An R-Tree Collision Detection Algorithm for Polygonal Models

Conference Paper · January 2009

CITATIONS
0

READS
77

1 author:

Mauro Figueiredo
Universidade do Algarve
88 PUBLICATIONS **237** CITATIONS

SEE PROFILE

AN R-TREE COLLISION DETECTION ALGORITHM FOR POLYGONAL MODELS

Mauro Figueiredo
Algarve University
Estrada da Penha, Faro
Portugal
mfiguei@ualg.pt

ABSTRACT
This paper presents a novel collision detection algorithm for polygonal models. It describes the implementation of an R-tree traversal algorithm using the Overlapping Axis-Aligned Bounding Box (OAABB) to improve performance. Experimental results show an order of magnitude improvement compared with previous implementation.

KEY WORDS
Collision detection, virtual environments, Computer Graphics.

1. Introduction

Collision detection is a very time consuming task, making it difficult to find collisions in virtual environments in real-time. 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 millions of primitives. Therefore developing real-time collision detection algorithms for complex environments is still a challenging research issue. Furthermore, in virtual 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 often divided in two phases: broad and narrow. 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 the new traversal algorithm based on an R-tree structure of Axis-Aligned Bounding Boxes (AABB) to solve the narrow phase of the collision detection problem. It finds intersecting triangles between two objects. The new collision detection pipeline compares favourable with our previous implementation [1]. Experimental results show that the algorithm presented in this paper determines intersections at interactive rates.

2. Previous Work

This section describes previous work in the implementation of a collision detection algorithm to determine intersecting triangles between two three-dimensional models, solving the narrow phase of the collision detection problem.
The collision detection approach is supported by axis-aligned bounding boxes and uses the *Overlapping Axis-Aligned Bounding Box* (*OAABB*) and the R-tree structure for improving performance. The *OAABB* is used in this approach with polygonal models and the R-tree is a familiar concept in the database field.
The *OAABB* is an approach introduced by [2] to improve collision detection performance. Consider two objects, *A* and *B*, whose corresponding axis-aligned bounding boxes are overlapping and therefore are candidates for collision. The *OAABB* is defined as the volume that is common to two axis-aligned bounding boxes.
An R-tree is a multi-ary tree in which the non-leaf node has between *m* and *M* children, except for the root node [3]. The values *m* and *M* are the minimum and maximum number of children of a node, respectively. The main idea behind using an R-tree structure is to exploit spatial coherency. Each object is represented by an R-tree of bounding volumes in its own local coordinate system, grouping neighbouring triangles.
The following paragraphs describe the collision detection algorithm.
It uses mainly two processes (Figure 1): 1) Filter Triangles using the Overlapping AABB (lines 4 and 6); and 2) Intersect Triangles (line 7).
First, the collision detection process checks if the two objects *A* and *B* are candidate for collision (lines 1 and 2 of Figure 1). Objects *A* and *B* are approximated by axis-aligned bounding boxes defined in their own local coordinate system, $AABB_A(A)$ and $AABB_B(B)$, respectively. The $AABB_{cs}(A)$ is the axis-aligned bounding box of object *A* defined in the coordinate system *cs*. Therefore, the first step is to transform the axis-aligned bounding box of object *A* in to the coordinate system of object *B* (line 1). This is performed by executing a *cover bounding volume update* on object *A*, using the transformation matrix from coordinate system of object *A* into *B*, $M_{B \leftarrow A}$ (line 1). To compute the *cover* $AABB_B(A)$, the eight vertices of the AABB of object *A* are transformed into the coordinate system of *B*. The extents of the cover AABB are determined by finding the minimal and maximal values of these transformed eight vertices. After this step, the AABBs of the two objects are

intersected. A pair of objects cannot intersect if the corresponding bounding volumes are not intersecting. In such situations, the process ends in line 2. If the AABBs of the two objects are intersecting, then objects *A* and *B* are considered to be candidates for collision.

```
Collide version 1 (A, B)
1: AABB_B(A) = M_B_A • AABB_A(A) //update Cover BV oper.
2: if (AABB_B(B) do not intersect AABB_B(A)) then return
3: Determine OAABB_B(A, B)
4: mT_B =FilterTrianglesOAABB(TBV(B), OAABB_B(A,B), T_B)
5: OAABB_A(A,B) = M_A_B • OAABB_B(A,B) //update Cover BV oper.
6: mT_A =FilterTrianglesOAABB(TBV(A), OAABB_A(A,B), T_A)
7: IntersectTriangles(A, B, T_A, mT_A, T_B, mT_B)
```

Figure 1: Collide function that computes intersecting triangles between objects A and B.

To determine intersecting triangles, the process continues and the next step uses the overlapping axis-aligned bounding box (OAABB) approach and the R-tree data structure to filter out triangles that are not intersecting. The overlapping axis-aligned bounding box of the two objects ($OAABB_B(A, B)$) is determined in line 3. Once this is done, line 4 calls the FilterTrianglesOAABB() function to determine the triangles of object *B* whose AABBs intersect the OAABB. The triangles of object *B* are organized in a Triangle Bounding Volume R-tree, *TBV*(*B*). The total number of intersecting triangles is returned in mT_B and their IDs are returned in an array T_B. The same process is also executed for object *A*. However, the OAABB is defined in the coordinate system of object *B* and the bounding volume R-tree of object *A* is defined in the coordinate system of *A*. Therefore, line 5 executes a cover bounding volume update to determine an overlapping axis-aligned bounding box, $OAABB_A(A,B)$, in the coordinate system of object *A*. After executing this step, the bounding volume is defined in the coordinate system of object *A*. The function FilterTrianglesOAABB() (line 6) is then called to determine the number of potentially intersecting triangles of object *A* that intersect the OAABB. The total number of potentially intersecting triangles of object *A* is returned in mT_A and their IDs are returned in an array T_A.

Once these operations are carried out, the collision detection process has one set of triangles from object *A* and another set of triangles from object *B* that are candidates for collision, T_A and T_B, respectively. The next step is to intersect every possible pair of triangles from these two sets, which is implemented with the IntersectTriangles() process. This process determines pairs of intersecting triangles from objects *A* and *B*.

2.1 Filter Triangles with the OAABB Process

The processing of filter triangles uses the overlapping axis-aligned bounding box (OAABB) of two objects, to improve the performance of the collision detection process.

The OAABB is used to filter out triangles that cannot intersect. A triangle from object A intersects another triangle from B, if it also intersects the overlapping axis-aligned bounding box of the two objects. In this way, triangles whose AABBs do not intersect the $OAABB(A,B)$ are filtered out.

To improve performance furthermore, the FilterTrianglesOAABB() function uses the R-tree structure to speed up the calculation of triangles intersecting the OAABB.

In this way, a depth-first traversal scheme implements the Filter Triangles process to search bounding volumes that intersect the OAABB (algorithm presented in Figure 2).

```
FilterTrianglesOAABB (TBV(A), OAABB_A(A,B), T_A)
if TBV(A) intersect OAABB_A(A,B) then
   if TBV(A) is leaf then
     Store A in array T_A //is candidate for collision
   else
    for all children TBV(A[i]) do
      FilterTrianglesOAABB(TBV(A[i]),OAABB_A(A,B),T_A)
```

Figure 2: A depth-first recursive algorithm to traverse the R-tree and determine triangles whose BV intersects the OAABB.

2.2 Intersect Triangle Process

The previous calls to the FilterTrianglesOAABB function determined two sets of triangles from *A* and *B* candidates for collision. The Intersect Triangles Process is responsible for determining intersecting triangles, implemented with the function IntersectTriangles presented in Figure 3.

```
IntersectTriangles (A, B, T_A, mT_A, T_B, mT_B)
1:for n = 1 to mT_B
2:  T'_n =M_A_B, T_n //Update triangle of B into cs of A
3: Compute AABB_A(T'_n) // Update Optimal BV
4: for m = 1 to mT_A
5:   if (AABB_A(T_m) intersects AABB_A(T'_n)) then
6:     if (T_m intersects T'_n) then
7:           store pair T_m of A intersects T_n of B
```

Figure 3: IntersectTriangles function implementation.

The intersection of two triangles is an expensive computational operation. The axis-aligned bounding box of each triangle is used to reduce the number of such operations and improve performance.

For this purpose, each one of the candidate triangles from object B is transformed into the coordinate system of object *A* (line 2) for the determination of its *optimal* axis-aligned bounding volume (line 3). The new *optimal* $AABB_A(B)$ is then 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. The optimal AABB is computed after transforming three vertices from a triangle while a cover AABB is computed after transforming the eight vertices of an AABB. Furthermore, the optimal AABB encloses the triangle better than a cover AABB.

Those triangle pairs with intersecting bounding volumes are candidates for collisions and therefore the algorithm next checks for intersection of the correspondent triangles (line 6). If a pair of triangles intersects then it is stored (line 7).

3. Traversal Algorithm for Collision Detection

This section presents the new algorithm for determining intersecting triangles between a pair of 3D objects. The approach presented is supported by an R-tree hierarchy of axis-aligned bounding boxes and the Overlapping Axis-Aligned Bounding Box to reduce the number of intersection operations and therefore improve performance.

The collision detection approach uses axis-aligned bounding boxes for four reasons: i) they are fast to intersect; ii) use less memory; iii) hierarchies of AABBs are faster to build; and iv) faster to update.

It was found that intersecting two axis-aligned bounding boxes is about forty times faster than intersecting oriented bounding boxes and about six times faster than 18-dops [4].

Axis-aligned bounding volumes also use less memory. An AABB is represented with only six scalars for representing its extents. An oriented bounding box is represented with fifteen scalars. Nine scalars to store a 3×3 transformation matrix, three scalars for position and three for storing its extent. An 18-dop is represented with eighteen scalars to represent the volume extent in each one of the nine directions. OBBs and 18-dops require 2.5 and 3 times more storage than AABBs, respectively.

In some applications it is also necessary to insert and delete 3D models interactively, without expending too much time re-computing the data structures. In [5], Van Der Bergen found that building an OBB tree takes three times more time than building an AABB tree. Furthermore, Van Der Bergen showed that updating an AABB tree as a model is deformed is significantly faster than in an OBB tree. Hence, a bounding volume hierarchy of axis-aligned bounding boxes also offers the flexibility to develop an efficient collision detection algorithm for models that can deform with time.

Hierarchical bounding volume structures are frequently used to organize the 3D objects data to improve the performance of the collision detection process.

The triangles of an object are organized in a hierarchical tree of bounding volumes (BV). The classic scheme for hierarchical collision detection is a simultaneous, recursive traversal of two bounding volumes trees *A* and *B*. The approach presented uses an R-tree for each object in the scene. It differs from the classical traversal approach taking advantage of the OAABB and is implemented to avoid visiting the same node several times to improve performance. It visits the nodes of object *A* once.

Figure 4 presents the pseudo code of the novel approach.

```
Collide Version 2 (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(TBV(B), OAABB_B(A,B))
5: for each triangle T(B) from TBV(B) intersecting OAABB_B(A,B)
6:    Update triangle T(B) geometry into coord. system of A
7:    Compute the new optimal AABB_A(T(B)) for T(B)
8:    DescendRtree(BV(A), OAABB_A(A,B), AABB_A(T(B)))
9:    Intersect T(A) and T(B)
```

Figure 4: Pseudo-code for finding intersecting triangles.

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

The triangles of object *B* intersecting the OAABB are transformed into the coordinate system of object *A* (line 6). At this point is also determined the triangle's optimal bounding volume (line 7). 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 algorithm descends the Triangle Bounding Volume *TBV(A)* R-tree for object *A* (line 8 of Figure 4). In this step it finds triangles of object *A* intersecting both the OAABB and the triangle's optimal $AABB_A(T(B))$ of object B.

The final step is to proceed with the intersection between pairs of candidate triangles (line 9) implemented with Philippe Guige's algorithm [6].

4. Experimental Results

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

For this purpose, both implementations were tested using a benchmarking suite [7] to compare pair wise static collision detection algorithms for rigid objects. This benchmark generates a number of positions and orientations for a predefined distance in close proximity. It is available to download and can be downloaded 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.

A synthetic model of a grid (Figure 5) with 46 080 triangles was used to test the collision detection implementations versions 1 and 2. Figure 6 shows the timing results obtained for the benchmark application using two grid objects.

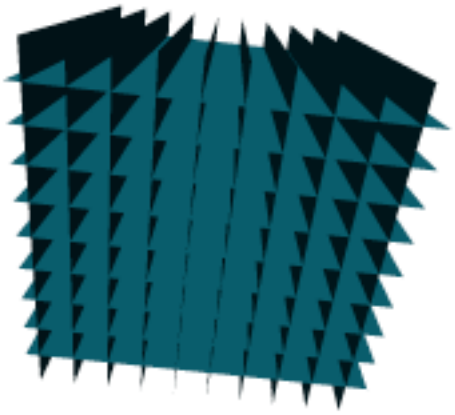

Figure 5: Model of a grid object with 46 080 faces.

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 6 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 6) 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.

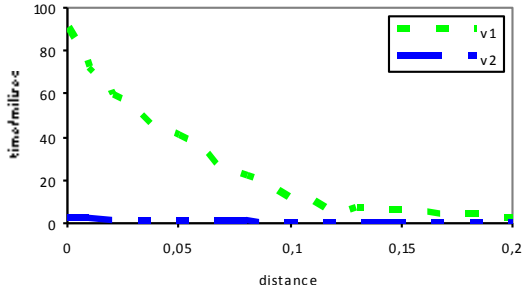

Figure 6: Results of the benchmark for the grid scene with 46 080 faces. The abbreviations are v1 for the previous work and v2 for the novel algorithm presented.

Figure 6 shows that version 2 presents a great improvement over the first version. The new implementation runs interactively at distance 0.0 as desired. For virtual environment applications a frame rate of at least 20Hz is desired and with version 2 is possible to achieve frame rates higher than 100Hz.

Table 1 explains the reason for this improvement. The cost of finding collisions between a pair of 3D models using bounding volume hierarchies depends on the cost of testing bounding volumes and triangles for collisions and the number of such operations. When one of the grid objects moves from distance 0.0 to 0.20, Table 1 shows that the average number of AABBs checks per position reduces from 1.85 million for version 1, to about six thousand for the second collision detection implementation. This a very significant performance improvement that contributes for the overall performance improvement of the novel implementation presented in this paper. The number of triangle checks does not change significantly.

Table 2 clearly emphasizes the improvement achieved. As previously mentioned the most time consuming position is at distance 0.0. Table 2 shows that the number of bounding volume checks reduces from about 6 million for version 1, to about sixteen thousand for version 2. At position distance 0.10 the two grid objects are clearly separated and the bounding volume hierarchies filter out all volumes and for this reason the number of triangle checks is zero.

Table 1: Average number of operations to determine intersections for the complete simulation.

Average number of	Version 1	Version 2
AABBs tests/step	1.85E+006	6303
Triangles tests/step	12	14

Table 2: Average number of operations per position to determine intersections.

Position	Version 1		Version 2	
	AABBs tests	Triangles tests	AABBs tests	Triangles tests
0.0	6,15E+006	103	16315	149
0.10	0,868E+006	0	4257	0
0.20	0,209E+006	0	2438	0

5. Conclusion

This paper presents a novel collision detection algorithm that computes intersecting triangles between polygonal models.
It describes the use of R-trees and the new traversal approach for the implementation of an efficient method to find collisions in real time for virtual reality applications.
The traversal algorithm takes advantage of using overlapping axis-aligned bounding boxes to reduce the number of intersecting operations. To improve performance, the collision detection uses the overlapping axis-aligned bounding box approach, together with a novel use of the 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 novel traversal algorithm also reduces the number of node visits to one for object *A*, improving the overall performance.
This paper showed that this implementation performs significantly better than our previous work and runs interactively.

References

[1] Mauro Figueiredo, Terrence Fernando. "A Collision Detection Algorithm for Polygonal Models that uses Sequential and Parallel Techniques for Improving Performance". In *Proc. of the Central European Multimedia and Virtual Reality Conference 2005*, Prague, Czech Republic. 2005, 149-153.

[2] M. Figueiredo, K. Boehm, and J. Teixeira. Precise Object Interactions using Solid Modeling Techniques. *Proc. of IFIP TC 5/WG 5.10 Conference on Modeling in Computer Graphics*, 1993, 157-176.

[3] GUTTMAN A., "R-trees: A dynamic index structure for spatial searching,"*Proceedings of the ACM SIGMOD International Conference On Management of Data*, 1984, 47-57.

[4] Figueiredo M., Marcelino L., Fernando T., "A Survey on Collision Detection Techniques for Virtual Environments". In *Proc. of V Symposium in Virtual Reality*, Brasil, 2002, 285-307.

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

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

[7] 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.