Interactive Collision Detection for 3D Environments

Mauro Figueiredo
Univ. do Algarve
Faro
mfiguei@ualg.pt

Abstract

This paper presents a collision detection algorithm for 3D simulated environments. It describes the implementation of a collision detection approach using the Overlapping Axis-Aligned Bounding Box (OAABB) and R-trees to improve performance. Experimental results show that this implementation is effective in determining interactively intersections between 3D models.

Keywords

Collision detection, virtual environments, Computer Graphics.

1. INTRODUCTION

It is a complex problem to find collisions in virtual environments in real-time. Collision detection is a very time consuming task. In some environments it can easily consume up to 50% of the total run time. In real industrial case studies, 3D virtual prototypes can be very complex with thousands of primitives. Therefore developing real-time collision detection algorithms for complex environments is still a challenging research issue. In such environments, it is important to maintain at least a frame rate of 20Hz to create a usable simulator.

The problem of finding collisions in an environment is more complex for the narrow phase. The broad phase of the collision detection problem is responsible for discarding pairs of objects that do not collide. The narrow phase determines if two objects are colliding.

This paper describes a collision detection algorithm based on an R-tree structure of Axis-Aligned Bounding Boxes (AABB) and using the Overlapping Axis-Aligned Bounding Box (OAABB) to solve the narrow phase of the collision detection problem. It finds intersections between two objects. Experimental results show that the algorithm presented in this paper determines intersections at interactive rates.

2. BACKGROUND

To find collisions between two 3D objects it is frequently used bounding volume hierarchies (BVH) that organize the triangles of an object. In this way, performance is improved by reducing the number of pairs of bounding volume tests.

Suppose that the bounding volumes of two objects are organized in two hierarchical trees of bounding volumes (BV) *A* and *B*. The classic scheme for hierarchical collision detection is a simultaneous, recursive traversal of two bounding volumes trees of *A* and *B*. This can be described by the algorithm presented in Figure 1.

```
TraverseTrees (A, B)
update BV of B into coordinate system of A
if A and B do not intersect then
   return
if A and B are leaves then
  update triangle B into coordinate system of A
  return intersection of triangles enclosed by A and B
else
  if A is leaf and B is an inner node then
      for all children B[i] do
         TraverseTrees (A, B[i])
   else
      if A is an inner node and B is a leaf node then
         for all children A[i] do
            TraverseTrees (A[i], B)
      else
         for all children A[i] and B[j] do
               TraverseTrees (A[i], B[j])
```

Fig. 1: Classical hierarchical traversal scheme for collision detection.

Nevertheless, Zachmann and Knittel [Zachmann03] identify several problems in the traditional traversal. In particular, they showed that several nodes in the hierarchy trees of *A* and *B* are visited more than once. The second problem is that the nodes of the bounding volume tree of *B*, for example, have to be transformed into the local coordinate system of *A*. As a consequence of the first problem, this transformation is repeated several times for the same node, reducing performance. The algorithm presented in this paper solves these problems.

There are several public toolkits to solve the narrow phase of the collision detection based on bounding volume hierarchies. They differ on the type of bounding volume implemented.

SOLID [Bergen97] and OPCODE [Terdiman01] uses axis-aligned bounding boxes (AABB). RAPID [Gottschalk96], V-COLLIDE [Hudson97], PQP [Larsen99], H-COLLIDE [Gregory99], use oriented bounding boxes (OBB). QuickCD [Klosowski98] and Dop-Tree [Zachmann98] uses k-dops; and Swift++ [Ehmann01] uses convex hulls (CH). There are also hybrid approaches like QuOSPOs [He99] that use a combination of OBBs and k-dops.

It is very difficult to compare different approaches since performance also depends on the shapes of the models, type of contact, size of the models and others.

The main advantage of SOLID, OPCODE and Box-Tree is that AABBs are faster to intersect. When using AABBs, only six comparisons are required to find out if two axis-aligned bounding boxes are overlapping. It is also possible to say that two AABBs are disjoint, in the best case situation, with only one comparison. Another advantage of using AABBs is that it is simple to update these volumes as an object rotates and translates.

RAPID approximates 3D objects with hierarchies of oriented bounding boxes (OBBs). An OBB is a rectangular bounding box with an arbitrary orientation so that it encloses the underlying geometry more tightly. The representation of an oriented bounding box encodes position, widths and orientation. The main advantage of RAPID is that OBBs are better approximations to triangles reducing effectively the number of intersecting operations.

V-COLLIDE solves the broad-phase of the collision detection using a sweep-and-prune operation to find pairs of objects potentially in contact. It uses RAPID to find in the narrow phase which pairs of objects intersect.

PQP solves the narrow phase and is also based on the RAPID library. It uses oriented bounding boxes to find intersecting objects. PQP also computes the distance between closest pair of points using swept spheres.

H-COLLIDE is a framework to find collisions for haptic interactions. It uses a hybrid hierarchy of uniform grids and trees of OBBs to exploit frame-to-frame coherence. It was specialized to find collisions between a point probe against 3D objects.

The QuickCD and Dop-Tree implementations build a hierarchy tree of discrete orientation polytopes. Discrete orientation polytopes, or k-dops, are convex bounding volumes whose faces are determined by halfspaces whose outward normals come from a small fixed set of *k* orientations. The main advantage of using discrete orientation polytopes is that *k*-dops are better approximations to the underlying geometry than AABBs with the advantage of its low cost compared to OBBs. A major drawback of QuickCD is that allows only one moving object.

Swift++ builds a hierarchy of convex hulls and intersection is tested using a modified Lin-Canny [Lin91] closest feature algorithm.

He [He99] uses a hybrid approach that combines OBBs and *k*-dops called QuOSPOs. This approach provides a tight approximation of the original model at each level.

Since collision detection is a very demanding task, researchers are also working in using existing graphics accelerated boards (GPU) [Baciu03, Knott03, Govindaraju03, Yoon04] or dedicated hardware [Raabe06] to accelerate collision detection by hardware.

Algorithms using graphics hardware use depth and stencil buffer techniques to determine collisions between convex [Baciu03] and non-convex [Knott03] objects. CULLIDE [Govindaraju03] is also a GPU based algorithm that uses image-space occlusion queries and OBBs in a hybrid approach to determine intersections between general models with thousands of polygons. MRC [Yoon04] deals with large models composed of dozens of millions of polygons by using the representation of a clustered hierarchy of progressive meshes (CHPM) as a LOD hierarchy for a conservative error bound collision and as a BVH for a GPU-based collision culling algorithm.

These GPU-based algorithms are applicable to both rigid and deformable models since all the computations are made in the image-space. Collision detection methods using GPUs have the disadvantage that they compete with the rendering process, slowing down the overall frame rate. Furthermore, some of these approaches are pure image based reducing their accuracy due to the discrete geometry representation.

3. COLLISION DETECTION

This section presents the latest implementation details of the initial ideas [Figueiredo03] of an approach for determining intersecting surfaces and triangles between a pair of 3D objects. For virtual prototyping applications, three-dimensional objects can be defined by a collection of surfaces, where each surface is tessellated individually and represented as a collection of triangles. For virtual environment or gaming applications, 3D objects can be simply defined as a polygonal soup.

The time to determine collisions between two objects using BVH depends on: (1) the cost of intersecting and updating bounding volumes; (2) the cost of intersecting triangles; and (3) on the number of such operations.

The approach presented in this paper is supported by R-tree hierarchies of axis-aligned bounding boxes and the Overlapping Axis-Aligned Bounding Box to reduce the cost and the number of updating and intersection operations and therefore improve performance.

The choice of bounding volume type influences performance of the collision detection process. The implementation of the collision detection algorithm presented in this paper uses axis aligned bounding boxes because they are faster to intersect.

It was decided to use R-trees [Guttman84] to build bounding volume hierarchies and organize 3D geometry of objects to improving the performance of the collision

detection process. R-trees are a good choice for collision detection because first, at any level of the tree, each primitive is associated with only a single node. Secondly, in an R-tree all leaf nodes appear on the same level. Third, because the depth of a R-tree storing n primitives is $\log_m n$, m is the minimum number of children of a node.

To speed up the process of finding collisions each object is represented by an R-tree data structure in its own local coordinate system (Figure 1). A hierarchical tree is built, grouping neighbouring surfaces. The leaf nodes of the R-tree point to the geometry of the surfaces that define the object. For two objects, it checks for collisions between surfaces which are in the neighbourhood, eliminating comparisons with those that are far away.

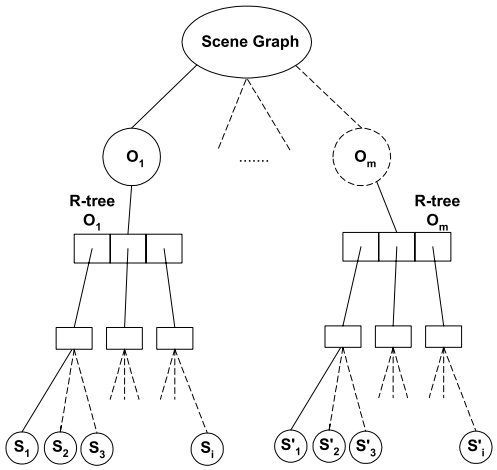

Fig. 1: Each object of the scene graph is represented by an R-tree to organize its surfaces [Figueiredo03].

Surfaces from a three-dimensional model can be complex with a large number of triangles. For this reason, the implementation described in this paper also uses an R-tree to organize the triangles spatially (Figure 2) and hence to quickly reject triangles that cannot intersect. In this approach, an R-tree is computed for each surface, grouping neighbouring triangles to eliminate comparisons with those that are faraway from the area of intersection.

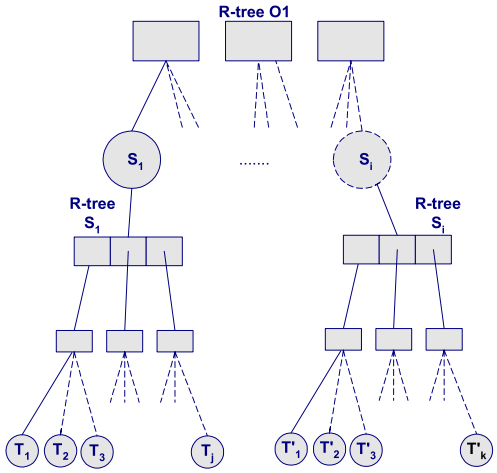

Fig. 2: Each surface of the 3D model is also represented by an R-tree data structure to organize spatially its triangles.

The proposed algorithm is also based in the use of the Overlapping Axis-Aligned Bounding Box, OAABB (A, B), of two geometric primitives, to improve the performance of the collision detection process. The OAABB is defined as the volume that is common to two axis-aligned bounding boxes of A and B.

The OAABB is used to filter out primitives that cannot intersect. Consider the example of Figure 3. Surface S_2 from object A cannot intersect object B since it is not intersecting the OAABB. In this way, it is possible to quickly filter out all surfaces from both objects that do not intersect the OAABB.

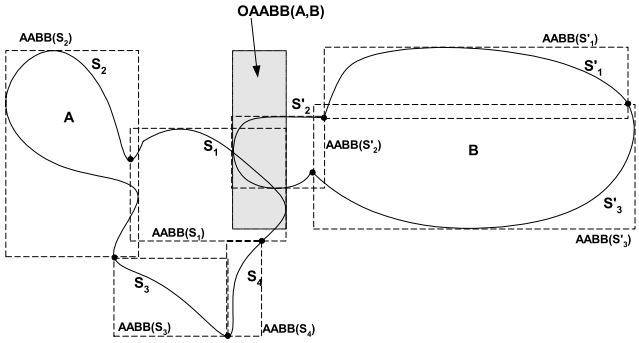

Fig. 3: The Overlapping Axis-Aligned Bounding Box (OAABB) concept shown in 2D [Figueiredo03].

The algorithm to find intersecting surfaces is presented in Figure 4.

```
S-CD_Collide (A, B)
1:AABB_B(A)=M_B←A•AABB_A(A) //update Cover BV
2:if (AABB_B(A) do not intersect AABB_B(B))     return
3:Determine OAABB_B(A, B)
4:DescendRtree(SBV(B), OAABB_B(A,B))
5:for each surface from SBV(B) intersecting OAABB_B(A,B)
6: DescendRtree(TBV(B), OAABB_B(A,B))
7: for each triangle T(B) from TBV(B) intersecting OAABB_B(A,B)
8:  Update triangle T(B) geometry into coord. system of A
9:  Compute the new optimal AABB_A(T(B)) for T(B)
10: DescendRtree(SBV(A), AABB_A(T(B)))
11: for each surface from SBV(A)
12:  DescendRtree(TBV(A), AABB_A(T(B)))
13:        Intersect T(A) and T(B)
```

Fig. 4. Pseudo-code for finding intersecting surfaces and triangles.

The collision detection process first checks if objects A and B are disjoint (line 1-2 in Figure 4). The bounding volumes of each object are originally computed in the object's local coordinate system, $AABB_A(A)$ and $AABB_B(B)$, respectively. The transformation matrix that converts the local representation of object A into the local coordinate system of object B is defined as $M_{B \leftarrow A}$. The bounding volume of object A is updated to the coordinate

system of object *B*, by computing the cover axis-aligned bounding box, $AABB_B(A)$. Once the bounding volumes of each object are in the same coordinate system they can be checked for overlap. If this pair of AABBs does not overlap, then the corresponding two objects are not intersecting and the process ends. If they overlap, then the system determines the Overlapping Axis-Aligned Bounding Box, $OAABB_B(A,B)$ of the two objects (line 3 in Figure 4), which is defined in the local coordinate system of object *B*.

The next step of the collision detection process determines the surfaces from object *B* intersecting the OAABB (line 4 of Figure 4). As mentioned before, the surfaces of object *B* are organized in a Surface Bounding Volume R-tree called *SBV(B)*. The surfaces of *B* are stored at the leaf nodes of the *SBV(B)* R-tree. By descending this R-tree, the surfaces of object *B* that do not intersect the $OAABB_B(A,B)$ are filtered out. Only the surfaces at the leaf nodes intesecting the $OAABB_B(A,B)$ are candidate for collision.

Each surface of object *B* is tessellated and represented by a collection of triangles. The triangles of each surfaces are also represented in its own Triangle Bounding Volume R-tree data structure, organizing its triangles into subregions. The next step of the collision detection process descends to the Triangle Bounding Volume R-tree *TBV(B)* of each candidate surface from object *B* (lines 5 and 6 of Figure 4). This stage determines the triangles of object *B* intersecting the OAABB.

Using the OAABB, the collision detection manager filters out surfaces and triangles of object *B* that cannot intersect.

The triangles of object *B* intersecting the OAABB are transformed into the coordinate system of object *A* (line 8). At this point is also determined the triangle's optimal bounding volume (line 9). The new optimal AABB is built by computing the new minimal and maximal values in the *x*, *y* and *z* axes, defining the new extents of the bounding volume. This stage computes the optimal bounding volume, since it is faster than computing the cover AABB. Furthermore, the optimal AABB encloses the triangle better than a cover AABB.

Then, the collision detection process descends the Surface Bounding Volume *SBV(A)* R-tree for object *A* (line 10 of Figure 4). In this step it finds surfaces of object *A* intersecting the triangle's optimal $AABB_A(T(B))$ of object B.

To determine if a pair of surfaces intersects, it is necessary to find a pair of intersecting triangles. Then, the collision detection process descends to the Triangle Bounding Volume R-tree *TBV(A)* of each candidate surface from object A (lines 11 and 12 of Figure 4). This stage determines the triangles of object *A* who's bounding volumes intersect the triangle's optimal AABB of object B.

The final step is to proceed with the intersection between pairs of candidate triangles (line 13) implemented with [Guige03] algorithm. If there is a pair of intersecting triangles then the corresponding surfaces is intersecting.

4. EXPERIMENTAL RESULTS

This section presents the performance evaluation results of the proposed collision detection algorithm described in this paper.

Two examples are presented: i) a process plant case study and ii) a grid. All the experiments conducted in the following paragraphs run in an Intel Core 2 Duo T7300, 2GByte of RAM memory at 2GHz.

The first example is a real application from maintenance simulation in virtual prototyping environments. It is a case study (Figure 5), from UMIST in Manchester that shows the assembly of a process plant. This example is used to test the collision detection in virtual prototyping environments where it is important to find intersecting surfaces. This scenario has a total number of eight parts, one thousand and seventy three surfaces and 24 415 triangles.

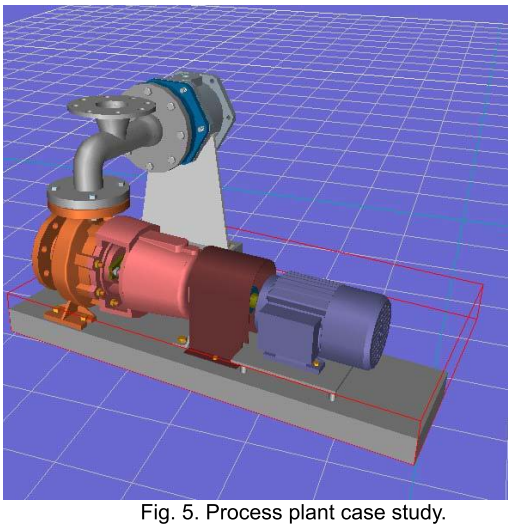

Fig. 5. Process plant case study.

The collision detection implementation can be configured to determine first intersecting surface, all intersecting surfaces or all the intersecting triangles between three-dimensional components in a virtual prototyping environment. The process plant case study was tested with an interaction sequence of 233 intersecting steps that simulate assembly operations. Table 1 presents the time to find first, all intersecting surfaces and all intersecting triangles.

Table 1. Collision detection time to find intersecting surfaces for the process plant.

Time in milliseconds per step to determine:		
Using OAABB	Yes	No
(i) first intersecting surface per step	0.09	0.28
(ii) all intersecting surfaces	0.94	1.75
(iii) all intersecting surfaces and all intersecting triangles	1.26	2.06

The proposed collision detection algorithm achieves interactive rates in real industrial applications as desired.

Table 1 also presents the time obtained when the overlapping axis-aligned bounding box (OAABB) approach is not used. It is seen a significant improvement in performance by using the overlapping axis aligned bounding box.

This improvement is explained by table 2. The cost of finding collisions depends on several factors. The choice of type of bounding volume influences the number and the cost of executing bounding volume intersections. For AABBs and *k*-dops the cost and the number of updating bounding volumes also influences performance. Table 2 shows an effective reduction in the number of AABBs tests and update bounding volume operations explaining the better performance obtained by the OAABBs.

Table 2. Average number of operations per step to determine intersections for the process plant.

Number of operations:	*First Triangle*		*All Triangles*	
Using OAABB	Yes	No	Yes	No
AABBs tests	1263	2915	21836	30663
AABBs updates	102	713	537	3279

It is also important to compare the performance of this algorithm with other collision detection toolkits. Table 3 presents the times obtained comparing the implementation described in this paper, named as S-CD (Surface Collision Detection), with PQP, RAPID, OPCODE and Dop Tree. The times presented were obtained in the determination of the first intersecting triangle and also for all the intersecting triangles. S-CD is effective in the determination of intersections interactively.

Table 3. Time to find intersections for the process plant.

Time to find intersections (milliseconds):	*First Triangle*	*All Triangles*
PQP	0.40	4.96
RAPID	0.12	3.62
Dop Tree	0.04	17.29
OPCODE	1.52	2.09
S-CD	0.09	1.26

The second is a synthetic model of a grid (Figure 6) with 414 720 triangles available from a benchmarking suite [Trenkel, *et al*. 2007] to compare in close proximity pair wise static collision detection algorithms for rigid objects.

This example is used to test the performance of collision detection in finding intersecting triangles in environments where objects are defined as polygonal soups.

The benchmark generates a number of positions and orientations for a predefined distance in close proximity. It does not test performance of collision detection approaches when intersections occur. It is available to download together with a set of objects that cover a wide range of possible scenarios for collision detection algorithms, and a set of precomputed test points for these objects.

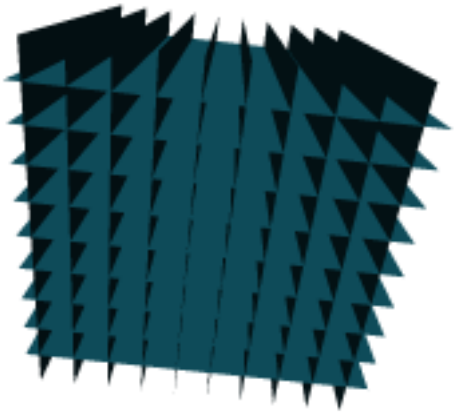

Fig. 6. Model of a grid object with 414 720 faces.

Figure 7 shows the timing results obtained for the benchmark application using two grid objects.

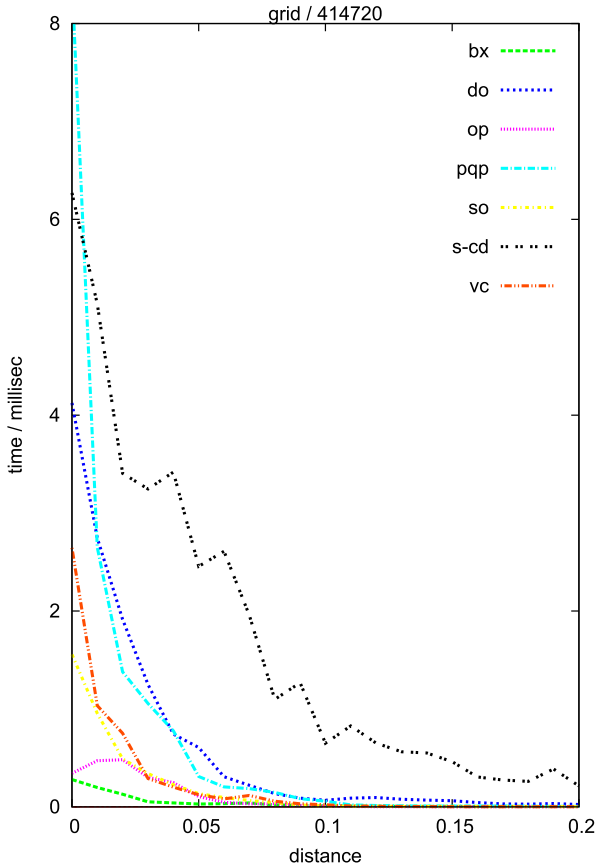

Fig. 7: Results of the benchmark for the grid scene with 414 720 faces. The x-axis denotes the relative distance between the objects, where 1.0 is the size of the object. Distance 0.0 means that the objects are almost touching but do not collide. The abbreviations for the libraries are as follows: bx=BoxTree, do=Dop-Tree, pqp=PQP, vc=V-Collide, op=Opcode, so=FreeSOLID, s-cd=S-CD.

To test performance the benchmark manager keeps one grid object static. The second grid object is placed at specified test positions and orientations. The x-axis in Figure 7 denotes the relative distance between the objects, where 1.0 is the size of the object. Distance 0.0 means that the objects are almost touching but do not collide. Since collision detection is mostly used to avoid interpenetrations, this position is the most important in many simulations, because most of the times, objects are allowed to collide only a little bit and then the collision handling resolves the collision by backtracking. Distance 0.0 is also the most time consuming (Figure 7) because the bounding volume hierarchies for the two objects overlap, but objects do not.

The time to find intersections is greater at position 0.0 and then as the distance between objects increases the time to find intersections reduces. As distance between the two objects increase then the overlap between the two bounding volume hierarchies decrease. In this case, the bounding volume of an object is found to do not intersect the bounding volume of the other object higher in the tree, then it cannot intersect any object bellow that node. Thus, they are all rejected quicker and it is faster to conclude that objects do not intersect.

Figure 7 shows that S-CD runs interactively at distance 0.0 as desired. For virtual environment applications a frame rate of at least 20Hz is desired and is possible to achieve frame rates higher than 100Hz. It should be regarded that in this benchmark S-CD does not uses surface knowledge and in this way the approach described in this paper does not use all its features to improve performance.

5. CONCLUSION

This paper presents a novel collision detection algorithm that computes intersecting surfaces and triangles solving the narrow phase problem of the collision detection.

It describes the use of R-trees and the OAABB for the implementation of an efficient method to find collisions in real time for virtual reality applications.

The collision detection algorithm presented in this paper uses the overlapping axis-aligned bounding boxes together with an R-tree structure, to filter out bounding volumes of primitives that cannot intersect. Two primitives intersect if the corresponding bounding volumes also intersect the OAABB. The traversal algorithm presented also reduces the number of node visits to one for object *A*, improving the overall performance and solving outlined problems of the traditional traversal scheme. The OAABB also contributes for the significant reduction of the number of bounding volume update operations.

This paper also showed that the collision detection runs interactively. The S-CD toolkit is publicly available for download at http://w3.ualg.pt/~mfiguei.

6. REFERENCES

[Baciu03] Baciu, G. and Wong, S., 2003. Image-based Techniques in a Hybrid Collision Detector. IEEE Trans. On Visualization and Computer Graphic, 9, 2, 254-271.

[Bergen97] Van Der Bergen, G., 1997. Efficient Collision Detection of Complex Deformable Models using AABB Trees. Journal of Graphics Tools 2, 4, 1-13.

[Ehmann01] Ehmann, S. and Lin, M., 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. Computer Graphics Forum. 20, 500–10.

[Figueiredo03] Figueiredo, M. and Fernando, T., “An Unified Framework to Solve the Broad and Narrow Phases of the Collision Detection Problem in Virtual Prototype Environments”. In Proc. of 2003 International Conference on Geometric Modelling and Graphics, Londres, Reino Unido, pp. 130-136, IEEE Computer Society, 16-18 Julho, 2003.

[Gottschalk96] Gottschalk, S. , Lin, M. and Manocha, D., 1996. Obb-tree: A hierarchical structure for rapid interference detection. Proc. of ACM Siggraph'96, 171-180.

[Govindaraju03] Govindaraju, N., Redon, S., Lin, M. and Manocha, D., 2003. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. Graphics Hardware 2003, 25–32.

[Gregory99] Gregory, A., Lin, M., Gottschalk, S. and Taylor, R., 1999. A Framework for Fast and Accurate Collision Detection for Haptic Interaction. Proc. of IEEE Virtual Reality Conference, 38-45.

[Guige03]Guige, P. and Devillers, O., 2003. Fast and Robust Triangle-Triangle Overlap Test using Orientation Predicates. Journal of Graphics Tools, 8, 1, 25-42.

[Guttman84] Guttman A., 1984. R-trees: A dynamic index structure for spatial searching. Proc. of the ACM SIGMOD International Conference On Management of Data, 47-57.

[He99] He, T., 1999. Fast collision detection using QuOSPO trees. Proc. of the Symposium on Interactive 3D graphics, 55–62.

[Hudson97] Hudson, T., Lin, M., Cohen, J., Gottschalk, S. and Manocha, D., 1997. VCollide: Accelerated Collision Detection for VRML. Proc. of VRML.

[Klosowski98] Klosowski, J., Held, M., Mitchell, J., Sowizral, H. and Zika, K., 1998. Efficient Collision Detection using Bounding Volume Hierarchies of k-DOPs. IEEE Trans. On Visualization and Computer Graphics 4, 1, 21-36.

[Knott03] Knott, D. and Pai, D., 2003. ClnDeR: Collision and Interference Detection in Real-time using Graphics Hardware. Proc. of Graphics Interface 2003, 73-80.

[Larsen99] Larsen, E., Gottschalk, S., Lin, M. and Manocha, D. 1999. Fast Proximity Queries with Swept Sphere Volumes. Technical report TR99-018, UNC.

[Lin91] Lin, M. and Canny, J. 1991. Efficient algorithms for incremental distance computation. IEEE Conference on Robotics and Automation, 1008–1014.

[Raabe06] Raabe, A., Hochgurtel, S., Anlauf J. and Zachmann, G., 2006. Space-efficient FPGA-accelerated collision detection for virtual prototyping. Proc. of Design, Automation and Test in Europe, 206-211.

[Terdiman01] Terdiman, P., 2001. Memory-optimized bounding-volume hierarchies. http://www.codecorner.com/Opcode.pdf.

[Trenkel07] S Trenkel, R Weller, G Zachmann. A Benchmarking Suite for Static Collision Detection Algorithms. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, Czech Republic, 2007.

[Yoon04] Yoon, S., Salomon, B., Lin, M. and Manocha, D., 2004. Fast Collision Detection between Massive Models using Dynamic Simplification. Eurographics Symposium on Geometry Processing, 136-146.

[Zachmann98] Zachmann, G., 1998. Rapid Collision Detection by Dynamically Aligned DOP-Trees. In Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS ’98, pages 90–97. Atlanta, Georgia.

[Zachmann03] G. Zachman, and G. Knittel, “An Architecture for Hierarchical Collision Detection,” *Proceedings of WSCG' 2003, the 11th International Conference in Central Europe on* Computer Graphics, Visualization and Computer Vision'2003, Czech Republic, 2003.