EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (2002), Number 3

Fast Continuous Collision Detection between Rigid Bodies

Stéphane Redon*, Abderrahmane Kheddar† and Sabine Coquillart*

*i3D - INRIA - France, [stephane.redon,sabine.coquillart]@inria.fr
†CEMIF-SC - Université d'Evry - France, kheddar@iup.univ-evry.fr

Abstract

This paper introduces a fast continuous collision detection technique for polyhedral rigid bodies. As opposed to most collision detection techniques, the computation of the first contact time between two objects is inherently part of the algorithm. The method can thus robustly prevent objects interpenetrations or collisions misses, even when objects are thin or have large velocities. The method is valid for general objects (polygon soups), handles multiple moving objects and acyclic articulated bodies, and is efficient in low and high coherency situations. Moreover, the method can be used to speed up existent continuous collision detection methods for parametric or implicit rigid surfaces. The collision detection algorithms have been successfully coupled to a real-time dynamics simulator. Various experiments are conducted that show the method's ability to produce high-quality interaction (precise objects positioning for example) between models up to tens of thousands of triangles, which couldn't have been performed with previous continuous methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation - Virtual Reality

1. Introduction

Collision detection (CD) is still a fundamental problem in numerous domains. Typical examples are computer graphics (physically-based modeling, animation), robotics (motion planning, collision avoidance), industrial applications (virtual prototyping, assembly tests) and video games. Moreover, haptics research has created the need of algorithms able to achieve kilohertz rates. Collision detection methods are usually split into two categories:

Discrete methods Most previous collision detection methods are *discrete*: they sample the objects motions and detect objects *interpenetrations* (see for example [1, 2, 6, 10, 11, 12, 13, 17, 25, 29]). As a result, these methods may miss collisions (tunneling effect). While an adaptative time-step and predictive methods can be used to correct this problem in offline applications, this may not be suitable in interactive applications when a relatively high and constant frame-rate is required. Moreover, discrete collision detection requires backtracking methods to compute the first contact time, which is necessary in constraint-based analytical dynamics simulations. Depending on the

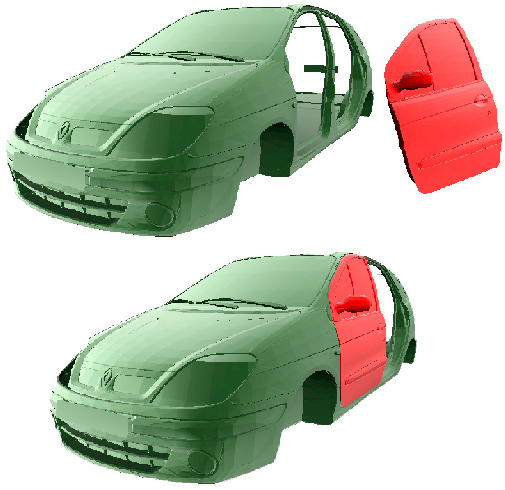

Figure 1. *Precise car door positioning*. The continuous collision detection technique described in this paper allows to precisely (without any objects interpenetration) and interactively position the door. The car skeleton is about 29000 triangles. The door is about 16000 triangles (3d models ©Renault).

object complexity, however, the computational cost of backtracking may be unpredictably large, mainly because estimating the penetration depth is a difficult problem, for example when many triangles have penetrated or if the object is concave or non-connex. Such typical problems are encountered in the interaction technique described in Snyder[27]. In haptics, the penetration problem is a major cause of instability[15].

Continuous methods As opposed to these methods, *continuous* methods compute the first time of contact *during* the collision detection. This computation is inherently part of the algorithm. While more suitable to robust interactive dynamics simulations (to guarantee collision-free motions), continuous methods are usually slower than discrete methods, and are often abandonned for discrete ones[27].

This paper contributes to the field by introducing a fast *continuous* method, able to compute collision times between rigid polyhedral objects composed of tens of thousands of triangles at interactive rates (Figure 1). To our knowledge (see Section 2), this is not possible for previous continuous methods. The collision detection algorithms require no particular topology (objects can be polygon soups), can handle multiple moving objects and are efficient for slow and fast objects. Moreover, the method handles acyclic articulated bodies and can be used to speed up previous continuous collision detection methods for parametric or implicit rigid surfaces.

The algorithms described in this paper rely upon the effective integration of interval arithmetic (IA) and hierarchies of oriented bounding boxes (OBBs). Both approaches benefit from each other. Interval arithmetic is used to robustly compute collision times between objects features (vertices, edges and faces), and to derive a conservative *continuous* overlap test between moving OBBs from a well-known discrete overlap test. Conversely, the bounding boxes help to cull many irrelevant elementary tests (edge/edge, vertex/face and face/vertex tests), which made previous interval methods unpractical for complex polyhedral objects.

The next section describes previous work on *continuous* methods. Section 3 provides an overview of the method. It details the fixed in-between rigid motion assumption, recalls basic principles of interval arithmetic and of bounding volume hierarchies. Section 4 introduces the continuous overlap test between moving OBBs and presents the elementary CD tests (edge/edge, vertex/face and face/vertex). Section 5 briefly explains how our method may be used to speed up CD methods for parametric or implicit rigid surfaces. Section 6 describes several optimizations. The algorithms described in this paper have been coupled to dynamics algorithms[24]. Section 7 presents various experiments conducted to test the interactive simulator. Finally, Section 8 concludes and gives future research directions.

2. Previous work

There are relatively few *continuous* collision detection (CD) techniques. Canny [5] uses a parameterization of the objects trajectories based on quaternions and computes the collision time by solving low-order polynomials, but the algorithm's high complexity doesn't allow real-time interaction between large models. Redon *et al.*[22] use screwings to parameterize the trajectories of rigid polyhedral bodies and obtain a motion similar to the one in Canny [5]. The approach is extended to hierarchies of bounding spheres in Redon *et al.*[23] to continuously detect collisions between rigid polyhedral bodies in real-time. However, since spheres don't fit objects well, the approach can only handle moderately complex objects (a few thousands of triangles). Moreover, the algorithm handles only one mobile object. It is noted in [23], though, that using arbitrary in-between rigid motions over successions of small time intervals allows realistic real-time dynamics simulations.

Cameron [4] introduces spatio-temporal extrusion to determine contact between moving CSG objects. However, due to the high computational cost of the extrusion operation, the object motions are piecewise translational.

Von Herzen *et al.*[30] use Lipschitz bounds and binary subdivision to find the first contact time between time-dependent parametric surfaces. Hierarchies of bounding spheres and axis-aligned bounding boxes are computed during collision detection to speed up the method. Duff [8] uses interval arithmetic and binary subdivision to detect collisions between boolean combinations of implicit surfaces. Snyder *et al.*[28] use interval arithmetic and interval Newton methods to significantly speed up these approaches, and use a simple culling test based on bounding spheres to adress the *n-body* problem. While the algorithm has many interesting features and is pratical for computer graphics animations, it is not efficient enough to handle real-time interaction, even for rigid bodies[27]. General polyhedral objects may be considered as unions of parametric surfaces. However, none of interval-based methods able to handle parametric surfaces ([30, 28]) use hierarchies of bounding volumes to speed up the detection between *unions* of objects. Consequently, detecting a collision between two complex polyhedral objects is turned into an artificial and *unpractical n-body* problem (for example, see the multiple elements algorithm in Snyder *et al.*[28]).

Mirtich[21] uses physical laws to bound the times of impact, and is able to perform collision checks only when necessary. However, the lower bounds may be difficult to obtain for multibodies or complex motions, and the resultingly varying timestep may not be suited for interactive applications.

Note that some methods perform discrete CD on bounding volume hierarchies and continuous CD between polyhedral primitives[19]. Some other methods perform pseudo-continuous CD by bounding at runtime the initial and final object's (or bounding-volume's) positions[9]. However,

these methods don't ensure that the *whole* object trajectory is bounded and thus may miss collisions. The only case when bounding the initial and final object positions is valid is the much simpler one of a single moving vertex. In this case, some methods achieve continuous collision detection at haptic rates[14].

3. Overview

3.1. Arbitrary in-between rigid motions

As in Redon *et al.*[23], this paper uses *arbitrary in-between rigid motions*. Successive objects' positions, determined at fixed instants by the dynamics simulator or by the user interface, are interpolated with an arbitrarily fixed rigid motion. *Arbitrary* meaning that the in-between motion must be continuous and rigid, in order to get a truly continuous collision detection method. Precisely, interpolations are performed between the objects' positions at time t_i and the *intentional* objects' positions at time t_{i+1}. Typically, $t_0, t_1, \ldots$ correspond to the display times. Thus, the interpolations occur between frames. Provided that the simulation timestep is small, the difference between the actual objects' motions and the interpolated ones is negligeable. Note that this approximation principle is similar to the one used in dynamics simulations: the dynamics equations are discretized and low-order approximations are used to move the objects between the instants chosen by the simulator.

The in-between rigid motion used in this paper, though, differs from the one in[23]. The arbitrary in-between motion used in[23] is designed to obtain an algebraic collision test, for which there is a closed-form contact time, and thus is non-natural in dynamics simulations. This isn't required in the present paper since interval arithmetic is used to find roots of functions. Consequently, the in-between rigid motion we use is simply a continuous screwing, with constant rotational and translational velocities.

For clarity, let us assume that the current time interval is $[0,1]$ (between two frames for example). The screwing-based in-between motion in a reference frame $\mathcal{R}_0$ is a 4×4 homogeneous matrix:

$$\mathbf{S}(t) = \mathbf{P}^{-1}\mathbf{V}(t)\mathbf{P} \qquad t \in [0,1] \tag{1}$$

where $\mathbf{V}(t)$ is a z-axis screwing, and $\mathbf{P}$ is the transformation matrix from $\mathcal{R}_0$ to the screwing local frame. If ω and s are respectively the total amount of rotation and translation during the current time interval, then:

$$\mathbf{V}(t) = \begin{pmatrix} \cos(\omega.t) & -\sin(\omega.t) & 0 & 0 \\ \sin(\omega.t) & \cos(\omega.t) & 0 & 0 \\ 0 & 0 & 1 & s.t \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

for $t \in [0,1]$. For acyclic articuled bodies, these screwing-based motions can be composed, starting from the root of the graph describing the body.

Note that other in-between motions can be arbitrarily chosen, provided they are continuous and rigid. For example, one solution consists in linearly interpolating the six independent parameters describing the object's position and orientation between successive frames.

3.2. Interval arithmetic

A good introduction to interval arithmetic for computer graphics can be found in Snyder[26]. The use of interval arithmetic to detect collisions between parametric or implicit surfaces is explored in Von Herzen *et al.*, Duff[8], and Snyder *et al.*[28].

Briefly, interval arithmetic consists in computing with intervals instead of numbers. The definition of a real interval $[a,b]$ is:

$$I = [a,b] = \{x \in \mathbb{R},\ a \leqslant x \leqslant b\}$$

This can be generalized to vector-valued intervals:

$$\begin{aligned} I_n &= [a_1,b_1] \times ... \times [a_1,b_1] \\ &= \left\{\mathbf{x} = (x_1,...,x_n) \in \mathbb{R}^n,\ a_i \leqslant x_i \leqslant b_i \quad \forall i,\ \ 1 \leqslant i \leqslant n\right\} \end{aligned}$$

The set of intervals of real numbers is denoted by $\mathbb{IR}$, while the set of vector-valued intervals is denoted by $\mathbb{IR}^n$. Elementary operations on real numbers can be transposed to intervals:

$$\begin{aligned} [a,b] + [c,d] &= [a+c, b+d] \\ [a,b] - [c,d] &= [a-c, b-d] \\ [a,b] \times [c,d] &= [min(ac,ad,bc,bd), max(ac,ad,bc,bd)] \\ 1/[a,b] &= [1/b, 1/a] \quad \text{if } a>0 \text{ or } b<0 \\ [a,b]/[c,d] &= [a,b] \times (1/[c,d]) \end{aligned} \tag{3}$$

For vector-valued intervals in $\mathbb{IR}^n$, the operations are performed for each coordinate.

In the algorithms described in this paper, intervals are used to bound function ranges on intervals. Precisely, for any given function $f : \mathbb{R} \to \mathbb{R}$, an *inclusion function* $\tilde{f} : \mathbb{IR} \to \mathbb{IR}$ is associated to f, such as:

$$x \in I \Rightarrow f(x) \in \tilde{f}(I) \tag{4}$$

for any interval I. The ideal inclusion functions are those which exactly bound the function range, whatever the interval I.

From equations (1) and (2), the only inclusion functions required are those of the sine and cosine functions. Moreover, in the continuous overlap test derived in Section 4, one more inclusion function is required for the *abs* function ($abs : x \to |x|$). For these functions, inclusion functions are easily computed[26]. Then, appropriate elementary operations (equation (3)) are recursively applied to compute inclusion

functions for the coordinates of a vertex or a vector. The coordinates of a mobile vertex in frame $\mathcal{R}_0$ are:

$$\mathbf{x}_G(t) = \mathbf{P}^{-1}\mathbf{V}(t)\mathbf{P}\mathbf{P}_o\mathbf{x}_o \tag{5}$$

where $\mathbf{x}_o$ are the vertex coordinates in the object frame, and $\mathbf{P}_o$ is the transformation matrix from the local object frame to the reference frame $\mathcal{R}_0$. Consequently, the coordinates inclusion functions are:

$$\tilde{\mathbf{x}}_G(t) = \mathbf{P}^{-1}\tilde{\mathbf{V}}(t)\mathbf{P}\mathbf{P}_o\mathbf{x}_o \tag{6}$$

where operations are performed on intervals, according to equation (3).

The interest of inclusion functions is that they provide a simple way to robustly compute the roots of a function $f : I \rightarrow \mathbb{R}$. The simplest of these methods is the recursive binary subdivision method. Let's assume that an inclusion function $\tilde{f}$ is available. Then $\tilde{f}(I) = [a,b]$ is computed. If $a > 0$ or $b < 0$, then there can't be any root in I. Otherwise, there *may* be a root (may only since $\tilde{f}(I)$ may not fit exactly f's range over I, and/or f may not be continuous). In this case, interval I is cut into two equal intervals $[a,m]$ and $[m,b]$, where $m = 1/2(a+b)$, and the computations are now performed on smaller intervals. This process is recursively performed until an interval width is smaller than a pre-determined threshold (the user-defined precision of the collision detection, see Section 4). In this case, the algorithm declares that a root has been found. It can be shown that if f is continuous on interval I, then the algorithm can't miss any root.

Note, however, that if $f(x) = 0$ over a non-empty interval, then the method returns only a finite number of intervals containing all the roots. The number of intervals returned depend on the pre-determined threshold. In Snyder *et al.*[28], multidimensional interval-based root-finding is performed to detect collision between time-varying parametric or implicit surfaces.

3.3. Bounding volume hierarchies

Using bounding volume hierarchies (BVH) is a common strategy in collision detection and other domains (ray-tracing for example), which is naturally related to interval-based root-finding. Briefly, overlap tests between bounding volumes are used to cull many irrelevant elementary tests between objects parts. Let's assume, for example, that each of the two objects currently processed by the CD algorithm is bounded by a sphere. If the spheres don't overlap, then there can't be any collision between the objects. If the spheres *do* overlap, however, then there *may* be a collision between the objects (may only since the spheres probably don't exactly fit the objects). In this case, the spheres are replaced by unions of smaller spheres and overlap tests between spheres are recursively performed. When spheres sizes are smaller than a pre-determined threshold, exact tests are performed between the object geometries (for example, triangle/triangle collision tests, in the case of triangle soups). For rigid objects, the hierarchies of bounding volumes are usually computed offline. Typical bounding volumes are spheres[16], axis-aligned bounding boxes, oriented bounding boxes[12, 13], *k*-dops[17] and spherical shells[18].

We choose to use OBB hierarchies because of their performance in a wide range of applications, especially close-proximity situations. The hierarchies are binary trees built in a classical way[12, 13]. We noticed that the tightest hierarchies were usually obtained with the *min-max* method[12].

4. Continuous collision detection

This section describes the integrated collision detection algorithms: a well-known discrete overlap test between OBBs is extended to the continuous case, and the continuous CD functions for polyhedral primitives (vertices, edges and faces) are detailed. The precision problem, which is naturally handled by interval arithmetic, is also adressed.

4.1. Collision detection between OBBs

4.1.1. Discrete overlap test

The most efficient discrete overlap test between two static OBBs is probably the one described by Gottschalk *et al.*[13], which relies upon the *separating axis theorem*.

Let us assume that the first OBB is described by three axes $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, a center $\mathbf{T}_A$, and its half-sizes along its axes a_1, a_2 and a_3. In the same way, the second OBB is described by its axes $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$, its center $\mathbf{T}_B$, and its half-sizes along its axes b_1, b_2 and b_3. The separating axis theorem states that two static OBBs overlap if and only if all of fifteen separating axis tests fail. A separating test is simple: the axis $\mathbf{a}$ separates the OBBs if and only if

$$|\mathbf{a}\cdot\mathbf{T}_A\mathbf{T}_B| > \sum_{i=1}^{3} a_i|\mathbf{a}\cdot\mathbf{e}_i| + \sum_{i=1}^{3} b_i|\mathbf{a}\cdot\mathbf{f}_i| \tag{7}$$

The fifteen sufficient axes are deduced from the OBBs axes:

$$\mathbf{a} \in \{\mathbf{e}_i,\ \mathbf{f}_j,\ \mathbf{e}_i \times \mathbf{f}_j,\ 1 \leqslant i \leqslant 3,\ 1 \leqslant j \leqslant 3\} \tag{8}$$

4.1.2. Continuous overlap test

Since we want to continuously detect collisions, we must determine whether two moving OBBs overlap *during* a time interval $[t_0, t_1]$, and not only at the initial or final position. An important point is that we only need a *conservative* overlap test. While an overlap occuring between two OBBs *must* be detected, it is not fundamentally a problem to declare that an overlap has occured when it hasn't. When the OBB hierarchies traversal terminates, no collision can have been missed. Note that conservative tests are used in other collision detection methods, for example *k*-dops[17]. While the discrete OBB test is an exact test, the continuous test introduced in this section is a conservative one.

The continuous overlap test is constituted of two steps:

1. Perform a continuous version of the fifteen separating axis tests.
2. If the OBBs are found to overlap for the current time interval, perform a *subdivision test*, to determine whether the current time interval should be subdivided.

Continuous separating axis tests The first step is easily derived from the discrete overlap test and interval aritmetic. Both sides of inequality (7) are continuous functions of time depending on OBBs time-dependent positions. Both functions can be bounded using interval arithmetic, as described in Section 3. Now let $[l_1, l_2]$ denote a bound on the left side of inequality (7), and let $[r_1, r_2]$ denote a bound on the right side of the same inequality. If $l_1 > r_2$, then the axis **a** is separating the OBBs over the whole time interval. The first step thus consists in performing fifteen such continuous separating axis tests.

This first step, however, can only detect an axis which separates two OBBs *during the whole interval*. Yet, the OBBs may not overlap during a time interval, and be separated by different axes during the motion. This can't be detected by the first step of the continuous test: the fifteen continuous separating axis tests fail, and the OBBs are found to overlap. In interval-based root-finding methods, this problem is solved by subdividing the interval, or by using more sophisticated methods to reduce the interval width, like Newton interval methods[26, 28]. In order to avoid automatic subdivisions or computationally intensive methods, we propose as a second step a simple *subdivision test*. This test is an heuristic similar in spirit to Newton interval methods, since it depends on the OBBs velocities, but is far cheaper to compute.

Subdivision test Generally, an axis separating the OBBs at a given instant won't be separating during the whole time interval when objects move too fast *relatively to their sizes*. This is exactly what the subdivision test adresses: briefly, the OBBs are projected on their relative velocity directions. First, the relative velocity of the OBBs centers at the beginning of the time interval are computed. This amounts to consider that the second OBB is static. Assuming $\mathbf{T}_A$ is expressed in the reference frame $\mathcal{R}_0$, then its velocity can be derived from equation (5):

$$\mathbf{v}(\mathbf{T}_A) = \mathbf{P}_A^{-1}\mathbf{V}'_A(t_0)\mathbf{P}_A\mathbf{T}_A$$

where t_0 is the lower bound of the current time interval, $\mathbf{P}_A$ and $\mathbf{V}_A(t_0)$ describe the screwing associated to the first object. $\mathbf{V}'_A(t_0)$ is computed simply from equation (2) by differentiating the matrix elements with respect to time. Similar relations hold for the second OBB. Let $\mathbf{v}_r = \mathbf{v}(\mathbf{T}_A) - \mathbf{v}(\mathbf{T}_B)$ denote the OBBs centers relative velocity. Then $(t_1 - t_0)|\mathbf{v}_r|$ is approximately the length of the relative path followed by the OBBs centers during the time interval. Thus, the time interval $[t_0, t_1]$ is subdivided if and only if:

$$\sum_{i=1}^{3} a_i|\mathbf{v}_r \cdot \mathbf{e}_i(t_0)| + (t_1 - t_0)|\mathbf{v}_r| > k.(\sum_{i=1}^{3} b_i\mathbf{v}_r \cdot \mathbf{f}_i(t_0)|)$$

where k is a pre-determined constant. If the time interval isn't subdivided, then the OBBs are declared to have overlapped. Our experiments indicate that most false hit results can be culled when $k = 0.2$.

4.2. Collision detection between primitives

For polyhedral objects, continuous collision detection is somewhat simpler, though more time-consuming, than discrete collision detection since all contact configurations imply at least one of the three non-degenerate contact types: vertex/face, face/vertex, and edge/edge. Moreover, each CD test can be formulated so that the first contact time is a root of a time-dependant function[5, 22, 23].

For the edge/edge case, a collision is first detected between *the lines* containing the edges. If $\mathbf{a}(t)\mathbf{b}(t)$ denotes the first edge and $\mathbf{c}(t)\mathbf{d}(t)$ denotes the second edge, then a collision occurs when:

$$\mathbf{a}(t)\mathbf{c}(t) \cdot (\mathbf{a}(t)\mathbf{b}(t) \times \mathbf{c}(t)\mathbf{d}(t)) = 0 \qquad (9)$$

The solutions of this equation are computed thanks to the interval-based root-finding method described in Section 3. A solution in $[0,1]$ is kept if and only if the corresponding contact point belongs to the *edges*.

For the vertex/face and face/vertex cases, a collision is first detected between the vertex and the plane containing the face. If $\mathbf{a}(t)$ denotes the vertex and $\mathbf{b}(t)\mathbf{c}(t)\mathbf{d}(t)$ denotes the triangle, then a collision occurs when:

$$\mathbf{a}(t)\mathbf{b}(t) \cdot (\mathbf{b}(t)\mathbf{c}(t) \times \mathbf{b}(t)\mathbf{d}(t)) = 0 \qquad (10)$$

In this case, too, solutions are computed using an interval-based root-finding method. A solution t_r in $[0,1]$ is kept if and only if $\mathbf{a}(t_r)$ is inside the triangle $\mathbf{b}(t_r)\mathbf{c}(t_r)\mathbf{d}(t_r)$.

Note that during the recursive root-finding, when a time interval $[t_l, t_r]$ is subdivided into two intervals $[t_l, t_m]$ and $[t_m, t_r]$, solutions are first recursively looked for in $[t_l, t_m]$. If *and only if* no valid solution is found in this interval, then the second interval $[t_m, t_r]$ is recursively examined. This allows the process to be stopped as soon as the first contact time has been found.

4.3. Collision detection precision

The subdivision method used in the primitive-primitive tests depends on a threshold, which defines the maximal width of an interval that can be subdivided, and thus determines the collision detection precision. The threshold is computed at runtime and depends on the location of the object features.

If dt denotes a small time interval, then the length dl of the helical path followed by an object vertex $\mathbf{p}$ is:

$$\begin{aligned} dl &= |\mathbf{p}'(t)|dt \\ &= \sqrt{s^2+\omega^2(x^2+y^2)}dt \end{aligned}$$

where s and ω are the associated screwing parameters and x and y are the first two coordinates of $\mathbf{p}$ in the screwing local frame. Let ε_0 denote the maximum error allowed on $\mathbf{p}$'s position when the subdivision method completes. Then $dl < \varepsilon_0$ must hold for the last time interval returned. To enforce this, a valid time interval (one which may contain a root) must be subdivided as long as

$$dt > \frac{\varepsilon_0}{\sqrt{s^2+\omega^2(x^2+y^2)}} \tag{11}$$

Since the contact position can't be known in advance, a similar bound is computed for each of the four vertices **a**, **b**, **c** and **d** of the current elementary CD test. The interval width threshold is then defined as the lowest of these four bounds. Since the objects primitives (vertices, edges and faces) are convex, this ensures that the error allowed on the contact position will be enforced.

5. Extension to parametric or implicit surfaces

Using interval analysis to detect collisions between parametric or implicit surfaces amounts to compute *at runtime* axis-aligned bounding-boxes on the objects. For example, let

$$\mathcal{P} : \left\{ \begin{array}{c} \mathbb{R}^3 \rightarrow \mathbb{R}^3 \\ \mathbf{x} \rightarrow \mathbf{y} = \mathcal{P}(\mathbf{x}) \end{array} \right. \tag{12}$$

denote a parametric surface, and let X denote an interval in $\mathbb{IR}^3$. Then $Y := \tilde{\mathcal{P}}(X)$ is an interval in $\mathbb{IR}^3$. From the inclusion function definition, Y is an axis-aligned box bounding the object part described by X. However, as noted in Snyder *et al.*[28], evaluating Y may be highly time-consuming for complex objects, since the structure describing the object (or its inclusion function) has to be traversed.

For rigid parametric or implicit surfaces, a hierarchy of oriented bounding boxes can be precomputed and our continuous OBB/OBB overlap test may be used to quickly cull irrelevant tests, whatever the underlying objects complexity. The actual objects geometry is then tested using Snyder *et al.*'s algorithms only when two leaf-nodes collide.

6. Optimizations

This section describes several optimizations added to the system.

Discrete tests In order to avoid unnecessary continuous OBB/OBB overlap tests, a discrete overlap test is performed for the OBBs initial and final positions before each continuous one. If these discrete tests find that the static OBBs overlap, then the continuous test isn't required and thus isn't performed. Since our implementation of the continuous overlap test is approximately five times slower than the one of the discrete overlap test, this results in a significant speedup in high coherency situations. However, it is unuseful in general configurations.

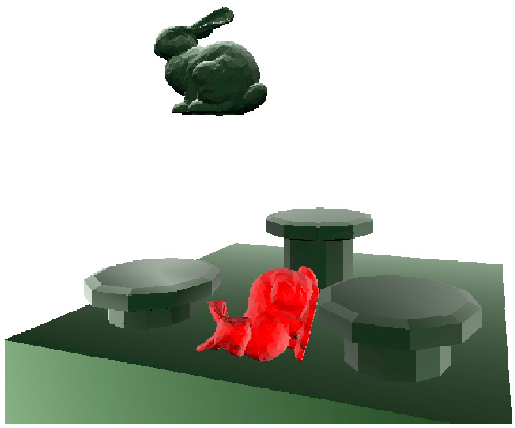

Figure 2. *First test application: playing with bunnies*. The user can test typical interaction situations (object positioning, slow and fast motions resulting in collisions).

Passing the current collision time As in Redon *et al.*[23], when two objects are processed by the CD functions, a *current collision time* (CCT) is maintained and passed to the CD functions. For example, when a collision between two edges has been detected at $t_c \in [0,1]$, then the next tests (for the pair of objects currently processed) examine only the smaller time interval $[0,t_c]$.

Directional traversal When descending an OBB, the child which center is the closest to the center of the OBB belonging to the other object is processed first. Combined to the previous optimization, this allows to rapidly find a preliminary CCT when two fast objects collide, and thus help avoid a quadratic growth of the number of OBB/OBB tests, as would happen if one object passed completely through the other.

Partial tests Van den Bergen[29] notices that an axis separating two OBBs is generally an OBB axis, and removes the other nine axes. This results in a moderate speedup in our system. We observed that these nine axes are often useful to separate rotating objects.

Coherence tables We have implemented the coherence tables method introduced in Gottschalk[12]. Briefly, the method consists in storing the terminal nodes of the bounding volumes test tree. When detecting a collision, the bounding volume hierarchies are tested beginning from the terminal nodes of the previous frame. This may be highly memory-consuming, especially for large objects. Moreover, because of the use of the preliminary discrete test, this results in a low speedup (15%), which roughly corresponds to the theoretical limit given by Gottschalk, since

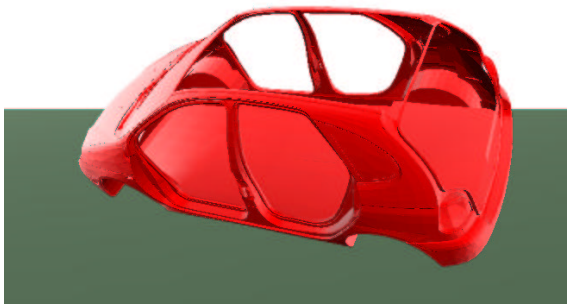

Figure 3. *Second test application: a car skeleton bounces on a large cubic floor*. The car is manipulated by the user.

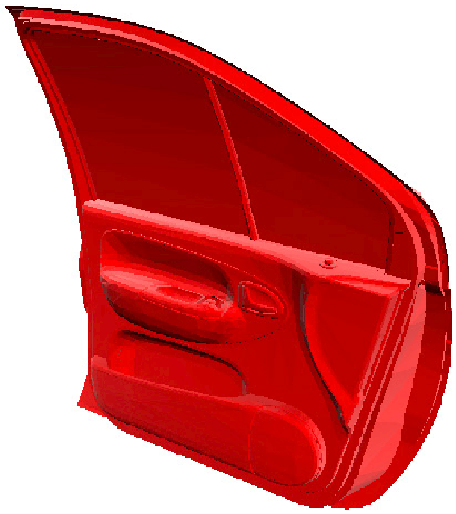

Figure 4. *A highly-detailed door (16000 triangles)*. This door is used in the third test application.

a continuous test is approximately five times slower than a discrete one.

Surprisingly, a usual 'trick' didn't speed up the computations. Pre-computed values of required trigonometric functions (equation (2)) were stored in look-up tables. However, this gave no significant result. We conjecture that the compiler already uses such a strategy.

Some low-level optimizations have yet to be tried. For example, many processors offer local parallelism through SIMD instructions (single instructions, multiple data). Lin *et al.* report that a SIMD version of the discrete OBB/OBB overlap test allows a factor 2-3 speedup[20]. Our continuous version should probably benefit from a SIMD implementation. Since profiling reveals that roughly half of the total time is spent during overlap tests between OBBs, this could speed up the whole interactive dynamics simulation significantly. For now, one continuous overlap test is about a few microseconds on a 1 GHz Pentium PC.

7. Results

A portable collision detection library based upon the algorithms described in this paper has been implemented in C++ and has been successfully tested on Windows and Unix systems. The library has been coupled to analytical (constraint-based) dynamics algorithms[24] which take advantage of the precise contact information (first contact instant, contact position and contact normal) sent by the CD functions. The coupling is performed in a classical way[3]. The resulting simulator, running on a 1GHz Pentium PC with 256 Megabytes of memory and the Windows 2000 operating system, is able to perform interactive simulations with models up to tens of thousands of triangles. While it is generally difficult to compare collision detection methods (mainly because CD librairies are not public domain, and because it is difficult to estimate librairies performance on nowadays faster processors from old data), we believe that this couldn't be achieved with previous continuous methods, essentially for the reasons exposed in Section 2.

Several interactive sessions have been specifically designed to test the simulator. No 3D peripherals have been used for interaction. The interface is a 2D mouse, which is sometimes tedious for precise tasks. For sure, the use of a spaceball and stereo glasses would greatly increase the quality of interaction. During an interactive session, the user is able to navigate the scene and can check the validity of collision detection. Moreover, he or she can choose at runtime between a first-order or a second-order simulation. Depending on the world order, the mouse commands the object's velocities or accelerations.

The first application involves 3D models of a few thousand triangles (see Figure 2). It has been designed to test several typical interaction situations (object positioning, slow and fast motions resulting in collisions). This application allows the simulator to be tested for correctness and robustness. As predicted, using arbitrary in-between motions on small time intervals isn't conflicting with realism requirements (according to the user's eye).

Two other applications have been designed to test the approach's scalability. The first one involves a skeleton of a "Renault Scénic" car model (29000 triangles). The car skeleton, manipulated by the user, bounces on a large cubic floor (Figure 3). In the second application, the user must position a car door (16000 triangles, Figure 4) in the car skeleton used in the bouncing test application. Despite unadapted interaction peripherals, the car door can be positioned precisely and interactively with no difficulty (Figures 1 and 5). Continuous

Figure 5. *Third test application: the door being interactively positioned in the car skeleton*. Continuous collision detection and constraint-based simulation allows the user to precisely and intuitively position the door. No interpenetration ever occurs.

collision detection, which allows us to achieve very precise object/object interaction, seems essential to the success of this kind of task.

8. Conclusion and future work

This paper has described a fast continuous collision detection technique which relies upon the effective integration of arbitrary in-between rigid motions, interval arithmetic and OBB hierarchies. An efficient continuous overlap test between two OBBs has been derived from a well-known discrete test. The continuous test consists of two steps. The first step is an interval-based version of the separating tests. The second step is a subdivision test which heuristically determines whether the current time interval should be subdivided when the OBBs are found to overlap. Various optimizations have been described and experimented with. The collision detection algorithms have been coupled to an analytical dynamics simulator which takes advantage of the contact information (first contact time, contact position and contact normal). This interactive dynamics simulator has allowed us to demonstrate the resulting high-quality interaction for models composed of tens of thousands of triangles: the interactive simulation is both robust and precise, since no interpenetration ever occurs.

As noted in the introduction, a major cause of instability in haptics comes from uncontrolled object inter-penetration, since in most algorithms the force exerted by the haptic peripheral depends on the amount of penetration. Actually, the amount of penetration isn't required when a virtual coupling method is used[7]. Thus, we plan to explore haptic interaction with the simulator based upon the algorithms described in this paper.

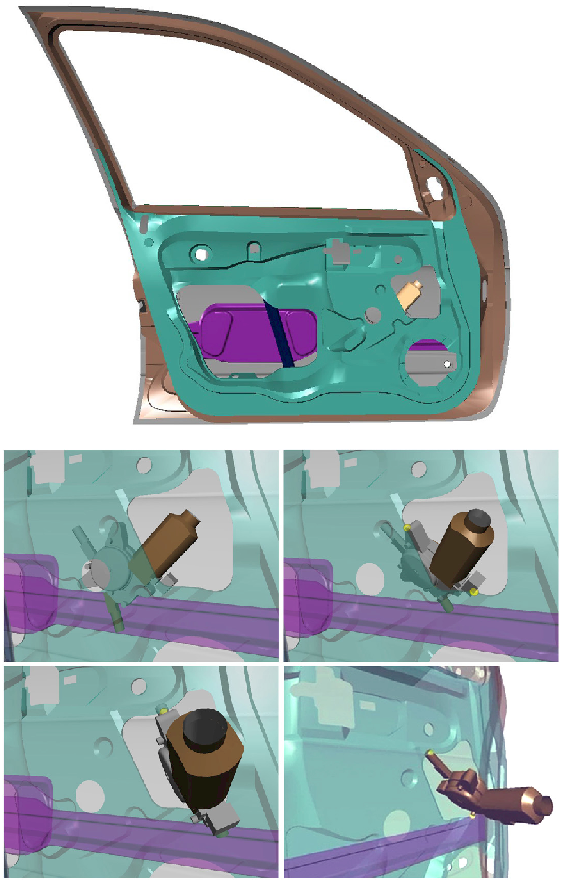

Figure 6. *Engine removal*. The engine is easily removed from the car door thanks to a 2D mouse (20000 triangles) (3d models ©Renault).

Aknowledgements

The authors would like to thank Arnaud Buissé for proofreading the submission version, Tangui Morvan for implementing an inventor parser able to convert the Renault models, and Renault for providing the car models. The model parts are ©Renault. The authors would also like to thank the INRIA DISC Multimedia team for producing the accompanying videos. Also, many thanks go to the anonymous reviewers for interesting and useful remarks. This work has been funded by the French Ministry of Research through an AMX grant and the RNTL PERF-RV project.

References

1. G. Baciu, S. K. Wong, and H. Sun. *RECODE: An Image-Based Collision Detection Algorithm*. Journal of Visualization and Computer Animation, Vol. 10, No. 4, 1999 pp. 181-192. 1

2. D. Baraff. *Curved surfaces and coherence for non-penetrating rigid body simulation*. Computer Graphics, Vol. 24, No. 4, 1990, pp. 19-28. 1

3. D. Baraff. *Interactive simulation of solid rigid bodies.* IEEE Computer Graphics and Applications, Vol. 15, 1995, pp. 63 - 75. 7

4. S. A. Cameron. *Collision detection by four-dimensional intersection testing.* IEEE Trans. Robotics and Automation. 6, 3 (June 1990), pp 291-302. 2

5. J. F. Canny. *Collision detection for moving polyhedra.* IEEE Trans. Patt. Anal. Mach. Intell. 8,2 (March 1986), pp 200-209. 2, 5

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

7. J. Colgate, M. Stanley, and J. Brown. *Issues in the haptic display of tool use*. In Int. Conf. on Intelligent Robots and Systems, (Pittsburgh), August 1995. 8

8. T. Duff. *Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry.* Computer Graphics, 26(2), July 1992, pp. 131-138. 2, 3

9. J. Eckstein and E. Schoemer. *Dynamic collision detection in virtual reality applications.* In 7th International Conference in Central Europe on Computer Graphics and Visualization and Interactive Digital Media, WSCG'99, pp. 71-78. 2

10. A. Garcia-Alonso, N. Serrano and J. Flaquer. *Solving the collision detection problem*. IEEE Computer Graphic and Applications, 13(3), 36-43 (1994). 1

11. E. G. Gilbert, D. W. Johnson and S. S. Keerthi. *A fast procedure for computing the distance between objects in three-dimensional space*. Journal of Robotics and Automation, 4, 193-203 (1988). 1

12. S. Gottschalk. *Collision queries using oriented bounding boxes*. PhD Thesis. 1999. 1, 4, 6

13. S. Gottschalk, M. C. Lin, and D. Manocha. *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*. In SIGGRAPH 96 Conference Proceedings, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1996. 1, 4

14. A. Gregory, M. Lin, S. Gottschalk and R. Taylor. *Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback device.* In Computational Geometry: Theory and Applications. 3

15. A. Gregory, A. Mascarenhas, S. Ehmann, M. Lin and D. Manocha. *Six degree-of-freedom haptic display of polygonal models*.In Proc. IEEE Visualization, 2000. 2

16. P. M. Hubbard. *Collision detection for interactive graphics applications*. Ph.D. Thesis, April 1995. 4

17. 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. 1, 4

18. 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.* 4

19. C. Lennerz, E. Schoemer and T. Warken. *A Framework for Collision Detection and Response.* In 11th European Simulation Symposium, ESS'99, pp. 309-314. 2

20. M. C. Lin, A. Gregory, S. Ehmann, S. Gottschalk, and R. Taylor. *Contact Determination for Real-Time Haptic Interaction in 3D Modeling, Editing and Painting*. Proc. 1999 Workshop for PhanTom User Group. 7

21. B. Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD Thesis. Fall 1996. 2

22. S. Redon, A. Kheddar and S. Coquillart. *An Algebraic Solution to the Problem of Collision Detection for Rigid Polyhedral Objects.* In Proceedings of International Conference on Robotics and Automation, pp 3733-3738, April 2000. 2, 5

23. S. Redon, A. Kheddar and S. Coquillart. *CONTACT: arbitrary in-between motions for continuous collision detection.* In Proceedings of IEEE ROMAN'2001, Sep. 2001. 2, 3, 5, 6

24. S. Redon, A. Kheddar and S. Coquillart. *Gauss' least constraints principle and rigid body simulations.* In proceedings of IEEE International Conference on Robotics and Automation, may 2002 2, 7

25. M. Shinya and M. Forque. *Interference detection through rasterization*. The Journal of Visualization and Computer Animation, 2, 131-134 (1991). 1

26. J. Snyder. *Interval analysis for Computer Graphics*. Computer Graphics, 26(2),pages 121-130, July 1992. 3, 5

27. J. Snyder. *An interactive tool for placing curved surfaces without interpenetration.* In Proceedings of ACM SIGGRAPH, pages 209-218, 1995. 2

28. J. Snyder, A. Woodbury, K. Fleischer, B. Currin, and A. Barr, *Interval Methods for Multi-point Collisions between Time-Dependent Curved Surfaces*. Computer Graphics, 27(2), pp. 321-334, Aug. 1993. 2, 3, 4, 5, 6

29. G. Van den Bergen. *Efficient collision detection of complex deformable models using AABB trees.* Journal of Graphics Tools, 2(4):1-14, 1997. 1, 6

30. Von Herzen, B., A.H. Barr, and H.R. Zatz, *Geometric Collisions for Time-Dependent Parametric Surfaces.* Computer Graphics, 24(4), August 1990, pp. 39-48. 2