INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Monte-Carlo collision detection

Stéphane Guy — Gilles Debunne

N° 5136

Mars 2004

THÈME 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5136--FR+ENG

Monte-Carlo collision detection

Stéphane Guy*, Gilles Debunne†

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projets Prima et Artis

Rapport de recherche n° 5136 — Mars 2004 — 20 pages

Abstract: This paper presents a method for detecting collisions between objects under the hard real-time constraints of a virtual reality simulation. A list of potential collision regions is computed and updated over time, using temporal coherence to reduce the cost of this update. New samples are constantly randomly generated on every object in order to discover new interesting regions. The objects are then efficiently tested for collision using a multiresolution layered shell representation, which is locally fitted according to an evaluation of the objects' distance. Amortized algorithms allow the user to trade accuracy for speed, in order to reach real-time performances.

Deformable objects and auto-collisions are handled by our algorithm without any change, with a validity that decreases with the amplitude of the deformation. We show how a multiresolution deformable object simulation can be linked with the collision detection process in order to optimize the simulation.

We demonstrate our method in a context of virtual reality by simulating realistic dynamic collisions between several and possibly deformable objects, with a guaranteed frame rate. Benchmarks indicate that the method favorably compares to alternative methods, including those which are restricted to (and optimized for) rigid objects collision detection.

Key-words: collision detection, stochatic method, deformable objects, multiresolution methods

guy|debunne@inrialpes.fr

* Prima
† Artis

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Détection de collision par méthode de Monte-Carlo

Résumé : Ce rapport de recherche présente une méthode pour détecter les collisions entre objets, sous des contraintes de temps-réel. Une liste des régions de collision potentielles est mise à jour, en utilisant la cohérence temporelle. De nouveaux échatillons sont constamment crées sur les objets afin de découvrir de nouvelles régions d'intêret. Les objets sont entourés de couches, localement adaptées selon la distance inter-objets qui a été évaluée.

Les objets déformables et les auto-collisions sont également gérées, avec une validité qui décroit avecf l'amplitude de la déformation. Nous montrons comment lier efficacement la détection de collision et une animation multi-résolution.

Nous faisons la preuve de noter méthode dans un contexte de réalité virtuelle, en simulant les collisions entre plusieurs objets, possiblement déformables, avec une fréquence d'affichage garantie. Nos tests indiquent que notre méthode se compare favorablement à d'autres, y compris celles qui se limitent au cas des objets non déformables.

Mots-clés : détection de collision, méthodes stochatiques, objets déformables, méthodes multirésolution

Contents

1 Introduction

Collision detection is a critical problem for computer simulation and animation. Applications may also be found in domains that involve spatial reasoning among moving objects, including motion planning, robotics and computer-aided design. In most such domains, collision detection is a major computational bottleneck.

Different types of queries can be performed depending on the application. In the simplest case, we only need to determine if there is a collision in the scene. In other cases, we can also ask for a list of collision regions or a minimum Euclidean distance between the objects.

Different models can be used to represent the objects, leading to specific algorithms. Parametric surfaces, implicit surfaces and Constructive Solid Geometry may be suited for *analytical* collision detection, with a possible *exact* computation of the intersection surface. However, most of the models encountered in Computer Graphics and virtual reality are composed of polygons, that generally only represent an approximation of the actual geometry of the object and are not suited for collision detection based on computational geometry.

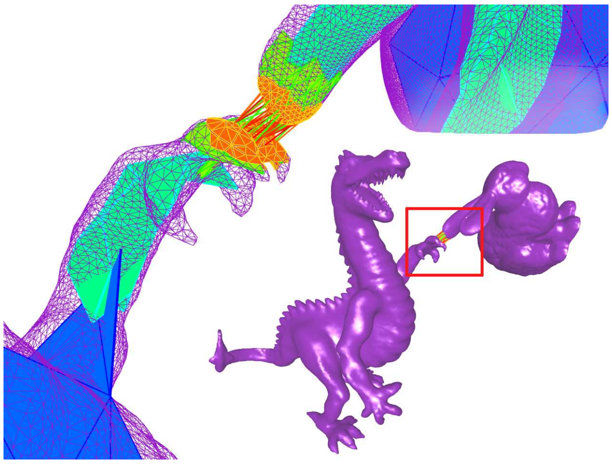


Figure 1: *Collision detection between complex and possibly deformable objects can be performed in real-time using this technique. Presented meshes are made of 50,000 and 70,000 triangles. Color represents the mesh multiresolution levels.*

In this article, we assume that the scene is composed of many polygonal objects that may collide as they are manipulated by the user and may have their own laws of motion (such as Newton's mechanics). Their trajectory can not be expressed in a closed form. However, we can impose reasonable restrictions on their maximum speed and acceleration as well as on the shapes of the objects, which can however be non convex and can have an arbitrary topology.

The targeted application domain of this article is virtual reality simulations. Our goal is to provide a very fast collision detection algorithm that is compatible with the real-time, memory and CPU usage constraints of this kind of simulations. Furthermore, we also want to be able to detect the collisions with objects that may deform over time without increasing the computational cost of the algorithm.

These constraints are hard to satisfy simultaneously, thus we must introduce some restrictions. The amplitude of the objects' deformations must be bounded to ensure the validity of the method. The algorithm is also optimized for *smooth* surfaces and is less efficient for degenerated object geometries. Finally, we can tolerate an *error* (which can be tuned or set to zero) in the collision process in order to speed up the response and to reach the real-time constraints.

During the simulation, we compute and update a *list* of potential collision regions. The update of this list employs *temporal coherence* and permits to simultaneously watch over different potentially colliding regions of the objects. New distance estimations between other random regions of the objects are also constantly generated in order to discover new proximity regions, as will be explained in Section 3. This random research of new collision regions is one of the contributions of the paper and gives it the name of Monte-Carlo.

In order to perform such complex collision detection, the algorithm is based on a *multiresolution* representation of each object of the scene, using different discretizations. Those object representations are based on a simple piece-wise planar approximation that allows for very fast collision queries. This *layered shells* representation of the objects is computed as a pre-process using standard algorithms, such as surface simplification, as will be detailed in Section 4.

The use of this algorithm with a multiresolution deformable model provides an appropriate criterion to drive the model's discretization and to compute the collision response. A simplification of the models when the collision stabilizes is proposed (which does not affect the precise collision contact surface that we have computed), in order to save computations in stable regions. Section 5 will focuss on the case of deformable objects.

This method handles a broad range of virtual reality simulations. One of its advantages is that the same algorithm handles the case of complex rigid objects as well as deformable objects in a unified manner. The update of a list of potential collision regions and the random search of new ones as well as the light-weight memory layered shell hierarchy are some of the main contributions of this work. Benchmarks demonstrate that our approach is faster than classical methods, which are restricted to rigid bodies. The computational cost of the method can be *bounded*, at the expense of its accuracy, thus guaranteeing a true real-time simulation which is critical for virtual reality applications.

2 Previous work

Two recent surveys describe the previous works in collision detection [19, 17] and they should be read for a more complete comparison.

Most of the earlier work in collision detection focused on algorithms for *convex* polytopes. Linear programming gives good theoretical and practical approaches with a linear complexity [29]. An expected constant time can also be achieved using incremental distance computation algorithms on convex objects [21, 3]. Non convex object may be treated using the same methods by using their decomposition into convex part, but this naive approach can not be used in practice on highly non-convex objects.

The currently available efficient techniques for arbitrary objects are mostly based on a *hierarchical subdivision* of the models. Typical examples of bounding volumes include spheres [27, 15], axis-aligned [31] or oriented bounding boxes [12], octrees, BSP trees and many others. Recent works focused on finding tighter fitting bounding volumes [12, 18]. All of these methods do very well in performing rejection tests whenever two objects are far apart. However their performance slows down considerably when the objects are in close proximity as a large number of bounding volumes pairs have to be tested for contact.

Collision detection between deformable objects is "a research area that will certainly deserve attention in the near future" [17]. For the 2D cases, an update of a pseudo-triangle envelope of the object was proposed in [1] and a method based on graphics hardware was presented in [14]. However, most of the approaches for deformable objects collision detection are based on an efficient update of the hierarchical representations of the objects as they deform. Two approaches use hierarchies of axis-aligned bounding boxes to subdivide the model [31, 8]. These hierarchies can easily be recursively updated when the objects undergo small deformations and hence provide a good trade-off between the efficiency of collision queries and the cost of the structure update. Smith *et al.* use an octree after the bounding boxes tests in order to speed up the cases of intersection [30]. An update of the radii and positions of Quinlan' sphere hierarchy [27] is proposed by Brown *et al.* [2]. In order to keep all these hierarchy updates efficient, the deformation must remain *local* so that only some branches of the hierarchy need to be updated. This is unfortunately not the case with physically-based global simulations, where the local deformations are propagated to the entire object. This is a major drawback of such hierarchical update methods for a use in a physically-based simulator.

Some methods take advantage of *temporal coherence* and re-use some of the computations done at the previous time steps [20, 25, 13] in order to speed up the algorithm. Real-time methods are usually based on a simplification of the scene or of the collision response, as a recent work demonstrates using an approximate voxel representation of the scene [22]. An error in the distance/collision queries can also be tolerated in order to accelerate the algorithm [27].

However, none of the presented approaches achieves a true real-time simulation for an arbitrary scene. Deformable object are especially a problem as they often jeopardize the expensive pre-computed collision data structures. The presented method uses a temporally coherent distance estimation between hierarchical representations of the objects as in H-Walk [13], with a layered shell representation that can be compared to the convex polyhedra used in [6]. It tries to generalize these approaches to non-convex and possibly deformable objects. It is especially targeted for real-time simulations, where it may be needed to trade accuracy for speed.

3 Potential collision regions

This section details how the algorithm determines and updates the potential collision regions between two objects. New interesting regions are constantly discovered using a Monte-Carlo-like random method. These potential collision regions are represented as a list of *sample pairs*, one sample on each object, that are tested for collision. The goal is to create and update this list, which should be as small as possible, and yet able to identify all the regions of interest. A unique list can store all the interesting pairs that were detected between all the different objets of the scene.

3.1 Monte-Carlo random search

During all the simulation, and for each couple of objects, new pairs of samples are created, one on each object, and are compared to the pairs that are allready present in the list. Intuitively, the interest of a pair of samples is measured by the *distance* between the samples, and only the shortest distances are kept in the list.

Lets consider a couple of objects of the scene. A new pair of sample points is created using the following process : 1. a *random* sample seed is selected on one of the objects, 2. the closest sample $\mathscr{C}$ on the other object is then computed, 3. the created pair is made of $\mathscr{C}$ and its closest sample on the first object. This method creates a *local* estimate of the distance between the two objects, which depends on the choice of the initial random sample.

This naive approach may lead to a large number of (squared) distance computations. However, the multiresolution scheme presented in Section 4 reduces the original number of samples on each object to a few tens. A space partition can also be used to limit the number of samples that have to be tested on each object. However, the main benefit of this Monte-Carlo approach when applied to real-time simulations is the possible *amortization* of the algorithm.

3.2 Amortized algorithm

The cost of this new pair creation algorithm is $n+n'$ distance computations, where n and n' are the number of considered samples on the two objects $\mathcal{O}$ and $\mathcal{O}'$. This is the price to pay to discover new potentially interesting (close) regions between two arbitrary objects.

However, it is intuitively clear that there is no need to test *all* the possible random seeds at each time step to perform a good probing of these regions of interest and few new pairs actually need to be generated.

The key idea is then to amortize this new pair creation over *several time steps*. At each time step, only a given number of new pairs are created between all the objects of the scene. This bounds the computational cost of the method. This number can be (automatically) tuned during the simulation to ensure that the real-time constraint is satisfied.

With this method, regions may be detected with a little delay, but as it is usually *before* the collision takes place, no collision will be missed. In practice, with a simulation frequency of 30Hz, and with reasonable objects' motions, the creation of only a few pairs of new samples per second is able to capture all the potentially regions of interest. This algorithm is not exact, but the benefits of a bounded and tunable computational time is essential for real-time simulations.

The choice of the random seed must be as random as possible. Ideally, all the possible seeds should be tested sequentially, two consecutive seeds being as far as possible in order to test the maximum number of regions in the minimum number of steps. This is easily achieved using a pre-processed numbering of the samples. Note that the random seed only gives a local starting point for the research, and that the choice of an other close seed would however have probably resulted in the same resulting pair.

The seed can also be chosen from a priority queue, for instance based on the object moving direction or on a display-dependent criterion which favorizes the regions of the objects that are the most visible on screen.

Once a pair has been initialized as described, it is tracked over time. The cost of its creation is then also amortized over several time steps, as its update is easy and cheap.

3.3 Update of a pair

Thanks to temporal coherence, if the two samples of a pair are close at a given time step, they are still interesting for the *next* collision detection. The closest samples of this region may however have slightly changed and the pair must be updated, using the previous pair as an *initial guess*.

A sample *neighborhood* data structure is used to update the pair. This data structure is infered from the mesh created by the samples, using the mesh connectivity information (edges, faces...). A simple pair update method is then to search among the neighbors of a sample for a better (closer) pair.

For each pair, each neighbor of a sample parses the other smaple's neighbors, searching for a closest one. All the "interesting" pairs that are created by this process are kept as is detailed in section 3.4. If the objects move quickly, this simple neighbor update may have to be iterated until a local minimum distance is found.

Once again and as detailed in Section 3.2, an amortized algorithm can here also be used to only test *a part* of all the possible neighbors' pairs at each time step. One can then choose to trade accuracy for speed, an exact solution still being computable if needed.

3.4 Multiple collision regions

This simple update algorithm has been proven to be exact in the case of convex objects [21] and its *constant* practical complexity is very interesting. However this method does not work anymore for non convex objects where a small motion can drastically change the closest distance location. This is the reason why a *list* of the closest pairs between the two objects is maintained, instead of the single closest feature.

This list is made of pairs which almost represent the same distance between the objects. It actually makes sense to use and to update several pairs, which approximately have the same short distance, as all these regions may reveal intersesting when the objects move. Updating several pairs instead of just the closest one is not very expensive with the temporal coherent update and is much more efficient than trying to find the closest pair from scratch at each time step (no temporal coherence would be available for non convex objects). A single closest pair would anyway not be satisfactory in the case of multiple equal closest distances (two parallel planes getting in contact for instance).

This list is constantly refreshed by the addition of new random pairs created using the Monte-Carlo process described in Section 3.1. The important benefit is to cope with *non convex* objects which would otherwise be a problem, even with the pair update mechanism described in Section 3.3.

For efficiency reasons, the number of pairs has to be limited. However, the number of pairs that should be conserved after the Monte-Carlo and the pair update processes depends on the simulation. More precisely, the shortest distance (d_{min}) pair *and* those that may also be interesting (*i.e.* close enough) should be kept. Different criterion were tested and we found that keeping the pairs which distance d satisfies $d < k * d_{min} + \varepsilon$ gives the best results. The k coefficient (typically around 1.2) tunes the number of pairs we want to keep. The ε term is determined from the objects' sizes (say 1% of their sizes) and allows us to keep pairs that may reveal interesting, even when a collision occurs ($d_{min} = 0$).

Note that this criterion no longer bounds the computational cost of the method (in the case of two parallel surfaces for instance), as the number of pairs is only limited by the sampling of the objects. The (automatic) tuning of the amortization coefficients (see Section 3.2) from a real-time measure of the computational cost of the method can reduce this problem. A simple limit on the total number of pairs will also solve this (rare) problem, at the optional expense of a distance sorting of the pairs.

The method presented in this section locates and updates the multiple possible collision regions between the objects of the scene. If the nxn' distances between the samples of two objects were stored in a triangular nxn' matrix (with an appropriate neighborhood-based numbering of the lines and columns), a false color representation would exhibit local areas of short distance pairs. The goal of the Monte-Carlo random pair creation is to find new areas of interest in this matrix. Once a new pair has been found, the pair update process creates new neighboring pairs that soon cover the entire connex area. The process takes advantage of the temporal coherence of the matrix evolution to amortize its computations.

4 Layered shells representation

4.1 Surface approximations

The method takes advantage of multiresolution by considering the object's surface at different resolutions, depending on their collision probability. This probability is related to an evaluation of the distance between the objects, which is a good approximation of their possible next collision. The key is to use coarse representations of the objects when they are distant, thus allowing fast collision queries, and to locally refine this representation as the objects get closer.

As multiresolution techniques start to develop for modeling deformable objects, one can think of linking the collision *and* the animation multiresolution methods. However, we will focuss on the case of rigid objects in the next sections, the application of the method to deformable objects being defered to Section 5.

Layered shells The objects' surfaces is represented with a hierarchical *piece-wise planar* approximation (see Fig. 2). This *layered shells* structure "protects" the object and is tested for a possible collision. For collision consistency reasons, one has to guarantee that for each of these shells, the object and all the finer shells are *entirely included* inside the shell. This is a complex geometrical task and we will detail for each step of the shells construction a practical and fast approximation of the exact solution which is easier to compute and guarantees that no collision is missed.

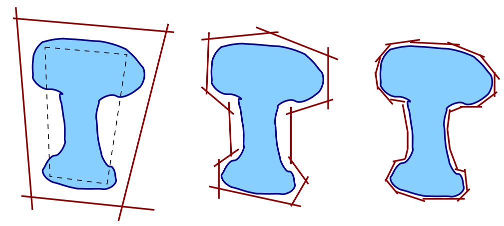


Figure 2: *Different discretizations of the model lead to different layered shell surface approximations, which converge to the real surface of the object.*

Surface simplification is an easy way of creating multiresolution representations of a given object [9]. We used the Garland publically available surface simplification algorithm in the examples, but other methods such as subdivision surfaces [5] or wavelet representation [28] could also be used. The number of generated meshes depends on the initial object geometry and on the user's desires, but an empirical heuristic consists in dividing the number of triangles by a constant factor (around 4 in our case) between each generated mesh, until a coarse approximation of a few tens of triangles is reached. This method gives from 2 to 5 simplified meshes of the original geometry, with a "linearly" increasing complexity.

The next step of the pre-processing is to link these meshes in a parent-child relationship. This hierarchy links the *polygons* of the meshes (triangles in our case). The children of a given triangle are simply defined as the triangles of the finer mesh which represent the same region of the original geometry. This information can be extracted from the surface simplification process or re-created from the topology of the meshes and some distance criterions. Each such child triangle can then be linked to a unique parent, thus creating a transversal tree structure in the meshes.

Shell facets The way the surface shells are computed and represented is related to the multiresolution deformable models that we use to compute the object deformations (see Section 5), but it is also a natural and efficient way to handle rigid objects.

The shell structure is made of planar "facets", each of them being associated with a unique mesh triangle. One shell layer is associated with each simplified mesh of the object, plus one associated with the original geometry. The shape of a shell facet is a complex Voronoi-like polygon which is difficult to compute (and to update in the case of deformable objects). This geometry is simplified by considering each facet as a simple disk. This simplification is justified by the average disk shape of the actual facets and we will detail how to choose the radius of the disks so that no collision can be missed because of this simplification. Each shell facet is then defined by its associated mesh triangle T, its normal $\vec{\mathbf{n}}$, its radius r and an offset distance d as depicted in Figure 3a. The center of the facets c is a simple translation of the triangle barycenter b : $\mathsf{c} = \mathsf{b} + d\vec{\mathbf{n}}$. All those values (and hence the entire shell geometry) are build in a classical bottom-up manner, starting from the finer discretization.

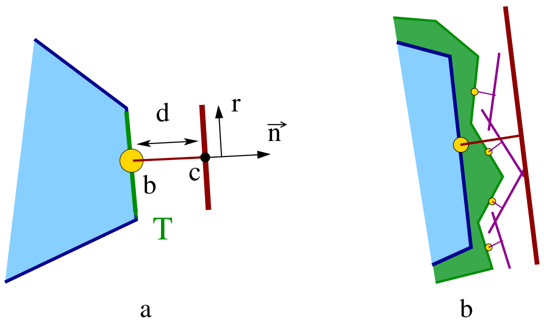


Figure 3: *A facet is attached to each triangle of the object, at different discretizations (a). Succesive shells are constructed around the object (b).*

The offset associated with a facet of the *finer* shell level (i.e. linked with the non simplified object geometry) is set to zero, and its normal is simply the triangle normal. The radius of the facet disk is the radius of the triangle circumscribing circle.

The shell facets' parameters of the coarser levels is then determined from those of the finer level, and more precisely from those of the *children* of their associated triangle. The goal is to set the offset and normal of the parent facet so that all the "child facets" are located *behind* it, and to make sure that the facet is the best fit of its children's facets (minimization of the offset distance). A good heuristic is to set the facet normal to the one of its associated simplified mesh triangle. The offset distance is then computed to make the parent facet external and tangent to its children disk facets (see Figure 3b).

The radii of the created facet disks should be computed from the local surface curvature. The goal is to make sure that the union of all the disks of a given shell level covers the entire surface, with no "holes" between them, where an other object could enter. This is a complex geometric task, and a pretty tight upper bound of this radius, given by $r_i = max_j \, d_{ij}$ (where d_{ij} is the distance between the centers of facets i and j) is used instead.

Note that this radius estimation, as well as the assimilation of a facet to a disk does not compromise the collision queries as it may simply detect a collision between disks while the actual facets are not colliding. However this case rarely happens with smooth surfaces and it simply means that finer levels of the hierarchy are tested, leading to extra computations. In all the cases, no collision is missed because of this slight over estimation of the facets.

Due to the surface simplification "shrinking" artefact, the simplified meshes are usually located *inside* the object original surface. Note that this is not a problem in our case, as the triangles of the simplified geometry have their associated facets projected *outside* of the surface by the offset. The topology and the sampling of the simplified object is closely related to the *shape* of the object and create a good shell representation of the object at this resolution.

For each object, the surface approximation that is actually be used to compute collision queries highly depends on the region of the object. At a given time of the simulation, the geometry of some regions of the same object is precisely sampled by many facets while others are coarsely represented by a single plane. All the hierarchical levels are still available in these regions, but only one is actually considered at a given instant for this specific region, thus saving a lot of computations.

As the discretization of the multiresolution model increases, the facet approximation of the surface refines, converging to an arbitrarily precise representation of the true surface of the object. The finer level of the shell hierarchy does not need to correspond to the object surface geometry. Limiting the shell hierarchy to a *coarser* representation is an easy way of speeding-up the collision query process, at the expense of a distance-bounded error. This is especially important for true real-time application which may need to be able to tune the computational cost on the fly.

4.2 Collisions between shells

We will reduce the following explanations to the study of the collisions between two objects, its generalization simply being the study of all the pairs of objects in the scene.

During the simulation, the different facets of the two shells of the objects must be tested for a possible collision. As detailed in Section 3, only a reduced list of interesting facet *pairs* has to be tested. We now simply consider the collision detection between two facets and the local update of the shell level that it may require.

Distance to an average plane We introduce the notion of *average planes* which can be seen as a piece-wise approximation of the surface contact between the two objects. This idea can be related to the *separating plane* idea introduced in [11]. The average plane of a facet pair is located in the middle of the two facets, with a normal that is the average of the two facets' normals. This plane represents the frontier between the two facets disks.

As the two objects' normals are almost opposite in a collision region, the facets and the average planes are roughly parallel. However, the facets orientation has to be taken into account for determining if the facet crosses the average plane: $r.sin(\theta)$ (see Figure 4) is the closest distance between the disk extremities and the average plane. The difference $\Delta = d - r.sin(\theta)$ (d is the distance from the facet center to the average plane) corresponds to the distance between a local representation of the surface of the object (the facet) and the potential contact surface (the average plane) and it gives an estimation of the facet collision probability.

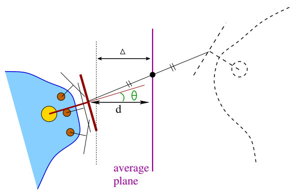


Figure 4: *A fast evaluation of the distance between the facet and the average plane is given by* Δ*. This distance indicates which shell level should be used in this region.*

Changing the shell resolution By construction, the real surface is always *behind* the facet, and the distance Δ to the average plane is always an under-estimation of the actual distance between the object and this potential local collision surface. This estimation can be used to change the local representation of the surface, *i.e.* the shell level that is used. When Δ decreases to zero (and becomes negative, meaning that the facet has crossed the average plane), the shell facet is no longer a good enough local representation of the surface to guarantee that there is no collision. The shell representation is hence refined, replacing this facet by its children, in order to compute a better estimations of the distance between the objects.

The $\Delta <= 0$ criterion is usually sufficient, but we also have to check that the two facets are actually in contact, *i.e.* not too far away. The distance between the two centers of the facets is compared to the sum of their respective radii to make sure that there can really be a collision. The choice of the list of interesting facet pairs limits the necessicity of this test as the compared facets are chosen as allready being close.

Symmetrically, a shell representation is coarsified, a facet being replaced by its parent, when the distance to the average plane exceeds a given threshold. This threshold can for instance be set to the distance between the child facet and its parent facet. It must be greater or equal than this value anyway, in order to prevent a newly created facet to merge back immediately. For precision reasons, a parent facet is never used to replace its children as long as *at least one* of them has a distance to its average plane that is greater than its threshold, so that no usefull child facet are removed.

Collision detection As the objects get closer to one an other, the local shell representation of the surface is refined. When the last finer level of the hierarchy is reached, the facets that are tested for collision are supported by the actual polygons of the object geometry. When the signed distance Δ to the average plane becomes negative, the two associated object polygons have to be tested for a possible collision. This is done using any standard polygonal geometric intersection test.

A null Δ distance does not indeed necessarily means collision as the facets are simply the circumscribing disks of the polygons. However, if one is simply interested in a visually realistic collision, this final geometric test can usually simply be skipped to speed up the algorithm. This assumption is valid only if the objects' geometry is not degenerated and has no ill-shaped polygons, for which an assimilation with their facet would lead to a large error.

5 Deformable objects

One of the main advantages of the method is that it can naturally handle the simulation of *deformable* objects, which then react to the collisions of the simulation. For these objects, we assume that a proper model is able to animate their global deformation when their surface is deformed during a collision (see [10] for a survey on deformable modeling). The algorithm can compute the displacement that should be applied to each collision node in order to prevent the inter-penetration of the objects. The possible *auto-collisions* that may happen are also handled with no change in the algorithm.

5.1 Changes and approximations

The shell structure can be compared with the convex hull shells created by Ehmann and Lin [6]. However, the use of facets attached to the object instead of a fixed polygonal mesh was targeted at handling the case of deformable objects. When the normals of the facets are linked with the ones of the object, the shell structure naturally deforms with the object.

Updating the shell facets parameters so that all the assumption we did on the layered shells remain valid and efficient when the object deforms is a difficult task which is clearly not compatible with the strict real-time constraint. However, we claim that this data structure remains valid (although it can be proven inexact for extreme cases) under reasonable limitations on the amplitude of the deformation, and that it is a efficient tradeoff for designing a real-time collision detection method for a scene that contains deformable objects.

In order to ensure the validity of the shell structure, some minor changes have to be done. The first question is how to update the coarse shell levels as the object deform. We clearly can not afford to compute their simplified geometry at each time step. A first solution is to leave them unchanged, even when the object deform (only the global translation and rotation of the object are taken into account). For small deformations, this strategy proves to be sufficient as the shells are external to the surface, with a margin.

An other solution is to update the coarse levels as the object deforms. This can be done by recomputing the position of the interesting triangles (those that belong to the facet list) from a weighted average of the actual geometry. An other eaier solution is to use a multiresolution deformable model, which actually uses and updates the different representations of the object. We will detail in section 5.2 how this can be done.

The distance under-estimation hypothesis made in section 4.2 may no longer be valid as the object deforms, and a collision may be missed. A slightly over estimated offset and disk radius values (computed according to foreseen object's maximum deformation) can limit imprecisions while preventing from having to compute an exact value during the real-time simulation. A small over estimation is actually desired as it ensures that we switch to a better discretization *before* the collision, so that the finer level is reached when the collision occurs.

5.2 Sampling from collision probability

The switches of the object shell representation can be linked to a multiresolution deformable object simulation, in order to provide an optimal sampling of the object and hence of their deformation. Recent publications introduced multiresolution for animating deformable objects. The idea is to optimize the computational load by increasing the discretization of the physical model in the interesting deformation regions, while using a coarser representation in the stable regions. Hutchinson first presented a 2D mass-spring cloth model which subdivided in high curvature regions, in order to model creases [16]. O'Brien subdivided a Finite Element mesh to model brittle fractures propagation in [26]. 3D deformable objects were recently animated using multiresolution by Ganovelli using mass-spring systems [7] and by Debunne using Finite Elements [4].

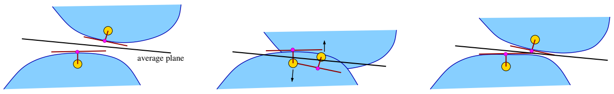
A collision algorithm between deformable objects should be able to take advantage of the efficiency of those methods, which will surely develop during the next years. It actually makes sense to use a precise animation model in regions of high collision probability as the model is hence subdivided to *anticipate* the possible collision, thus ensuring an optimal response when collision actually occurs. The presented algorithm chooses to take advantage of those multiresolution schemes and *links* collision surface approximation and deformable model discretization. This multiresolution aspect of the problem only concerns the collision (and possibly the object deformation) process: the *display* of the objects remains the same during the simulation, thus completely hiding the multiresolution scheme to the user.

The link between the multiresolution deformable model and the shell representation depends on the deformable model. This model is usually based on a hierarchy of meshes which can easily be related to the simplified surface meshes. In all the cases, the local refinements and simplifications of the shell representation *drives* the internal sampling of the deformable object.

5.3 Collision response

When a collision is detected, a possible collision response simply consists in projecting back the facet centers on the average plane (see Fig. 5 b,c), such that Δ is restored to 0. This projection is natural as this plane represents the local contact surface between the two objects. This response is also coherent with the collision detection method as it creates a stable limit collision configuration.

Projecting back the facet on the plane actually means moving its attached triangle *so that* the facet is tangent to the average plane. For a deformable object, this displacement locally affects the shape of the object, the global behavior (bulging, oscillation and displacement) being handled by the underlying physical model.



a b c

Figure 5: *(a) The average plane represents the local collision contact surface. A collision is detected when the facet crosses this plane (b). A possible response is then to simply back project the centers on the contact surface (c).*

If the object is rigid, one can simply transfer the imposed displacement to the frame of the object, but in case of multiple collisions (between two or more objects), this displacement is just the input set of constraints has of a more complex solver [24, 23].

5.4 Collision behavior

This simple collision formalism allows for a wide range of intuitive collision behaviors. The average plane position can for instance be inferred from the deformable physical coefficient (set closer to the stiffer object) so that the deformations applied to the objects depends on their material (a smaller amplitude on the stiff object).

A "sticky" collision can easily be simulated by deciding that a collision is maintained as long as Δ is lower than a given threshold. The objects have to separate at a certain minimum speed (tuned by the threshold) in order to break the collision.

Simulating a friction in the collision is as simple as slightly shifting the back projection of the facet center on the average plane according to the relative speeds of the two samples. Many other interesting behaviors can easily be implemented.

5.5 Stable regions simplification

Subdividing the model in collision regions creates extra computations but is needed to ensure an optimal surface deformation. However, when the two objects do not move anymore, and even if there is still a collision, one might want to simplify back the discretization of the objects in order to save some computational load, which can be useful for other active collisions.

Each pair is given an age which is incremented after each collision detection as long as the two samples of the pair remain the same and have almost a null relative speed. If one of these two conditions is not satisfied, the age of the pair is reset to zero. When the age of a pair reaches a given threshold, the pair is declared *old*. Note that this threshold is expressed in seconds as a real-time simulation guarantees the number of collision detection done every second.

The samples that belong to old pairs are allowed to simplify back, over-passing the distance criterion defined in section 5.2. A sample should decide to merge only when *all* its children belong to old collision pairs. The pairs that are created from a simplification are also declared old, thus ensuring a fast complete simplification of the model when the region is stable. If the objects were to move again after this simplification, the pairs would no longer be old and the distance criterion defined in 5.2 would immediately subdivide back the model. The resulting algorithm is rather complex and special care is needed in order to prevent successive splits and merges of the same pair.

Such simplification of the inner physical model may affect the surface of the object and distract the user. In order to prevent this, the position of the surface vertices which are linked to an old pair sample are then simply "frozen". The precise contact surface that was computed using the subdivided physical model is still displayed although the inner model is completely simplified.

6 Results

Using this method, the simulation can find and update hundreds of pair between the objects of the scene. All the potential collision regions are discovered and complex object topologies are handled. Multiresolution deformable models benefit from the collision detection and subdivide the model before the collision take place. We computed collision detections between models of more than 50000 triangles at 30Hz on a regular PC (Fig. 1). Parallelization of the method would be easy. Figure 6 shows snapshots of the simulations.

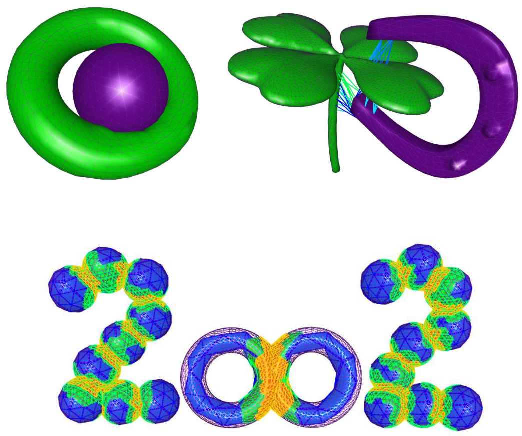


Figure 6: *The algorithm computes collision detection and response on arbitrary objects : deformable object and complex non-convex geometries.*

6.1 Comparison with other methods

We compared the algorithm with two hierarchical bounding boxes methods : the RAPID library (OBB: oriented bounding boxes, rigid objects only [12]) and the AABB method (axis aligned BB, possible object deformation [31]). The models were compared on 3 scenario of increasing complexity, with rigid and deformable objects whose motion was recorded and replayed using the different collision methods. The benchmark exemples are shown in the video and in Figures 1&6.

Models (# of triangles)	AABB	OBB	Shells
Sphere(980)-Tore(2300)	9.5	42/4.4	6.9/4.7/4.2
Horseshoe(2K)-Clover(10K)	33.7	230/1.6	4.9/2.7/0.6
Dragon(50K)-Bunny(70K)	434	N/A	140/51/5.5

Table 1: *Comparison with other methods. Times are expressed in milliseconds. Increasing the number of multiresolution levels speeds up the collision process.*

For OBB, the times represent the structure construction and the collision query. For deformable objects, the time is the sum of all the construction and query times corresponding to the different time steps. The different times given for the multiresolution method correspond to the use of different numbers of shell levels (from one to four). Note how the times regularly decreases as we use more shells around the object.

The finer shell always corresponds to the actual geometry of the object, so that the comparisons with the other methods are fair. For non-convex objects, a valid comparison requires that all the potential collision regions are properly detected, which is well the case, as demonstrated in the video.

OBB could not fit in the 512Mb of memory on the dragon-bunny example. Table 1 demonstrates that the presented method, although more general, performs faster than the two other ones. This is probably due to the fact that it is *collision dependent* and not *geometry dependent* like classical hierarchical volumes structures. The cost of the method only depends on the number of potential collision faces (reduced by multiresolution) whereas classical approaches are limited by the complexity of the geometry. Note that the number of pairs can easily be tuned by the k and ε coefficients defined in 3.4. This number can also be reduced in order to satisfy the real-time constraints of the simulation.

7 Conclusion and discussion

We introduced a novel algorithm for computing collision detection in a context of real-time virtual reality. The potential collision regions are identified, updated and randomly discovered. A fast facet collision test uses a multiresolution representation of the objects' surfaces to detect actual collisions.

The main advantage of the method is the fact that it can handle complex object, including deformable objects, with no change in the algorithm. A multiresolution deformable object simulation can then be linked to the simulation in order to optimize the deformation computation. The benchmarks demonstrate that, although more general, the method performs faster than efficient hierarchical methods. The multiresolution scheme also allows for a tuning of the computational load, which is critical for real-time virtual reality simulations.

The different assumptions that were made on the amplitude of the deformations and on the smoothness of the geometry are usually respected, or at least create no visible problem. In the context of a virtual surgery simulator, we were able to increase the realism of the simulation with deformable organs that interacted as the surgeon practices.

The ideas of the method could be applied to a wider class of object representation. Implicit and parametric surfaces could be represented in a multiresolution framework which would allow the same kind of simulation.

Surface simplification tries to preserve the surface features. However, finding a coarse representation of the surface which provides bounds on the distance to the real surface would allow for a better multiresolution construction, with tighter offset plane distances. It could also handle the case of sharp edges which otherwise need to be modeled using several attached planes.

References

[1] P. Agarwal, J. Basch, L. Guibas, J. Hershberger, and L. Zhang. Deformable free space tilings for kinetic collision detection, March 2000.

[2] Joel Brown, Stephen Sorkin, Cynthia Bruyns, Jean-Claude Latombe, Kevin Montgomery, and Michael Stephanides. Real-time simulation of deformable objects: Tools and application. In *Proceedings of Computer Animation*, November 2001. Seoul, Korea.

[3] J. Cohen, M. Lin, D. Manosha, and M. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, pages 189–196, Aug 1995.

[4] G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, Los Angeles, USA, August 2001.

[5] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proc. of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, pages 85–94, July 1998.

[6] S. A. Ehmann and M. C. Lin. Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. In *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[7] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. In *Proceedings of Eurographics*, 2000.

[8] F. Ganovelli, J. Dingliana, and C. O'Sullivan. Buckettree: Improving collision detection between deformable objects. In *Spring Conference on Computer Graphics*, 2000.

[9] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH*, volume 31 of *Computer Graphics Proceedings*, pages 209–216, 1997.

[10] S. F. Gibson and B. Mirtich. A survey of deformable modeling in computer graphics. Technical report, MERL, 1997. TR-97-19.

[11] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three dimensional space. *IEEE J. Robotics and Automation*, 4(2):193–203, 1988.

[12] S. Gottschalk, M. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proceedings of Siggraph*, Conference Proceedings, pages 171–180, Aug 1996.

[13] L. Guibas, D. Hsu, and L. Zhang. H-walk: Hierarchical distance computation for moving convex bodies. In *Proceedings of the ACM Symposium on Computational Geometry*, 1999.

[14] Kenneth E. Hoff, Andrew Zaferakis, Ming Lin, and Dinesh Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *Proceddings of ACM Symposium on Interactive 3D Graphics*, 2001.

[15] Phil Hubbard. Interactive collision detection. In *Proc. of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.

[16] D. Hutchinson, M. Preston, and T. Hewitt. Adaptative refinement for mass/spring simulations. In *Proc. of the Eurographics Workshop on Computer Animation and Simulation*, pages 31–45, Poitiers, sep 1996.

[17] P. Jiménez, F. Thomas, and C. Torras. 3d collision detection: A survey, 2001.

[18] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. In *IEEE Transactions on Visualization and Computer Graphics*, volume 4(1), pages 21–36, 1998.

[19] M. Lin and S. Gottschalk. Collision detection between geometric models: A survey. In *IMA Conf. on Mathematics of Surfaces.*, 1998.

[20] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, 1993.

[21] M. C. Lin and J. F. Canny. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1008–1014, 1991.

[22] W. McNeely, K. Puterbaugh, and J. Troy. Six-degree-of-freedom haptic rendering using voxel sampling. In *Proc of Siggraph*, pages 401–408, 1999.

[23] V. Milenkovitch, H. Schmidt, and A. McMahon. Optimization-based animation. In *Proceedings of SIGGRAPH*, Computer Graphics Proceedings, Annual Conference Series, 2001.

[24] B. Mirtich. Timewarp rigid body simulation. In *Proceedings of SIGGRAPH*, Computer Graphics Proceedings, 2000.

[25] B. Mirtich and J. Canny. Impulse-based dynamic simulation. In *Workshop on Algorithmic Foundations of Robotics*, 1994.

[26] J. O'Brien and J. Hodgins. Graphical models and animation of brittle fracture. In *Proceedings of SIGGRAPH'99*, pages 137–146, 1999.

[27] Sean Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of the IEEE International conference on Robotics and Automation*, pages 3324–3329, 1994.

[28] P. Schröder, W. Sweldens, M. Cohen, T. DeRose, and D. Salesin. Wavelets in computer graphics. Siggraph course notes, 1996.

[29] R. Seidel. Linear programming and convex hulls made easy. In *6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, 1990.

[30] A. Smith, Y. Kitamura, H. Takemura, and F. Kishino. A simple and efficient method for accurate collision detection among deformable objects in arbitrary motion. In *Proc. of the IEEE Virtual Reality Annual Int. Symposium*, pages 136–145, 1995.

[31] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399