A Framework for Fast and Accurate Collision Detection for Haptic Interaction

Arthur Gregory Ming C. Lin Stefan Gottschalk Russell Taylor
Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{gregory,lin,stefan,taylorr}@cs.unc.edu

Abstract

We present a framework for fast and accurate collision detection for haptic interaction with polygonal models. Given a model, we pre-compute a hybrid hierarchical representation, consisting of uniform grids and trees of tight-fitting oriented bounding box trees (OBB-Trees). At run time, we use hybrid hierarchical representations and exploit frame-to-frame coherence for fast proximity queries. We describe a new overlap test, which is specialized for intersection of a line segment with an oriented bounding box for haptic simulation and takes 6-36 operations excluding transformation costs. The algorithms have been implemented as part of H-COLLIDE and interfaced with a PHANToM arm and its haptic toolkit, GHOST*, and applied to a number of models. As compared to the commercial implementation, we are able to achieve up to 20 times speedup in our experiments and sustain update rates over 1000Hz on a 400MHz Pentium II.*

1 Introduction

Virtual environments require natural interaction between interactive computer systems and users. Compared to the presentation of visual and auditory information, methods for haptic display are not as well developed. Haptic rendering as an augmentation to visual display can improve perception and understanding both of force fields and of world models populated in the synthetic environments [6]. It allows users to reach into virtual worlds with a sense of touch, so they can feel and manipulate simulated objects.

Haptic display is often rendered through what is essentially a small robot arm, used in reverse. Such devices are now commercially available for a variety of configurations (2D, 3D, 6D, specialized for laparoscopy or general-purpose). The system used in this work was a 6DOF-in/3DOF-out SensAble Technologies PHANToM arm.

"Real-time" graphics applications have display update requirements somewhere between 20 and 30 frames/second. In contrast, the update rate of haptic simulations must be as high as 1000 updates/second in order to maintain a stable system. This rate varies with the spatial frequency and stiffness of displayed forces, and with the speed of motion of the user. Also, the skin is sensitive to vibrations of greater than 500 Hz, so changes in force at even relatively high frequencies are detectable [10].

In order to create a sense of touch between the user's hand and a virtual object, contact or restoring forces are generated to prevent penetration into this virtual object. This is computed by first detecting if a collision or penetration has occurred, then determining the (projected) contact point on the object surface. Most of the existing algorithms are only sufficient to address the collision detection and contact determination problems for relatively small models consisting of only a few thousand polygons or a few surfaces. Our ultimate goal is to be able to achieve smooth, realistic haptic interaction with CAD models of high complexity (normally consisted of tens of thousands of primitives) for virtual prototyping applications. In addition, we are aiming at designing algorithms that are easily extensible to support a wide range of force-feedback devices (including 6 degree-of-freedom arms) and deformable surfaces.

Main Contribution: In this paper we present a framework for fast and accurate collision detection for haptic interaction. It consists of a number of algorithms and a system specialized for computing contact(s) between the probe of the force-feedback device and objects in the virtual environment. To meet the

stringent performance requirements for haptic interaction, we use a *hybrid* approach that specializes many earlier algorithms for this application. Our framework utilizes:

- **Spatial Decomposition:** It decomposes the workspace into uniform grids or cells, implemented as a hash table to efficiently deal with large storage requirements. At runtime, the algorithm can quickly find the cell containing the path swept out by the probe.
- **Bounding Volume Hierarchy based on OBBTrees:** An OBBTree is a bounding volume hierarchy [15] of tight-fitting oriented bounding boxes (OBBs). For each cell consisting of a subset of polygons of the virtual model, we pre-compute an OBBTree. At run-time, most of the computation time is spent in finding collisions between an OBBTree and the path swept out by the tip of the probe between two successive time steps. To optimize this query, we have developed a very fast specialized overlap test between a line segment and an OBB, that takes as few as 6 operations and only 36 arithmetic operations in the worst case, not including the cost of transformation.
- **Frame-to-Frame Coherence:** Typically, there is little movement in the probe position between successive steps. The algorithm utilizes this coherence by caching the contact information from the previous step to perform incremental computations.

The algorithm pre-computes a hybrid hierarchy. Our framework also allows the application program to select only a subset of the approaches listed above.

We have successfully implemented all the algorithms described above, interfaced them with *GHOST* (a commercial haptic library) [34] and used them to find surface contact points between the probe of a PHANToM arm and large geometric models (composed of tens of thousands of polygons). Their performance varies based on the geometric model, the configuration of the probe relative to the model, machine configuration (e.g. cache and memory size) and the combination of techniques used by our system. The overall approach results in a factor of $2-20$ speed improvement as compared to earlier algorithms and commercial implementations. For a number of models composed of $5,000-80,000$ polygons, our system is able to substain a KHz update rate on a 400M Hz PC.

The results presented in this paper are specialized for a point probe against 3D object collision detection. We conjecture that it can be extended to compute object-object intersection for a six-degree-of-freedom haptic device.

Organization: The rest of the paper is organized in the following manner. Section 2 provides a brief survey of related research. Section 3 describes the system architecture and algorithms used in the design of our system. We discuss the implementation issues in Section 4, present our experimental results and compare their performance with a commercial implementation in Section 5.

2 Related Work

Collision detection and contact determination are well-studied problems in computer graphics, computational geometry, robotics and virtual environments. Due to limited space, we refer the readers to [22, 17] for recent surveys. In the ray-tracing literature, the problem of computing fast intersections between a ray and a three-dimensional geometric model has also been extensively studied [1]. While a number of algorithms have been proposed that make use of bounding volume hierarchies, spatial partitioning or frame-to-frame coherence, there is relatively little available on hybrid approaches combining two or more such techniques.

Bounding Volume Hierarchies: A number of algorithms based on hierarchical representations have been proposed. The set of bounding volumes include spheres [18, 30], axis-aligned bounding boxes [5, 17], oriented bounding boxes [15, 4], approximation hierarchies based on S-bounds [7], spherical shells [21] and k-dop's [20]. In the close proximity scenarios, hierarchies of oriented bounding boxes (OBBTrees) appear superior to many other bounding volumes [15].

Spatial Partitioning Approaches: Some of the simplest algorithms for collision detection are based on spatial decomposition techniques. These algorithms partition the space into uniform or adaptive grids (i.e. volumetric approaches), octrees [33], k-D trees or BSP's [27]. To overcome the problem of large memory requirements for volumetric approaches, some authors [29] have proposed the use of hash tables.

Utilizing Frame-to-Frame Coherence: In many simulations, the objects move only a little between successive frames. Many efficient algorithms that utilize frame-to-frame coherence have been proposed for convex polytopes [23, 8, 3]. Cohen et al. [9] have used coherence-based incremental sorting to detect possible pairs of overlapping objects in large environments.

Research in Haptic Rendering: Several techniques have been proposed for integrating force feedback with a complete real-time virtual environment to enhance the user's ability to perform interaction tasks

[10, 11, 12, 24, 25, 28, 35]. The commercial haptic toolkit developed by SensAble Technologies, Inc. also has a collision detection library probably using BSP-Trees [32, 34]. Ruspini et al. [31] have presented a haptic interface library "HL" that uses a multi-level control system to effectively simulate contacts with virtual environments. It uses a bounding volume hierarchy based on sphere-trees [30].

Nahvi et al. [26] have designed a haptic display system for manipulating virtual mechanisms derived from a mechanical CAD design. It uses the Sarcos Dexterous Arm Master and Utah's Alpha_1 CAD system, with algorithmic support from a tracing algorithm and a minimum distance framework developed by Johnson, Cohen et al. [36, 19]. They utilized a variety of algorithmic toolkits from RAPID [15] to build an OBBTree for each object, Gilbert's algorithm [14] to find distances between two OBB's and a tracing algorithm for parametric surfaces. Their system takes about $20-150$ milliseconds for models composed of $500-23,000$ triangles on an SGI Indigo2 and 4 milliseconds for models composed of 3 parametric surfaces on a Motorola 68040 microprocessors.

Gibson [13] and Sobierajski [2] have proposed algorithms for object manipulation including haptic interaction with volumetric objects and physically-realistic modeling of object interactions.

3 Fast Proximity Queries for Haptic Interaction

In this section, we describe the haptic system setup and algorithmic techniques that are an integral part of the collision detection system framework for haptic interaction, H-COLLIDE.

3.1 Haptic System Architecture

Due to the stringent update requirements for real-time haptic display, we run a special stand-alone haptic server written with the VRPN library (http://www.cs.unc.edu/Research/nano/manual/vrpn) on a PC connected to the PHANToM. The client application runs on another machine, which is typically the host for graphical display. Through VRPN, the client application sends the server the description of the scene to be haptically displayed, and the server sends back information such as the position and orientation of the PHANToM probe. The client application can also modify and transform the scene being displayed by the haptic server.

3.2 Algorithm Overview

Given the last and current positions of the PHANToM probe, we need to determine if it has in fact passed through the object's surface, in order to display the appropriate force. The probe movement is usually small due to the high haptic update rates. This implies that we only need to check a relatively small volume of the workspace for collision detection.

Approaches using spatial partitioning seem to be natural candidates for such situations. For large and complex models, techniques based on uniform or adaptive grids can be implemented more efficiently using hash tables. However, to achieve the desired speed, these approaches still have extremely high storage requirements even when implemented using a hashing scheme.

Despite its better fit to the underlying geometry, the hierarchical bounding volume method based on OBBTrees may end up traversing trees to great depths to locate the exact contact points for large, complex models. To take advantage of each approach and to avoid some deficiency of each, we propose a hybrid technique.

Hybrid Hierarchical Representation: Given a virtual environment containing several objects, each composed of tens of thousands of polygons, the algorithm computes a *hybrid hierarchical representation* of the objects as part of the off-line pre-computation. It first partitions the entire virtual workspace into coarse-grain uniform grid cells. Then, for each grid cell containing some primitives of the objects in the virtual world, it computes the OBBTrees for that grid cell and stores the pointer to the associated OBBTrees using a hash table for constant-time proximity queries.

Specialized Intersection Tests: The on-line computation of our collision detection system consists of three phases. In the first phase, it identifies "the region of potential contacts" by determining which cells were touched by the probe path, using the precomputed look-up table. In the second phase, it traverses the OBBTree(s) in that cell to determine if collisions have occurred, using the specialized fast overlap test to be described later. In the third phase, if the line segment intersects with an OBB in the leaf node, then it computes the (projected) surface contact point(s) (SCP) using techniques similar to those in [34, 36].

Frame-to-Frame Coherence: If in the previous frame the probe of the feedback device was in contact with the surface of the model, we exploit *frame-to-frame coherence* by first checking if the last intersected triangle is still in contact with the probe. If so, we cache this contact witness. Otherwise, we check for collision using hybrid hierarchical representation of the objects.

3.3 H-COLLIDE

H-COLLIDE, a framework for fast and accurate collision detection for haptic interaction, is designed based on the hybrid hierarchical representation and the algorithmic techniques described above. Figure 1 shows the system architecture of H-COLLIDE.

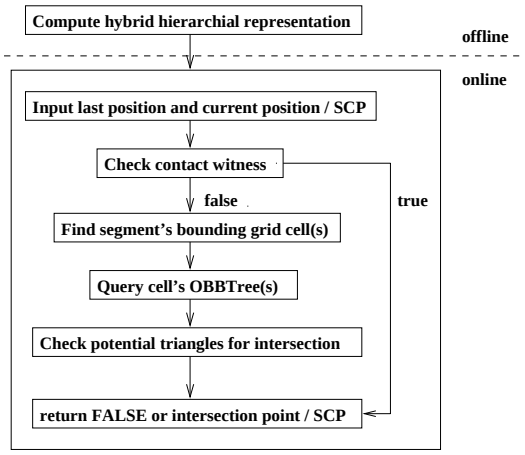

Figure 1. The System Architecture of H-COLLIDE

3.4 Overlap Test based on a Line Segment against an OBBTree

For haptic display using a point probe, we can specialize the algorithm based on OBBTrees by only testing a line segment (representing the path swept out by the probe device between two successive steps) and an OBBTree. (The original algorithm [15] uses a overlap test between a pair of OBBs and can take more than 200 operations per test.) At run time, most of the computation is spent in finding collisions between a line segment and an OBB. To optimize this query, we have developed a very fast overlap test between a line segment and an OBB, that takes as few as 6 operations and only 36 arithmetic operations in the worst case, not including the cost of transformation.

At the first glance, it is tempting to use sophisticated and optimized line clipping algorithms. However, the line-OBB intersection problem for haptic interaction is a simpler one than line clipping and the environment is dynamic and consisting of many OBBs. Next we'll describe this specialized overlap test between a line segment and an oriented bounding box for haptic rendering. Without loss of generality, we will choose the coordinate system centered on and aligned with the box – so the problem is transformed to an overlap test between a segment and a centered axis-aligned bounding box. Our overlap test uses the Separating-Axis Theorem described in [15], but specialized for a line segment against an OBB.

Specifically, the candidate axes are the three box face normals (which are aligned with the coordinate axes) and their cross-products with the segment's direction vector. With each of these six candidate axes, we project both the box and the segment onto it, and test whether the projection intervals overlap. If the projections are disjoint for any of the six candidate axes, then the segment and the box are disjoint. Otherwise, the segment and the box overlap.

How are the projection intervals computed? Given a direction vector v of a line through the origin, and a point p, let the point p' be the axial projection of p onto the line. The value $d_p = v \cdot p/|v|$ is the signed distance of p' from the origin along the line. Now consider the line segment with midpoint m and endpoints $m + w$ and $m - w$. The half-length of the line segment is $|w|$. The image of the segment under axial projection is the interval centered at

$$d_s = v \cdot m/|v|$$

and with half-length

$$L_s = |w \cdot v|/|v|$$

Given a box centered at the origin, the image of the box under axial projection is an interval with midpoint at the origin.

Furthermore, if the box has thicknesses $2t^x, 2t^y$, and $2t^z$ along the orthogonal unit directions u^x, u^y, and u^z, the half-length of the interval is given by

$$L_b = |t^x v \cdot u^x/|v|| + |t^y v \cdot u^y/|v|| + |t^z v \cdot u^z/|v||$$

With the intervals so expressed, the axis v is a separating axis if and only if (see Figure 2)

$$|d_s| > L_b + L_s$$

If we assume that the box is axis-aligned, then $u^x = [1,0,0]^T, u^y = [0,1,0]^T$, and $u^z = [0,0,1]^T$, and the dot products with these vectors become simple component selections. This simplifies the box interval length computation to

$$L_b = |t^x v_x| + |t^y v_y| + |t^z v_z|$$

Now, recall that the candidate axis v is either a box face normal, or a cross product of a face normal with the line segment direction vector. Consider the former case, when v is a box face normal, for example $[1,0,0]^T$. In this case, the components v_y and v_z are zero, and the component v_x is one, and we are left with

$$L_b = t^x$$

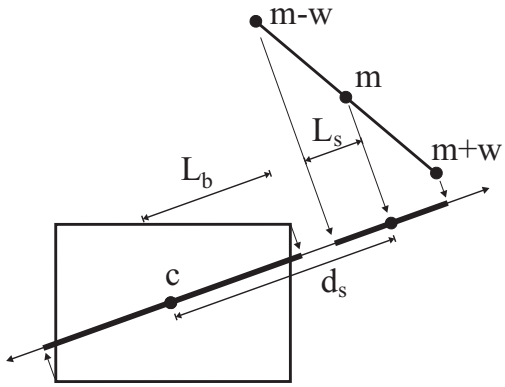

Figure 2. Overlap test between a line segment and an OBB

The projection of the line segment onto the $x-$axis is also simple:

$$L_s = |w_x|$$

So, the test for the $v = [1,0,0]^T$ axis is

$$|m_x| > t^x + |w_x|$$

The tests for the candidate axes $v = [0,1,0]^T$ and $v = [0,0,1]^T$ have similar structure.

The three cases where v is a cross product of w with one of the box faces are a little more complex. Recall that in general,

$$L_b = |t^x v \cdot u^x| + |t^y v \cdot u^y| + |t^z v \cdot u^z|$$

For the sake of concreteness, we will choose $v = w \times u_y$. Then this expression becomes

$$L_b = |t^x (w \times u^y) \cdot u^x| + |t^y (w \times u^y) \cdot u^y| + |t^z (w \times u^y) \cdot u^z|$$

Application of the triple product identity

$$(a \times b) \cdot c = (c \times a) \cdot b$$

yields

$$L_b = |t^x (u^y \times u^x) \cdot w| + |t^y (u^y \times u^y) \cdot w| + |t^z (u^y \times u^z) \cdot w|$$

All of these cross products simplify, because the u vectors are mutually orthogonal, $u^x \times u^y = u^z, u^y \times u^z = u^x$, and $u^z \times u^x = u^y$, so

$$L_b = |t^x (-u^z) \cdot w| + |t^y (0) \cdot w| + |t^z (u^x) \cdot w|$$

And again, using the fact that $u^x = [1,0,0]^T$, and so forth,

$$L_b = t^x |w_z| + t^z |w_x|$$

The half-length of the segment interval is

$$L_s = |w \cdot (w \times u^y)| = |u^y \cdot (w \times w)| = |u^y \cdot 0| = 0$$

which is what we would expect, since we are projecting the segment onto a line orthogonal to it.

Finally, the projection of the segments midpoint falls at

$$d_s = (w \times u^y) \cdot m = (m \times w) \cdot u^y = m_z w_x - m_x w_z$$

which is just the $y-$component of $m \times w$. The final test is

$$|m_z w_x - m_x w_z| > t^x |w_z| + t^z |w_x|$$

Similar derivations are possible for the cases $v = w \times u^x$ and $v = w \times u^z$.

Writing out the entire procedure, and precomputing a few common subexpressions, we have the following pseudo code:

let $X = |w_x|$
let $Y = |w_y|$
let $Z = |w_z|$
if $|m_x| > X + t_x$ return disjoint
if $|m_y| > Y + t_y$ return disjoint
if $|m_z| > Z + t_z$ return disjoint
if $|m_y w_z - m_z w_y| > t_y Z + t_z Y$ return disjoint
if $|m_x w_z - m_z w_x| > t_x Z + t_z X$ return disjoint
if $|m_x w_y - m_y w_x| > t_x Y + t_y X$ return disjoint
otherwise return overlap

When a segment and an OBB are disjoint, the routine often encounters an early exit and only one (or two) out of the six expressions is executed. Total operation count for the worst case is: 9 absolute values, 6 comparisons, 9 add and subtracts, 12 multiplies. This does not include the cost of transforming, i.e. 36 operations, the problem into a coordinate system centered and aligned with the box.

4 Implementation Issues

H-COLLIDE has been successfully implemented in C++. We have interfaced H-COLLIDE with *GHOST*, a commercial software developer's toolkit for haptic rendering, and used it to find surface contact points between the probe of a PHANToM arm and large geometric models (composed of tens of thousands of polygons). Here we describe some of the implementation issues.

4.1 Hashing Scheme

Clearly it is extremely inefficient to allocate storage for all these cells, since a polygonal surface is most likely to occupy a very small fraction of them. We use a hash table to alleviate the storage problem. From each cell location at (x, y, z) and a grid that has len cells in each dimension, we can compute a unique key using

$$key = x + y * len + z * len^2.$$

In order to avoid hashing too many cells with same pattern into the same table location we compute the actual location for a grid cell in the hash table with

$$TableLoc = random(key)\%TableLength.$$

Should the table have too many cells in one table location, we can simply grow the table. Hence, it is possible to determine which triangles we need to check in constant time and the amount of storage required is a constant factor (based on the grid grain) of the surface area of the object we want to "feel".

Determining the optimal grid grain is a non-trivial problem. Please refer to [16] for a detailed retreatment and a possible analytical solution to this problem. We simply set the grain of the grids to be the average length of all edges. If the model has a very irregular triangulation it is very possible that there could be a large number of small triangles in a single grid cell.

Querying an OBBTree takes $O(logn)$ time, where n is the number of triangles in the tree. During the off-line computation, we can ensure that n is a small number compared to the total number of triangles in the model; thus the overall running time of our hybrid approach should be constant.

4.2 User Options

Since the hybrid approach used in H-COLLIDE has a higher storage requirement than either the individual technique alone, the system also allows the user to select a subset of the techniques, such as the algorithm purely based on OBBTrees, to opt for better performance on a machine with less memory.

5 System Performance

For comparison, we have implemented adaptive grids, our hybrid approach and an algorithm using only OBBTrees and the specialized overlap test described in Section 3.4. We have applied them to a wide range of models of varying sizes. (Due to the page limit, we invite the readers to view the Color Plates of these models at http://www.cs.unc.edu/~geom/HCollide/model.pdf.)

Their performance varies based on the models, the configuration of the probe relative to the model, machine configuration (e.g. cache and memory size) and the combination of techniques used by our system. Our hybrid approach results in a factor of 2-20 speed improvement as compared to a native *GHOST* method. For a number of models composed of $5,000 - 80,000$ polygons, our system is able to compute all the contacts and response at rates higher than 1000 Hz on a 400MHz PC.

5.1 Obtaining Test Data

We first obtained the test data set by deriving a class from the triangle mesh primitive which comes with SensAble Technologies' *GHOST* library, version 2.0 beta. This records the start and the endpoint of each segment used for collision detection during a real force-feedback session with a 3-DOF PHANToM arm. We then implemented the three techniques mentioned above to interface with *GHOST* for comparison with a native *GHOST* method, and timed the collision detection routines for the different libraries using the data from the test set. The test set for each of these models contains 30,000 readings.

The distinction between a collision and an intersection shown in the tables is particular to *GHOST*'s haptic rendering. Each haptic update cycle contains a "collision" test to see if the line segment from the last position of the PHANToM probe to its current position has intersected any of the geometry in the haptic scene. If there has been a collision, then the intersected primitive suggests a surface contact point for the PHANToM probe to move towards. In this case it is now necessary to perform an "intersection" test to determine if the line segment from the last position of the PHANToM probe to the suggested surface contact point intersects any of the geometry in the scene (including the primitive with which there was a "collision").

The timings (in milliseconds) shown in Tables 1-5 were obtained by replaying the test data set on a 4 processor 400 MHz PC, with 1 GB of physical memory. Each timing was obtained using only one processor. For comparison, we ran the same suite of tests on a single processor 300 MHz Pentium Pro with 128 MB memory. The hybrid approach appeared to be the most favorable as well.

5.2 Comparison between Algorithms

Since the algorithms run on a real-time system, we are not only interested in the average performance, but also the worst case performance. Tables 1-5 show the

Method	Hash Grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0122	0.00883	0.0120	0.0917
Worst Col. Hit	0.157	0.171	0.0800	0.711
Ave Col. Miss	0.00964	0.00789	0.00856	0.0217
Worst Col. Miss	0.0753	0.0583	0.0683	0.663
Ave Int. Hit	0.0434	0.0467	0.0459	0.0668
Worst Int. Hit	0.108	0.102	0.0793	0.100
Ave Int. Miss	0.0330	0.0226	0.0261	0.0245
Worst Int. Miss	0.105	0.141	0.0890	0.364
Ave. Query	0.019	0.014	0.017	0.048

Table 1. Timings in msecs for Man Symbol, 5K tris

Method	Hash Grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0115	0.0185	0.0109	0.131
Worst Col. Hit	0.142	0.213	0.138	0.622
Ave Col. Miss	0.0104	0.00846	0.0101	0.0176
Worst Col. Miss	0.0800	0.0603	0.0813	0.396
Ave Int. Hit	0.0583	0.0568	0.0652	0.0653
Worst Int. Hit	0.278	0.200	0.125	0.233
Ave Int. Miss	0.0446	0.0237	0.0349	0.0322
Worst Int. Miss	0.152	0.173	0.111	0.287
Ave. Query	0.030	0.025	0.028	0.070

Table 2. Timings in msecs for Man with Hat, 7K tris

Method	Hash Grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0138	0.0101	0.0134	0.332
Worst Col. Hit	0.125	0.168	0.0663	0.724
Ave Col. Miss	0.00739	0.00508	0.00422	0.0109
Worst Col. Miss	0.0347	0.0377	0.0613	0.210
Ave Int. Hit	0.0428	0.0386	0.0447	0.0851
Worst Int. Hit	0.0877	0.102	0.0690	0.175
Ave Int. Miss	0.0268	0.0197	0.0213	0.0545
Worst Int. Miss	0.0757	0.0697	0.0587	0.284
Ave. Query	0.022	0.016	0.039	0.18

Table 3. Timings in msecs for Nano Surface, 12K tris

Method	Hash Grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0113	0.00995	0.0125	0.104
Worst Col. Hit	0.136	0.132	0.177	0.495
Ave Col. Miss	0.0133	0.00731	0.0189	0.0280
Worst Col. Miss	0.128	0.0730	0.137	0.641
Ave Int. Hit	0.0566	0.0374	0.609	0.0671
Worst Int. Hit	0.145	0.105	0.170	0.293
Ave Int. Miss	0.0523	0.0225	0.0452	0.0423
Worst Int. Miss	0.132	0.133	0.167	0.556
Ave. Query	0.027	0.014	0.028	0.048

Table 4. Timings in msecs for Bronco, 18K tris

Method	Hash Grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0232	0.0204	0.0163	1.33
Worst Col. Hit	0.545	0.198	0.100	5.37
Ave Col. Miss	0.00896	0.00405	0.00683	0.160
Worst Col. Miss	0.237	0.139	0.121	3.15
Ave Int. Hit	0.228	0.0659	0.0704	0.509
Worst Int. Hit	0.104	0.138	0.103	1.952
Ave Int. Miss	0.258	0.0279	0.0256	0.229
Worst Int. Miss	0.0544	0.131	0.0977	3.28
Ave. Query	0.030	0.016	0.016	0.320

Table 5. Timings in msecs for Butterfly, 79K tris

timings in milliseconds obtained for both cases on each model and each contact configuration.

All our algorithms are able to perform collision queries at rates faster than the required 1000 Hz force update rate for *all* models in the worst case. Although the hybrid approach often outperforms the algorithm based on OBBTrees, it is sometimes slightly slower than the alogrithm based on OBBTrees. We conjecture that this behavior is due to the cache size of the CPU (independent of the memory size) and memory paging algorithm of the operating system. Among techniques that use hierarchical representations, cache access patterns can often have a dramatic impact on run time performance.

The hybrid approach requires more memory and is likely to have a less cache-friendly memory access pattern than the algorithm purely based on OBBTrees, despite the fact that both were well within the realm of physical memory available to the machine. Furthermore, by partitioning polygons into groups using grids, the hybrid technique can enable real-time local surface modification.

The adaptive grids-hashing scheme, a commonly used technique in ray-tracing, did not perform equally well in all cases. Once again, our hypothesis is that its inferior worst case behavior is due to its cache access patterns, in addition to its storage requirements. We believe the native *GHOST* method uses an algorithm based BSP trees. While it is competitive for the smaller model sizes, its performance fails to scale up for larger models. Our hybrid approach and our algorithm purely based on OBBTrees and the specialized overlap test appear to be relatively unaffected by the model complexity.

6 Conclusion

We have presented a framework, H-COLLIDE, that consists of a suite of algorithms and a system implementation for fast and accurate collision detection for haptic interaction with polygonal models at rates higher than 1000Hz on a desk-top PC. This framework may be extended for supporting 6-DOF haptic devices to perform collision tests between a pair of 3D objects and flexible surfaces that may deform due to manipulation. In addition, it can be combined with the tracing algorithm [36] to handle complex sculptured models more efficiently, by using their control points.

Acknowledgements: We are grateful to National Science Foundation, National Institute of Health National Center for Research Resources and Intel Corporation for their support and the reviewers for their comments.

References

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In *An Introduction to Ray Tracing*, pages 201–262, 1989.

[2] R. S. Avila and L. M. Sobierajski. A haptic interaction method for volume visualization. *Proceedings of Visualization'96*, pages 197–204, 1996.

[3] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, 24(4):19–28, 1990.

[4] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal. Boxtree: A hierarchical representation of surfaces in 3d. In *Proc. of Eurographics'96*, 1996.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

[6] Frederick P. Brooks, Jr., Ming Ouh-Young, James J. Batter, and P. Jerome Kilpatrick. Project GROPE — Haptic displays for scientific visualization. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 177–185, August 1990.

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

[8] Stephen Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics and Automation*, 13(6):915–920, December 1996.

[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. of ACM Interactive 3D Graphics Conference*, pages 189–196, 1995.

[10] J. E. Colgate and J. M. Brown. Factors affecting the z-width of a haptic display. *IEEE Conference on Robotics and Automation*, pages 3205–3210, 1994.

[11] J. E. Colgate and et al. Issues in the haptic display of tool use. *Proceedings of the ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pages 140–144, 1994.

[12] M. Finch, M. Falvo, V. L. Chi, S. Washburn, R. M. Taylor, and R. Superfine. Surface modification tools in a virtual environment interface to a scanning probe microscope. In Pat Hanrahan and Jim Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 13–18. ACM SIGGRAPH, April 1995.

[13] S. Gibson. Beyond volume rendering: Visualization, haptic exploration, an d physical modeling of element-based objects. In *Proc. Eurographics workshop on Visualization in Scientific Computing*, pages 10–24, 1995.

[14] 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.

[15] 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.

[16] A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. H-collide: A framework for fast and accurate collision detection for haptic interaction. Technical report, Department of Computer Science, University of North Carolina, 1998.

[17] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conference on Computational Geometry*, 1995.

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

[19] D. Johnson and E. Cohen. A framework for efficient minimum distance computation. *IEEE Conference on Robotics and Automation*, pages 3678–3683, 1998.

[20] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. In *Siggraph'96 Visual Proceedings*, page 151, 1996.

[21] S. Krishnan, A. Pattekar, M. Lin, and D. Manocha. Spherical shell: A higher order bounding volume for fast proximity queries. In *Proc. of Third International Workshop on Algorithmic Foundations of Robotics*, pages 122–136, 1998.

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

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

[24] William Mark, Scott Randolph, Mark Finch, James Van Verth, and Russell M. Taylor II. Adding force feedback to graphics systems: Issues and solutions. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 447–452, 1996.

[25] T. M. Massie and J. K. Salisbury. The phantom haptic interface: A device for probing virtual objects. *Proc. of ASME Hpatic Interfaces for Virtual Environment and Teleoperator Systems*, 1:295–301, 1994.

[26] A. Nahvi, D. Nelson, J. Hollerbach, and D. Johnson. Haptic manipulation of virtual mechanisms from mechanical cad designs. In *Proc. of 1998 Conference on Robotics and Automation*, pages 375–380, 1998.

[27] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yield polyhedral modeling results. In *Proc. of ACM Siggraph*, pages 115–124, 1990.

[28] M. Ouh-Young. *Force Display in Molecular Docking*. PhD thesis, University of North Carolina, Computer Science Department, 1990.

[29] M. H. Overmars. Point location in fat subdivisions. *Inform. Proc. Lett.*, 44:261–265, 1992.

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

[31] D.C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, pages 345–352, 1997.

[32] K. Salisbury, D. Brock, T. Massie, N Swarup, and C. Zilles. Haptic rendering: Programming touch interaction with virtual objects. *Proc. of 1995 ACM Symposium on Interactive 3D Graphics*, pages 123–130, 1995.

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

[34] Inc. SensAble Technologies. $ghost^{TM}$: Sofware developer's toolkit. *Programmer's Guide*, 1997.

[35] R. M. Taylor, W. Robinett, V.L. Chii, F. Brooks, and W. Wright. The nanomanipulator: A virtual-reality interface for a scanning tunneling microscope. In *Proc. of ACM Siggraph*, pages 127–134, 1993.

[36] T.V. Thompson, D. Johnson, and E. Cohen. Direct haptic rendering of sculptured models. *Proc. of ACM Interactive 3D Graphics*, pages 167–176, 1997.