Proceedings of the 1997 IEEE
International Conference on Robotics and Automation
Albuquerque, New Mexico - April 1997

Fast Swept-Volume Distance for Robust Collision Detection*

Patrick G. Xavier
Sandia National Laboratories, Albuquerque NM 87185-1008

Abstract: The need for *collision detection* arises in several robotics areas, including motion-planning, online collision avoidance, and simulation. At the heart of most current methods are algorithms for *interference detection* and/or *distance computation*. A few recent algorithms and implementations are very fast, but to use them for accurate collision detection, very small step sizes can be necessary, reducing their effective efficiency. We present a fast, implemented technique for doing exact distance computation and interference detection for translationally-swept bodies. For rotationally swept bodies, we adapt this technique to improve accuracy, for any given step size, in distance computation and interference detection. We present preliminary experiments that show that the combination of basic and swept-body calculations holds much promise for faster accurate collision detection.

1 Introduction

1.1 Overview

Collision detection is a basic problem in robotics and related areas, arising in motion planning, control, graphical programming, motion-preview, virtual reality, and dynamical simulation. The collision detection problem asks whether a rigid body moving along a given path intersects with any of a set of obstacles at any point on that path. In a fuller version of the problem, all contacts must also be determined. In both cases, accuracy is of extreme importance when the results of collision detection between modeled objects affect the behavior of physical robots or influence the outcomes of physical simulations, such as those used in process and product design and evaluation.

Most current methods for collision detection rely on *interference detection* and/or *distance computation*. A few recent algorithms are very fast, but to use them for accurate collision detection, very small step sizes between queries become necessary because of the difficulty in determining whether a collision occurs between consecutive interference or distance computations. This reduces effective efficiency.

We recently developed a technique, based on a novel use of Gilbert's Algorithm [12], for computing exact distances between linear-translationally swept rigid bodies. For bodies swept with a rotational component, we have extended this technique to improve accuracy, for any given step size, in distance computation. The same techniques trivially suffice for interference detection between the swept bodies. In both cases, the bodies can be non-convex, and the combination of basic and swept-body distance/interference calculation is suitable for fast, accurate collision detection. We report on our algorithms, our implementation, and preliminary tests involving 10K-50K polygon examples that illustrate the potential of our swept-body distance techniques.

1.2 Background: Collision, Interference, and Distance

We first review definitions of interference detection, distance computation, and collision detection for a single moving body. *Interference detection* (or *clash detection*) for a rigid body $\mathcal{R}$ at a position and orientation $\mathbf{x}$ and obstacles $\mathcal{O}$ means to determine whether $\mathcal{R}(\mathbf{x}) \cap \mathcal{O}$ is non-empty, with $\mathcal{R}(\mathbf{x})$ denoting the image of $\mathcal{R}$ in the world. *Distance computation* means to determine the minimum distance between all points in $\mathcal{R}(\mathbf{x})$ and those in $\mathcal{O}$. Simple *collision detection* for a rigid body $\mathcal{R}$ moving

*This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

0-7803-3612-7-4/97 $5.00 © 1997 IEEE

over a path segment among obstacles $\mathcal{O}$ means to determine whether at any point along the path the body contacts or intersects any obstacles. In other words, if we let $\mathcal{C}$ denote the space of positions and orientations and let $\mathbf{p} : [0,1] \rightarrow \mathcal{C}$ denote the path segment, then collision detection asks whether there is any $t \in [0,1]$ such that $\mathcal{R}(\mathbf{p}(t)) \cap \mathcal{O}$ is non-empty.

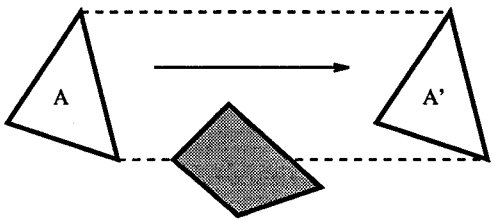

Figure 1: Between queries, the triangle moves linearly from A to A'. Because there is no interference with the obstacle at the query times, simple interference checking fails to detect the collision with the obstacle. Basic distance queries at the query points would indicate that it might be possible that there is a collision, but further computations would be needed to decide. The swept hull of the moving triangle intersects the obstacle, so a swept-body interference check would detect the collision.

Exact or accurate collision detection is often avoided for the sake of speed. Usually, either interference detection or distance computation queries are applied at closely spaced points on the path. Simple use of interference detection can obviously miss collisions. (See Figure 1.) Growing the objects by a safety margin can prevent this, but at the risk of detecting false collisions. With distance computation, one can vary the step size to reduce the number of queries needed to obtain a given resolution, but resolution is limited to the minimum step size. Alternatively, one can collect all pairs of polygons closer than the step size and then apply more sophisticated methods to these pairs, and our discussion attempts to account for this.

1.3 Results

We present results that help to bridge the gaps between robust space-time methods for collision detection (e.g., [4, 7]) and fast exact interference detection (e.g., [13]) and distance computation (e.g., [22]) techniques. Specifically, we present an easily implemented technique for exact distance computation for linear-translationally swept bodies. We then extend this technique to the improve accuracy, for any given step size, in distance computation and interference detection for rotationally swept bodies. Our techniques enable conservative approximations when desired. We assume that all bodies, previous to sweeping, are represented by boundaries composed of convex polygons and/or by unions of convex polygons and polyhedra. We do not require the bodies to be manifolds.

In Section 2 we describe our techniques, which combine a convex-hull-based hierarchy with a previously unreported use of Gilbert's Algorithm. In Section 3 we present preliminary experimental results obtained using our implementation. We compare the linear-translational swept-body technique against its non-swept-body counterpart in both distance computation and interference detection. Both are implemented in our geometry library, the C-Space Toolkit. Our test scenarios have 10K to 50K polygons and include semi-pathological cases with over 500 pairs of polygons in contact per step.

1.4 Previous and Related Work

We review some of the extensive body of work on interference detection and distance computation between two rigid bodies. An overview covering a more complete set of previous and related work (from three research communities) can be obtained by consulting [13–15]. A related and important problem concerns collision detection among members of a collection of objects that move arbitrarily but continuously with respect to each other. For example, see [2, 9, 17, 20].

For a number of years, hierarchical geometric representations have been used to avoid *all pairs comparisons* in interference detection, distance computation, and collision detection. The binary-space partitioning tree (BSP-tree) [11, 21] and variants (e.g., [24]) have been successfully used in exact interference checking and contact determination, but they do not readily yield distance information. The octree [16, 19] is another space-partitioning data-structure that has been used in interference detection (for example, [1]), but it must be used with other geometric objects to obtain computations matching the accuracy of faceted boundary representations or BSP-trees.

Other hierarchical structures use various primitives to bound an object or its surface at each level of the hierarchy, although sometimes holes are treated explicitly, as in [9, 10]. Successful implementations for exact interference detection and distance computation have been based on several geometric primitives. The convex-hull based implementations of [5, 6, 9, 23], the sphere-based techniques of [22], and bounding-prism methods of [13] are among those known to be fast. However, none of these results cover swept-body distance or swept-body interference detection.

Much work has also been done on optimizing the minimum-distance computation between primitives. The convex-hull distance algorithm of [12], referred to as "Gilbert's Algorithm" is used by many of the systems mentioned above. [5, 8] have presented fast incremental convex-hull distance schemes, which exhibit constant-time complexity when the object configurations change only by small amounts between queries. There is potential to adapt these techniques into our system.

[7] describes a technique for exact collision detection for convex polyhedra moving along path segments linear in translation and the quaternion representation of $SO(3)$. Finally, [4] uses space-time techniques and develops algorithms and hierarchical structures for exact collision detection for non-convex objects. Both these results are more general than ours in allowing exact time intervals of contact and penetration to be computed directly, but they are considerably more expensive.

2 Algorithm

We now describe our techniques. First, we review a basic method of hierarchical exact distance calculation. We then describe our extension of this method to exact distance calculation for linear-translationally swept bodies. Finally, we describe two extensions that compute approximate distances for bodies that are swept linearly and rotationally simultaneously.

2.1 Basic Hierarchical Distance Calculation

We represent the boundary of a body with a bounding-volume hierarchy generated with a variant of our algorithm described in [25]. The hierarchy is a binary tree whose nodes each contain a convex polygon or convex polyhedron. The subtree rooted at a node represents the union of the primitives at its leaves. Thus, each node of our hierarchical geometric representation contains a *conservative approximation*, or *wrapper*, of the object represented by its subtree. In particular, our trees contain a convex hull (polyhedron) at each interior node, and a convex polygon or convex polyhedron at each leaf.

To perform basic distance computation, we use a recursive algorithm similar to those of [22] and others. (See Figure 2.) At each stage we consider a pair of nodes, one from each tree. We begin at the roots, with the distance *dist* set at infinity. In the base case, we simply calculate the distance between two convex primitives, and set *dist* to this distance if it is smaller than the current value. In

Simple Basic Distance

```
real cstkDist(body *R, body *S)
{
    real dist ← ∞;
    pairStack stack;
    body *b1,*b2;
    stack.push(R,S);
    while (!stack.isEmpty()) {
        stack.pop(&b1,&b2);
        if (isLeaf(b1) ∧ isLeaf(b2))
            dist ← min(dist,primDist(b1,b2));
        else if (hullDist(b1,b2) > dist)
            continue;
        else if (isLeaf(b1) ∨
                (!isLeaf(b2) ∧ len(b1) < len(b2))) {
            stack.push(b1, b2→child1);
            stack.push(b1, b2→child2);
        } else {
            stack.push(b1→child1, b2);
            stack.push(b1→child2, b2);
        }
    }
    return dist;
}
```

Figure 2: This pseudo-code gives the simple basic algorithm, using a stack to implement recursion. It should look familiar to many readers.

the recursive case, if the distance between the hulls of the current nodes is no greater than *dist*, then we recurse using the current node from one tree paired with each of the children of the current node from the other; otherwise, we cut the current branch of the recursion. The *len(b1) < len(b2)* test is a heuristic, comparing principal axis lengths to decide which one of *b1* and *b2* should be descended.

Both *hullDist(..)* and *primDist(..)* compute the distance between two convex polygons or polyhedra. This can be efficiently done with Gilbert's Algorithm [12].

2.2 Hierarchical Distance Calculation for a Translationally-Swept Body

We now make three observations.

First, although Gilbert's Algorithm is usually applied to convex polyhedra and convex polygons, it can do more: given two finite sets of points, it computes a minimum distance vector between their convex hulls. To see that this is true without going into the details, we observe that points in the interior of the hulls are never returned by the

support functions in [12].

Second, the convex hull of the vertices of a polyhedron (polygon) and their images under a translation $\mathbf{x}$ is equal to the swept hull of that polyhedron (polygon) under translation $\mathbf{x}$. Thus, we can use Gilbert's Algorithm to compute the distance between the volumes swept by two polygons or polyhedra under linear translations. We do this by adding the new positions of the vertices to the lists of input vertices.

Third, if A, B, and C are convex and $A \cup B \subset C$,then

$$sweep(A,\mathbf{x}) \cup sweep(B,\mathbf{x}) \subset sweep(C,\mathbf{x}).$$

This means that the *conservative approximation* nature of our hierarchies is preserved under translation.

Together, this means we can do the following to the algorithm in Figure 2. First, we add vectors $\mathbf{x}$ and $\mathbf{y}$, which give the translations to be applied to R and S, to the parameter list. We then replace the function calls *primDist(b1,b2)* and *hullDist(b1,b2)* with the call *transGilbert(b1,*$\mathbf{x}$*, b2,* $\mathbf{y}$*)*, which calls Gilbert's Algorithm on the vertices of *b1* unioned with their images under translation $\mathbf{x}$, and on the vertices of *b2* unioned with their images under translation $\mathbf{y}$.

These extensions are clearly easy to implement. To see heuristically that the extended distance algorithm should be fast, we first observe that the divide-and-conquer effects of the geometric hierarchy are still valid in $d-1$ of d dimensions, i.e., in the directions normal to the sweep direction. Second, the loss of divide-and-conquer effectiveness in the sweep direction varies roughly with the ratio of the sweep length to the diameter of the environment in the sweep direction. Finally, since the number of vertices in each call to Gilbert's algorithm only doubles, the cost of each call should at most double or triple in typical cases, in which Gilbert's algorithm is $(O(N \log N)$.

Finally, we observe that when both bodies are in motion, computing the swept-body distance gives a conservative estimate of how close the bodies come during the motion. However, when two bodies each undergo a linear translational motion, the relative motion is also a linear translation. Therefore, we can transform the problem to one with one linearly translating body and one stationary body, and get an exact answer.

2.3 A Rotational Sweep Approximation

For motions that include a small single-axis rotation,[1] we describe two approximation techniques — a simple one that might underestimate or overestimate the distance, and a conservative one that will never overestimate it but might underestimate it. Both require the rotation angle to be less than $\frac{\pi}{2}$.

The simple approximation technique replaces the function calls to *primDist(b1,b2)* and *hullDist(b1,b2)* with calls to a function that calls Gilbert's Algorithm on the unions of the vertices of *b1* and *b2* and their final images under the sweep motions, just as in the translational case. By convexity, it suffices to consider the case of a line segment. First, we consider a vertex rotated by angle θ about an axis a distance r away. (See Fig. 3a.) The distance

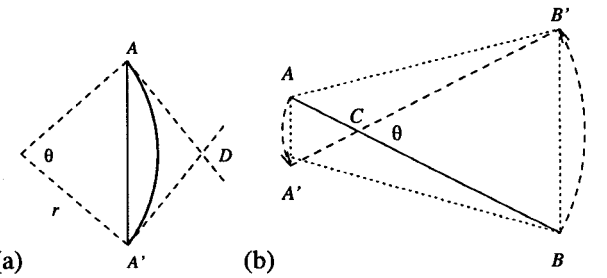

Figure 3: (a) Line segment $\overline{AA'}$ approximates arc AA'. If segments $\overline{AD}$ and $\overline{DA'}$ are tangents, then triangle ADA' bounds the arc. (b) Line segment $\overline{AB}$ rotates by θ about C; new term must bound the maximum distance between segment $\overline{AB'}$ (or $\overline{A'B}$) and the union of sectors ACA' and BCB'.

between the arc the vertex follows and the line segment between its original and final locations is bounded by

$$r\left(1-\cos\left(\frac{\theta}{2}\right)\right). \tag{1}$$

For a line segment that has projected length l perpendicular to the axis of rotation, we bound the maximum distance between the region swept by the segment and the (4-vertex) convex hull of its vertices at their original and final locations. This is (1) plus the greatest minimum distance possible between an axis of rotation on the segment and the boundary of the 4-vertex convex hull (see Fig. 3b):

$$r\left(1-\cos\left(\frac{\theta}{2}\right)\right)+\frac{l}{2}\sin\theta. \tag{2}$$

(2) thus bounds the error in distance computation that can result. Compared to the $r\sin(\frac{\theta}{2})$ error bound resulting from just computing distance before and after the rotation, we see that the part of the error dependent on rotational radius r decreases quadratically with the rotation angle instead of linearly. (Consider the series expansions.)

A more sophisticated technique, which builds on one from [18], is similarly bounded in error but conservative,

[1] A mathematical description of the transformation is given by (4).

returning distances no greater than the actual distance. In the planar case, the position of each vertex during motion is bounded by the triangle formed by its initial and final positions and the intersection of the tangents to the path at those positions. (See Fig. 3a.) The distance from the two new edges to the arc is bounded by $r \tan\left(\frac{\theta}{2}\right) \sin\left(\frac{\theta}{2}\right)$, which is greater than (1). Substituting this into (2), we obtain an error bound for the line segment case:

$$r \tan\left(\frac{\theta}{2}\right) \sin\left(\frac{\theta}{2}\right) + \frac{l}{2} \sin\theta. \tag{3}$$

Now, suppose that a vertex $\mathbf{p}$ is subjected to a simultaneous translation normal to the rotation axis, so that its position for $s \in [0, 1]$ is given by

$$\mathbf{R}_{\mathbf{n}_\theta}(s\theta)(\mathbf{p} - \mathbf{r}_0) + \mathbf{r}_0 + s\mathbf{x}, \tag{4}$$

where $\mathbf{R}_{\mathbf{n}_\theta}(\alpha)$ is the rotation matrix for angle α about $\mathbf{n}_\theta$ and the original axis of rotation goes through $\mathbf{r}_0$. Then the same construction bounds the motion of the vertex, and the error bound for a line segment remains (3).

The case with a translational component parallel to the rotation axis is similar, but requires a four-vertex hull instead of a triangle for each vertex. We again use the tangents to the vertex's path at the endpoints, but instead find their intersections with the plane bisecting the line segment between the initial and final vertex positions. The four-vertex hull then bounds the path. A simple but not tight error bound doubles (4), so that for a swept line segment of projected length l, we have the error bound,

$$2r \tan\left(\frac{\theta}{2}\right) \sin\left(\frac{\theta}{2}\right) + l \sin\theta. \tag{5}$$

Note that this bound *is independent of the translation* $\mathbf{x}$.

Now, let $\mathcal{H}(\mathbf{p}, \mathbf{n}_\theta, \mathbf{r}_0, \mathbf{x})$ denote the vertices so constructed for a vertex $\mathbf{p}$. Then for a convex polygon with vertex set $\mathcal{V}$, the convex hull of the set of vertices given by

$$\bigcup_{\mathbf{p}_i \in \mathcal{V}} \mathcal{H}(\mathbf{p}_i, \mathbf{n}_\theta, \mathbf{r}_0, \mathbf{x})$$

bounds the volume swept by the polygon. By convexity, the distance error bound (5) for the line-segment case also holds for a polygon(dron) that projected has diameter l perpendicular to the rotation axis. This is particularly significant because l is typically much smaller than r, and because a more careful analysis often eliminates the $l \sin\theta$ term when it is not.

Using $\mathcal{H}$ to expand the vertex sets in the calls to Gilbert's Algorithm in the hierarchical distance algorithm results in a conservative distance approximation algorithm. An error bound can be computed from (5) and the l and r for the closest pair of polygons (polyhedra) found. The expanded vertex sets and the distance underestimation makes the cost greater than the simpler approximation technique, but guarantees conservative results.

3 Implementation, Examples, and Discussion

3.1 Implementation

The algorithms sketched in Section 2 have been implemented in C++ as a part of our geometry engine, the C-Space Toolkit. Our implementation contains several optimizations that we describe briefly in the Appendix and is still under development. We use the Quickhull code [3] from the University of Minnesota for computing convex hulls. The runs described in this paper were done on an SGI Indigo2 R4400/250.

To get a rough idea of our implementation's speed, we tested it on the "Complex Torus" motion sequences [13] from the U. North Carolina at Chapel Hill, featuring a 20K-polygon body moving along a 98K polygon environment. It averaged 44Hz when performing complete interference detection between the bodies (collecting all pairs of interfering polygons), and at 31Hz when doing distance computation.

3.2 Interference Detection and Distance Computation: Basic Versus Swept-Body

In this section and the next we present examples comparing the performance of our swept-body distance code with the basic distance code. We include interference detection computations, which we implement as distance computation subject to maximum and minimum thresholds. We consider only linear translations; the rotational case will be covered in future work.

Our examples covered two model scenarios. In the first (Figure 4), coded "FT", a truck chassis (10K polygons) moves through an environment containing other truck chassis and a teapot (42K polygons total). The objects are up to 10 units in scale, and the translational path is 20 units long. Zero-distance occurs near both ends of the path, and the interference and distance queries are typical of gross-motion problems or free-flight. In the second scenario (Figure 5), coded "GPM", a gear (3920 poly-

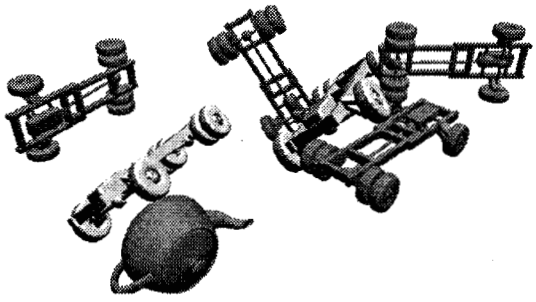

Figure 4: FT, an example typical of gross-motion problems or free-flight. The light-colored truck chassis translates from right to left, shown at path start and finish.

gons) drops through a hole in a flat plate (4054 polygons) and onto the spindle of a motor (2252 polygons). The motor is roughly unit scale, and the translational path is one unit long. The clearance between the gear and the motor spindle is 0.00125 units. Except where noted, an initial

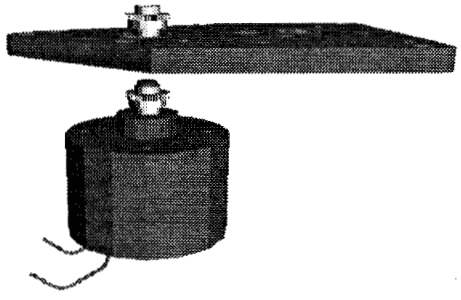

Figure 5: GPM, an example with tight tolerances and semi-pathological difficulty due to concavities and symmetries. The gear is shown at the middle and the end of its path downwards.

call is always made to the distance function to initialize the caches, so that the runs reflect steady-state behavior.

Our first series of experiments compares the costs of four types of queries: basic distance (D), interference detection (I), swept-body distance (DS), and swept-body interference detection (IS). For each model scenario, we broke the motion into 25-400 steps. The results are shown in Table 1. They show swept-body interference detection and distance computation costing up to 30-50 percent more time than their basic counterparts. Swept-body interference detection was typically half as expensive as basic distance computation.

An interesting observation is that despite the much smaller polygon count, the GPM examples are computationally more intensive than the FT examples. Not only are clearances small in the hole and on the spindle, but in

num steps		25	50	100	200	400
scenario	expt	time (secs.)				
FT	D	1.6	1.9	2.2	2.9	5.3
FT	I	.32	.49	.83	1.5	2.7
FT	DS	1.5	1.9	2.5	3.8	6.4
FT	IS	.44	.64	1.1	2.0	3.5
GPM	D	2.0	3.3	5.9	10.7	20.9
GPM	I	.59	1.1	1.7	3.1	6.0
GPM	DS	3.1	5.1	9.3	17.3	31.9
GPM	IS	1.0	1.6	2.6	4.6	8.6

Table 1: Distance computation (D) and interference detection (I) in both the basic and swept-body (S) cases. Scenario FT: 10K polygons moving, 42K polygons stationary; scenario GPM: 3.9K polygons moving, 6.3K polygons stationary.

both cases, the geometric symmetry means that there are many pairs of polygons approximately the minimal distance (0.00125) apart. This is a semi-pathological but realistic occurrence. While the gear moves onto the spindle in the last 20-25 percent of the motion, there are typically 500 pairs of polygons that are about the same, minimal distance apart, and distance calculation slows to 10-15Hz. For comparison, if we set the interference threshold to 0.00128, our implementation exploits cached information to detect interference at over 2kHz in that segment.

3.3 Translational Sweeps and Collision Detection

We now consider how basic and swept-body distance computation might be used in robust collision detection.

Two important problems that arise in assembly planning are to determine: (i) whether a linear translation causes interference, and (ii) which pairs of polygons interfere during the motion. Basic interference detection can only answer (i) and (ii) to a resolution equal to the step size. Accuracy is attainable if both the step size and the interference threshold (safety margin) can be precisely manipulated. Basic distance algorithms have the advantage of efficient step size estimates. Finally, both (i) and (ii) can also be reduced to polygon-polygon collision detection problems by stepping through the motion and collecting all polygon pairs less than a step size apart.

In contrast, a single swept-body interference query is sufficient to answer (i) exactly. On an implementation that can automatically collect all pairs of interfering polygons (i.e., *witness pairs*), a single query is also sufficient for (ii). Because the translational swept-body distance algo-

rithm is exact, it is not necessary to step through the motion. In the GPM scenario, our implementation required 0.45 seconds each for (i) and (ii) when there was no contact (ϵ = 0.00124), and 0.45 seconds for (i) and .89 seconds (1335 witness pairs) for (ii) when there was contact (ϵ = 0.00126). For this test, no cache initialization queries were done, since it is likely that this sort of query would be done "cold" in practice.

Sequential-motion collision detection queries occur in dynamical simulation and other applications. An additional problem that arises is (iii) to determine contact initiation and termination points during a timestep. Computing a conservative approximation of (ii) — finding a guaranteed superset of polygon pairs that interfere sometime during a step — reduces this problem to a collection of polygon-polygon problems.[2]

For an example of this sub-task, we considered two cases of the gear-plate-motor example, one in which the 0.00125 unit clearance is considered interference/collision (ϵ = 0.00128), and one in which it is not. We conducted five experiments. (See Table 2.) Using basic thresholded interference detection with witness collection, we set the interference threshold to half the step size (TW) and to the same value plus ϵ (TWϵ). Swept-body interference detection with witness collection was tested both with the ϵ (ISWϵ) and without (ISW). Our swept-body tests use a hybrid implementation — the swept-body code is triggered when basic code would collect witnesses because of the interference threshold. This accounts for GPM/ISW being faster than GPM/IS (see Table1). We also ran the swept-body interference detection with the ϵ but without witness collection (ISϵ).

num steps		25	50	100	200	400
scenario	expt	time (secs.)				
GPM	TW	3.7	3.6	5.7	9.8	7.8
GPM	TWϵ	3.7	3.7	5.8	9.4	17.8
GPM	ISW	.83	1.2	2.1	3.5	7.8
GPM	ISϵ	.44	.6	.84	1.49	2.85
GPM	ISWϵ	1.9	3.0	5.1	8.7	16.8

Table 2: Some queries that might be used in collision detection.

We see two important comparisons. First, we compare GPM/ISW and GPM/ISϵ against GPM/TW and GPM/TWϵ to consider how useful swept-body interference detection might be in answering problem (i) — determining whether a collision takes place during a motion step. The data shows the swept-body method to be much faster except when there is no interference and the step size is smaller than the clearance. To consider efficiency in collecting witnesses for accurately answering problems (ii) and (iii), we compare GPM/ISW and GPM/ISWϵ to GPM/TW and GPM/TWϵ. We find that the basic code is several times as costly as the swept-body code when the former must collect many polygon pairs that the latter safely rules out, and that the two methods cost about the same when they find similarly high numbers of collision candidates. Given the algorithms and analysis in Section 2.3, we expect that these more general algorithms could similarly increase performance in accurate collision detection for motions with a rotational component.

[2]These, in turn could be solved by using sub-steps, binary search and our swept-polygon distance algorithm, or a space-time method [4, 7].

4 Conclusions and Future Work

We have presented methods for extending basic hierarchical distance computation to swept-body distance computation, both in linear translational and combined translational and rotational sweeps. Our methods are exact for the translational case, and include an improved conservative approximation in the rotational case. The methods apply Gilbert's Algorithm in a simple but previously unreported way. They have been implemented as a part of our geometry library, the C-Space Toolkit.

We have presented simple experiments for the linear-translational case comparing the swept-body and basic techniques in distance computation, interference detection, and collision detection. These experiments indicate that computing linear-translational swept-body distance is no more than 50 percent more expensive than the basic technique in practice, and that our methods hold the potential to speed up robust collision detection.

We are currently building a fast, robust, full-contact collision detection system around our basic and swept-body distance code. In addition, we see room for speeding up our code by upgrading our implementation of Gilbert's Algorithm to use incremental computations, e.g., [5, 23]. We hope soon to report soon on improved results.

Acknowledgements

The author thanks Randy Wilson, Peter Watterberg (Sandia), Philip Hubbard (Washington U. at St. Louis), and Ming Lin and Stephan Gottschalk (UNC) for providing the geometric models used in our experiments. The

author also thanks Randy Wilson and Arlo Ames (Sandia) for their comments and suggestions.

A Main Optimizations in the C-Space Toolkit

We list five main optimizations our distance-computation code uses that result in much greater speed over the algorithm in Figure 2. They will be reported in detail in a forthcoming publication. (1) For a given body pair, our implementation caches which leaf pair results in the minimum distance; on the subsequent call, it uses this pair in intially bounding the distance. (2) Our version of Gilbert's Algorithm takes an optional *maxdist* argument and can cut short computation if the actual distance exceeds it. (3) Our implementation keeps a *cut-table* of pairs of interior nodes for which it believes a comparison against the current best distance might cause a cut in the recursion; the pre-recursion check is only done on node-pairs in this table. (4) At interior nodes, our hierarchies use convex hulls with no more than some *a priori* fixed number of vertices. (5) Our implementation caches minimum-distance simplices found in calls to Gilbert's Algorithm and uses them as initial simplices the when same arguments are encountered again; this was suggested in [12].

References

[1] S. Arimoto, H. Noborio, S. Fukuda, and A. Noda. A feasible aprach to automatic planning of collision-free robot motions. In *Int'l Symp. on Robotics Reasearch*, 1988.

[2] D. Baraff. Curved surfaces and coherence for nonpenetrating rigid body simulation. *Computer Graphics* (Proc. SIGGRAPH), 24(4):19–28, August 1990.

[3] C. B. Barber, D .P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. To appear in *ACM Trans. on Mathematical Software*. Also appears as Tech. Rept. GCG 53, Geometry Center at U. Minnesota, Minnesota, MN, 1993.

[4] S. Cameron. Collision detection by 4D intersection testing. *Int'l Journal of Robotics Research*, 6(3):291–302, June 1990.

[5] S. Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *submitted to IEEE Trans. on Robotics and Automation*, July 1996.

[6] S. Cameron. Dealing with geometric complexity in motion planning. In *IEEE ICRA 1996 Workshop on Practical Motion Planning in Robotics*, Minneapolis, MN, 1996.

[7] J. Canny. Collision detection for moving polyhedra. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(2):200–209, 1986.

[8] J. Canny and M. C. Lin. A fast algorithm for incremental distance calculation. In *Proc. IEEE ICRA 1991*, pages 1008–1014, Sacremento, California, 1991.

[9] J. Cohen, M. Lin, D. Manocha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. ACM Interactive 3D Graphics Conf.*, pages 189–196, 1995.

[10] B. Faverjon. Hierarchical object models for efficient anti-collision algorithms. In *Proc. IEEE ICRA 1989*, pages 333–340, Scottsdale, AZ, 1989.

[11] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. In *Proc. of ACM SIGGRAPH*, pages 124–133, 1980.

[12] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2), April 1988.

[13] S. Gottschalk, M.C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proc. ACM SIGGRAPH '96*, 1996.

[14] M. Held, J. Klosowski, and J. S. B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Proc. 7th Canadian Conf. on Computational Geometry*, Quebec, Canada, 1995.

[15] P. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. on Graphics*, 15(3), July 1996.

[16] C. L. Jackins and S. L. Tanimoto. Octtrees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, (14):249–270, 1980.

[17] M. Lin, D. Manocha, and J. Canny. Fast contact determination in dynamic environments. In *Proc. IEEE ICRA 1994*, pages 602–607, San Diego, CA, May 1994.

[18] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. on Computers*, C–32(2):108–120, 1983. Also MIT A.I. Memo 605, December 1982.

[19] D Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, (19):129–147, 1982.

[20] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proc. Monterey Symp. on Real-Time Interactive Graphics*, April 1995.

[21] B. Naylor. Interactive solid geometry via partitioning trees. In *Proc. of Graphics Interface*, pages 11–18, May 1992.

[22] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. 1994 IEEE Int'l Conf on Robotics and Automation*, San Diego, CA, 1994.

[23] Y. Sato, M. Hirata, T. Maruyama, and Y. Arita. Efficient collision detection using fast distance calculation algorithms for convex and non-convex objects. In *Proc. 1995 IEEE Int'l Conf. on Robotics and Automation*, pages 772–778, Minneapolis, MN, April 1996.

[24] G. Vanecek. Brep index, a multi-dimensional space partitioning tree. In *Proc. 1st ACM-SIGGRAPH Symp. on Solid Modeling Foundations and CAD/CAM Applications*, pages 35–44, Austin,TX, June 1991.

[25] P. Xavier. A generic algorithm for constructing hierarchical representations of geometric objects. In *Proc. 1996 IEEE Int'l Conf. on Robotics and Automation*, pages 3644–3651, Minneapolis, MN, April 1996.