SIACG 2002 - 1st Ibero-American Symposium on Computer Graphics
1-5 July 2002, Guimarães - Portugal

Surface Collision Detection for Maintenance Support in Virtual Environments

Luis Marcelino[1]
L.Marcelino@salford.ac.uk

Mauro Figueiredo[1,2]
Mfiguei@ualg.pt

Terrence Fernando[1]
T.Fernando@salford.ac.uk

[1] Centre for Virtual Environments
University of Salford
University Road, Salford, UK

[2] Escola Superior Tecnologia
Universidade do Algarve
Portugal

Abstract
Detecting collision between surfaces is an important feature for supporting interactive assembly operations based on the geometric constraint-based simulation. CAD models define the geometry of components using surfaces as the basic primitive while current collision detection toolkits, disregard this information and use only polygons. The automatic recognition of geometric constraints can use colliding surfaces to simulate realistic behaviour interactively. This paper presents our research into supporting surface based collision detection for use in immersive Virtual Environments. That resulted in the implementation of a surface collision detection module built on top of RAPID.

Keywords
Collision Detection, Surface, Virtual Environment, Virtual Prototyping

1. INTRODUCTION

Due to many research efforts over the last decade, virtual reality technology is now considered as a viable design tool for industrial applications. In his assessment of VR, Brooks [Brooks99] concludes that VR industrial applications are limited to visualization task or tasks with simple interaction. The industry now requires that these systems integrate realistic behaviour of virtual 3D models, making use of surface geometric data provided by CAD systems.

The Virtual Prototyping Group at the Centre for Virtual Environments at Salford has been exploring the applicability of VR in different product development stages such as maintenance simulation, which involves complex object interaction and control [Fernando01] [Fernando00]. The aim of this research is to develop a simulation environment that allows designers and engineers to assess maintenance tasks before any physical prototype is built. One of the key challenges presented in this research is the simulation of physical realism within the environment to support assembly and disassembly operations on the virtual prototypes. The group has chosen the geometric constraint-based approach to simulate assembly and disassembly operations since this approach is faster and robust for virtual environment applications. In this geometric constraint-based approach, the simulation of an assembly relationship is considered as a constraint specification and satisfaction problem. For example, the creation of an *against* assembly relationship between two blocks involves satisfying an *against* constraint between two planar surfaces. Once this constraint is satisfied, the geometric constraint-based approach allows the user to perform relative constrained motion between the two objects. In this approach, disassembly operations involve breaking the previously defined constraints by applying an external force.

The paradigm presented within the maintenance simulation environment, developed by the Virtual Prototyping Group, is to assemble and disassemble parts within immersive environments such as a CAVE or a Workbench using direct interaction techniques. A constraint manager based on the geometric constraint-based modelling monitors the user manipulations within the environment. While an object is being manipulated, the position of the moving object is sampled to identify collisions between objects. When a collision occurs between two objects, checks are made to identify collisions between surface pairs in order to identify potential assembly relationships. The detection of collision between surface pairs is time consuming and therefore one of the technical challenges in this research is to develop a fast collision detection algorithm that can perform surface collisions.

This paper presents the development of a surface based collision detection toolkit. This toolkit, built on top of RAPID [Gottschalk96], can determine all colliding surfaces of virtual prototypes. RAPID is a public domain collision detection library developed to investigate colliding polygons in a dynamic scenarios, disregarding all the surface data of the CAD model. The awareness of all the colliding surfaces is a valuable information to the constraint manager that can use it to automatically recognize constraints between objects.

The next section gives a brief discussion of the techniques used within the RAPID library. Section 3 describes the implementation of the surface collision detection module, which includes spatialization of surfaces. Section 4 evaluates the implementation of this toolkit using a real industrial case study. Conclusions are reported in section 5.

2. RAPID COLLISION DETECTION

RAPID [Gottschalk96] is a public domain collision detection toolkit developed at the University of North Carolina at Chapel Hill. RAPID defines the geometry of objects by a set of triangles (polygon soup), which are organised spatially using a hierarchy of oriented bounding boxes. An oriented bounding box (OBB) is a rectangular bounding box but with an arbitrary orientation so that it encloses the underlying geometry more tightly. The representation of an oriented bounding box encodes not only position and widths, as in the case of the axis aligned bounding boxes, but also orientation.

For the representation of an oriented bounding box A, fifteen parameters are used to define the centre point C, edge half-lengths a_1, a_2, a_3 and an orientation specified as three mutually orthogonal unit vectors $\mathbf{A}^1$, $\mathbf{A}^2$ and $\mathbf{A}^3$, which are the columns of a 3x3 rotation matrix. With these parameters we can define the bounded region of the oriented bounding box A as:

$$R_A = \{\ \mathbf{P} \in \Re^3 : \mathbf{P} = \mathbf{C} + ka_1\mathbf{A}^1 + sa_2\mathbf{A}^2 + ta_3\mathbf{A}^3 \wedge k,s,t \in [-1,1]\ \}$$

Using a top-down approach, RAPID builds a hierarchical tree of oriented bounding volumes. This tree is constructed as a recursive application of "fit-and-split" operations. Given a collection of polygons, it fits a bounding volume to them, and then partitions the collection into two groups. To each group, fit a new bounding volume and partition again until all leaf nodes are indivisible. The subdivision rule used in the RAPID system is to split the longest axis of an oriented bounding box with a plane orthogonal to one of its axes.

The overall time to build the oriented bounding tree is not an essential issue for virtual environments applications. This is a pre-processing step that does not influence the interactive component of a virtual environment. On the other hand, it is important that the algorithm for the intersection of three-dimensional models, supported by this hierarchical tree of oriented bounding boxes, to run in real time.

In the RAPID system, the algorithm for finding if two OBBs are overlapping is based on the "separating axis" theorem, knowing that two convex polytopes are disjoint if there exists a separating axis orthogonal to a face of one of the polytopes, or orthogonal to an edge from each polytope.

An oriented bounding box has three unique face orientations and three unique edge directions. This leads to fifteen potential separating axes to test, three faces from one OBB, three faces from the other OBB, and nine pairwise combinations of edges. If two oriented bounding boxes are not in contact, then a separating axis exists, and at least one of the fifteen axes mentioned above will be a separating axis. If the polytopes are overlapping, then clearly no separating axis exists. So, testing the fifteen given axes is a sufficient test for determining overlap status of two OBBs.

To find out if two oriented bounding boxes overlap, RAPID's strategy is to project the centres and the half-lengths of the OBBs onto the separating axis **L**. If the distance between the projected box centres is greater than the sum of the projected half-lengths, then the intervals and the corresponding oriented bounding boxes are disjoint. If the two oriented bounding boxes are overlapping then the corresponding children nodes may or may not be disjoint, further tests are required. If this is the case, this procedure is recursively repeated with the children nodes of the oriented bounding box tree current node until the leaf nodes are reached.

Several others public domain collision detection systems developed based on RAPID are also available for polygonal models.

V-COLLIDE [Hudson97] is built on top of the RAPID library and does a sweep-and-prune operation using an axis-aligned bounding box (AABB) for each object. In this first level V-COLLIDE finds all pairs of overlapping bounding boxes eliminating pairs of objects that cannot intersect. Then for each pair of objects, which are potentially in contact, it uses RAPID algorithm based on oriented bounding boxes to detect collisions.

The PQP library, Proximity Query Package, uses two different type of bounding volumes [Larsen99]. It uses rectangle swept spheres (RSS) for distance queries and oriented bounding boxes (OBBs) for collision detection. It is also based on RAPID library.

These public libraries implement different approaches based on bounding volumes hierarchies to speed up collision detection algorithms between general polygonal models with no topological data. The basic idea of these approaches is to approximate the three-dimensional models with bounding volumes to reduce the number of pairs of objects or primitives that are needed to be checked for contact.

Bounding volumes used in these implementations are mainly axis-aligned bounding boxes or oriented bounding boxes. These volumes are used to reject objects or pairs of polygons, which cannot intersect, providing a small number of polygons to be checked pair wise.

3. SYSTEM DESCRIPTION

Industrial designs, developed using modern CAD systems, are defined using geometric surfaces. Such surface details are required to be maintained within our constraint-based design environment to support collisions between surfaces and to simulate interactive assembly modelling operations between parts. However, typical graphical toolkits use polygonal representation to visualise objects. Similarly, the collision detection toolkits, developed for VR applications, are based on triangles. Therefore there is a need for extending the current polygonal representations, used for rendering and collision detection, to perform surface based collisions for supporting interactive assembly modelling operations within virtual environments.

In this research, for supporting surface based collisions, we have explored three different mapping techniques for determining colliding surfaces based on polygon collision detection: surface-polygon mapping, surface-object mapping and spatial mapping. These three methods are designed to optimise the existing features offered by current collision detection toolkits such as RAPID. These are designed for polygonal models and only provide pairs of colliding polygons. We have chosen the RAPID algorithms as our collision detection toolkit to conduct this research. The following section presents the key features of these three approaches.

3.1 Surface-Polygon mapping

This implementation associates polygons with their surfaces. In this approach, a geometric kernel called Parasolid is used to extract surface information and to tessellate each surface individually. These individual surfaces and their corresponding polygonal data are stored in a scenegraph (based on the SGI OpenGL Optimizer). Each object is considered as a collection of surfaces and each surface is maintained within the scenegraph as a separate node. Each surface node in the scenegraph maintains the surface information and its corresponding polygonal data. For example, for a planar surface it maintains a point, normal, bounding box and polygonal data for that surface.

The polygonal data from each surface is then inserted into the collision detection engine (figure 1). Each polygon, inserted into the collision detection module, maintains a pointer onto its parent surface in the scenegraph.

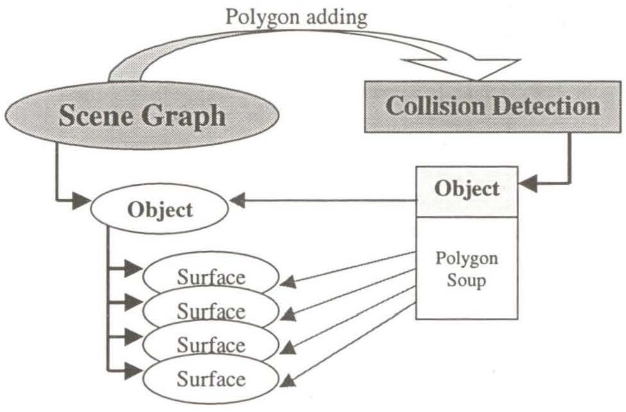

Figure 1: The surface-polygon mapping data structure.

When two objects are tested for collision, the detection engine is configured to return all intersecting triangles. These triangles are then processed to identify all the colliding surface pairs.

A consequence of this approach is that the potentially large amount of redundant computation is required since the algorithm has to check collisions between all the possible polygonal pairs. However, it is obvious that we conclude that two surfaces are colliding when two polygons (one from each surface) are colliding and hence avoid further collision checks between the colliding surfaces. However, since typical collision detection libraries, such as RAPID, maintain polygons for the objects as "polygon soups" it is not possible to eliminate redundant collision checks between polygons.

3.2 Surface-Object mapping

A different implementation creates multiple objects in the collision detection engine from one single virtual object. The virtual object is first added into the detection engine as a whole. Then surfaces are added into the collision engine as separate objects: one surface is one colliding object. We named this approach surface-object mapping (figure 2).

All objects added to the collision detection engine are organized hierarchically and a connection is maintained between each surface and its representation as a colliding object. The whole object is the top node of this hierarchy of objects. Surface-objects are underneath this node. This hierarchy minimize the tests of non-colliding objects.

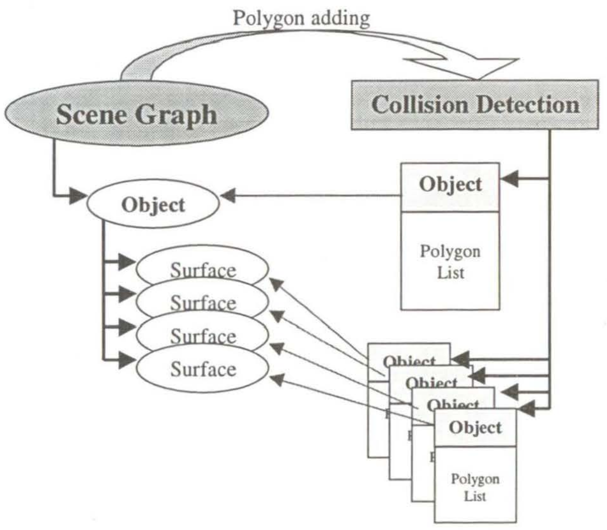

Figure 2: The surface-object mapping data structure.

Our current implementation of the surface-object collision detection starts by testing the each object's top nodes. If both objects' top nodes are colliding, in principal all their surfaces needs to be tested for intersection. However, the testing of all possible surface pairs will lead to O(N^2) complexity and therefore techniques are required to reduce the number of checks made. The following approach was initially used to eliminate unnecessary checks.

The first collision test determines whether there is any contact between both objects or not. This test uses both objects' complete representation (i.e. the top node of the colliding-objects hierarchy). If objects are colliding, their surfaces are screened in a second series of intersection tests. Here, each surface from one object is tested against the complete model of the other colliding object. If the result is positive, the surface is added to a list of colliding surfaces. This test is repeated to all surfaces of both objects. At the end of these series of tests there are two lists of colliding surfaces, one for each object. The next step is to test the surfaces on the lists among themselves. The positive intersection tests of this last step are the colliding surfaces.

Table 1 presents the surface-object algorithm described above, where the keyword `colliding` indicate a RAPID collision detection test. Although these RAPID collision test stop at the first intersecting polygon, the overhead associated with the integration of surfaces is significant.

```
if object1 colliding object2
{
      collList // List of colliding surfaces
      for each surfaceX of object1
      {
         if surfaceX colliding object2
            add surfaceX to collList
      }
      for each surfaceY of object2
         if surfaceY colliding object1
            for surfaceX of collList
                  if surfaceX colliding surfaceY
                        add colliding pair to result
}
```

Table 1: Suface-object mapping algorithm

The surface-object collision detection is better suited for large surfaces models. This approach eliminates redundant tests between multiple colliding triangles of the same surface. We believe that a more sophisticated hierarchy of colliding objects can improve the performance of this object-surface implementation. That may be a subject for further research.

3.3 Spatial Mapping

Spatial mapping consists in partitioning the volume of the object into cells. These cells are divisions of the object's bounding box used to localise the area of collision and efficiently eliminate surfaces that are not in this area. The spatial mapping is a pre-processing stage that does not determine whether surfaces are intersecting or not. It reduces the number of surfaces to be tested by the collision detection toolkit.

The cells result from the division of each edge of the object's bounding volume in three equal segments, dividing the bounding volume into 27 identical cells. Each of these cells corresponds to a bit mask that uniquely identifies this cell. Each surface is then signed with a bit mask that determines which cells are occupied by this surface. A surface can be in more than one cell and one cell can include multiple surfaces.

The spatial organization of data can be used to eliminate some surfaces that are not colliding. It is an inexpensive test compared to the one described in the surface-object mapping, although it only eliminates surfaces that are not in colliding cells. Surfaces that are in these cells but are not colliding are not filtered. For this reason a combination of both approaches is necessary with surfaces still being added to the polygonal collision detection toolkit as objects (as described in the surface-object mapping).

We implemented and assessed different spatial mapping algorithms, attempting to reduce efficiently the number of surfaces to be tested by the polygonal collision detection. Two of the algorithms that performed better are described bellow.

A first implementation of this algorithm tested all cells of one object for intersection with all cells of another object. All the 27 cells of each object were checked for intersection, resulting in 27x27 intersection tests. These tests yielded pairs of intersecting cells that defined the pairs of surfaces that were potentially colliding. By checking all cells we eliminated pairs surfaces that were in colliding cells but that were not colliding with each other. Although this filtering was faster than the one described in the surface-object mapping algorithm, it eliminated fewer surfaces that the surface object mapping. This ineffective filtering passed more surfaces to the last stage of the collision detection process, requiring more tests at the latest stage, using the polygon-based collision detection. As a result this algorithm performed worse than both the surface-object and the surface-polygon approaches.

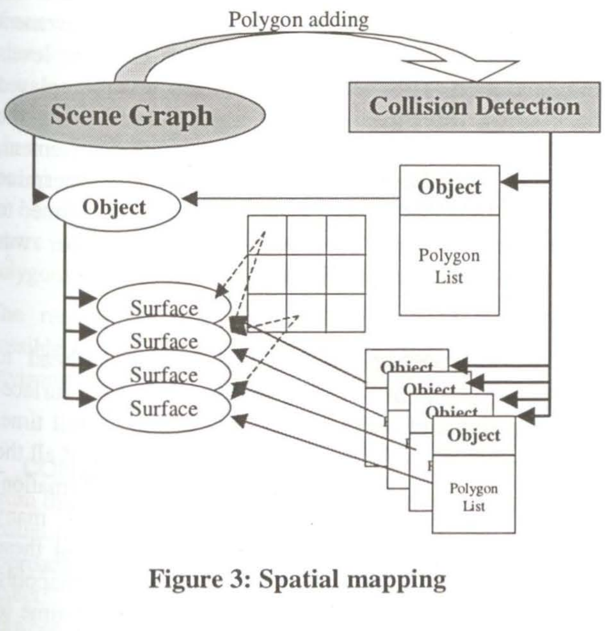

Figure 3: Spatial mapping

A refined algorithm benefits from the speed of spatial mapping to complement the accuracy of surface-object mapping. It started with an intersection test between the bounding boxes of both objects. If these volumes were intersecting further tests were required to determine the colliding surfaces. Then each cell of one object was tested with the bounding box of other object. This only involved 2x27 intersection tests: the 27 cells of a component A against the bounding volume of a component B and the 27 cells of a component B against the bounding volume of a component A. These tests determined the colliding cells of each component and not the pair of colliding cells like in the 27x27 intersection tests. This resulted in more surfaces being passed to the following stage. The cells intersection test is like a coarse filter that eliminates surfaces from further testing. This filtering is not very tight and surfaces that are not colliding but occupying an intersecting cell are passed to the next filtering stage. However, it is faster than other filtering processes, which make it ideal for an initial screening. The surfaces occupying colliding cells are then passed to the next filtering stage: the one described in the surface-object mapping. Table 2 outlines this implementation algorithm.

```
if object1 intersects object2
{
      collCells1 // colliding cells of obj 1
      collCells2 // colliding cells of obj 2

      BB1 // Bounding box of object1
      BB2 // Bounding box of object2

      For each cellX of object1
         if cellX intersects BB2
            add cellX to collCells1

      for each cellY of object2
         if cellY intersects BB1
            add cellY to collCells2

      collList // List of colliding surfaces
      for each surfaceX of object1
         if surfaceX in collCells1
            if surfaceX colliding object2
               add surfaceX to collList

      for each surfaceY of object2
         if surfaceY in collCells2
            if surface Y colliding object1
               for surfaceX of collList
                     if surfaceX colliding surfaceY
                           add colliding pair to re-
                           sult
}
```

Table 2: Spatial mapping algorithm

4. EXPERIMENTAL RESULTS

We conducted a set of experiments to assess the different algorithms for determining colliding surfaces and the overhead of integrating surface information into the collision detection process.

We executed several different tests that showed that the time to determine all colliding surfaces depended on the characteristics of the colliding models. The surface-polygon approach is better suited for surfaces with small number of polygons, while surface-object performs better when used with surfaces with many polygons. In this situation, the surface-object collision detection can even outperform the RAPID collision detection of all colliding triangles.

From the tests involving objects of different complexities we concluded that for objects with an average of up to 30 polygons per surface the surface-polygon mapping performed better in determining the colliding surfaces. For more complex objects, the surface object mapping showed to be more adequate. The virtual prototypes that were provided to us by our industrial partners, when tessellated with a tolerance of 0.5 millimetres, have approximately this average of polygons per surface. Therefore, it is important to investigate the surface collision detection performance in this situation. The experiments described in this section aim to illustrate a realistic scenario in a virtual prototyping environment.

A real industrial case study from Rolls Royce was used in this assessment. The maintenance scenario was composed of a Fuel Metering Unit and an Oil Pump from the Trent 800 engine. This is the engine used in the Boeing 777.

The Fuel Metering Unit is a component with 142 surfaces tessellated to 5472 triangles and the Oil Pump has 912 surfaces, totalling 25302 triangles. Figure shows the fuel-metering unit with its surfaces coloured randomly and the cells dividing the object's space. Figure 5 shows the oil pump with all its surfaces also coloured randomly.

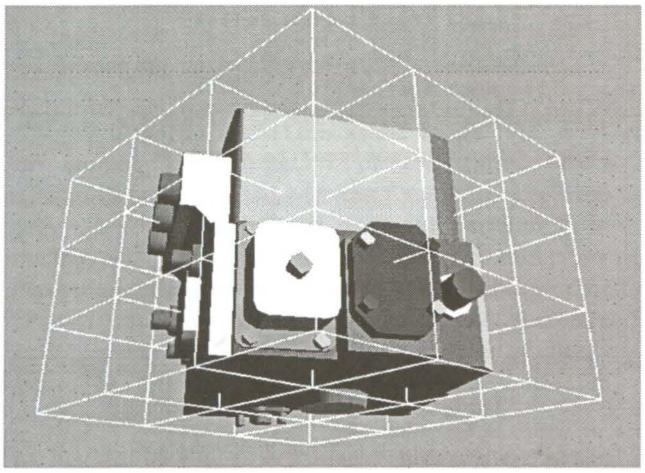

Figure 4: The Fuel Metering Unit with coloured surfaces

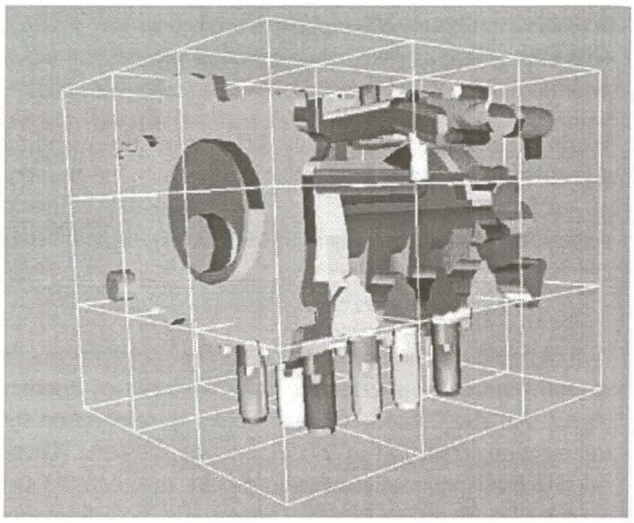

Figure 5: The Oil Pump with coloured surfaces

These tests were performed on an ONYX2 with MIPS10000 processors clocked at 250MHz and 4 gigabytes of RAM. Only one CPU was used, because this collision detection software is not a multithreaded application.

All trials were performed under identical circumstances. To achieve this an interaction was recorded where the user moved the Fuel Metering Unit around the Oil Pump, which remained static. During this movement both objects collided several times with different levels of penetration. The recorded interaction was then played back using the three different algorithms. During the playback of the fuel-metering unit's recorded movement, the collision detection was timed. The time to determine the colliding polygons, their number, the time to determine colliding surfaces and their number was recorded.

The recorded results revealed that the overhead to convert from polygons to surfaces using the surface-polygon mapping was less than 6% of the overall time. The expensive operation in this case is the test of all the polygons, many of which are just redundant information. The surface-polygon mapping consists of many inexpensive tests, but is the required quantity of these tests that limits the performance. The spatial mapping reduces the number of these tests, reducing the time to determine colliding surfaces.

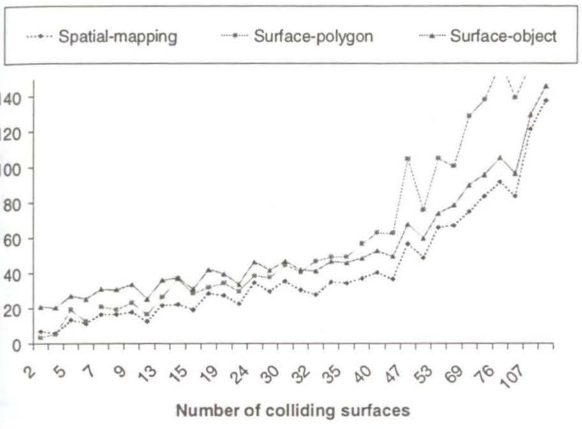

Figure 6 –Performance of the three implemented mapping techniques

Figure 6 shows the time to determine all colliding surfaces as a function of the number of colliding surfaces. The graph shows a deterioration of the surface-polygon mapping results when compared to the other two approaches. The performance of surface-polygon mapping is comparable with the spatial mapping when objects are colliding superficially, involving few surface intersections. Once objects are penetrating more deeply the surface polygon mapping degrade considerably when compared to the spatial mapping. The use of a spatial data structure to localise the colliding area allows the use of complex virtual prototypes in interactive systems. For a number of colliding surfaces between 30 and 50 there is a significant improve of performance, pushing to interactive rates values that are beyond interactivity when using a simple surface polygon mapping. Using a spatial approach together with surface object mapping we have achieved interactivity in very complex situations where we have found about two thousand intersecting polygons representing two hundred intersecting surfaces.

The results presented above demonstrate that it is possible to use surface collision detection in interactive environments.

5. CONCLUSION

From the results we conclude that the hybrid approach of spatial mapping and surface-object mapping are the best approach to determine colliding surfaces of complex object. The time to determine all colliding surfaces between objects leaves a limited time to compute the collision response at interactive rates. Nevertheless, the surface collision detection is essential for the automatic recognition of geometric constraints that determine the collision response. This work presents a solution for determining surface intersection between complex objects like real case studies of virtual prototypes.

This work resulted in a library built on top of RAPID that is able to identify all colliding surfaces. Currently it implements a surface-object mapping, a surface-polygon mapping and a spatial mapping approach for surface collision detection. The library operation mode can be adjusted to the complexity of the models to be manipulated. For use in virtual prototyping environments we find the spatial mapping algorithm to yield better performance.

6. ACKNOWLEDGEMENTS

Mauro Figueiredo is grateful to Calouste Gulbenkian Foundation for financing and supporting his work.

We would like to thank to Rolls Royce for providing the case study used for these experiments.

7. REFERENCES

[Brooks99] Brooks, F.P., "What's Real About Virtual Reality?", IEEE Computer Graphics and Applications, November/December 1999

[Fernando01] Fernando, T, Marcelino, L, Wimalaratne, P, "Constraint-based Immersive Virtual Environment for Supporting Assembly and Maintenance Tasks", HCII 2001, August 2001, New Orleans, pp

[Fernando00]_Fernando, T, Marcelino, L, Wimalaratne, P, Tan, K, "Interactive Assembly Modelling within a CAVE Environment", 9 Eurographics Portuguese Chapter Meeting, February 2000, Marinha Grande, pp.43-49

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

[Hudson97] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk and D. Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. In *Proc. of VRML*, pp. 119-125, 1997.

[Larsen99] E. Larsen, S. Gottschalk, M. C. Lin, D. Manocha. *Fast Proximity Queries with Swept Sphere Volumes*. Technical report TR99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, 1999.