A Collision Detection Algorithm for Polygonal Models that uses Sequential and Parallel Techniques for Improving Performance

M. Figueiredo and T. Fernando
University Algarve, Portugal; Salford University, UK

Abstract
This paper presents an efficient collision detection algorithm for polygonal models. It is based on the Overlapping Axis-Aligned Bounding Box (OAABB) and R-tree structures to gain sequential improvements. It also takes advantages of multiple processors platforms.

1. Introduction
Collision detection plays an important role in virtual reality applications for the simulation of interactions between virtual objects. This is a very time consuming task that can easily consume up to 50% of the total run time, making it difficult to simulate collision detection for complex environments in real-time.

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, as many pairs of objects as possible, that do not collide. The narrow phase determines if two objects are colliding.

This paper presents a novel approach to solve the narrow phase of the collision detection problem. This approach determines intersecting polygons between three-dimensional models.

2. Collision detection algorithm
This section describes the collision detection algorithm proposed in this paper to determine intersecting triangles between two three-dimensional models, solving the narrow phase of the collision detection problem.

This 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 [FMF02].

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 $3 \times 3$ 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 [Ber97], 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.

The main problem of using AABBs is that they do not enclose the object geometry tightly as 18-dops and OBBs. For this reason, it is expected that in a collision detection cycle, more axis-aligned bounding boxes checks will be required than 18-dops or oriented-bounding boxes intersections.

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).

<table>
<thead>
<tr>
<th>Collide (A, B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: ( A_{\text{AABB}}(B) \times M_{A} \times A_{\text{AABB}}(B) ) //update Cover BV oper.</td>
</tr>
<tr>
<td>2: if ((A_{\text{AABB}}(A) \text{ do not intersect } A_{\text{AABB}}(B))) then return</td>
</tr>
<tr>
<td>3: Determine ( \text{OAABB}(A, B) )</td>
</tr>
<tr>
<td>4: ( m_{T} = \text{FilterTrianglesOAABB}(A, \text{OAABB}(A, B), T_{B}) )</td>
</tr>
<tr>
<td>5: ( \text{OAABB}(A, B) = M_{A} \times A_{\text{AABB}}(A, B) ) //update Cover BV oper.</td>
</tr>
<tr>
<td>6: ( m_{T} = \text{FilterTrianglesOAABB}(B, \text{OAABB}(A, B), T_{B}) )</td>
</tr>
<tr>
<td>7: Intersect Triangles ( (v_{i}, B_{i}, \tau_{i}, m_{i}, n_{i}) )</td>
</tr>
</tbody>
</table>

**Figure 1**: Collide function that computes intersecting triangles between objects \( A \) and \( B \).

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, \( A_{\text{AABB}}(A) \) and \( B_{\text{AABB}}(B) \), respectively. \( A_{\text{AABB}}(A) \), 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 \( A \) (line 1). This is performed by executing a cover bounding volume update on object \( B \), using the transformation matrix from coordinate system of object \( B \) into \( A, M_{A}, B \) (line 1). To compute the cover \( A_{\text{AABB}}(B) \), the eight vertices of the AABB of object \( B \) are transformed into the coordinate system of \( A \). 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.

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 \( (OAABB(A, B)) \) is determined in line 3. Once this is done, line 4 calls the FilterTrianglesOAABB() function to determine the triangles of object \( A \) whose AABBs intersect the OAABB. The total number of intersecting triangles is returned in \( mT_{A} \) and their IDs are returned in an array \( T_{A} \). The same process is also executed for object \( B \). However, the OAABB is defined in the coordinate system of object \( A \) and the bounding volume R-tree of object \( B \) is defined in the coordinate system of \( B \). Therefore, line 5 executes a cover bounding volume update to determine an overlapping axis-aligned bounding box, \( OAABB_{B}(A, B) \), in the coordinate system of object \( B \). After executing this step, the bounding volume is defined in the coordinate system of object \( B \). The function FilterTrianglesOAABB() (line 6) is then called to determine the number of potentially intersecting surfaces of object \( B \) that intersect the OAABB. The total number of potentially intersecting triangles of object \( B \) is returned in \( mT_{B} \) and their IDs are returned in an array \( T_{B} \).

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 \( B \).

3. **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 an approach introduced by [FF03] to implement a collision detection algorithm to find intersecting surfaces. In this paper, it is used to determine intersecting triangles in a novel collision detection algorithm for polygonal models.

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:

\[
OAABB(A, B) = \{ (x, y, z) \in \mathbb{R}^3 : A_{\text{AABB}}(A) \cap A_{\text{AABB}}(B) \}
\]

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. Instead of cross checking all pairs of triangles from \( A \) and \( B \), which is of complexity \( O(n^2) \), first, every triangle of \( A \) and \( B \) is tested against the overlapping bound-
The candidate triangles that intersect the OAABB are then cross-checked.

To improve performance furthermore, the FilterTrianglesOAABB function uses the R-tree structure to speed up the calculation of triangles intersecting the OAABB. 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 [Gut84]. 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 neighboring triangles.

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

\[
\text{FilterTrianglesOAABB}(A, \text{OAABB}(A, B), T_n)
\]

if \( \text{BV}(A) \) intersect \( \text{OAABB}(A, B) \) then
  if \( A \) is leaf then
    Store \( A \) in array \( T_i \) //is candidate for collision
  else
    for all children \( A[i] \) do
      FilterTrianglesOAABB\((A[i], \text{OAABB}(A, B), T_n)\)
end if

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

4. Intersect Triangles 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.

The intersection of two triangles is an expensive computational operation. Two techniques are used to improve performance of this process. It uses the axis-aligned bounding boxes of each triangle to reduce the number of such operations; and, it uses OpenMP to automatically parallelize code and distribute work to available processors.

OpenMP is an extension to a sequential programming language to support shared-memory computation. OpenMP was used as the basis for parallelising the algorithm due to three main reasons: portability, shared memory single source code and incremental development of parallel code.

Parallel regions are more tied to loop constructs. The nested loop (line 2 of Figure 3) can efficiently be executed in parallel to reduce the computation time. This for loop is automatically parallelized by OpenMP by applying the \#pragma omp parallel for directive (line 1).

Lines 2 to 9 of Figure 3 perform intersection operations between every pair of candidate triangles to determine intersecting triangles. To reduce the number of these checks, the bounding volume of each triangle is used to filter out those pairs of triangles that cannot intersect (line 6).

5. Experimental Results

This section presents the performance evaluation results of the novel collision detection algorithm described in this paper. It is used an industrial maintenance simulation case study of a simplified digger mechanism (Figure 4).
This test case simulates the building procedure of the digger in a virtual prototyping environment. This model has 2,452 triangles. To evaluate the performance of this system in more complex environments the number of triangles of the digger model is also multiplied by 5, 10, 15 and 20, and the corresponding models will be referred in the following paragraphs as diggerx5, diggerx10, diggerx15, and diggerx20, respectively. The same trajectories are repeated for the five scenarios to simulate the complete assembly.

All the experiments described in the following sections were run on an ONYX2 with MIPS10000 processors clocked at 250MHz, 4 gigabytes of RAM.

Table 1 presents the times to determine intersecting triangles. The proposed collision detection algorithm achieves better results than RAPID for the five scenarios. The original digger and diggerx5 scenario run interactively.

The time to determine collisions between two objects depends on: (1) the cost of intersecting and updating bounding volumes and triangles; and (2) on the number of such operations [LG98, FMF02]. Tables 2 and 3 show the number of operations executed to determine intersecting triangles for the proposed collision detection algorithm and for RAPID, respectively.

These tables show that the novel approach effectively reduces the number of bounding volume updates when compared to RAPID. The number of cover bounding volume updates is always of two, which is much lower than the number of optimal updates. Optimal updates are less expensive to execute and are tighter to the underlying geometry. RAPID executes fewer bounding volume intersections, but intersecting OBBs is more expensive than intersecting AABBs.

The proposed algorithm is very effective in rejecting pairs of polygons that cannot intersect since it executes a number of triangle intersections comparable to RAPID.

From these results, it is evident that the novel collision detection algorithm is effective in reducing the number of intersecting operations to achieve better performance in a single processor system.

Table 4 shows that the proposed algorithm takes advantage of a system with multiple processors. Results show that performance is improved in a parallel system with a small number of processors.

<table>
<thead>
<tr>
<th>Scene size (# triangles)</th>
<th>Digger</th>
<th>x5</th>
<th>x10</th>
<th>x15</th>
<th>x20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2452</td>
<td>12260</td>
<td>24520</td>
<td>36780</td>
<td>49040</td>
</tr>
<tr>
<td>Number of intersecting steps for the entire assembly simulation</td>
<td>507</td>
<td>507</td>
<td>507</td>
<td>507</td>
<td>507</td>
</tr>
<tr>
<td>Average number intersecting triangles per step</td>
<td>141</td>
<td>4153</td>
<td>14058</td>
<td>37379</td>
<td>66451</td>
</tr>
<tr>
<td>Time to determine intersecting triangles per step (milliseconds)</td>
<td>2.37</td>
<td>62</td>
<td>250</td>
<td>595</td>
<td>1017</td>
</tr>
<tr>
<td>RAPID (milliseconds)</td>
<td>7.73</td>
<td>77</td>
<td>274</td>
<td>632</td>
<td>1030</td>
</tr>
</tbody>
</table>

Table 1: Collision Detection time to find intersecting triangles.

<table>
<thead>
<tr>
<th>Number of AABBs tests</th>
<th>Digger</th>
<th>x5</th>
<th>x10</th>
<th>x15</th>
<th>x20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of AABBs tests</td>
<td>332</td>
<td>81843</td>
<td>326800</td>
<td>734910</td>
<td>130620</td>
</tr>
<tr>
<td>Number of AABBs optimal updates</td>
<td>145</td>
<td>731</td>
<td>1460</td>
<td>2188</td>
<td>2917</td>
</tr>
<tr>
<td>Number of AABBs cover updates</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2: Average number operations per step to determine intersecting triangles.

<table>
<thead>
<tr>
<th>Number of AABBs tests</th>
<th>Digger</th>
<th>x5</th>
<th>x10</th>
<th>x15</th>
<th>x20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of AABBs tests</td>
<td>436</td>
<td>10854</td>
<td>43417</td>
<td>97689</td>
<td>173670</td>
</tr>
</tbody>
</table>

Table 3: Average number operations to determine intersecting triangles with RAPID.
6. Conclusions

This paper presents a novel collision detection algorithm that computes intersecting triangles between polygonal models. This paper showed sequential and parallel techniques incorporated in the design, for the implementation of an efficient collision detection algorithm. Results show that in systems with one processor it compares favourable with RAPID. Furthermore, it can take advantage of multiple processors and achieve additional speed improvements.

7. Acknowledgments

Mauro Figueiredo is being supported by FSE and PRODEP III program project reference 5.3/ALG/194.019/01.

References


