Collision Detection and Force Response in Highly-Detailed Point-Based Hapto-Visual Virtual Environments

Naim R. El-Far, Nicolas D. Georganas, Abdulmotaleb El Saddik
Distributed and Collaborative Virtual Environments Research Lab – University of Ottawa
{naim, georganas, abed}@discover.uottawa.ca

Abstract

In this paper, we present a collision detection algorithm and a force response algorithm both for use in dynamic, rigid-bodied, highly-detailed, hapto-visual virtual environments in which the models' geometry is point-based. Our collision detection algorithm partitions the virtual space into a modified octree in a preprocessing step. At runtime, collision detection involves querying the octree for the octant where the end-effector currently is, as well as the indices of neighboring octants. After the world space is narrowed down to a volume of interest, the algorithm checks to see if the end-effector falls inside any axes-aligned bounding box that is centered at the model surface points that reside in the aforementioned volume of interest. A collision is defined as the haptic end-effector being found inside an axes-aligned bounding box centered at a model surface point. After a collision is detected, the force response algorithm calculates a force vector that starts at the end-effector's current position and ends at the model surface point closest to the end-effector. This is an adaptation of the common god/proxy-object approach but for use in point-based models.

1. Introduction

A hapto-visual application can be conceptually broken down into three major components: the simulation engine which is responsible for determining the virtual environment's behavior throughout the life of the application; the rendering block which encompasses algorithms for graphics, haptic, and audio rendering; and the display block which includes haptic, video, and audio transducers that physically interface the environment with the human user.

Of interest to us in this paper is the rendering block, and, more specifically, its haptic rendering component. Just like the process of graphic rendering essentially takes a graphic representation and displays it on a computer screen, haptic rendering takes a haptic representation and "displays" it on a haptic device. This is the definition of haptic rendering on an intuitive level. More formally, haptic rendering is defined as the collection of algorithms and techniques needed to impose on the user kinesthetic and tactile information via a haptic (force feedback) device.

There are three main steps in haptic rendering: The first is collision detection (CD). As the name implies CD is the process in which the environment is examined to see if, when, and where any two or more objects come in contact with one another. More specifically, haptic collision detection is the process in which collision detection techniques are applied to examine whether the haptic end-effector (or any other means of haptic transduction) has come in contact with a touchable object.

Should a haptic collision be detected, the second component of the haptic rendering loop comes into play: the force response algorithm. Given the position of the end-effector which is in contact with a haptic object, the force response algorithm determines the force vector that ideally should be felt by the human user that is handling the haptic device. Once the force is calculated, it is passed onto the final component of the haptic rendering cycle, the control algorithm.

The control algorithm determines how to best "display" the force calculated by the force response algorithm on the haptic device. Physical imitations on the haptic device hardware, as well as operational limitations on the hapto-visual software, prevent the haptic device from being able to display as is the force vector calculated by the force response algorithm; the control algorithm ensures that the difference between the ideal force and the actual force being displayed is minimal.

After this brief introduction to hapto-visual applications and haptic rendering, we turn our attention to model representation.

Graphics rendering and display is a well-researched field that has been around since the day computers had visual displays. One of the main pillars of graphics rendering today is the work done in model representation; how to represent a model in the virtual world given its geometric and surface properties. Many answers exist to the aforementioned problem, however

1550-6525/07 $25.00 © 2007 IEEE
DOI 10.1109/DS-RT.2007.29

the answer most prevalent is the technique that is most commonly used today; polygonal representation (in this work we are only concerned with model geometries, so we disregard model surface information). Typically, a polygonal representation of a model's geometry uses 2D polygons, such as triangles, that are connected in order to display complex geometries. This approach has become so prevalent that graphics hardware is designed to optimize triangle rendering. However, there are many applications where polygonal representation is not ideal. One such application is the virtualization, or digitization, of real-life objects.

The work in [45] is a prime example of a virtualization application where polygonal representation is unfitting. In the Mona Lisa Exploration Project, undertaken by Canada's National Research Council, the famous da Vinci painting was scanned using a very high-resolution laser scanner. The result was a very dense point cloud representing the painting's surface geometry. Although haptic manipulation was not a stated goal or vision for the scanning project, the ability to interface this highly-detailed model with a haptic device is truly intriguing and very enriching to any museum-visitor's experience.

In this work we focus on the haptic rendering of models such as the Mona Lisa scans. We introduce a novel collision detection algorithm and an adapted force response algorithm both of which are designed for use with dense point clouds. We assume that haptic interaction is done via a 3 degrees-of-freedom (DOF) point-based haptic device (e.g. Phantom Omni [42], Phantom Desktop [43], and MPB Freedom 6S [44]). We also speak throughout this paper of a virtual environment containing only one model; this is done for the sake of simplicity. Neither of our two algorithms places any limitation on the number of models in the environment.

2. Related Works

Collision detection in graphic environments is a mature field. Several surveys exist in the literature that present the most common approaches used today [2 – 4]. The same cannot be said of haptic environments for which CD literature is poor with most approaches borrowing from graphic CD.

Graphic CD is not optimized for haptic environments. The biggest difference between rendering graphic scenes and haptic scenes is the rendering rate. To display continuous graphics, a scene needs to be rendered at a rate of 25 – 60 Hz. For haptics, the rendering rate has to be around 1000 Hz in order for surfaces to feel continuous and for the haptic device to behave coherently. Another difference between haptic rendering and graphic rendering is that humans are more sensitive to touch that they are to sight, so while graphics can get away with inexact collision detection for instance, haptics cannot.

Approaches that use graphic CD in haptic environments overcome the rendering rate increase obstacle by either forcing a slower rendering rate (approximately 100 Hz) and placing restrictions on the environment [5], or distributing the hapto-visual application assigning the haptic loop a dedicated machine [6].

Some graphic CD algorithms that can be adapted for use in haptics include those described in [7 – 16]. All of these algorithms use a variation of bounding volumes to build a hierarchy that is traversed in order to spatially narrow the area of interest to a small volume. Other algorithms directly partition the world space using spatial partitioning structures [17 – 20]. Special hashing functions are also used in spatial partitioning by mapping a 3-dimensional coordinate to a 1-dimensional structure holding pointers to volumes in the world space [21].

In our review of the state of the art in haptic-specific collision detection, we looked at two open-source haptic API's, namely CHAI3D 1.51 [22] and H3D 1.5 [23], as well as the proprietary SensAble Open Haptics Toolkit [24]. All three APIs use a variation of the bounding-volume hierarchy (BVH) approach to perform a broad-phase narrowing of the model down to the area closest to the haptic probe, and then, in narrow-phase processing, determine collision by using an accurate (but expensive) method such as segment-triangle intersection detection.

Our literature review produced only two works that looked at models primarily as point clouds for the purpose of collision detection. The first is a US patent [25], and the second is a recent paper [5].

The US patent in [25] uses uniform spatial partitioning to arrive at a voxel containing a portion of the object's surface that is then decomposed into points. The distances between the point representing the haptic probe and all points in the aforementioned voxel are computed, and should any of them fall below a given threshold, a collision is detected. This method is rather incomplete since it does not account for points in neighboring voxels, and it also places no demand on the density of the model point cloud, so there is significant room for false results. The work also does not address how a model is decomposed into its constituent vertices.

[5] differs from [25] in two areas: how the algorithms find vertices neighboring the haptic end-effector's position, and how narrow-phase CD is performed after the neighboring vertices are found. As opposed to the approach in [25] explained above, [5] extracts, in a preprocessing phase, neighbor relationships between the vertices using the Voronoi theory. Then, at runtime, the CD algorithm uses these relationships to find vertices in the neighborhood of the haptic probe finally arriving at the one closest to the probe. Narrow-phase CD involves a standard segment-triangle intersection test. The approach in [5] is particularly well-suited for highly-detailed, complex virtual environments since it deals with the models as point-clouds.

Other approaches and adaptations exist, and a more thorough literature review is outside the scope of this work, however, we point the reader to: [31] for GPU-based CD; [32] for hapto-visual OBB-tree-based CD; [39] for level-of-detail-oriented dynamic BVH work, and to [33 - 35] for works in ray/segment intersection detection.

We now turn our attention to force response.

For translational 3-DOF devices, there are two main methods to compute ideal forces by the collision response module of the haptic rendering loop. The first is Vector Field Mapping, a concept borrowed from physics, which assigns a force vector to each point in a given space.

An alternative, much more widely used method has been put forward concurrently but independently by [46] and [47]. The notion of a god object in the former and a proxy object in the latter places a spring between where the end-effector actually is and where it should be. Haptic devices could have their end-effector's very well penetrate haptic models that are meant to be impenetrable, so although the actual haptic end-effector is inside the haptic model, it should be on its surface at the point closest to where its actual position is. This closest-point is dubbed the god-object or the proxy-object, which is essentially the point where the haptic end-effector should be.

Since [46] and [47], there have been many works to try to improve the fidelity of the haptic force response. One of the more significant works is the one in [48] which borrowed from the Phong shading techniques in computer graphics to smooth out force changes throughout the surface of an object. Another approach [49] optimizes the proxy object technique for use with models that have a high number of polygons in their triangular meshes.

3. Our Approach

If it were possible to store all vertices that made up a model's surface, then collision detection between a haptic probe and the model would be a matter of searching for the probe's coordinates in whatever structure held the model's vertices. Of course it is not possible to store the infinite number of vertices that make up the model's surface (not explicitly anyway), but even if we could, there are limitations on the positional resolutions of haptic devices, as well as the precision of floating number representations in computers. So, practically speaking, if we were able to produce a large and dense (but finite) data set of model vertices that were separated by distances less than or equal to a given haptic device's positional resolution, we could perform collision detection by searching for the probe's coordinates in the model vertices' data set. By definition, such a collision detection query would be 100% accurate, and depending on how the vertices are stored, it could also be fast.

Highly-detailed models are in essence a large and dense (but finite) data set of model vertices that are separated by very small distances. Granted, such distances could be larger than haptic hardware's positional resolutions, but highly-detailed models are a prime candidate for the accurate collision detection approach explained above. Provisions must be made, however, to account for the gaps between the vertices that are separated by a distance greater than the resolution of a haptic device.

We propose to fill these gaps by surrounding the model vertices with very small, but overlapping, axes-aligned bounding boxes (AABBs). Collision detection is then a matter of finding the probe's coordinates in the model dataset, or if this search fails, finding a vertex-AABB that bounds the probe. Point-in-AABB tests are cheap and fast, and they are the reason we chose AABBs over other bounding volumes.

As you might have already realized from the description of our approach, the accuracy of our CD algorithm is highly dependent on the density of the vertex cloud that is the model. The more vertices there are in a unit volume, the smaller the overlapping AABBs need to be, and the more accurate the CD result is.

False positives reported by our CD algorithm can result if the AABBs that are centered at the model vertices are too large. False negatives can result if the AABBs are too small or non-overlapping. Both cases are shown in Figure 1 where the haptic probe (indicated by a grey dot) is colliding with the object's surface at position 1 but since it is outside any of the

AABBs, no collision is detected (false negative), while in position 2 it is inside an AABB but not colliding with the surface although a collision is falsely detected (false positive).

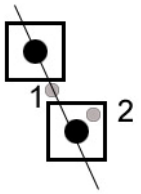

Figure 1: AABBs must be big enough to avoid false negatives (1) and small enough to avoid false positives (2).

The right size for the AABBs is highly dependent on the separation between the model vertices; our experimental results (discussed later) show that it is best to set it to a value approximately equal to the model vertex separation.

Now that we have reformulated the CD problem into a vertex search problem and a point-AABB bounding test, we shift our focus to the structure we use to store the model vertices.

We use a variation of octrees to partition our model space, thus speeding up the vertex search problem to logarithmic time O(*log n*) where *n* is the number of vertices that make up the model. The octree that we use is special in that each octant can contain more than one vertex (more details in the Experimental Results section). This is done to limit the depth of the tree, and does not cause a significant performance hit because the two checks that happen in leaf octants (namely checking for vertex equality and checking for point-AABB intersection) are both inexpensive computationally and memory-wise. Note that vertex AABB's are never constructed and kept in memory, but are rather only used for point-AABB intersection at the octant leaf level when that octant is reached vis-à-vis the octree search algorithm.

Keeping in mind that our CD test is done by either searching for the probe coordinates in the octree or searching for an AABB that bounds the probe's coordinates, we introduce the notion of neighbor search to our variation of the octree. Neighbor search is the price we pay for the benefit of a logarithmic search time. Figure 2 shows how simply searching for an AABB in the octant where the probe is can lead to a false negative CD result. Neighbor search insures that all octants neighboring our octant of interest are examined to avoid missing a bounding AABB.

[40] and [41] review several methods for performing neighbor search in octrees, however, we feel that for the purposes of our CD algorithm, a much simpler approach can produce results that are equally good. Again, taking advantage of the high density of vertices in the point cloud, we observe that examining octants containing points at a fixed distance from the probe is enough to retrieve all AABBs near the probe. We propose a six-point-test that we use to find neighboring octants (those containing points an equal distance to the left, right, top, bottom, front, and left of the probe – experimental data has shown no significant improvement in examining more than six). The distance between the probe's actual location and the six points we examine around it must be small enough to not miss octants in the immediate vicinity of the probe, and large enough to ensure that neighboring octants are retrieved. Worthy of note is that our implementation of the octree indices all leaf octants in a depth-first traversal of the tree after it is constructed, so a six-point-search would return the indices of that octants where the neighboring points reside.

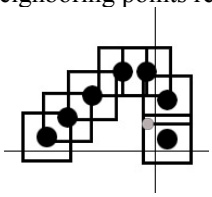

Figure 2: The haptic probe (grey point) is in an octant where no collision occurs, however, the probe is colliding with an AABB in the neighboring octant.

Our CD algorithm handles the translation and rotation of the scene's models by storing for each a 4x4 homogenous coordinate transformation matrix that is updated every time an object undergoes a transformation. The transformation matrix is used to find out the coordinates of the haptic probe in the local model space (the space where the octree is built) so that an octree query could be run.

Our approach is unique from the ones presented in [5] and [25]. In contrasting it with the Voronoi theory approach, we claim that ours is simpler to implement yet achieves comparable results (see Experimental Results below). Trials have shown that the six-point-test, paired with an octree that has leaves containing several vertices, performs very well in finding possible neighbors of the haptic probe. Furthermore, the point-AABB test is much cheaper than the intersection test in [5]. Accuracy between the two algorithms is comparable when the point cloud is dense enough for our algorithm. Admittedly, however, for less dense models, [5] performs better. In comparing our approach with that presented in [25] we conclude that although both use a point-centric view of the model, ours is specific to dense models, however the approach

in [25] is not (we believe from its description that it should be).

Now that we have presented our CD algorithm, we present our collision response algorithm which is an adaptation of the classic god/proxy-object approach found in [46] and [47]. The god-object we want to find is an object on the surface of our haptic model where the end-effector should be once a collision occurs. As explained earlier, the actual end-effector's position and the position of the god-object don't necessarily coincide because of the nature of haptic device use. So in creating a force response, we create a force vector pointing from the actual end-effector's position to the god-object's position (i.e. the surface position nearest to the end-effector). The collision response algorithm is called after collision is detected. It runs as follows:

Preprocessing phase:

1. For each octant in the octree, retrieve the indices of the 6 neighboring ones (as per the six-point rule) and store them in a table.

Run-time phase if a collision is detected:

1. Retrieve the octant where collision has been detected.
2. In that given octant, calculate the squared distances between the end-effector and all contained model surface points. Save the minimum.
3. If applying the six-point rule to the end-effector's position is enough to cross from the collision octant to any of its neighbors, retrieve the neighboring octant's model surface points and calculate the square distances. Save the minimum.
4. The point that is closest to the end-effector after steps 2 and 3 are run is assumed to be the point on the model's surface. The force vector is that which starts at the end-effector and ends at the closest point.

Given that each octant contains up to m points and that the six-point rule may result in up to six more octants searched, the time taken to search for the closest point as defined above has an upper bound of $7 \times m \times T$ where T is the time needed to perform a squared distance calculation. Since we are interested in the minimum number among a list of distances, we can cut down T by calculating the squared distance, as opposed to the distance, thereby saving the time it takes to perform square root operations.

At their current stage of development, our algorithms apply to rigid bodies only. This is due to the problems that arise in updating the octree should the model deform.

Finally, we draw the reader's attention to the fact that our algorithm is a point-based algorithm, so it is susceptible to classic temporal aliasing. However, as our experimental results show below, the CD and response algorithms perform well despite not explicitly accounting for temporal aliasing issues. This is due to the relatively slow speed which humans move a haptic device at, the small workspace most haptic devices have, and the high update rate of the haptic rendering loop.

4. Experimental Results

In testing our CD algorithm, we chose the surface described by the function $z = cos(x) + cos(y)$. We chose an implicit representation of a model because it would be easy to extract as many model vertices as we wanted out of it, it would be easy to verify collision points (they would satisfy the equation), and it should serve to highlight temporal aliasing issues since it is a "thin" object. Figure 3 shows the graph represented by model vertices and by vertex-AABB's. It shows a good example of why our CD algorithm will not work for models that have high vertex separation (0.1 m in the figure). Note how a sparse model results in AABBs that are too large to form a smooth surface. The finer the AABBs are, the smoother the surface and the better performing our algorithm. The fact that our AABBs in the figure are completely overlapping insures that we are unlikely to get false negative results; however, we expect a lot of false positives for such a coarse AABB approximation.

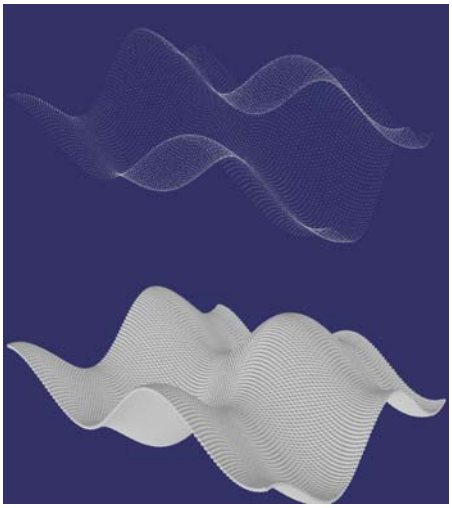

Figure 3: A sparse (0.1m resolution) point cloud of the surface $z = cos(x) + cos(y)$ (top) and the corresponding AABB-bound model (below).

The vertex points for our test model were generated by code. Three versions differing only by the separation between the vertices were created and stored offline. The largest separation we tested was 0.01 m, and the smallest was 0.0025 m. Figure 3 was graphed with a larger vertex separation than the test

models, and a larger range of *x* and *y* values, for clarity in print form.

Worthy of note here is that we used OpenSceneGraph (an OpenGL-based object-oriented graphics library) to display our model and test results graphically albeit at a lower resolution than our haptic scene. This is because OpenSceneGraph struggled understandably with the large number of vertices and AABBs that we were asking it to render. The test data were collected on an HP Pavilion dv9000 laptop with 1 Gb of RAM and AMD Turion64 dual processors, running Windows XP SP2. The code was timed using the Win32 API QueryPerformanceCounter. We also note that we built in a margin of error equal to 0.005 mm which is a typical positional resolution of a haptic device. So in other words if a point 0.005 mm or less away from our probe's location would have returned from the model surface equation a result equal to what our CD algorithm reported, our CD algorithm was granted a positive characterization score.

Table 1 shows test results from manipulating a static model and a dynamic model where the octree contained 10 points per leaf, the AABB cube was of length 0.0025 m, and the six-point-test separation was 0.0025 m. As expected, the results show that the sparser the point cloud is, the less accurate the CD results are. The haptic manipulation is simulated by a probe path made up of 6820 points tracing or hovering over the model surface. When the separation between the model vertices is 0.01 m, a large number of false negatives were registered because the gaps between the vertices were not filled by the AABBs (which were too small to overlap). As the separation between the vertices shrank, the AABB's overlapped enough to cover the gaps, and at 0.0025 m, they were still small enough to not cause excessive false positives. The size of the AABBs is crucial; with larger sizes, we see more false positives, but less false negatives, and vice versa. We arrived at the figure 0.0025 m for this specific model following a rule of thumb; an AABB size equal to or slightly larger than the model vertex separation works best. As for the six-point-test separation distance, a figure equal to the model vertex separation works best since it maximizes the probability that relevant neighboring octants will be visited (note that the six-point-test also saves us the trouble of checking a neighboring octant if it's too far from the probe's location). We note that the run-times are a function of the octree's depth (i.e. the number of vertices in the model) and that, by definition, traversing the octree is done in logarithmic time.

In theory, whether our model undergoes some transformation(s) or not, the accuracy of our algorithm should not change. This is simply because when we query the octree, we search for the probe's coordinates in local space, thanks to the transformation matrices we keep updated. The only overhead is the time it takes to transform a point from one coordinate system to the other (i.e. the time it takes to multiply a 4x1 vector by a 4x4 matrix). This reasoning is supported by the results in Table 1.

Table 1: Average test results showing better accuracy for denser point clouds.

Separation (*m*)	Accuracy (6820 test points)	Worst run-time (*ms*)	Average run-time (*ms*)
Static model			
0.01	77.889% (100 false positives, 1408 false negatives)	0.28408	0.02865
0.005	89.633% (291 false positives, 416 false negatives)	0.31549	0.03233
0.0025	98.431% (1 false positive, 106 false negatives)	0.32758	0.04198
Dynamic model			
0.0025	98.18% (3 false positives, 179 false negatives)	0.35625	0.03025

We now turn our attention to the force response algorithm.

In order to test the performance of our closest-point algorithm (and hence our force response algorithm) described in the previous section, we devised the following test. For a portion of the model surface $z = cos(x) + cos(y)$, we simulated a probe path that crisscrossed the model surface. This path was described by $z = cos(x) + cos(y) + r$ where r is a random number in the range [-0.05m, 0.01m]. The probe path constituted of 3,000 distinct test points. The other parameters were kept the same as those used in the two CD tests above. Table 2 shows the time it took to run our algorithm and its accuracy in comparison with the optimal brute-force approach in which the distance between the end-effector and all the points in the model was measured.

Our experimental results demonstrate that the six-point rule performs very well in determining the closest distance between the end-effector and the model's surface; the worst difference between our approach's results and the results of a brute force approach is approximately 0.3 micrometer for a point cloud that has vertex separation of 0.0025 m. Runtime is acceptable but at the expense of increased preprocessing time, which is necessary to create an octant neighborhood table.

Table 2: Test results showing the runtimes of our closes-point algorithm and comparing its output with that of a brute force approach.

Our algorithm's best runtime (ms)	Our algorithm's worst runtime (ms)	Our algorithm's average runtime (ms)
0.173	0.621	0.233
Smallest distance difference between our algorithm's calculation and the brute force calculation (m)	**Biggest distance difference between our algorithm's calculation and the brute force calculation (m)**	**Average distance difference between our algorithm's calculation and the brute force calculation (m)**
0.0	0.0000341795	0.0000019716

5. Conclusion

In this paper, we presented a collision detection algorithm and a force response algorithm that are designed to work with highly-detailed point-based haptic models. Our CD algorithm redefines the collision problem into an octree search and a point-in-AABB collision problem. We have shown that it has logarithmic complexity and runs fast. It is also highly accurate. Our force response algorithm is in essence an adaptation of the god/proxy-object approach to point-based models. We have shown that it produces fast results that are very close to the optimal brute-force approach results. Both algorithms use what we call the six-point rule or six-point test which we have shown works very well with dense point clouds. The six-point test hypothesizes that we can determine all the neighbors of a given point in the point cloud by querying the octree for the octant indices that contain the front, back, top, bottom, left, and right neighbors of a given point. After retrieving the octant indices, we can determine all the points that are contained therein, which are the neighboring points.

Our algorithms support model transformations through use of a transformation matrix that is kept up-to-date with the translation and rotation that a model undergoes. This matrix is used to localize the probe's coordinates into the model's space for the purpose of querying the octree.

Experimentation has shown that our algorithms perform best when vertex separation is at or below 25 mm, which is significantly larger than most haptic devices' positional resolutions. We also showed the effects that the size of our vertex AABBs have on the algorithm's accuracy and recommended that the six-point-tests have a point separation equal to the model vertex separation.

We are working on an approach to support deformable bodies, which our algorithms do not at the moment because we can neither rebuild the octree at run-time, nor can we efficiently update it when deformation occurs (both actions are too lengthy).

A second research track we are working on is to load the Digital Michelangelo models [1] and the Mona Lisa model [45] into a hapto-visual environment so that we could "touch" them.

We conclude by reiterating the usefulness and simplicity of our algorithms in highly-detailed virtual environments such as those found in museum and archiving applications, prototyping applications, and scientific modeling and visualization applications.

6. References

[1] M. Levoy et al. The digital Michelangelo project: 3D scanning of large statues, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, p.131-144, July 2000

[2] S. Hadap et al. Collision detection and proximity queries. In ACM SIGGRAPH 2004 Course Notes (Los Angeles, CA, August 08 - 12, 2004). SIGGRAPH '04. ACM Press, New York, NY

[3] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. Proc. of IMA Conference on Mathematics of Surfaces, 1:602–608, 1998

[4] P. Jimenez, F. Thomas, and C. Torras. 3D collision detection: a survey. Computers and Graphics, 25(2):269–285, 2001

[5] M. de Pascale and D. Prattichizzo, 2006. A framework for bounded-time collision detection in haptic interactions. In Proceedings of the ACM Symposium on Virtual Reality Software and Technology (Limassol, Cyprus, November 01 - 03, 2006). VRST '06

[6] N.R. El-Far, X. Shen, N.D. Georganas, "Applying Unison, a Generic Framework for Hapto-Visual Application Development, to an E-Commerce Application", Proc. IEEE Workshop on Haptic Audio Visual Environments and their Applications, Ottawa, ON, Canada, October 2004

[7] P. Hubbard. Interactive collision detection. Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in, pages 24--31, 1993

[8] S. Quinlan. Efficient distance computation between non-convex objects. Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, pages 3324--3329, 1994

[9] N. Beckmann et al, The R*-tree: an efficient and robust access method for points and rectangles, Proceedings of the 1990 ACM SIGMOD international conference on Management of data, p.322-331, May 23-26, 1990, Atlantic City, New Jersey, United States

[10] M. Held, J. Klosowski, and J. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. Proc. 7th Canad. Conf. Comput. Geom, pp. 205--210, 1995

[11] Gino van den Bergen, Efficient collision detection of complex deformable models using AABB trees, Journal of Graphics Tools, v.2 n.4, p.1-13, April 1997

[12] S. Gottschalk , M. C. Lin , D. Manocha, OBBTree: a hierarchical structure for rapid interference detection, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, p.171-180, August 1996

[13] S. Redon, A. Kheddar, and S. Coquillart. Collision Detection and Augmented Reality: Fast Continuous Collision Detection between Rigid Bodies. Computer Graphics Forum, 21(3):279, 2002
[14] S. Cameron, Approximation hierarchies and S-bounds, Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, p.129-137, June 05-07, 1991, Austin, Texas, United States
[15] S. Krishnan, A. Pattekar , M. Lin , D. Manocha, Spherical shell: a higher order bounding volume for fast proximity queries, Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective: the algorithmic perspective, p.177-190, August 1998, Houston, Texas, United States
[16] J.T. Klosowski et al, Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, IEEE Transactions on Visualization and Computer Graphics, v.4 n.1, p.21-36, January 1998
[17] H. Samet and R.E. Webber, Hierarchical Data Structures and Algorithms for Computer Graphics. Part I., IEEE Computer Graphics and Applications, v.8 n.3, p.48-68, May 1988
[18] B. Naylor, J. Amanatides, and W. Thibault, Merging BSP trees yields polyhedral set operations, Proceedings of the 17th annual conference on Computer graphics and interactive techniques, p.115-124, September 1990, Dallas, TX, USA
[19] H. Konig and T. Strothotte. Fast Collision Detection for Haptic Displays Using Polygonal Models. Proceedings of the Conference on Simulation and Visualization, 300, 2002
[20] D. Borro, A. Garcia-Alonso, and L. Matey. Approximation of Optimal Voxel Size for Collision Detection in Maintainability Simulations within Massive Virtual Environments. Computer Graphics Forum, 23(1):13--23, 2004
[21] C. Ericson: Real-Time Collision Detection. Morgan Kaufmann, 2004
[22] CHAI 3D: The Open Source Haptics Project. http://www.chai3d.org/. Last accessed on Mar 10, 2007
[23] H3D: Open Source Haptics. http://www.h3d.org/. Last accessed on Mar. 10, 2007
[24] SensAble 3D Touch SDK OpenHaptics Toolkit version 1.0 Programmers' Guide. 2004. http://www.sensable.com/61614600231945743464549037/Link.htm. Last accessed on Mar. 10, 2007
[25] S.E. Lithicum et al. System and method for providing interactive haptic collision detection. US Patent 6714213
[26] D. Bielser, M.H. Gross, "Interactive Simulation of Surgical Cuts," pg, p. 116, Eighth Pacific Conference on Computer Graphics and Applications (PG"00), 2000
[27] Y. Kim, M. Lin, and D. Manocha, 2002. DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes. In IEEE Conference on Robotics and Automation
[28] T.C. Hudson et al. 1997. V-COLLIDE: accelerated collision detection for VRML. In Proceedings of the Second Symposium on Virtual Reality Modeling Language (Monterey, California, United States, February 24 - 26, 1997). VRML '97
[29] J. Cohen et al, 1995. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. In Proceedings of the 1995 Symposium on interactive 3D Graphics (Monterey, California, United States, April 09 - 12, 1995). SI3D '95. ACM Press, New York, NY
[30] SWIFT++: Speedy Walking via Improved Feature Testing for Non-Convex Objects. http://www.cs.unc.edu/~geom/SWIFT++/. Last accessed on Mar 1, 2007
[31] N. K. Govindaraju , M. C. Lin , D. Manocha, Fast and reliable collision culling using graphics hardware, Proceedings of the ACM symposium on Virtual reality software and technology, November 10-12, 2004, Hong Kong
[32] A. Gregory , M. C. Lin , S. Gottschalk , R. Taylor, A Framework for Fast and Accurate Collision Detection for Haptic Interaction, Proceedings of the IEEE Virtual Reality, p.38, March 13-17, 1999
[33] T. Möller, B. Trumbore, Fast, minimum storage ray-triangle intersection, Journal of Graphics Tools, v.2 n.1, p.21-28, 1997
[34] D. Badouel, An efficient ray-polygon intersection, Graphics gems, Academic Press Professional, Inc., San Diego, CA, 1990
[35] J. Arvo , David Kirk, A survey of ray tracing acceleration techniques, An introduction to ray tracing, Academic Press Ltd., London, UK, 1989
[36] FCS HapticMASTER User Guide. Moog FCS 2002.
[37] M. Eid. et al. Implementation and Evaluation of Several Haptic Collision Detection Algorithms. Technical report, University of Ottawa. 2007
[38] SensAble – Phantom Desktop. http://www.sensable.com/haptic-phantom-desktop.htm. Last accessed on Mar. 2, 2007
[39] P. Liu, X. Shen, N.D. Georganas, G. Roth, "Multi-resolution Modeling and Locally Refined Collision Detection for Haptic Interaction", Proc. 3DIM 2005: The Fifth International Conference on 3-D Digital Imaging and Modeling, Ottawa, ON, Canada, June 2005
[40] H. Samet: Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS. Addison-Wesley, 1993.
[41] H. Samet: The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1990.
[42] Phantom Omni. Technical Specifications. http://www.sensable.com/haptic-phantom-omni.htm. Last accessed on Apr. 15, 2007.
[43] Phantom Desktop. Technical Specifications. http://www.sensable.com/haptic-phantom-desktop.htm. Last accessed on Apr. 15, 2007.
[44] FCS HapticMASTER. Technical Specifications. http://www.est-kl.com/hardware/haptic/fcs/hapticmaster.html. Last accessed on Apr. 15, 2007.
[45] 3D Examination of the Mona Lisa. National Research Council Canada. http://iit-iti.nrc-cnrc.gc.ca/projects-projets/monalisa-lajoconde_e.html. Last accessed on May 9. 2007.
[46] C. Zilles and J.K. Salisbury, "A Constraint-Based God-Object Method for Haptic Display", Porc. IEE/RSJ Int'l Conf. Intelligent Robots and Systems, Human Robot Interaction, and Cooperative Robots, vol. 3, IEEE CS Press, 1995, pp 146-151.
[47] D.C. Ruspini, K. Kolarc, and O. Khatib, "The Haptic Display of Complex Graphical Environments", Proc. ACM Siggraph, ACM Press, 1997, pp. 345-352.