CONTACT: Arbitrary in-between motions for collision detection

S. Redon
INRIA-Rocquencourt

A. Kheddar
CEMIF-SC
Université d'Evry

S. Coquillart
INRIA-Rocquencourt

Abstract

A new approach to the collision detection problem was introduced in [8], that allows to detect collisions continuously and efficiently between polyhedral primitives (vertices, edges and faces). This paper extends the results of [8] to continuously detect collisions between pairs of complex polyhedral objects. A C++ library, CONTACT, has been developped. The tests of this library, reported here, seem to show that this approach is especially suited for precise real-time interaction in virtual environments.

1 Introduction

Collision detection (CD) has become an important research topic in many fields. In computer graphics, virtual reality, robotics, or engineering, efficient algorithms are needed to avoid the CD problem to become a major bottleneck of the applications developped. In virtual reality, especially, collision detection is needed in order to prevent objects from interpenetrating each other, and give them a physically correct behavior.

A recent survey can be found in Lin *et al.*[5]. Collision detection techniques are traditionnally split into two categories. Whereas *discrete* techniques detect interpenetrations between objects at successive discrete instants, *continuous* techniques use the object motion to compute the time of first collision. Discrete techniques are generally faster than continuous ones, but require *backtracking methods* to find the time of first collision[1]. These backtracking methods are difficult to implement as soon as the objects are complex (non-convex objects for example). Redon *et al.*[8] suggest to replace the complex (or unknown) object motion between successive instants (*eg* over successive *timesteps*) by an arbitrary rigid motion. This motion is general enough to interpolate any two successive positions, and simple enough to efficiently and continuously detect collisions between primitives (vertices, edges and faces): the time of first collision is a root of a polynomial whose degree is three or less. Since collision detection is efficient, the time interval between successive positions can be small and the difference between the real object motion and the arbitrary one tends to be negligible[2].

Whereas detecting collisions between primitives is theoretically sufficient to detect collisions between complete polyhedral objects[3], it is generally impractical to test for all possible pairs of *elementary* contacts (edge/edge or vertex/face). In order to eliminate quickly irrelevant pairs, many CD methods use *bounding-volume hierarchies* (BVHs). This paper adapts the specific case of sphere-trees to fit in the continuous approach given in [8], and describes a complete CD method that can efficiently detect collisions between pairs of complex rigid polyhedral objects.

Section 2 briefly recalls the results of Redon *et al.*[8]. Section 3 introduces the use of sphere-trees in a continuous CD scheme. Section 4 describes various optimizations. Especially we introduce a general optimization technique that can be used with any BVH method. Section 5 presents two applications that were designed to test the validity of the approach, while Section 6 concludes the paper.

2 Elementary collision detection

For simplicity, let's assume that the time interval between two successive positions is $[0,1]$ (*eg* the timestep size is 1). Finding an arbitrary in-between motion amounts to interpolate two successive rigid object positions. Using screwings[4], [8] suggested to define a class of in-between motions by choosing two

[1] For example, the time of first collision is required for analytical rigid body simulation methods.

[2] In a rigid body simulator, for example, the successive positions are computed by the constraints solver (one computation per frame), and the object's motions are replaced by arbitrary ones *between these positions only.*

[3] For rigid polyhedral objects, only three (non exclusive) contact types can occur for two rigid polyhedral objects A and B. Either an edge of A contacts an edge of B, or a vertex of A contacts a face of B, or a vertex of B contacts a face of A.

[4] A screwing $\mathcal{V}(\omega, s, O, \vec{u})$ is a composition of a rotation around the axis $(O, \vec{u})$ and a translation along the same axis. The rotation angle is ω, and the translation amount is s. It is well known that if A_0 describes an object at time 0 and A_1 describes the same object at time 1, then there exists a screwing $\mathcal{V}(\omega, s, O, \vec{u})$ such as $\mathcal{V}(A_0) = A_1$.

functions a, b: $\mathbb{R}^2 \times [0,1] \to \mathbb{R}$ such as, for every pair (ω, s) in $\mathbb{R}^2$, the functions:

$$a_{\omega,s} : \left\{ \begin{array}{c} [0,1] \to \mathbb{R} \\ t \mapsto \omega' = a(\omega, s, t) \end{array} \right.$$
$$b_{\omega,s} : \left\{ \begin{array}{c} [0,1] \to \mathbb{R} \\ t \mapsto s' = b(\omega, s, t) \end{array} \right.$$

are C^1, increasing, and such as $a_{\omega,s}(0) = b_{\omega,s}(0) = 0$ and $a_{\omega,s}(1) = \omega$ and $b_{\omega,s}(1) = s$.

For a given screwing, this defines a class of continuous in-between screwing-based motions, whose general form is:

$$\mathcal{M} : \left\{ \begin{array}{c} [0,1] \times \mathbb{R}^3 \to \mathbb{R}^3 \\ (t, A) \mapsto A' = \mathcal{V}(a_{\omega,s}(t), b_{\omega,s}(t), O, \vec{u})(A) \end{array} \right.$$

where A is the object at time 0 and A' the object during the in-between motion[5].

It is chosen to consider only the relative motion of two objects[6], so as to make the CD equation simpler by solving it in the screwing frame, in which the rotation axis is the third one. Thus, in the edge/edge problem, the second edge is always static and, in the vertex/face problem, the face is always static (in the face/vertex problem, the opposite relative motion is considered, so that the face is also static).

Two particular functions were shown to solve efficiently both elementary problems. These functions are:

$$\left\{ \begin{array}{c} a(\omega, s, t) = \omega t \\ b(\omega, s, t) = s f(t) \end{array} \right.$$

where f has the following form:

$$f(t) = \left\{ \begin{array}{ll} t & \text{if } \omega = 0 \\ \frac{\tan(\omega t/2)}{\tan(\omega/2)} & \text{else} \end{array} \right. .$$

Using these functions, it can indeed be shown that both problems lead to a polynomial CD equation whose degree is three or less. This polynomial equation is solved explicitly, and very quickly, using Cardano's fomulas.

[5] It is important to notice that the two functions a and b depend only on ω and s, the screwing parameters, and not on the object characteristics. Thanks to this, all of the object elements (vertices, edges and faces) have the same rigid motion.

[6] This can be a problem for multiple moving objects : if A has a particular screwing-based motion relatively to B, and another one relatively to C, then A has apparently two different motions in the global frame, leading to incoherent configurations. We believe that this problem can be avoided, however, by reducing the timestep size, as it has been shown in test applications. For exactness care, other functions a and b than the one given in this paper are currently under research.

3 Bounding Volumes Hierarchies

To detect collisions efficiently *between two objects*, it isn't enough to be able to solve efficiently the CD equation for pairs of primitives. Indeed, if v_A (resp. v_B), e_A (resp. e_B), and f_A (resp. f_B) denote the number of vertices, edges and faces of object A (resp. B), then the total computational cost of detecting a collision between A and B would be proportional to $v_A f_B + v_B f_A + e_A e_B$. A method is thus needed to eliminate quickly irrelevant pairs. Many collision detection algorithms use *bounding-volume hierarchies* (BVH).

The idea behind BVHs is quite simple: each rigid object is associated to a hierarchy of bounding volumes, organized in a tree[5]. Using BVHs is straightforward. If two rigid objects are respectiveley associated to two bounding volumes hierarchies, then a simple recursive function simultaneously descending both trees may eliminate many elementary CD tests, by detecting collisions between smaller and smaller BVs([5]). Note that in the case of a continuous CD technique, collisions must be detected between *moving* BVHs, and not just interpenetrations at fixed moments. Of course, over a timestep, the BVHs have the same motion $\mathcal{M}$ than the primitives they contain.

3.1 Sphere-trees

Various examples of bounding volumes have been studied in the literature, most of them for *discrete* collision detection. Well known examples are spheres [7, 3], axis-aligned bounding boxes, k-DOPs [4], spherical shells [11], or oriented bounding boxes [2].

It was chosen to use spheres because of their adaptability to our continuous case. Two spheres intersect if and only if the distance d between their center is less than the sum of their radius. Let c_A and c_B denote the centers, and r_A and r_B denote the radii. If the first sphere is moving according to $\mathcal{M}$, while the second one is static, then the problem is to find the first time in $[0,1]$ for which :

$$d(\mathcal{M}(t, c_A), c_B)^2 \leq (r_A + r_B)^2. \qquad (1)$$

However, equation (1) leads to a polynomial equation whose degree is four or less. Fortunately, it is not necessary to know *when* the spheres collide, during the timestep. While the exact collision time could be a helpful information, in order to speed up the detection (see Section 4), it is indeed enough to know that they collide, since they are just bounding volumes. Thus, it is enough to look for the roots of the derivative, whose degree is three or less. If no collision is detected, the function returns any value meaning that there is no collision during the timestep (for example, -1, since a valid collision time must be in $[0,1]$).

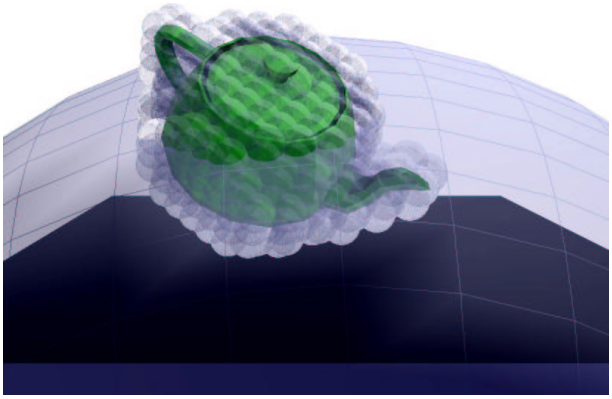

Figure 1: The teapot is half inside the cube's large leaf sphere, resulting in many irrelevant elementary tests. To avoid this situation, all the leaf spheres in a scene have to be approximately the same size.

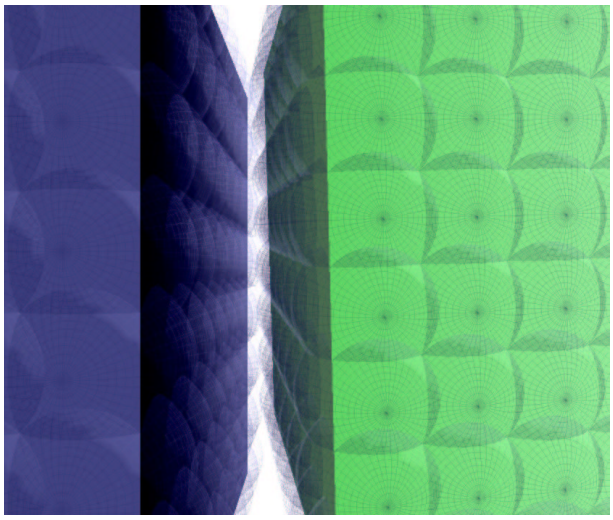

Figure 2: Problematic situation without labels. Numerous colliding leaf spheres will result in testing many times the same pair of faces.

Various methods exist to build sphere-trees[3]. We use an approach similar to that proposed by Quinlan[7] and detailed in Ruspini[9]. In this bottom-up approach, an object is first covered by small spheres, which will be the leaf-spheres of the final tree. All of these small spheres have the same user-defined size, which isn't *a priori* related to the size of the primitives. Thus, a leaf-sphere may contain a few vertices, (eventually parts of) edges, and (eventually parts of) faces. Conversely, an edge or a face may be covered by an arbitrary number of spheres. One traditionnal rule, however, is that the leaf-spheres contain only a few primitives (or parts of primitives). Once the leaf-spheres are created, a divide-and-conquer strategy creates the interior nodes (which are spheres too) to build a complete tree[9].

4 Speeding up the detection

4.1 Caching vertices coordinates

Each time an elementary CD occurs, the coordinates of the vertices characterizing the primitives (extremities of the edges and vertices of the faces) have to be given in the screwing frame. Unfortunately, some vertices can belong to many primitives, and some primitives can be located in many leaf spheres. Moreover, a leaf sphere associated to an object A can be tested with many leaf spheres associated to another object B. In order to prevent the re-calculation of the coordinates each time a leaf sphere is re-tested, these coordinates are cached.

4.2 Bounding the sphere's trajectory

In the general case of spheres collision detection (rotation *and* translation), where the degree of the polynomial equation is three, it is possible to test, first, for a collision between a volume bounding the first sphere's trajectory and the second sphere. This bounding volume is a cylinder whose axis is the z-axis of the screwing frame. Most of the time, this allows to avoid the resolution of the general equation. Moreover, in specific cases (pure rotations and pure translations), it is possible to achieve the tests without any division.

4.3 Passing the current time-of-first-collision

Let's consider a collision detection between two objects. Since we are designing a continuous CD method, we are only interested in the *first* collision time in $[0, 1]$ (and the primitives colliding at that time). Thus, during the CD process between the two objects (*eg* during both trees traversal), we maintain a *current time-of-first-collision* t_c. At the beginning of the CD process, t_c has any value meaning that no collision has been detected (for example, $t_c = -1$). Then, whenever a collision is detected between two primitives, t_c is updated if the new collision occurs in $[0, t_c]$ (if $t_c = -1$, then it is automatically set to the new collision time). Thus we can speed up the detection by passing t_c to the CD functions (*eg* the sphere/sphere, vertex/face and egdge/edge CD functions), so as to check for collisions in $[0, t_c]$ only instead of $[0, 1]$. This is especially

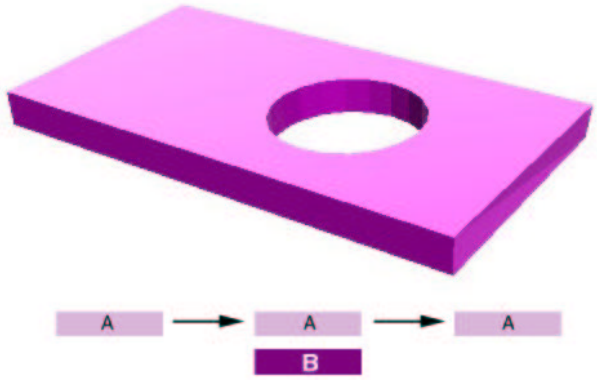

Figure 3: Test object for the label optimization technique. Object A comes close to object B and then goes far again.

useful in the sphere/sphere case, since many elementary tests can be avoided.

4.4 Labelling the spheres

Finally, an optimization technique proved to be most useful in some applications, and is based on a simple observation: the coherence in the BVHs. In a scene composed of mobile and static objects, indeed, a simple rule governs the size of the leaf spheres: they must be approximately the same size, for all objects. It is not enough that they contain only a few primitives, since a leaf sphere of an object A can contain many leaf spheres of an object B, as in Figure 1. In the case depicted in this figure, A is a simple cube, whose sphere-tree's depth is one, and B is a teapot, whose sphere-tree's depth is five (only the leaf spheres are drawn). Since the teapot is half entered into the sphere associated to the cube, many leaf spheres have to be tested, thus loosing the benefit of using a deep tree to speed up the collision detection.

This rule can lead, however, to many leaf spheres covering a single face. If two such faces, each one belonging to a different object, come close to each other (Figure 2), then the BVH algorithm will test the two faces as many times as there exists colliding pairs of leaf spheres. A way to avoid this is, for a given object, to add a *label* to the nodes of a BVH. Two leaf nodes that contain the same set of primitives have the same label (an integer for example). For the internal nodes, a simple rule is added. If all of the children of a node have the same label, then the same label is given to this node. If not, then a new label is created. When two nodes are processed, it is first checked whether the pair formed by their labels has already been tested. If it has, then the node-testing function returns. If not, the node-testing process continues. When two leaf nodes have been completely tested (that is, each possible elementary test has occured), then the pair formed by their labels is marked as tested. A hashtable is used to store the tested pairs of labels. To complete this technique, three more hashtables are used to store the tested pairs of primitives (one for edge/edge tests, one for vertex/face tests, and one for face/vertex tests). This optimization has proved to reduce significantly the total computation in cases similar to the one depicted in Figure 2. A simple test is reported in Table 1. In this test, two objects identical to the one of Figure 3 are placed in a scene. Object B is static, while object A is mobile and follows a recorded path. At the beginning of the path, the mobile object is far from the static one. It then comes very close to the static one, and finally goes far from it. Table 1 reports the total number of sphere/sphere (SS), edge/edge (EE), vertex/face (VF) and face/vertex (FV) tests along the whole path according to the hashtables used. It shows that in this example case, using the sphere/sphere hashtable alone allows to eliminate 17% of the CD tests and that using the four hashtables alltogether allows to eliminate more than 20% of the CD tests.

	No hasht.	SS	EE-VF-FV	Four hasht.
SS Tests	1.213.596	1.017.112	1.213.596	1.017.112
EE Tests	87.284	61.916	20.717	20.717
VF Tests	3.963	3.155	2.367	2.367
FV Tests	3.858	3.098	2.317	2.317
Total	1.308.701	1.085.281	1.238.997	1.042.513

Table 1: The label technique and the hashtables allows to reduce the total number of tests.

One final advantage of the label technique is that it can be used to factor the BVH associated to the object and thus reduce the total amount of memory required to store the tree. Note that this method only exploits the fact that the optimization structure is a tree, and does not depend on the nature of the bounding volumes.

5 Discussion

The algorithms described in this paper have been implemented in a portable C++ library: CONTACT (CONTinuous and Accurate Collision Tracking). This section describes two test applications designed to show the validity of the approach. The applications are portable, since the graphics are done with OpenGl, and have been tested on PCs and SGIs. Thanks to its stereo capability, the latter system permitted to check more easily the accuracy and the correctness of the collision detection.

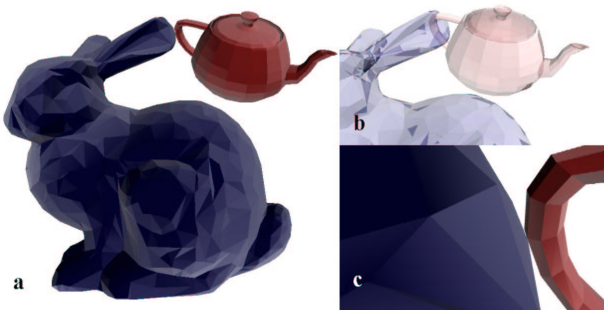

Figure 4: Object manipulation. In *a*, the objects contact but do not interpenetrate, as they would have if a discrete CD method had been used (*b*). The user can navigate the scene to check the collision detection (*c*).

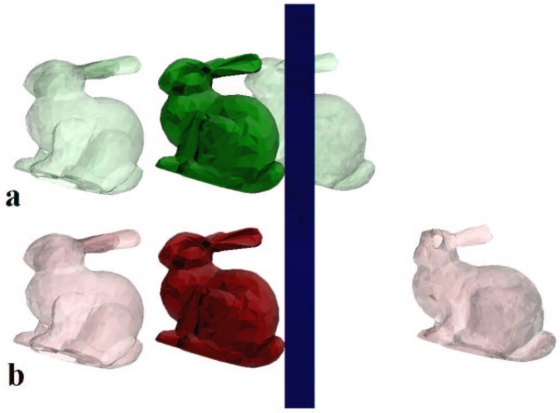

Figure 5: A problematic situation for a discrete CD method. Whereas such a method would detect an interpenetration in case *a*, it couldn't stop the bunny in case *b*. Using the arbitrary in-between motion given in Section 2 allows to stop the bunny in both cases.

5.1 Object manipulation

This first test was designed to show the comfort and the realism obtained thanks to a real-time continuous CD method. It consisted in a simple application in which the scene is composed of two objects: a mobile object, manipulated by the user, and a static one. The object manipulation was done with the keyboard on the PC and with a 6-degree-of-freedom device on the SGIs. Since the goal of the application was to study the CD efficiency and not the *collision response* problem, the mobile object was stopped whenever it contacted the static one. The user had then the opportunity to navigate the scene, in order to check that the objects *contacted* but did not interpenetrate. He or she had also the possibility to display the intentional final object position, that is, the position the object would have had if no collision had occured, or if a discrete CD method had been used.

Figure 4 shows a mobile teapot that has just contacted a static bunny. Part *c* of the figure is a close-up of the scene seen in part *a*, made by navigating in the scene, after the detected collision stopped the teapot motion. Part *b* of the figure shows the intentional final position of the teapot. Both objects are turned transparent, so as to better see the interpenetration that would have occured if a discrete CD method had been used.

Figure 5 shows a problematic situation for a discrete CD method, that doesn't occur in a continuous scheme. In cases *a* and *b*, the bunny is both translating, from left to right (the transparent bunnies are the initial and final positions of the plain one), and slightly rotating, which results in a complex motion. Since this motion occurs during a *single* timestep, a discrete CD method could detect a collision in case *a* only, and would fail in case *b*. In both cases, however, CONTACT detects a collision and stops the bunny.

5.2 Bouncing objects

This second test was designed to show the validity of the approach for real-time impulsive dynamics simulations. The laws of physics for impulsive contacts are well known ([6]), and readily implementable when the contact locations are known. Since they are given by our continuous CD method, we could easily add a simple dynamics engine, and create a test application in which various objects (one at a time) could bounce on a floor.

The CD library proved to be fast enough to allow objects with up to several thousands of triangles to bounce with a constant frame rate higer than 30 Hz on a PC, while maintaining a sufficiently short timestep to make the motion physically realistic. Moreover, the fact that no backtracking method was needed to give the objects a realistic position (and to locate the contact) when a collision occured helped to maintain a high frame rate. Finally, the simulation proved to be *extremely robust* (no incoherent behaviour, even with random initial velocities), which is most important in an interactive application. We found that some commercial products, even for simple similar simulations, have to be tuned carefully in order to produce realistic results. These systems use indeed a discrete collision detection technique and their collision response module does not seem to be able to handle unshared edges, or pure surfaces that are not actual volumes, such as

the teapot in Figure 4.

A demo mpeg file is available[12]. It is the recording of the real-time test application, running on a PC. It shows a simple teapot boucing on a floor, with random initial velocities. It is the same teapot that was problematic in some commercial products.

6 Conclusion

In [8], a new general approach to the collision detection problem was introduced, that consisted in using arbitrary in-between motions. The specific in-between motion that was given allowed to detect collisions between rigid polyhedral primitives (vertices, edges and faces). This paper has described how sphere-trees could be included in this approach to create a complete CD structure, enabling continuous collision detection between complex rigid polyhedral objects. Optimization techniques that were used to develop CONTACT, the C++ library based upon this approach, have been described, as well as test applications implemented thanks to this library. These tests allowed to demonstrate that this approach seems especially appropriate for virtual environments, where precise and fast manipulation of objects is required. Indeed, a coherent collision status (collision time and contact type), particularly useful to develop easily realistic interaction methods, is computed very efficiently.

6.1 Future work

The approach described in [8] opened many research directions, as it was both a general framework and a specific method (since two specific functions a and b were introduced). Many topics could benefit of further attention.

Thus, one could examine other arbitrary in-between motions (other functions a and b), more general, in order to avoid the relative-motion problem, or to achieve efficient collision detection for more general objects.

No effort was made to make this approach handle numerous models efficiently (*ie* the n-body problem). We believe, however, that existing methods[1, 10] could be easily adapted to produce a general continuous CD algorithm

Some optimizations given in section 4 seem to be extendible. The label method, for example, could benefit from more complex labelling rules (for internal nodes). Moreover, the poor assumptions needed to use this technique should make it usable for other algorithmic applications.

For thin objects, a sphere has a bad ratio $\frac{\text{bounding volume}}{\text{bounded volume}}$. Other bounding volumes could thus be profitably adapted or designed. The difficult point, of course, is to choose bounding volumes allowing fast CD tests between them.

For now, the temporal coherency in the scene hasn't been used. However, the primary goal of this method being precise manipulation, with strong frame-to-frame coherency, it should be useful to develop specific techniques exploiting it.

References

[1] J. Cohen, M. Lin, D. Manocha, M. Ponamgi. *I-Collide: an interactive and exact collision detection system for large-scale environments.* In Proceedings of ACM Interactive 3D Graphics Conference, pages 189-196, 1995.

[2] S. Gottschalk, M. Lin, D. Manocha. *OBB-tree: a hierarchical structure for rapid interference detection.* In Proceedings of ACM SIGGRAPH 1996, pp 171-180, 1996.

[3] P. M. Hubbard. *Collision detection for interactive graphics applications.* Ph.D. Thesis, April 1995.

[4] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, K. Zikan. *Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs.* IEEE Transactions on Visualization and Computer Graphics, March 1998, Volume 4, Number 1.

[5] M. Lin, S. Gottschalk. *Collision detection between geometric models.* In Proceedings of IMA Conference on Mathematics and Surfaces, 1998.

[6] W. D. MacMillan. *Dynamics of rigid bodies.* Dover, New-York, 1960.

[7] S. Quinlan. *Efficient distance computation between non-convex objects.* In Proceedings of International Conference on Robotics and Automation, pp 3324-3329, 1994.

[8] S. Redon, A. Kheddar, S. Coquillart. *An algebraic solution to the problem of collision detection for rigid polyhedral objects.* In Proceedings of International Conference on Robotics and Automation, April 2000.

[9] Diego C. Ruspini, Krasimir Kolarov and Oussama Khatib. *The haptic display of complex graphical environments.* In Computer Graphics (Proc. SIGGRAPH), pages 345-352. ACM, July 1997.

[10] A. Wilson, E. Larsen, D. Manocha and M. Lin. *IMMPACT: Partitioning and Handling Massive Models for Interactive Collision Detection* Appeared in Computer Graphics Forum, September 1999.

[11] S. Krishnan, A. Pattekar, M. Lin and D. Manocha. *Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries.* Appeared in Proceedings of WAFR'98.

[12] CONTACT demo mpeg file. *www-rocq.inria.fr/i3d/CONTACTdemo.mpg*