Hierarchical back-face culling for collision detection

Stéphane Redon, Abderrahmane Kheddar, Sabine Coquillart

Hierarchical Back-Face Culling for Collision Detection

Stéphane Redon*, Abderrahmane Kheddar†, Sabine Coquillart*

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

Abstract

A few years ago, Vanecek[16] suggested to apply a variant of back-face culling to speed-up collision detection between polyhedral objects. However, Vanecek's method is linear in the number of faces in the object, which is unpractical for large models. This paper suggests to add some geometrical information to hierarchies of bounding volumes, typically used in collision detection, and perform conservative back-face culling at the bounding-volume level in constant time. The method described in this paper can be applied to complement any kind of bounding-volumes hierarchy and allows a trade-off between memory and speed. Preliminary experimental results suggest that the method allows a significant speed-up, especially in close proximity situations.

1 Introduction

Collision detection (CD) is still a fundamental problem in numerous domains. Some typical examples are computer graphics (physically-based modeling, animation), robotics (path-planning, collision avoidance), industrial applications (virtual prototyping, assembly tests) and video games. Especially, haptics research has generated the need for algorithms able to achieve kilohertz rates.

A few years ago, Vanecek[16] suggested to apply a variant of back-face culling (BFC) to speed-up collision detection between polyhedral objects. The method consists in culling faces which are moving backwards, relatively to the other objects. Since it is insensitive to objects positions and orientations, the method is especially efficient in close-proximity configurations (a peg in a hole, for example). To our knowledge, this method is the only one offering such characteristics for general polyhedral objects[1, 2].

[1] For convex objects, however, separating plane algorithms are somewhat insensitive to the objects positions and orientations, even in close-proximity situations. Also, the *incoming constraint* in Snyder *et al.*[15] can be seen as back-face culling applied to collision detection between parametric or implicit surfaces.

[2] While some time-scheduling methods[9] use the objects motions to estimate lower bounds on time of impacts, they don't use the motion *direction*, as in Vanecek[16].

Vanecek's method, however, performs back-face culling at the *face* level. Consequently, the algorithm is linear in the number of faces in the object, which is unpractical for large models. This paper suggests to add some geometrical information to hierarchies of bounding volumes (BVs), typically used in collision detection, and perform conservative back-face culling at the *bounding-volume* level. Precisely, this geometrical information is used to perform a *constant-time* culling test which detects most situations where every triangle associated to a bounding-volume is moving backwards, *ie* situations where the bounding-volume hierarchies need not be further descended, even when the BVs are overlapping.

The method described in this paper can be applied to complement any kind of bounding-volumes hierarchy, and doesn't require a separate hierarchy. It allows a trade-off between the memory overhead due to the addition of geometrical information in the bounding-volumes and the provided speed-up. Moreover, it can be applied to discrete and continuous collision detection.

2 Background

2.1 Back-face culling for collision detection

Let's briefly recall Vanecek's results. More details can be found in the original paper[16]. Let's consider a point $\mathbf{p}$ belonging to a rigid body. Let $\mathbf{r}$ denote the body's center of gravity and let $\dot{\mathbf{r}}$ and ω denote respectively the object's translational and rotational velocities at a given time. Then, the instantaneous velocity of $\mathbf{p}$ is $\dot{\mathbf{p}} = \dot{\mathbf{r}} + \omega \wedge (\mathbf{p} - \mathbf{r})$. Now if i and j denote two objects, then the velocity of $\mathbf{p}$ from object i relatively to object j is:

$$\dot{\mathbf{p}}_{ij} = \dot{\mathbf{p}}_i - \dot{\mathbf{p}}_j = \mathbf{a}_{ij} + \mathbf{p} \wedge (\omega_j - \omega_i) \qquad (1)$$

where $\mathbf{a}_{ij} = \dot{\mathbf{r}}_i - \dot{\mathbf{r}}_j - \omega_i \wedge \mathbf{r}_i + \omega_j \wedge \mathbf{r}_j$. Relative velocities are linearly related. If $\mathbf{p}^1, \ldots \mathbf{p}^t$ denote some

points from object i, then

$$\mathbf{p} = \sum_{s=1}^{t} \alpha_s \mathbf{p}^s \Rightarrow \dot{\mathbf{p}}_{ij} = \sum_{s=1}^{t} \alpha_s \dot{\mathbf{p}}_{ij}^s \qquad (2)$$

for any $(\alpha_1, ..., \alpha_t) \in \mathbb{R}^t$. Now, if $\mathbf{p}$ denotes a point from face m in object i, and $\mathbf{n}_m$ denotes the outward normal to the face, then $\mathbf{p}$ is moving backwards relatively to object j if and only if

$$\dot{\mathbf{p}}_{ij}.\mathbf{n}_m < 0 \qquad (3)$$

Relation (2) and the linear property of the dot product leads to the following *convex property*:

$$(\dot{\mathbf{p}}_{ij}^s.\mathbf{n}_m < 0, \quad 1 \leqslant s \leqslant t) \Rightarrow (\dot{\mathbf{p}}_{ij}.\mathbf{n}_m < 0) \qquad (4)$$

for any point $\mathbf{p}$ in the convex hull of $\mathbf{p}^1,\dots \mathbf{p}^t$. In Vanecek[16], this property is used to cull faces in two steps[3]:

1. Compute extremal vertices' velocities
An enclosing convex polygon is associated to each face. The velocities of the vertices $\mathbf{p}^1,\dots \mathbf{p}^t$ of the enclosing polygon are computed from equation (1).

2. Perform culling tests
For each vertex $\mathbf{p}^s$ of the enclosing convex polygon, a culling test is performed as in equation (3). If every vertex $\mathbf{p}^s$ is culled, then the face is culled.

2.2 Bounding-Volumes Hierarchies

Using bounding-volumes hierarchies (BVH) is a common strategy in collision detection and other domains (rendering for example). 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. 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, bounding-volumes hierarchies are usually computed offline. Some typical examples of BVs are spheres[5], axis-aligned bouding boxes (AABBs), oriented bounding boxes[4] (OBBs), and k-dops[6].

In order to include the backward-motion (BM) culling test in the BVHs traversal algorithm, this paper suggests to extend the BV/BV overlap test in the following way:

1. If one of the two BVs has already been BM-culled (checked through a flag set in step 3), return, else go to 2.

2. Perform the BV/BV overlap test. If the BVs don't overlap, return, else go to 3.

3. Perform the BM culling test for each BV in turn. If one BV is BM-culled, mark it and return.

For clarity, a simple definition will be useful in the remaining of the paper: in the following, the triangles **associated** to any particular node of a bounding-volumes hierarchy are the ones found in its descendent leaf-nodes.

3 Method Overview

In order to get a constant-time backward-motion culling test for a bounding-volume, we adapt both steps from Section 2.1. Precisely, whatever the number k of triangles associated to the bounding volume, the following algorithm is used:

1. Compute characteristic points' velocities
Instead of computing the velocities of the $3k$ vertices defining the k triangles, we compute the velocities of a fixed number of *characteristic points* $\mathbf{c}^1, \dots \mathbf{c}^t$ whose convex hull contains the bounding-volume. For AABBs or OBBs, for example, the corners may by used ($t = 8$). From the linear property (2) of relative velocities, the characteristic points' velocities approximate the triangles velocities. Since the number of characteristic points is fixed for a given bounding-volume type, the velocities computation is performed in constant-time, whatever the number of triangles associated to the bounding-volume.

2. Perform conservative culling tests
From the convex property (4), the characteristic points' velocities can be used to cull a triangle associated to a bounding-volume. However, culling all the triangles would still be linear in k. Thus, we use a bounded number of precomputed *backward-motion vectors* (BM-vectors) $\tilde{\mathbf{n}}_1, \dots \tilde{\mathbf{n}}_r$ which suitably approximate

[3]Note that, actually, the velocities computations and the culling tests are interleaved in order to return earlier when a face is moving forwards. However, all faces still have to be tested.

the set of triangle normals. Precisely, each characteristic point $\mathbf{c}^s$ is tested against each precomputed vector $\tilde{\mathbf{n}}_m$ as in equation (3). The bounding-volume is culled if all of the $t \times r$ tests are successful.

The precomputed vectors are chosen so as to provide a constant-time conservative culling test. First, note that the elementary culling test (3) can be expressed geometrically. Denoting $\mathbf{H}(\mathbf{n}_m) = \{\mathbf{x} \in \mathbb{R}^3,\ \mathbf{x}.\mathbf{n}_m < 0\}$ the open half-space defined by $\mathbf{n}_m$, then $\mathbf{p}$ from i is moving backwards from j if and only if $\dot{\mathbf{p}}_{ij} \in \mathbf{H}(\mathbf{n}_m)$. Using this notation, the constraint on the precomputed BM-vectors can be expressed simply: $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_r$ are chosen such as

$$\mathbf{BM}_v = \bigcap_{m=1}^{r} \mathbf{H}(\tilde{\mathbf{n}}_m) \subset \bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m) \qquad (5)$$

where r is smaller than a user-defined constant r_{max}. $\mathbf{BM}_v$ is the polyhedral *backward-motion cone* (BM-cone) added to the bounding-volume. Thus, the bounding-volume v is BM-culled when

$$\dot{\mathbf{c}}^s_{ij} \in \mathbf{BM}_v,\ 1 \leqslant s \leqslant t \qquad (6)$$

r is the *size* of the BM-cone. Since t and r_{max} are fixed, the bounding-volume culling test is performed in constant time.

Since constraint (5) implies

$$\dot{\mathbf{c}}^s_{ij} \in \bigcap_{m=1}^{r} \mathbf{H}(\tilde{\mathbf{n}}_m) \Rightarrow \dot{\mathbf{c}}^s_{ij} \in \bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m) \qquad (7)$$

for $1 \leqslant s \leqslant t$, all triangles associated to the bounding-volume are *necessarily* moving backwards when equation (6) culls the bounding-volume. Yet, this culling test is only *conservative*: it may fail even though every triangle associated to the bounding-volume is moving backwards.

The reasons for this should be clear. First, triangles velocities are approximated through those of a few characteristic points. And second, triangles normals are approximated through the backward-motion cone. However, as the bounding-volumes hierarchy is being descended, the bounding volumes fit the object geometry better and better. Thus, these approximations are more and more precise and the backward-motion cone allows to cull more and more backward-moving triangles. Figure 1 depicts this phenomenon: as the hierarchy is descended, the BM-culling tests allows to cull more and more bounding volumes (and thus more and more groups of associated triangles).

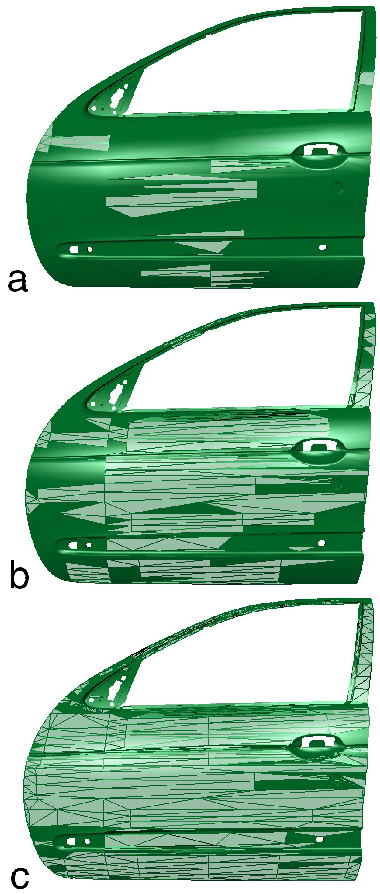

Figure 1: Hierarchical BM-culling of a moving door. *Faces moving forwards are shown filled. The door is translating away from the viewer (snapshots a, b and c are taken at the same time, though). As the hierarchy is descended, the BM-culling tests allows to cull more and more bounding volumes (and thus more and more groups of associated triangles). Levels of the BVH: a=7, b=9 and c=11.*

4 Building enhanced bounding-volumes hierarchies

4.1 Building hierarchies of BVs

Algorithms which compute bounding-volumes hierarchies can generally be split into two categories: top-down and bottom-up approaches. In the case of top-down approaches, a rule is used to split the triangles list for a given node into two sublists. Typical splitting rules include *Min Sum*, *Min Max*, *Splatter* and *Longest Side*[6]. Note that the choice of the splitting rule is generally independant of the choice of the bounding-volume.

In order to take advantage of BM-culling, a natural idea consists in deriving a splitting rule based upon the normals repartition. Thus, denoting $\mathbf{n}_1$, ... $\mathbf{n}_k$ the triangles normals associated to the current bouding-volume, the normals mean $\mu = \frac{1}{k}\sum_{i=1}^{k} \mathbf{n}_i$

is computed, as well as a covariance matrix $\mathbf{C}$:

$$\mathbf{C}_{lm} = \frac{1}{k}\sum_{i=1}^{k}(\mathbf{n}_i[l] - \mu[l])(\mathbf{n}_i[m] - \mu[m]) \quad (8)$$

where $1 \leqslant l, m \leqslant 3$, $\mathbf{n}_i[l]$ and $\mu[l]$ denote vectors coordinates, and $\mathbf{C}_{lm}$ denote the matrix components. This covariance matrix is diagonalized and the triangles are distributed into two sublists according to their normal projection along the greatest variance direction.

However, this idea misses the point of collision detection since the method is unable to compute tight bounding-volumes hierarchies. Instead, we suggest to use a traditional splitting rule (the MinMax rule for example). It turns out that, as the hierarchy is being descended, the local curvature is lower and lower and the normals vary less and less, enabling efficient BM-culling. While some future work include the search of a *mixed* splitting rule, which would use triangle normals and positions[4], traditional splitting rules proved to be sufficient to yield significant speed-ups (see Section 7).

4.2 Adding BM-cones

Adding BM-cones in bounding-volumes is independent of the kind of approach used to build the bounding-volumes hierarchy, and the addition itself can be performed top-down or bottom-up. In the top-down case, the BM-cones are computed from the triangles normals $\mathbf{n}_1, \ldots \mathbf{n}_k$ associated to the bounding-volumes. In the bottom-up case, the BM-cones are recursively computed from those of the bounding-volumes child-nodes according to the following rules:

1. The BM-cones of leaf bounding-volumes are computed from the triangles normals.
2. If (and only if) all child-nodes have a BM-cone, the BM-cones of internal nodes are computed from the BM-vectors in all child-nodes.

Note that using a bottom-up approach still provides conservative tests. If $\mathbf{BM}_{v_1}, \ldots \mathbf{BM}_{v_c}$ denote the BM-cones in the bounding-volume's child-nodes, then

$$\mathbf{BM}_v \subset \bigcap_{i=1}^{c} \mathbf{BM}_{v_i} \quad (9)$$

Whereas the top-down approach yields more precise results (and, thus, bouding-volumes are more often BM-culled), the bottom-up approach is typically much faster and is preferably used for large models.

[4]For example, Kumar *et al.*[7] describe a method to perform hierarchical back-face culling for rendering which computes clusters from both faces normals and positions.

5 Building the backward-motion cone

In this section, we describe an algorithm able to build a bounding-volume's BM-cone from a set of vectors $\mathbf{n}_1, \ldots \mathbf{n}_k$. From the previous section, these vectors are triangles normals or BM-vectors of all child-nodes.

5.1 Preliminaries

We need the following definitions:

Definition (Positive cone). *The positive cone $\mathcal{P}(\mathbf{v}_1, \ldots, \mathbf{v}_k)$ generated by k vectors $\mathbf{v}_1, \ldots \mathbf{v}_k$ in $\mathbb{R}^3$ is the set of positive linear combinations of these k vectors:*

$$\mathcal{P}(\mathbf{v}_1, \ldots, \mathbf{v}_k) = \left\{ \sum_{m=1}^{k} \lambda_m \mathbf{v}_m,\ \lambda_m \geqslant 0,\ 1 \leqslant m \leqslant k \right\}$$

Definition (Minimal subset). *Let $\mathbf{v}_1, \ldots \mathbf{v}_k$ be k vectors in $\mathbb{R}^3$. In this paper, a minimal subset of this vectors is a smallest set of s vectors $\mathbf{v}_{i_1}, \ldots \mathbf{v}_{i_t}$ of these k vectors such as $\mathcal{P}(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_t}) = \mathcal{P}(\mathbf{v}_1, \ldots, \mathbf{v}_k)$.*

In other words, a minimal subset is a smallest subset of vectors which generate the same positive cone. The vectors in a minimal subset are on the convex hull of the positive cone.

The constraint on the backward-motion cone is the inclusion property (5). Consequently, we are interested in building the BM-cone only when $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$ is non-empty[5]. The following lemma indicates when the search for a BM-cone can be avoided:

Lemma 1. *Assume $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k) = \mathbb{R}^3$, then $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$ is empty.*

Proof. Assume there exists $\mathbf{x} \in \mathbb{R}^3$ such as $\mathbf{x}.\mathbf{n}_m < 0, 1 \leqslant m \leqslant k$ (then $\mathbf{x} \neq \mathbf{0}$). Since $\mathbf{x} \in \mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$, there exists k positive values $\lambda_1, \ldots \lambda_k$ such as $\mathbf{x} = \sum_{m=1}^{k} \lambda_m \mathbf{n}_m$. Since $\mathbf{x} \neq \mathbf{0}$, at least one of these values is strictly positive. However, $||\mathbf{x}||^2 = \mathbf{x}.\mathbf{x} = \sum_{m=1}^{k} \lambda_m \mathbf{x}.\mathbf{n}_m < 0$. This is impossible. □

Note that this lemma only allows to avoid the construction of *most* unuseful (empty) BM-cones. Sometimes $\mathbf{n}_1, \ldots \mathbf{n}_k$ may not generate $\mathbb{R}^3$ while $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$ is empty. This occurs for example when $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$ contains a full line or a plane, but does *not* contain $\mathbb{R}^3$. In these (rare) cases, our algorithm computes a set of r vectors $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_r$. However, the resulting BM-cone is empty.

[5]when $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$ is empty, at least one of the triangle associated to the bounding-volume is moving forwards, whatever the object's motion.

In order to practically determine when $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k) = \mathbb{R}^3$, our algorithm uses the following lemma:

Lemma 2. *$\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k) = \mathbb{R}^3$ if and only if $(\pm 1, 0, 0)$, $(0, \pm 1, 0)$ and $(0, 0, \pm 1)$ are in $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$.*

Proof. Let $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$ and $\mathbf{k} = (0, 0, 1)$. Assume $\pm\mathbf{i}$, $\pm\mathbf{j}$ and $\pm\mathbf{k}$ are in $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$, and let $\mathbf{x} = x_i\mathbf{i} + x_j\mathbf{j} + x_k\mathbf{k}$. Assume for example that $x_i \geqslant 0$, $x_j < 0$ and $x_k \geqslant 0$. Then $\mathbf{x} = x_i\mathbf{i} + |x_j|(-\mathbf{j}) + x_k\mathbf{k}$. Since $\mathbf{i}$, $-\mathbf{j}$ and $\mathbf{k}$ are in $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$, $\mathbf{x}$ is also a positive linear combination of $\mathbf{n}_1, \ldots \mathbf{n}_k$. The seven other sign combinations are similar. □

One final lemma is used by our algorithm to search for vectors $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_r$ satisfying condition (5):

Lemma 3. *If $\mathbf{n}_m \in \mathcal{P}(\tilde{\mathbf{n}}_1, \ldots, \tilde{\mathbf{n}}_r)$, $1 \leqslant m \leqslant k$, then $\mathbf{BM}_v \subset \bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$.*

Proof. Let $\mathbf{x}$ denote a point in $\mathbf{BM}_v$, then $\mathbf{x}.\tilde{\mathbf{n}}_m < 0$, $1 \leqslant m \leqslant r$. For any $1 \leqslant m \leqslant k$, since $\mathbf{n}_m$ is a positive linear combination of $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_r$, and since the dot product is bilinear, $\mathbf{x}.\mathbf{n}_m < 0$. □

Since

$$(\mathbf{n}_m \in \mathcal{P}(\tilde{\mathbf{n}}_1, \ldots, \tilde{\mathbf{n}}_r), 1 \leqslant m \leqslant k) \Leftrightarrow (\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_r) \subset \mathcal{P}(\tilde{\mathbf{n}}_1, \ldots, \tilde{\mathbf{n}}_r)) \quad (10)$$

Lemma 3 indicates that a way to compute the BM-cone is to find a positive cone bounding $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_r)$. In order to BM-cull the bounding-volume as often as possible, the r vectors $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_r$ are chosen so as to minimize the difference between the BM-cone $\mathbf{BM}_v$ and $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$.

Thus, the positive cone $\mathcal{P}(\tilde{\mathbf{n}}_1, \ldots, \tilde{\mathbf{n}}_r)$ should be the tightest positive cone bounding $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$.

5.2 The algorithm

The interest in using positive cones representations of vectors sets is that they provide a way to measure the difference between $\mathbf{BM}_v$ and $\bigcap_{m=1}^{k} \mathbf{H}(\mathbf{n}_m)$: in our algorithm, this difference is measured by computing BM-vectors' distances to $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$.

Precisely, we use Wilhelmsen's algorithm[17] to compute the projections $\tilde{\mathbf{n}}_1^* = \mathcal{W}(\tilde{\mathbf{n}}_1), \ldots \tilde{\mathbf{n}}_r^* = \mathcal{W}(\tilde{\mathbf{n}}_r)$ of the BM-vectors on $\mathcal{P}(\mathbf{n}_1, \ldots, \mathbf{n}_k)$. The required distances are simply $d_1 = ||\tilde{\mathbf{n}}_1 - \tilde{\mathbf{n}}_1^*||, \ldots d_r = ||\tilde{\mathbf{n}}_r - \tilde{\mathbf{n}}_r^*||$. Especially, Wilhelmsen's algorithm allows to know that a point belongs to a positive cone when the distance between the point and its projection is zero.

The algorithm building a BM-cone can now be described:

1. ***Preliminary test*** Compute $\mathcal{W}(\pm\mathbf{i})$, $\mathcal{W}(\pm\mathbf{j})$ and $\mathcal{W}(\pm\mathbf{k})$. If every point projects on himself, then return. No non-empty BM-cone can be built. Else, go to step 2.
2. ***Minimal subset computation*** Use Wilhelmsen's algorithm to repeatedly remove vectors which are linearly dependent of the others and obtain a minimal subset $\mathbf{v}_{i_1}, \ldots \mathbf{v}_{i_t}$[6]. If t is smaller than r_{max}, then return the minimal subset as the set of BM-vectors. Else, go to step 3.
3. ***BM-cone initialization*** Compute the minimal vectors mean $\mathbf{m} = \frac{1}{t}\sum_{j=1}^{t} \mathbf{v}_{i_j}$. Randomly pick up r_{max} vectors $\tilde{\mathbf{n}}_1, \ldots \tilde{\mathbf{n}}_{r_{max}}$ from the minimal subset until $\mathbf{m}$ is in $\mathcal{P}(\tilde{\mathbf{n}}_1, \ldots, \tilde{\mathbf{n}}_{r_{max}})$ (r_{max} should be greater than 3). Push the r_{max} vectors outside $\mathcal{P}(\mathbf{v}_{i_1}, \ldots, \mathbf{v}_{i_t})$ (*ie* in the directions $\tilde{\mathbf{n}}_i - \mathbf{m}$).
4. ***BM-cone refinement*** Progressively refine the BM-cone thanks to a simulated annealing algorithm[10]. Briefly, BM-vectors are slightly modified step after step. Modifications are valid if the inclusion constraint (5) holds and if the increase in the distances $d_1, \ldots d_{r_{max}}$ ins't greater than a predetermined ratio. Wilhelmsen's algorithm is used to check the inclusion constraint (through Lemma 3) and to compute the distances.

6 Continuous culling

For now, the method only depends on relative velocities at a given instant. While this is valid for *discrete* collision detection methods, which detect objects interpenetrations at successive instants, this may not be suitable for *continuous* collision detection methods, which use the objects' motions *between* these instants to compute collision times (see for example [1, 2, 11, 15]). For these methods, the relative motion over a time interval may be general, and a triangle moving backwards at the beginning of the time interval may not do so during the *whole* time interval. However, the culling tests can be simply extended using interval arithmetic[14].

Essentially, interval arithmetic allows to bound functions ranges over 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:

$$t \in I \Rightarrow f(t) \in \tilde{f}(I) \quad (11)$$

for any interval I. Thus, the dot products in the culling test (6) yield interval results:

$$\dot{\mathbf{c}}_{ij}^{s}.\tilde{\mathbf{n}}_m \in [l_{s,m}, u_{s,m}] \quad (12)$$

[6] For a large-scale implementation, however, we suggest to adapt a convex hull algorithm.

and a bounding-volume is culled when $u_{s,m} < 0$, $1 \leqslant s \leqslant t$, $1 \leqslant m \leqslant r$.

7 Results

The method has been implemented and added to CONTACT Toolkit, our simulation system[11, 12, 13]. Preliminary experiments have been conducted with industrial data (two instances of the door in Figure 1, 3530 triangles). One test constituted the hard-case for any collision detection method: a mobile object comes in close-proximity with a static object and goes through different contact configurations (transient and permanent contacts). The object's motions were recorded and replayed without and with BM-culling. For the tests, r_{max} was set to 5 and the BM-cones were built using the bottom-up method. Table 1 reports the resulting numbers of BV/BV overlap tests, edge/edge, vertex/face and face/vertex tests in each case[7], and shows that BM-culling allows to reduce significantly the number of all kinds of tests, and results in an important speed-up.

	Without BM-culling	With BM-culling
BV/BV Tests	4.837.443	3.514.144
E/E Tests	3.572.298	1.622.297
V/F Tests	1.170.412	528.195
F/V Tests	1.169.060	528.553
Total time	15.46 sec.	11.85 sec.

Table 1: *BM-culling allows to significantly reduce the number of bounding-volume/bounding-volume (BV/BV), edge/edge (EE), vertex/face (VF) and face/vertex (FV) tests, which results in an important speed-up.*

8 Conclusion

This paper has described a backward-motion culling test that allows to detect in constant-time situations where every triangle associated to a bounding-volume is moving backwards, and thus cull this bounding-volume during the BVHs traversal. This test is based upon the addition of some geometrical information in the bounding-volumes. An algorithm able to compute this geometrical information has been described. The method has been implemented and the conducted experiments suggest that the method allows to significantly speed up the collision detection, especially in close proximity situations.

[7] Since our simulation system uses a continuous collision detection method, these are the only contact types that can occur.

Acknowledgments

The authors would like to thank Renault for providing the car model. The model parts are ©Renault. Tangui Morvan, especially, has to be thanked for coding an inventor parser able to load the provided models. This research was funded by the French Ministry of Research through an AMX grant and the RNTL PERF-RV project.

References

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] S. Kumar, D. Manocha, B. Garett and M. Lin. *Hierarchical back-face computation.* Computer and Graphics, vol. 9, no. 5, pp. 681-692. Special Issue on Visibility, 1999.

[8] 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.

[9] B. Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD Thesis. Fall 1996.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. R. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, Cambridge, second edition, 1992.

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

[12] S. Redon, A. Kheddar and S. Coquillart. *Gauss' least constraint principle and rigid body simulations.* In proceedings of IEEE International Conference on Robotics and Automation, may 2002.

[13] S. Redon. A. Kheddar and S. Coquillart. *Fast continuous collision detection between rigid bodies.* In Proceedings of Eurographics, September 2002.

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

[15] 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.

[16] G. Vanecek. *Back-Face Culling Applied to Collision Detection of Polyhedra.* Journal of Visualization and Computer Animation, Vol.5, no.1, pp.55-63, 1994.

[17] D. R. Wilhelmsen. A Nearest Point Algorithm for Convex Polyhedral Cones and Applications to Positive Linear Approximations. Mathematics of computation, 30, pp 48-57, 1976.