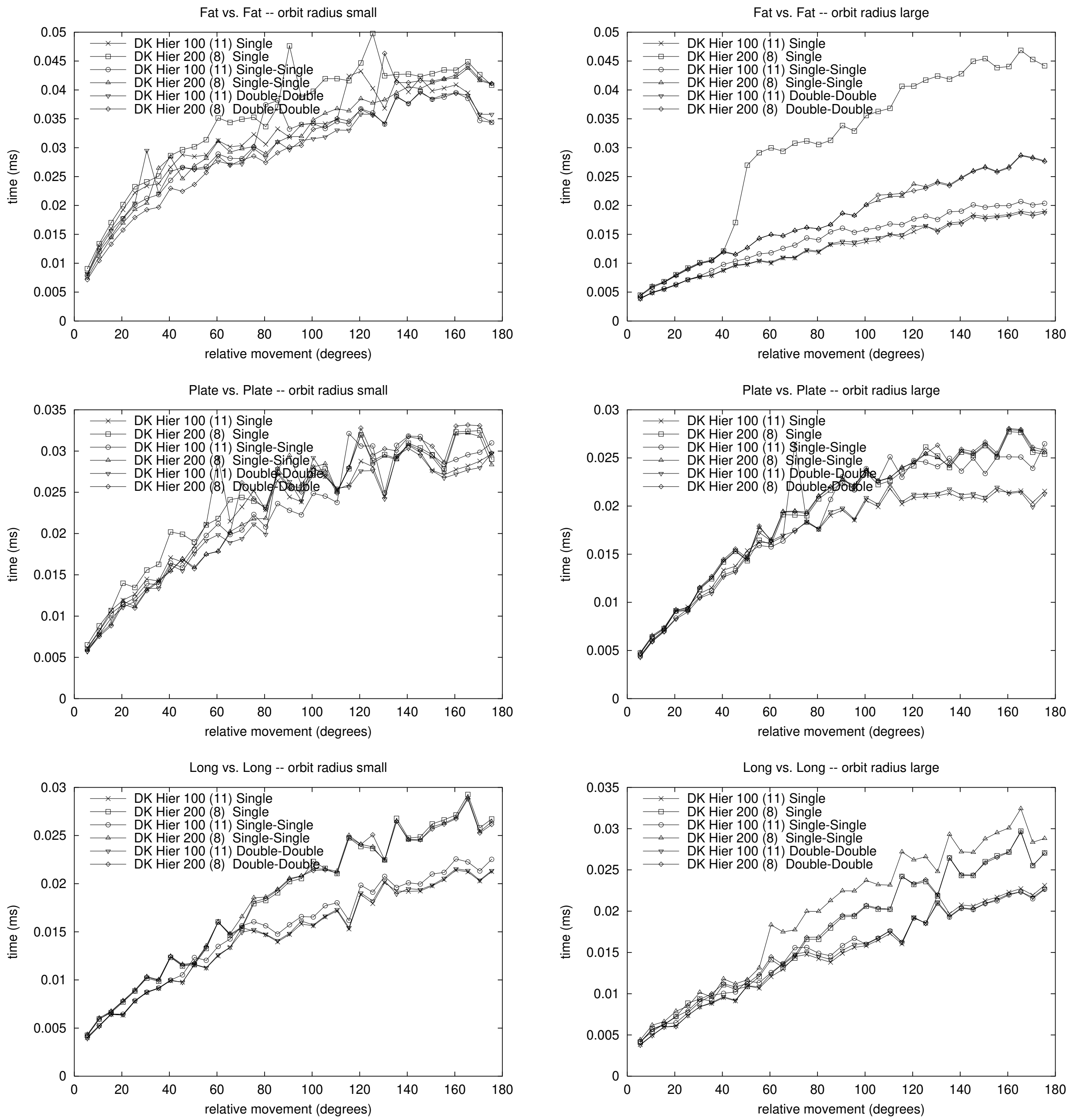

Figure 7. Performance of the Dobkin-Kirpatrick Hierarchy for Intersection Detection with Various Parameter Settings

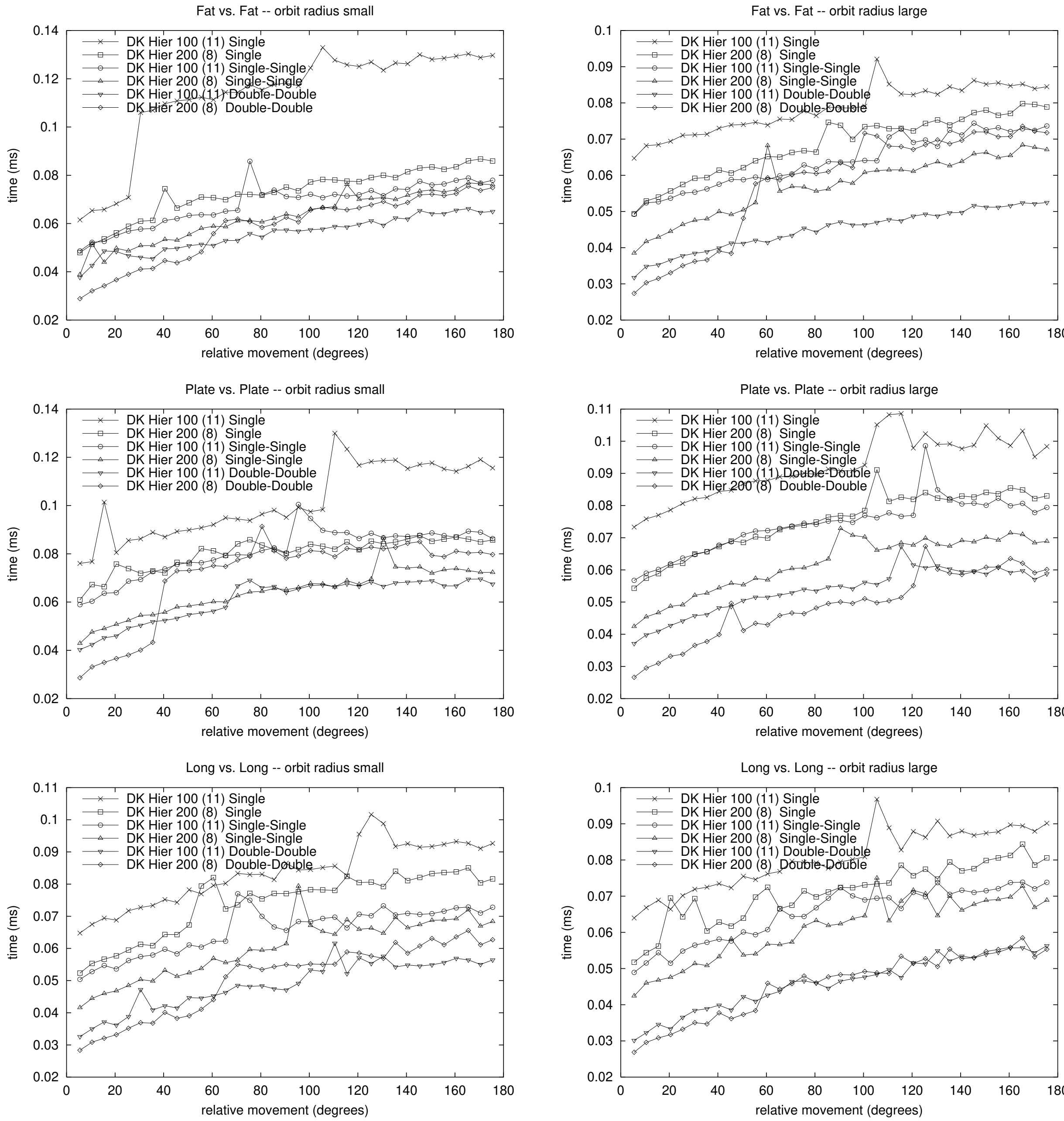

Figure 8. Performance of the Dobkin-Kirpatrick Hierarchy for Exact Distance Computation with Various Parameter Settings

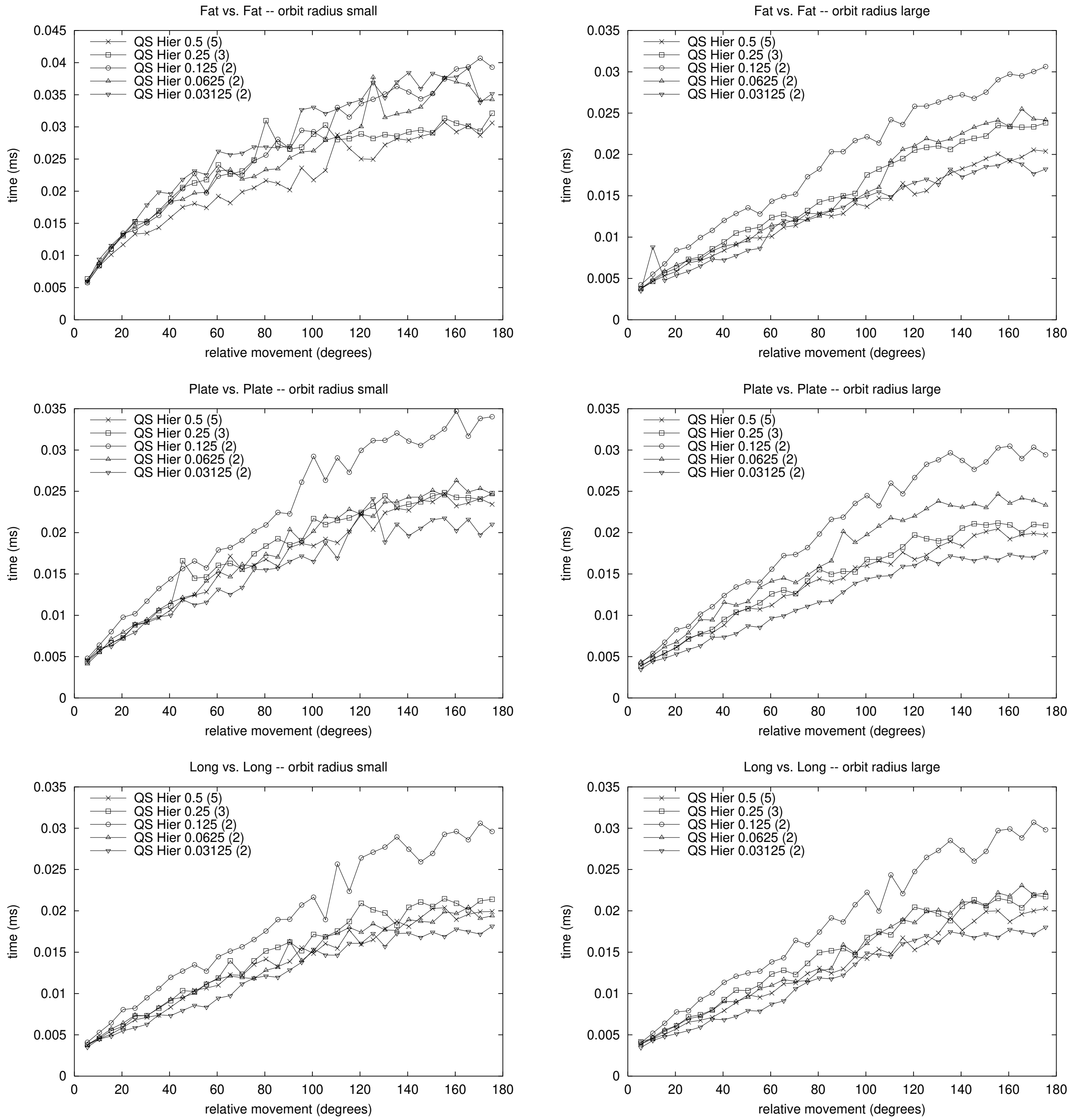

Figure 9. Performance of the QSlim Hierarchy for Intersection Detection with Various Parameter Settings

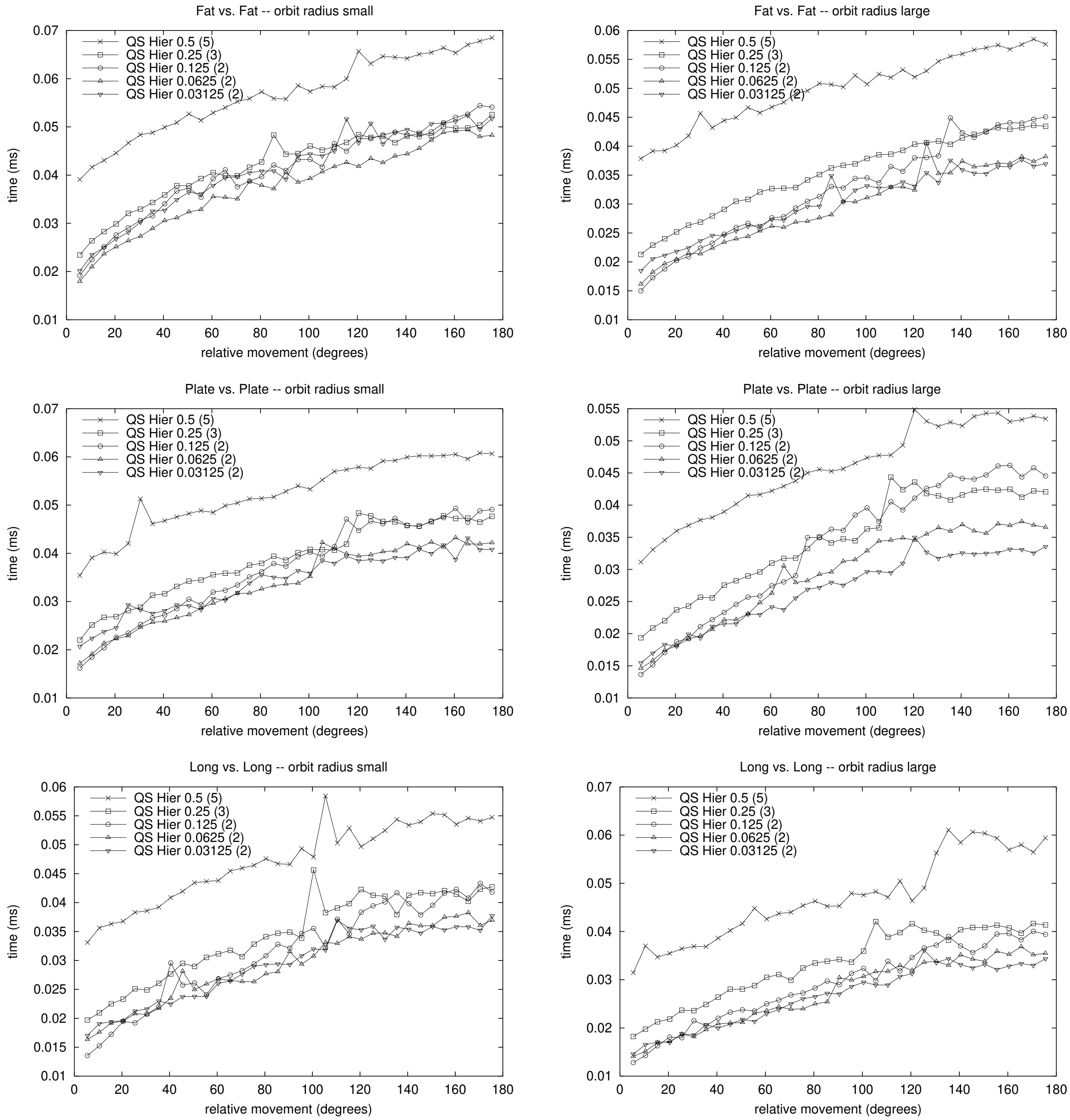

Figure 10. Performance of the QSlim Hierarchy for Exact Distance Computation with Various Parameter Settings

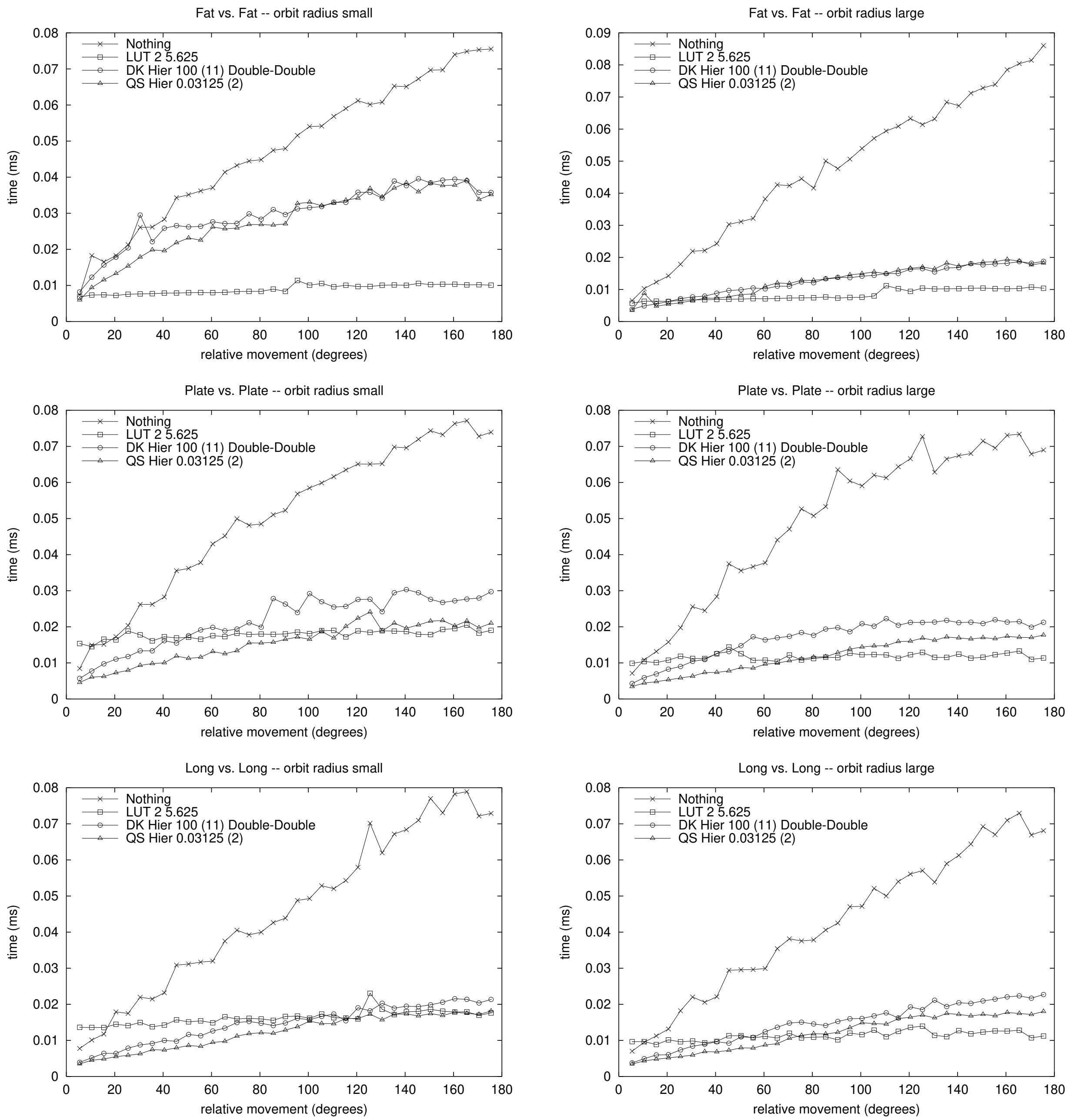

Figure 11. Performance of Best Methods for Intersection Detection

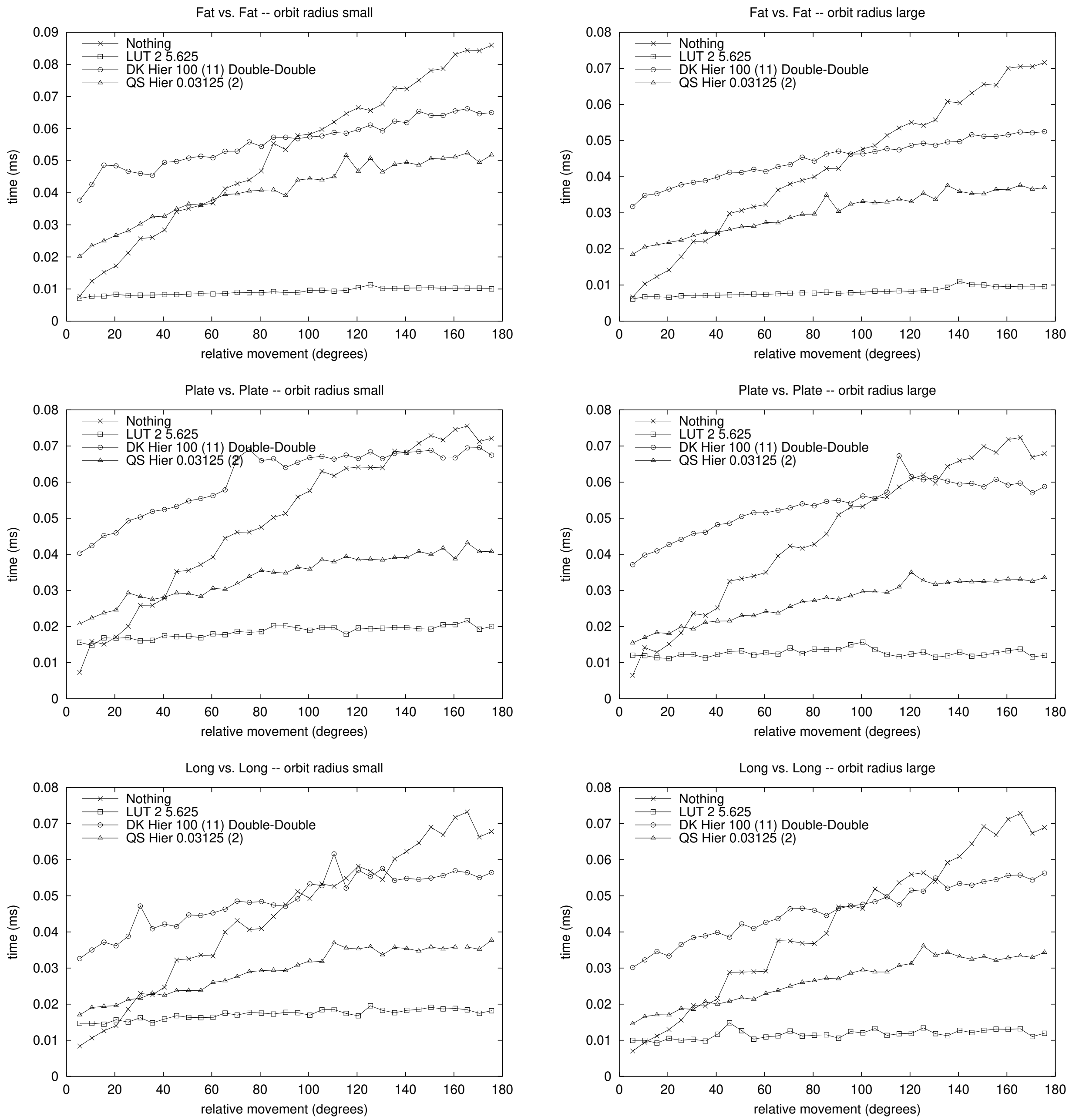

Figure 12. Performance of Best Methods for Exact Distance Computation

References

[1] S. Cameron. Approximation hierarchies and s-bounds. In *Proceedings. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 129–137, Austin, TX, 1991.

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

[3] J. Cohen. Appearance-Preserving Simplification of Polygonal Models. *Ph.D. Dissertation*. University of North Carolina at Chapel Hill. 1998.

[4] D. P. Dobkin and D. G. Kirkpatrick. Determining the separation of preprocessed polyhedra – a unified approach. In *Proc. 17th Internat. Colloq. Automata Lang. Program.*, volume 443 of *Lecture Notes Comput. Sci.*, pages 400–413. Springer-Verlag, 1990.

[5] P. Dworkin and D. Zeltzer. A new model for efficient dynamics simulation. *Proceedings Eurographics workshop on animation and simulation*, pages 175–184, 1993.

[6] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Algorithms*, 6:213–224, 1985.

[7] J. Erikson, L. Guibas, J. Stolfi, and L. Zhang. Separation-sensitive collision detection for convex objects. *Proc. of SODA*, 1999.

[8] M. Garland and P. Heckbert. Surface simplification using quadric error bounds. *Proc. of ACM SIGGRAPH*, pages 209–216, 1997.

[9] 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 RA-4:193–203, 1988.

[10] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. of ACM Siggraph'96*, pages 171–180, 1996.

[11] L. Guibas, D. Hsu, and L. Zhang. *H-Walk*: Hierarchical distance computation for moving convex bodies. *Proc. of ACM Symposium on Computational Geometry*, 1999.

[12] G. Hirota, R. Maheshwari and M. Lin. Fast volume-preserving free-form deformation using multi-level optimization. *Proc. Symposium on Solid Modeling and Applications*, 1999.

[13] P. M. Hubbard. Interactive collision detection. In *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[14] J. Klosowski, M. Held, Joseph S. B. Mitchell, K. Zikan, and H. Sowizral. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Trans. Visualizat. Comput. Graph.*, 4(1):21–36, 1998.

[15] M.C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.

[16] M.C. Lin and John F. Canny. Efficient algorithms for incremental distance computation. In *IEEE Conference on Robotics and Automation*, pages 1008–1014, 1991.

[17] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygon environments. In *Proc. of ACM SIGGRAPH*, 1997.

[18] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Computing*, 12:pp. 759–776, 1983.

[19] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[20] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329, 1994.

[21] H. Samet. *Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods*. Addison Wesley, 1989.

[22] P. Schröder and D. Zorin. Subdivision for modeling and animation. *ACM SIGGRAPH Course Notes*, 1998.

[23] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[24] E. Stollnitz, T. Derose, and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, 1996.

[25] Tiow-Seng Tan, Ket-Fah Chong, and Kok-Lim Low. Computing bounding volume hierarchies using model simplification. In *ACM Symposium on Interactive 3D Graphics*, pages 63–70, 1999.