An Efficient Parallel Collision Detection Algorithm for Virtual Prototype Environments

Mauro Figueiredo¹,², Terrence Fernando¹
¹Centre for Virtual Environments, University of Salford, UK
²Escola Superior Tecnologia, Universidade do Algarve, Portugal
MFiguei@ualg.pt, T.Fernando@salford.ac.uk

Abstract

The automatic recognition of geometric constraints in virtual assembly and maintenance operations relies in the determination of intersecting surfaces between virtual prototypes. This is a key challenge in many virtual prototype applications, where it is necessary to find collisions precisely and interactively. This paper presents a novel algorithm to determine intersecting surfaces at interactive speed in a virtual prototyping environment. The proposed algorithm is based on the Overlapping Axis-Aligned Bounding Box (OAABB). The OAABB concept is used effectively to eliminate the number of surfaces that cannot intersect and improve performance. The algorithm also facilitates the implementation using parallel computing methods. OpenMP is used, taking advantage of shared memory multiple processors and reducing the overall time complexity of the collision detection algorithm. To achieve an efficient parallel simulation, it is necessary to provide an efficient load balancing scheme. Our experiences in parallelising the code to achieve a better work distribution are also described. Results show that the proposed collision detection achieves interactive rates in real industrial applications as desired.

Keywords: Parallel applications, shared memory multiprocessors, OpenMP, collision detection, virtual prototype environments.

1. Introduction

The research work presented in this paper is a part of a larger research program, which is being conducted at the University of Salford, to develop a simulation environment that allows designers and engineers to assess assembly, disassembly and maintenance tasks before any physical prototype is built [1]. The motivation for this research arises from the fact that there is still a lack of software tools to address assembly/disassembly operations and maintenance verifications during design. Product development teams still continue to build a physical prototype to assess if a human worker can assemble and disassemble without difficulty later on for service and maintenance.

In an interactive virtual prototype environment, assembly and disassembly operations rely on a set of geometric constraint relationships that are automatically established, or removed, as the user manipulates assembly components. Collision detection plays an important role in recognizing assembly contacts and establishing and removing assembly relationships in an interactive manner. The collision detection process is responsible for: providing collision response to stop object penetration; assisting users to carry out object manipulations; recognition of constraints and many others.

There are several publicly available collision detection toolkits for virtual environments [2]. These collision detection managers support polygonal models and are effective in determining exactly intersecting polygons. However, virtual prototype models, generated by Computer-Aided Design (CAD) systems, are based on surfaces. These collision detection toolkits disregard all the surface data of the CAD model and do not provide information about intersecting surfaces. However the awareness of all the colliding surfaces provides valuable information for the constraint manager to dynamically manage assembly and disassembly relationships during virtual assembly and disassembly operations.

This paper presents an algorithm that determines intersecting surfaces between 3D models, for supporting assembly and maintenance simulation in virtual prototyping environments with many moving objects. The surface knowledge available from the CAD model is used to determine intersecting surfaces. This information is considered to build an effective collision detection algorithm that provides information about colliding surfaces to be used by the constraint manager. In order to determine collisions at interactive frame rates, the proposed algorithm uses the Overlapping Axis-Aligned Bounding Box (OAABB). Furthermore, the paper presents a parallel implementation of the collision detection algorithm using OpenMP. It presents the performance of the algorithm (sequential and parallel) using industrials case studies. The experimental results show that the
implemented algorithm determines intersecting surfaces at interactive rates.

Next section identifies the requirements of a collision detection manager for assisting assembly and maintenance operations in virtual prototyping applications. Section 3 presents the surface collision detection algorithm. The parallelisation of this algorithm with OpenMP is described in section 4. Section 5 presents results. Conclusions are discussed in section 6.

2. Requirements

A collision detection component for assisting in assembly and maintenance operations in virtual prototyping environments should have the following characteristics:

• **Determine Surface Intersections**: Wimalaratne [3] identifies the need for developing efficient surface-based collision software to integrate in a virtual prototyping system for assembly simulation. Virtual prototyping models, generated by CAD systems, are typically surface-based. The recognition of constraints for assembly and disassembly operations relies on the knowledge of the intersecting surfaces. However, current collision detection toolkits lack in providing this knowledge [2].

• **Detect Collisions Precisely**: In the simulation of assembly and disassembly operations, finding precise collisions is an important task for achieving realistic behavior [4]. When assembling two components, it is necessary to find which parts are colliding to determine possible constraints and simulate solidity. Collision detection algorithms implemented only with bounding volumes are ineffective in automatically identifying geometric constraints correctly.

• **Achieve Interactive Response**: Finding collisions in a three-dimensional environment can be a very demanding task. In some environments it can easily consume up to 50% of the total run time [5]. In real industrial case studies, 3D virtual prototypes can be very complex with thousands of primitives. Developing collision detection algorithms for complex environments is an open research issue [6]. Furthermore, rendering must be done in real time, 20Hz for graphics [7], to provide the user the illusion of being immersed and to work interactively.

3. Surface Collision Detection Algorithm

This section presents a novel algorithm for determining intersecting surfaces between a pair of 3D models in a virtual prototyping environment. The collision detection algorithm, described in the following paragraphs, takes advantage of the available 3D data model provided by the CAD model. A 3D virtual prototype model is defined as a collection of surfaces. Each surface is tessellated individually and represented as a collection of triangles.

The cost of finding collisions can be defined by the following expression:

\[ T = N_v \times C_v + N_p \times C_p + N_u \times C_u, \]

where \( T \) is the total time cost function for collision detection, \( N_v \) and \( N_p \) is the number of pairs of bounding volumes and primitives tested for intersection, respectively; \( C_v \) and \( C_p \) is the cost of testing a pair of bounding volumes and primitives for intersection, respectively. \( N_u \) and \( C_u \) is the number and the cost of update bounding volume transformations. The above three factors have been considered during the development of the interactive surface collision detection algorithm. First, we have decided to use axis-aligned bounding boxes (AABB). This type of bounding volumes has a very low value of \( C_v \). However, they do not tightly approximate the underlying geometry, increasing the number \( N_v \). The OAABB concept is used to decrease the number of bounding volumes tests.

The architecture of the proposed algorithm, for the determination of intersecting surfaces between a pair of colliding objects, is presented in Figure 1.
The Surfaces Intersection Manager first checks if the two objects are disjoint. For this purpose, the AABB of each object is considered. If this pair of axis-aligned bounding boxes does not overlap, then the corresponding two objects cannot intersect and the process ends.

Two objects, \( A \) and \( B \), are candidate for collision if the corresponding axis-aligned bounding boxes overlap. In this case, the Overlapping Axis-Aligned Bounding Box, \( OAABB(A,B) \) of the two objects (Figure 2) is determined. This is the volume that is common to the two axis-aligned bounding boxes. Figueiredo et al. [8] introduced this concept for improving performance in the determination of intersecting polygons. Smith et al. [9] also use the same concept together with an octree data structure for finding intersecting polygons. Our approach extends the OAABB concept for surfaces and uses parallel computing methods for improving performance in the determination of intersecting surfaces between a pair of objects. The \( OAABB(A,B) \) of the two objects is used in the next step of the collision detection process. It is considered that the OAABB is defined in the local coordinate system of object \( A \), and represent it by \( OAABB\_A(A,B) \).

Consider the use of the \( OAABB(A,B) \) determined from two candidate objects for intersection. Two surfaces, \( S_i \) and \( S'_i \), are candidate for intersection if they also intersect the \( OAABB(A,B) \) (Figure 3). Surfaces, whose AABBs do not intersect the OAABB, are filtered out. For the remaining \( m \) surfaces of each object, the corresponding AABBs are intersected. Those pairs of surfaces, whose AABBs do not overlap, are filtered out again. This step is also of complexity \( O(m^2) \), but \( m \) is the small number of remaining surfaces of each object.

![Figure 3. The surface filter process shown in 2D.](image)

The Surfaces Filter process receives a pair of objects and is responsible for determining the candidate surfaces for intersection. In this step, the collision process determines pairs of surfaces candidates for intersection, using bounding volumes to improve performance.

Consider that object \( A \) has \( n_s_A \) surfaces, and the object \( B \) has \( n_s_B \) surfaces. A “brute force” implementation for finding pairs of candidate surfaces for intersection, from the two candidate objects, would test for intersection between every AABB of each surface from one object and every AABB of each surface from the other object. The implementation of this step would have the following structure:

```plaintext
for k = 1 to \( n_s_A \)
  for l = 1 to \( n_s_B \)
    if ( \( AABB_k(S_l) \) intersects \( AABB_l(S'_l) \))
      \( S\), \( S'_l \) candidates for collision
```

This approach is of complexity \( O(n^2) \), where \( n \) is the number of surfaces of each object.

To achieve a better performance, the proposed algorithm uses the axis-aligned bounding box of each surface and the overlapping axis-aligned bounding box (Figure 3), to filter out surfaces that cannot intersect.

For every pair of candidate surfaces for collision, \( S_i \) and \( S'_i \), whose AABBs overlap, the \( OAABB(S_i,S'_i) \) of the two surfaces is also determined, to be used by the Polygons Intersection Manager. A list of pairs of candidate surfaces for collision and the corresponding \( OAABB(S_i,S'_i) \) is the output of the Surfaces Filter process. One advantage of using the OAABB concept is that when filtering surfaces from object \( B \), in loop index \( j \), it only needs to do one transformation of coordinate system of the OAABB, reducing the number of updates, \( N_o \), and hence improving performance.

The Polygons Intersection Manager is responsible for determining intersecting polygons. The previous stage determined a set of pairs of surfaces that can possibly collide. Finding if two objects intersect, needs checking for collision every pair of polygons from this set of
surfaces. Intersecting two polygons is an expensive computational operation. Axis-aligned bounding boxes of each polygon can be used to reduce the number of such operations in a process for filtering polygons.

Pairs of polygons whose AABBS do not overlap, cannot intersect and are discarded. At this point, the proposed algorithm again uses the overlapping axis-aligned bounding box (Figure 4).

![Figure 4. Polygon filtering using AABBS and OAABB in 2D.](image)

Two polygons can possibly collide if and only if their AABBS intersect the \( OAABB(S_k, S'_k) \). Consider that surface \( S_k \) has \( np_k \) polygons and \( S'_k \) has \( np_k' \). Therefore, the first operation accomplished in this stage consists in, for each polygon of \( S_k \) and \( S'_k \), testing if its ABB intersects the \( OAABB(S_k, S'_k) \). If there is no intersection, then the corresponding polygon is filtered out. AABBS’ pairs of the remaining polygons are then intersected. Those non-intersecting pairs are filtered out. In the end, there are a small number of pairs of candidate polygons for collision. The final step is to proceed with the intersection of pairs of polygons. If there is a pair of intersecting polygons then the corresponding surfaces intersect. A list of colliding surfaces is constructed during this stage. The implementation of the Polygons Intersection Manager can be described in pseudo-code as follows:

```plaintext
// Surface pair \( (S_u, S'_u) \) is candidate for collision
for u = 1 to \( np_u \)
   if \( ( \text{AABB}_u(P_u) \) intersects \( \text{AABB}_u(S_u, S'_u) \)) { 
      \( P_u \) is candidate for collision
      \( mp_u++ \)
   }

\( \text{OAABB}_u(S_u, S'_u) = M_{u,A} \cdot \text{OAABB}_u(S_u, S'_u) \) // update oper.
for v = 1 to \( np_v \)
   if \( ( \text{AABB}_v(P'_v) \) intersects \( \text{AABB}_v(S_u, S'_u) \)) { 
      \( P'_v \) is candidate for collision
      \( mp_v++ \)
   }
for n = 1 to \( mp_u \)
   \( \text{AABB}_u(P'_n) = M_{u,A} \cdot \text{AABB}_u(P'_n) \) // update oper.
for m = 1 to \( mp_v \)
   if \( ( \text{AABB}_u(P_n) \) intersects \( \text{AABB}_v(P'_n) ) \) { 
      \( P_n^{(A)} = M_{n,A} \cdot P_n^{(A)} \) // update oper.
      if \( (P_n \) intersects \( P'_n) \) { 
         \( S_u, S'_u \) intersect and exit
      }
   }
```

4. Shared-Memory OpenMP

In the development of the proposed surface based collision detection algorithm, for interactive virtual prototyping environments, parallel computing techniques were deployed to enhance the performance of the collision detection process, by utilizing multiple processors.

It was decided to use the shared-memory multiprocessing model since it supports fine-grain communication very effectively. Furthermore, the authors believe that this model is much more suitable for industrial settings since companies tend to have dedicated machines for supporting various design processes. In the future, multi-processor systems with several CPUs will be the norm in industry for supporting moderately compute intensive applications.

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. OpenMP is an extension to a sequential programming language to support shared-memory computation. Currently there are extensions for Fortran, C and C++. OpenMP is becoming the de facto standard for parallelising applications, making it possible to achieve portability across different shared-memory multiprocessors systems. Its directive-based approach for supporting parallelism has two main advantages: it allows the same code to be used for development on both single-processor and multiprocessor platforms; it allows an incremental approach to parallelism, starting from a sequential program, making it easier to parallelise existing code.

The proposed algorithm has a high degree of parallelism and it meets the three Bernstein’s conditions (output independence, flow independence and anti independence). As a result, there is no need to restructure the algorithm for parallelisation.

Parallel regions are more tied to loop constructs in the proposed algorithm. The sequential implementation of the surface collision detection algorithm involves several loops and nested loops. These loops can efficiently be executed in parallel, to reduce the computation time. Hence, first determine the most time consuming loops in the code, and for each loop, the iterations are divided up amongst the available processors. The resulting structure of the parallelised surface collision detection algorithm (Version 1), using OpenMP, is presented below:

```plaintext
Determine \( \text{OAABB}_v(A, B) \)

//Surfaces Filter process
for i = 1 to \( ns_A \)
```
if ( AABB_b(S_i) intersects OABB_b(A,B) ) {
    S_i is candidate for collision
    m_a ++
}

OABB_b(A,B) = M_{A,B} \cdot OABB_b(A,B) // update oper.
for j = 1 to n_{A,B}
    if ( AABB_b(S'_j) intersects OABB_b(A,B) ) {
        S'_j is candidate for collision
        m_a ++
    }
#pragma omp parallel for
for l = 1 to m_a { // parallel loop
    AABB_b(S'_l) = M_{A,B} \cdot OABB_b(S'_l) // update oper.
    for k = 1 to m_a
        if ( AABB_b(S_k) intersects AABB_b(S'_l) ){
            S_k, S'_l candidates for collision
            Determine OABB_b(S_k, S'_l)
        // PolygonsIntersectionManager process
            for u = 1 to n_{p_{A,B}}
                find P_u candidate for collision
                update OABB_b(S_k, S'_l)
            for v = 1 to n_{p_{A,B}}
                find P'_v is candidate for collision
                for n = 1 to m_{p_{A,B}}
                    update AABB_b(P'_v)
            for m = 1 to m_{p_{A,B}}
                if ( AABB_b(P_v) intersects AABB_b(P'_v) ){
                    update P'_v
                    if ( P_v intersects P'_v ){
                        #pragma omp critical
                        store intersecting pair S_k, S'_l;
                        exit
                    }
                }
        // for n
    // if
} // for l

There are three main loops in the algorithm. The first two loops, with indices i and j changing from 1 to n_{A,B} and n_{B} (number of surfaces of each object), determine the surfaces of objects A and B that intersect the OABB(A,B). The third loop, index l, is in fact a nested loop with four loops. Only the third loop, index l, is parallelised since this is the most time consuming loop.

When several nested loops are present, the simplest application of OpenMP applies a #pragma parallel for directive to only one of the loops. This should be the outermost loop so that each thread does as much work as possible to compensate the time spent in creation and termination.

In shared memory multiprocessors, the primary problem is work distribution. The way operations are distributed among the several processors is quite important for the reduction of computation time. In a loop body, the work distribution is better when the number of values of the loop index is much larger than the number of processors that are used to execute the parallel region. For this reason, the two loops with indices i and k are transformed into one loop of index p. Index pairs (l,k) are generated from a one dimensional index p. The resulting parallel loop has much larger p values and it is expected to obtain a better work distribution and consequently achieve a better load balance, improving the performance of the parallelisation process. Following is the resulting structure of the second version implementing this transformation:

SurfaceIntersectionManager Version2 (ObjA, ObjB)
Determine OABB_b(A,B)
//Surfaces Filter process
for i = 1 to n_{A,B}
    if ( AABB_b(S_i) intersects OABB_b(A,B) ) {
        S_i is candidate for collision
        m_a ++
    }
OABB_b(A,B) = M_{A,B} \cdot OABB_b(A,B) //update oper.
for j = 1 to n_{A,B}
    if ( AABB_b(S'_j) intersects OABB_b(A,B) ) {
        S'_j is candidate for collision
        m_a ++
    }
#pragma omp parallel for
for p = 1 to m_a * m_{p_{A,B}} { // parallel loop
    l = p div m_a + 1
    k = p mod m_a + 1
    AABB_b(S'_l) = M_{A,B} \cdot OABB_b(S'_l) // update oper.
    if ( AABB_b(S_k) intersects AABB_b(S'_l) ){
        S_k, S'_l candidates for collision
        Determine OABB_b(S_k, S'_l)
        // PolygonsIntersectionManager process
        for u = 1 to n_{p_{A,B}}
            find P_u candidate for collision
            update OABB_b(S_k, S'_l)
        for v = 1 to n_{p_{A,B}}
            find P'_v is candidate for collision
            for n = 1 to m_{p_{A,B}}
                update AABB_b(P'_v)
            for m = 1 to m_{p_{A,B}}
                if ( AABB_b(P_v) intersects AABB_b(P'_v) ){
                    update P'_v
                    if ( P_v intersects P'_v ){
                        #pragma omp critical
                        store intersecting pair S_k, S'_l;
                        exit
                    }
                }
        // for n
    } // if
} // for p

5. Experimental Results

The surface collision detection algorithm described in this paper was implemented in C++ and parallelised with OpenMP.

Two real industrial case studies of maintenance simulation were executed in a virtual prototyping environment. Case study 1 is a digger model (Figure 5). Case study 2 is based on an aircraft engine electronic control box from the bottom area of the aircraft (Figure 6), from SENER. Table 1 presents the complexity of the 3D models of the two test case scenarios. These two test cases are used to simulate maintenance operations in a virtual prototyping environment. In particular, the case study 2 is being used to demonstrate that the engine electronic control box can be removed and replaced from the
airframe, in less than 60 minutes, following the instructions defined in the standard maintenance handbook.

**Table 1. Complexity of the 3D models used in the test case scenarios**

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Objects</td>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>Number Surfaces / Object</td>
<td>Maximum: 32</td>
<td>510</td>
</tr>
<tr>
<td></td>
<td>Average: 22</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>Minimum: 8</td>
<td>8</td>
</tr>
<tr>
<td>Number Polygons / Surface</td>
<td>Maximum: 176</td>
<td>433</td>
</tr>
<tr>
<td></td>
<td>Average: 22</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Minimum: 2</td>
<td>2</td>
</tr>
<tr>
<td>Number Polygons / Object</td>
<td>Maximum: 736</td>
<td>8,973</td>
</tr>
<tr>
<td></td>
<td>Average: 492</td>
<td>1,000</td>
</tr>
<tr>
<td></td>
<td>Minimum: 104</td>
<td>292</td>
</tr>
</tbody>
</table>

Results were run on an ONYX2 with twelve MIPS10000 processors clocked at 250MHz and 4 gigabytes of RAM. In this research, Parasolid geometric kernel was used to extract surface information and to tessellate each individual surface. The individual surfaces and their corresponding polygonal data were stored in a scene graph (based on the SGI OpenGL Optimizer). The experiments were conducted by one user that executed the required assembly operations. The executed path was recorded and repeated to obtain the data values.

Table 2 presents results obtained for surface based collisions using the OAABB approach. It also presents results obtained when the surface collision detection manager does not use the OAABB concept, for the same objects and trajectories. On average, the use of the OAABB concept reduces the number of AABBS comparisons and updates about an order of magnitude. It also reduces the number of triangle tests. Table 2 shows the contribution of the OAABB concept in the performance of the surface collision detection algorithm.

**Table 2. Results for the surface collision detection algorithm supported by the OAABBs compared to the “Brute Force” approach.**

<table>
<thead>
<tr>
<th></th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average values for:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>case 1 with 508 steps;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>case 2 with 978 steps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number AABBs tests</td>
<td>2,880.8</td>
<td>19,269</td>
<td>22,752</td>
<td>192,000</td>
</tr>
<tr>
<td>Number AABBs updates</td>
<td>97.3</td>
<td>498.5</td>
<td>1,178</td>
<td>20,613</td>
</tr>
<tr>
<td>msA</td>
<td>9.9</td>
<td>25.1</td>
<td>23.6</td>
<td>104.7</td>
</tr>
<tr>
<td>msB</td>
<td>6.9</td>
<td>27.8</td>
<td>19.6</td>
<td>116.3</td>
</tr>
<tr>
<td>mpa</td>
<td>294.3</td>
<td>1,660.2</td>
<td>883.3</td>
<td>3,387.5</td>
</tr>
<tr>
<td>mps</td>
<td>170.8</td>
<td>500.8</td>
<td>825.2</td>
<td>9,910.9</td>
</tr>
<tr>
<td>Number triangle tests</td>
<td>110.2</td>
<td>2,939.2</td>
<td>168.5</td>
<td>4,829.8</td>
</tr>
<tr>
<td>Number intersecting</td>
<td>5.9</td>
<td>11.0</td>
<td>5.9</td>
<td>11.0</td>
</tr>
<tr>
<td>surfaces</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time (ms)</td>
<td>0.51</td>
<td>5.88</td>
<td>2.35</td>
<td>30.13</td>
</tr>
</tbody>
</table>

Table 3 shows the results gained from the RAPID [10] collision detection toolkit, based on Oriented Bounding Boxes (OBB), for the same objects and trajectories. The RAPID toolkit was used for comparison, since this approach presents good performance and functionality [2], when compared with other publicly available toolkits for collision detection.

Tables 2 and 3 shows that the OAABB approach is faster than RAPID. The surface collision detection presented determines intersecting surfaces, taking advantage of the scene graph structure available from the CAD model. For the simulation of maintenance operations, the application needs to know the intersecting surfaces. Two surfaces intersect if there is one pair of intersecting triangles. The use of the OAABBs, at both the object and surface levels, helps in reducing the number of axis-aligned bounding boxes and triangle tests. Rapid uses OBB tests which are about thirty times more expensive.
than AABBS comparisons [2]. Furthermore, the number of bounding volume updates is lower using OAABB than OBBs. When the OAABB approach is configured to determine intersecting surfaces and all intersecting polygons, it compares favourable to RAPID.

Table 3. Results to determine intersecting triangles using Rapid.

<table>
<thead>
<tr>
<th></th>
<th>OAABB configured to determine all intersecting triangles</th>
<th>Rapid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Case 2</td>
<td>Case 1</td>
</tr>
<tr>
<td>Case 2</td>
<td></td>
<td>Case 2</td>
</tr>
<tr>
<td>Number AABBS/OBBs tests</td>
<td>5 103</td>
<td>23 063</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 547.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 675.4</td>
</tr>
<tr>
<td>Number bounding volume updates</td>
<td>253.2</td>
<td>1 458.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 547.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 675.4</td>
</tr>
<tr>
<td>Number triangle tests</td>
<td>4 506</td>
<td>4 467.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>435.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>347.0</td>
</tr>
<tr>
<td>Number intersecting triangles</td>
<td>141.4</td>
<td>134.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>141.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>134.4</td>
</tr>
<tr>
<td>Time (ms)</td>
<td>1.8</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24.5</td>
</tr>
</tbody>
</table>

In these two environment scenarios, the interactions between objects are at a very superficial level. On average there is a small number of intersecting surfaces. The OAABB is particularly suitable for this type of interactions which occur in virtual prototyping environments.

Next, results gathered from our experiments in parallelising the code with OpenMP are presented. Guided, static and dynamic scheduling techniques from OpenMP were employed. The following results were obtained with guided scheduling. Each iteration of the for loop can have a different computation time and, in this case, when distribution of iterations were done at run time better results were achieved. The execution times, speedups and efficiency for the case 1 scenario are shown in Figures 7, 8 and 9, respectively. Theoretically, the use of multiple processors for case 1 improves performance with an efficiency factor greater than 50% until 10 processors. Theoretically, a speedup of 5.4 and execution time of 0.1ms are expected for ten processors. In practice, these values were not achieved. The main reason for this result is that, this scenario is not complex enough to achieve a good work distribution and consequently an efficient load balance. In this case, version 1 has better results than version 2, although the contrary was anticipated. Typically, the second version of the parallelised collision detection algorithm has better work distribution because the two loops are transformed in one loop at the extra cost of computing indices pairs and updating more axis-aligned bounding boxes. However, in case study 1, the small value of indices does not compensate the extra cost introduced with this loop transformation.

The execution times, speedups and efficiency for the case 2 scenario are shown in Figures 10, 11 and 12, respectively.

Figure 10 shows the initial extra cost of the loop transformation from version 1 to version 2 with one processor. In this case, the use of multiple processors improves performance with an efficiency factor greater than 88% until 10 processors. With this number of processors, a speedup value of 8.8 and execution time of 0.75ms was expected. The case 2 scenario is more complex and therefore the use of multiple processors yields better results. Efficiency values greater than 50% was achieved by using 6 and 4 processors for versions 1 and 2, respectively. In this case, Version 2 has better performance. The complexity of the problem compensates the additional cost of the loop transformation.
6. Conclusions

The main contribution of the present paper is the presentation of a faster algorithm that determines precise collisions between surfaces of 3D assembly models in virtual prototype environments. The algorithm presented in this paper uses the overlapping axis-aligned bounding box. The use of OAABBs filters out those primitives that cannot intersect, improving performance. The proposed approach also has high degree of parallelism. It is also shown the application of OpenMP to the parallelisation of the sequential algorithm for collision detection. The code is restructured to achieve better work distribution. Results obtained from real industrial maintenance applications shows that the proposed algorithms and techniques managed to achieve interactive speed and performance.

7. Acknowledgments

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

8. References


