

# Collision Handling and its Applications

Matthias Teschner<sup>1</sup>, Marie-Paule Cani<sup>2</sup>, Ron Fedkiw<sup>3</sup>, Robert Bridson<sup>4</sup>, Stephane Redon<sup>5</sup>, Pascal Volino<sup>6</sup>, Gabriel Zachmann<sup>7</sup>

<sup>1</sup> Freiburg University, Germany

<sup>2</sup> INP Grenoble, France

<sup>3</sup> Stanford University, USA

<sup>4</sup> University of British Columbia, Canada

<sup>5</sup> INRIA Rhone-Alpes, France

<sup>6</sup> University of Geneva, Switzerland

<sup>7</sup> Clausthal University, Germany

## 1. Introduction

In contrast to real-world scenarios, object representations in virtual environments have no notion of interpenetration. Therefore, algorithms for the detection of interfering object representations are an essential component in virtual environments. Applications are wide-spread and can be found in areas such as surgery simulation, games, cloth simulation, and virtual prototyping.

Early collision detection approaches have been presented in robotics and computational geometry more than twenty years ago. Nevertheless, collision detection is still a very active research topic in computer graphics. This ongoing interest is constantly documented by new results presented in journals and at major conferences, such as Siggraph and Eurographics.

In order to enable a realistic behavior of interacting objects in dynamic simulations, collision detection algorithms have to be accompanied by collision response schemes.

## 2. Summary

This tutorial discusses collision detection and response algorithms with a special emphasis on a wide range of applications, such as rigid objects, deformable objects, cloth, hair, point clouds and fluids. On one hand, the tutorial illustrates common aspects of collision handling approaches in these application areas. On the other hand, unique requirements of each application are outlined. The presentation of a variety of solution strategies for collision handling problems in terms of specific applications provides the participants of the tutorial with the ability to evaluate existing techniques in the context of their own application.

The presentation of rigid body collision handling illus-

trates the problem that collisions can occur in-between discrete time steps of a dynamic simulation. If these collisions have to be detected or the exact contact time is required, traditional discrete-time techniques cannot be employed. Instead, continuous collision detection has to be performed which is explained in the tutorial.

The presentation of deformable collision handling focuses on the interplay of collision detection and response. If a realistic interactive simulation of interacting three-dimensional deformable objects is desired, the collision detection algorithm has to provide collision information that can be used by the collision response scheme. If the collision detection is efficient, but does not provide useful collision information, the interplay of detection and response cannot be effective.

Cloth simulation is a challenging application area for collision handling approaches. In contrast to other areas, self-collision cannot be neglected and stacking has to be handled very carefully. While collision response methods for three-dimensional objects commonly employ penetration depth information, this information is not available for two-dimensional cloth models. The tutorial describes specific solutions to cloth collision handling.

Handling hair self collisions is one of the main challenges in hair simulation, due to the extremely high number of strands. These anisotropic interactions play a key role in hair shape and motion, being the cause for hair volume and for the damping of hair motion. The presentation analyzes and compares the different strategies that were used for tackling the problem, from volumetric methods to the adaptive, hierarchical clustering of hair strands.

Recent publications suggest that point-based object representations are useful in rendering and in animation. However, in the context of collision handling these object repre-

sentations present unique challenges since point-based objects are not characterized by explicit surface representations. Instead, efficient collision handling methods for implicit surfaces are required that are described in the tutorial.

Collision handling among fluids and solids is a very challenging topic. Different representations of fluids and solids have to be considered. If a collision occurs, both structures influence each other which has to be handled by a two-way collision response or two-way fluid / solid coupling. Additional problems have to be addressed when multi-phase fluids interact with arbitrarily thin solids such as cloth. Specific approaches to these problems are presented in the tutorial.

In summary, the tutorial illustrates specific problems in application areas related to collision handling. The tutorial shows that there does not exist an optimal approach that can be used in all cases. Instead, all prerequisites of a problem and all potential constraints have to be considered in order to choose a useful combination of collision detection and response techniques.

### 3. Proposed Length

- half-day tutorial

### 4. Topics

Collision handling for rigid objects, deformable objects, cloth, hair, point clouds, and fluids.

### 5. Tutorial Syllabus

*Introduction.* Matthias Teschner. 10 min. The relevance of collision detection and collision response is motivated. Challenges in the context of specific applications are illustrated.

*Rigid Bodies.* Stephane Redon. 30 min. Continuous collision detection approaches for rigid and articulated objects are presented. Methods for the discretization of trajectories and the computation of the exact contact time in discrete-time simulations are described. See Fig. 1.

*Deformable Objects.* Matthias Teschner. 25 min. The handling of collisions and self-collisions of dynamically deforming three-dimensional objects will be covered. The interplay of efficient collision detection, appropriate collision information, and physically based collision response is explained. See Fig. 2.

*Cloth.* Pascal Volino. 30 min. Collision detection and response techniques for non-volumetric objects are discussed. Since penetration depth information cannot be used for collision response, specific collision handling approaches will be presented that handle two-dimensional objects. See Fig. 3.

*Hair.* Marie-Paule Cani. 30 min. Different strategies for collisions detection and response are reviewed: we show that

volumetric approaches can be used for handling both hair-body and hair-hair interactions in real time. More intricate solutions yielding very realistic results are presented, such as the adaptive clustering of strands into interacting, anisotropic hair wisps. See Fig. 4.

*Points.* Gabriel Zachmann. 25 min. Point representations have shown to be attractive for rendering and animation. This part of the tutorial discusses approaches to detecting collisions among point-based objects. Efficient collision handling methods for implicit surfaces are discussed that do not explicitly reconstruct the object surface. See Fig. 5.

*Fluids.* Robert Bridson, Ron Fedkiw. 30 min. Solid / fluid coupling algorithms are presented. Challenges in the context of multi-phase fluids and arbitrarily thin solids are discussed. In particular, algorithms for cloth and water and their two-way interaction are described. See Fig. 6.

### 6. Prerequisites

The participants should have a working knowledge of spatial data structures and computational geometry. Further, participants should be familiar with dynamic simulation approaches for solids and fluids.

### 7. Organizer

Matthias Teschner

Computer Science Department, Freiburg University

Georges-Koehler-Allee 052

79110 Freiburg im Breisgau, Germany

mail [teschner@informatik.uni-freiburg.de](mailto:teschner@informatik.uni-freiburg.de)

http <http://cg.informatik.uni-freiburg.de/>

### 8. Speakers

Matthias Teschner received the PhD degree in Electrical Engineering from the University of Erlangen-Nuremberg in 2000. From 2001 to 2004, he was research associate at Stanford University and at the ETH Zurich. Currently, he is professor of Computer Science and head of the Computer Graphics group at the University of Freiburg. His research interests comprise real-time rendering, scientific computing, physical simulation, computer animation, computational geometry, collision handling, and human perception of motion. His research is particularly focused on real-time physically-based modeling of interacting deformable objects and fluids with applications in entertainment technology and medical simulation. Matthias Teschner has contributed to the field of physically-based modeling and collision handling in several papers. At Eurographics 2004, he organized a State-of-the-Art report on collision detection. At IEEE VR 2005 and Eurographics 2005 he participated and organized tutorials on collision detection and collision handling.

*Marie-Paule Cani* is a full Professor of Computer Science at the INPG, France. A graduate from the Ecole Normale Supérieure, she received a PhD in Computer Science from the University of Paris Sud in 1990, and was elected member of the Institut Universitaire de France (IUF) in 1999. She is vice-director of the research lab GRAVIR (Computer GRAPHics, Computer VIsion and Robotics), a joint lab of CNRS, INPG, INRIA and UJF, where she leads the research group EVASION, created in 2003. Her main research interests cover physically-based simulation, implicit surfaces applied to interactive modeling and animation and the design of layered models incorporating alternative representations and LODs. Recent applications include the animation of natural phenomena and virtual humans, real-time virtual surgery and interactive sculpting techniques. Marie-Paule Cani was paper co-chair of Eurographics 04, is co-chairing Shape Modeling International 2005 and has served in the program committee of major graphics conferences, including SIGGRAPH in 2001 and 2005.

*Ron Fedkiw* received his Ph.D. in Mathematics from UCLA in 1996 and did postdoctoral studies both at UCLA in Mathematics and at Caltech in Aeronautics before joining the Stanford Computer Science Department. He was awarded the National Academy of Science Award for Initiatives in Research, a Packard Foundation Fellowship, a Presidential Early Career Award for Scientists and Engineers (PECASE), a Sloan Research Fellowship, the ACM Siggraph Significant New Researcher Award, an Office of Naval Research Young Investigator Program Award (ONR YIP), a Robert N. Noyce Family Faculty Scholarship, two distinguished teaching awards, etc. Currently he is on the editorial board of the Journal of Computational Physics, Journal of Scientific Computing, IEEE Transactions on Visualization and Computer Graphics, and Communications in Mathematical Sciences, and he participates in the reviewing process of a number of journals and funding agencies. He has published over 60 research papers in computational physics, computer graphics and vision, as well as a book on level set methods. For the past five years, he has been a consultant with Industrial Light + Magic. He received screen credits for his work on "Terminator 3: Rise of the Machines" and "Star Wars: Episode III - Revenge of the Sith".

*Robert Bridson* received his Ph.D. in Scientific Computing and Computational Mathematics in 2003 at Stanford University before arriving at UBC. He has published several computer graphics papers on physics-based animation, from fluids to cloth, as well as a number of scientific computing papers, and is currently writing a book on physics-based animation for Cambridge University Press. He codeveloped physics-based animation software for feature film visual effects in use at Industrial Light + Magic and elsewhere, and has also consulted at Sony Pictures ImageWorks.

*Stephane Redon* is a Research Scientist at INRIA Rhone-Alpes, working with Dr. Sabine Coquillart in the i3D re-

search team. He graduated from Ecole Polytechnique in 1998, and received his M.S. in 1999 from Pierre and Marie Curie University, France. He received a Ph.D. in Computer Science in 2002 from INRIA Rocquencourt - Evry University, France, while working with Dr. Sabine Coquillart and Prof. Abderrahmane Kheddar on robust interactive simulation of rigid body systems and its applications to virtual prototyping and animation. He spent two years in the Department of Computer Science of the University of North Carolina at Chapel Hill as a Post-Doctoral Research Associate, working with Prof. Ming C. Lin in the GAMMA research team. His research interests include the design of robust and realistic real-time virtual environments, collision detection, haptics, motion planning, simulation levels of detail, and 3d interaction. His current research is centered on the development of scalable algorithms for interactive simulation and control of complex dynamical systems.

*Pascal Volino* is a computer scientist, working at MIRALab, University of Geneva. He is actually working on new models for cloth animation, involving versatile models for efficient simulations on situations involving high deformation, wrinkling and multilayer garments. The research is particularly focused on data structure, efficient collision detection, robust simulation and interactive cloth manipulation.

*Gabriel Zachmann* is professor for computer graphics at Clausthal University since 2005. Prior to that, he was assistant professor with the computer graphics group at Bonn University. He received a PhD in computer science from Darmstadt University in 2000. From 1994 until 2001, he was with the virtual reality group at the Fraunhofer Institute for Computer Graphics in Darmstadt, where he carried out many industrial projects in the area of virtual prototyping. Zachmann has published many papers at international conferences in areas like collision detection, virtual prototyping, intuitive interaction, mesh processing, and camera-based hand tracking. He has also served on various international program committees.

## 9. Course Notes Description

This tutorial builds on lecture material from the Freiburg University, INPG Grenoble, Stanford University, INRIA Rhone-Alpes, University of Geneva, and Clausthal University. Further, material from previous tutorials at Siggraph 2004, IEEE VR 2005 and Eurographics 2005 will be used. Since all presenters actively contribute to the area of collision handling, all presentations will be accompanied by videos and software demonstrations.

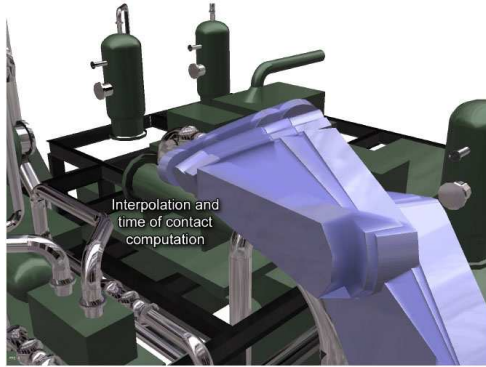


Figure 1: Continuous collision detection for rigid objects.



Figure 4: Volumetric versus hierarchical clustering methods for handling hair self collisions.

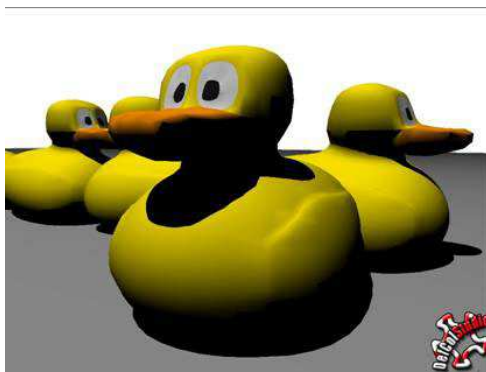


Figure 2: Handling of collisions and self-collisions among three-dimensional deformable objects.

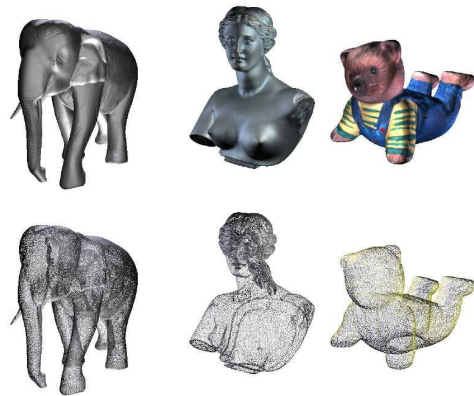


Figure 5: Collision detection for point-based objects using implicit surface reconstructions.



Figure 3: Cloth collision handling for two-dimensional objects without penetration depth information.

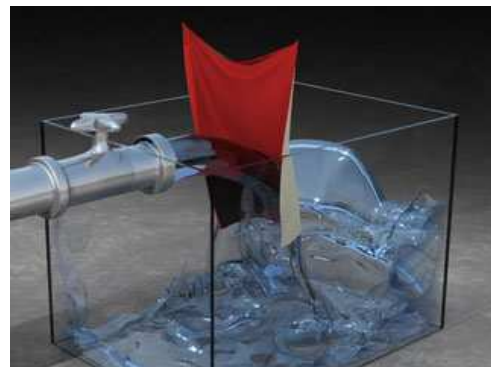


Figure 6: Two-way coupling of multi-phase fluids and arbitrarily thin solids.

# Hair interactions

Marie-Paule Cani and Florence Bertails

EVASION group (CNRS, UJF, INPG, INRIA), Grenoble, France

---

## Abstract

*Processing interactions is one of the main challenges in hair animation. Indeed, in addition to the collisions with the body, an extremely large number of contacts with high friction rates are permanently taking place between individual hair strands. Simulating the latter is essential: without hair self-interactions, strands would cross each other during motion or come to rest at the same location, yielding unrealistic behavior and a visible lack of hair volume.*

*This chapter reviews the most recent advances to tackle the specific problems of hair collision detection and response. The solutions presented here range from simple approximations that provide hair with a volumetric behavior in real-time to dedicated algorithms for efficiently yet robustly detecting collisions between hair guides and for generating a realistic response to hair interactions.*

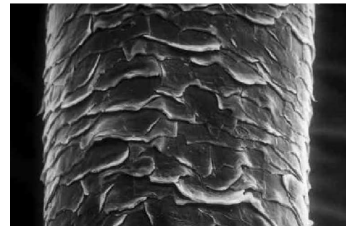
---

## 1. Introduction

Human hair is a composite, deformable material made of more than 100 000 individual fibers called hair strands. These thin tubular structures are elastic: after motion, they tend to come back to a rest shape, which is related to their individual natural curliness and to the set of external forces applied to them. Global hair motion and even the shape hair takes at rest highly depend on the nature of the multiple interactions taking place between hair strands: collisions and contacts between hair strands of different orientations cause hair to occupy a pretty high volume, especially in the case of irregular, curly or fuzzy hair. Due to this larger volume, tangled or fuzzy hair in motion is much more subject to air damping than smooth and disciplined hair.

The nature of interactions between hair strands is very complex. This is largely due to the surface of individual hair strands, which is not smooth but composed of tilted scales (see Figure 1). This irregular surface causes anisotropic friction inside hair, with an amplitude that strongly depends on the orientation of the scales and of the direction of motion [Zvi86]. Moreover, hair is very triboelectric, meaning it can easily release static charges by mere friction. This phenomenon, which has been measured in the case of combed hair, most probably impacts the hair-hair friction rates.

Because of the extremely large number of strands that compose a full head of hair, processing hair interactions is



**Figure 1:** An electron micrograph of a hair fiber that shows the structure of the outer cuticle surface, which is composed of thin overlapping scales [Rob94].

known as one of the main challenges in hair animation. Until the late nineties, most hair animation methods tackled hair collisions with the body, but were not processing self-interactions at all. This often resulted into an obvious lack of hair volume. The first methods that detected interactions between hair wisps spent more than 80% of the simulation time in this process. More recently, several interesting solutions that make hair interactions much more practical were developed: some of them mimic the effect of hair interactions globally, using a structure that stores the volumetric density of hair. Others achieve more accurate results by developing efficient algorithms for detecting collisions between hair-wisps and by setting up realistic models for response and friction forces.

This chapter presents those of these recent advances in which the authors participated: Section 2 briefly reviews the two main approaches for animating hair, namely modeling hair as a continuum or as a set of individual hair wisps. The associated methods for processing hair interactions with the body are presented and the issues raised by hair self-interactions are introduced. Section 3 presents a practical real-time solution, applicable in any hair animation system, which gives hair a volumetric behavior without requiring to detect individual interactions between the animated guide-strands. We then focus on more accurate methods, applicable for generating high quality animation of long hair: Section 4 reviews some recent methods for efficiently, yet robustly detecting the interactions between guide-strands. Section 5 discusses the anisotropic models that were set up to model response to these interactions. In particular, we describe a validated model for friction forces. In conclusion, we emphasize the steps forwards made in the last few years, but also the issues that were not tackled yet, showing that improving the efficiency and visual realism of hair animation is going to remain a hot research topic for a while.

## 2. Hair animation and interaction processing

### 2.1. Continuous versus wisp-based hair models

Hair animation was made practical in the early nineties [RCT91] by the idea of animating only a subset of the hair strands (typically one or two hundreds), which we will call here the *guide-strands*. This is made possible by the local spatial coherence of hair motion. Once the guide-strands have been animated (using for instance spring and masses, projective dynamics or chains of articulated rigid bodies), their position is used to generate the remaining hair strands at the rendering stage.

More precisely, two main families of approaches were developed for modeling hair: The first ones, more appropriate for smooth, fluid hair, consider hair as a continuum [AUK92, DTKT93, HMT01, CJY02, BCN03] and thus use interpolation between the animated guide-strands for generating a full head of hair. The second ones, which achieve their best results for wavy or curly hair, model hair as a set of disjoint wisps [CSDI99, KN00, PCP01, KH01, BKCN03, WL03, CCK05]. The animated guide-strands are assimilated to wisp skeletons and extrapolation is used for generating extra hair-strands within each wisp. Recently, Bertails [BAC\*06] bridged the gap between the two kinds of approaches by allowing the guide-strands to be used both for interpolation or approximation depending on the type of hair and on the current distance between neighboring guide-strands. This model captures hair that looks like a continuum near the head while well identified wisps can be observed at the tip.

In the remainder of this chapter, we will discuss hair interactions independently of the hair model used among the

approaches above: hair will be considered as a set of individual hair guides, each of them more or less explicitly modeling a volume of hair around it. Interactions will be detected and treated based on the position and motion of these guide-strands.

### 2.2. Processing hair interactions with the body

The first step towards processing hair interactions is to adequately model hair collisions and contacts with obstacles, starting with the body of the animated character. Since hair is animated using guide-strands, the latter and the wisp volumes around them (if any) should be prevented from penetrating inside the body. The latter is often approximated using sets of ellipsoids or stored in a spatial partitioning grid to accelerate this detection. Since hair is a very soft material, modeling a one way response is sufficient: the body can be considered as infinitely rigid and heavy compared with hair, so the collision has no effect on the subsequent body shape and motion. Moreover, hair is a very soft and light material: it does not bounce after collision, but rather experiment a strong static friction with the parts of the body it is in contact with. Collision response can thus be treated using methods set up for other very light material, such as clothing: when a penetration is detected, the guide-strand or the associated wisp volume is re-positioned as to be in resting contact with the body. The guide-strand is either given the velocity of this body part, or a static friction force is set up between them.

The remainder of the chapter focuses on the part of interaction processing most specific to hair and much more difficult to handle than collisions with obstacles: we are now addressing the challenging problem of self-interactions.

### 2.3. The issues raised by hair self-interactions

The interactions that occur between hair-strands are very difficult to simulate, For the following reasons:

Firstly, in real hair, the friction between neighboring strands of similar orientation plays an important part: it dissipates some kinetic energy and damps the overall motion. This phenomenon cannot be simulated properly in virtual hair, where only a few guide-hair distributed on the scalp are animated. The only way to capture this part of self-interaction is to add some internal damping - which should depend on the type of hair and is quite difficult to tune - on the individual motion of a guide strand.

Secondly, strands are very thin, so standard collision detection methods based on penetration cannot be used: strands or even small wisps of hair of different orientations might cross each other between two simulations steps and go to rest in the wrong positions, this interaction remaining unnoticed.

Lastly, once a collision between hair guides or hair wisps of different orientation have been detected, the response

model should account for the complex state of surface of a hair strand: the tilted scales that cover a strand result in strongly anisotropic static friction. Moreover, these friction forces are dominant: due to the lightness on a hair strand, the colliding strands will most probably remain in contact. One of the challenges of hair self-interactions it thus to define a response model that prevents strands from crossing each other while avoiding to generate any bouncing. The latter, often noticeable in hair animation systems, gives an overall unstable behavior to the full hair, due to the extremely large number of local collisions that occur at each time step, even when hair is at rest.

Historically, the continuous and wisp-based approaches have tackled hair self-interactions in dramatically different ways:

- **Volumetric interactions:** Continuum approaches such as Hadap's and Bando's methods relied on fluid-like internal viscosity to model hair friction and to prevent self-intersections in a rather global way [HMT01,BCN03]: no collision is detected between individual hair strands, but the latter interact (as fluid particles would do), depending on the local hair density and on the relative hair motion around them.
- **Guide-strands interactions:** In contrast, processing hair self-collision in discontinuous, wisp-based approaches has been done through the actual detection of penetration between moving hair wisps [PCP01]. This allows a more accurate modeling of the discontinuities that can be observed during fast motion of long, human hair: in these approaches, wisps of hair defined around a guide-strand are prevented from crossing each other and two wisps of different orientations can be in resting contact.

We believe that the general approach chosen for handling hair interactions can be chosen quite independently from the hair model, would it be a continuum model, an disjoint set of hair wisps, or something inbetween.

The remainder of this chapter presents the specific solution the authors have developed for tackling the problem of hair interactions. This chapter is not aimed as providing a state of the art in the area: the interested reader can find a recent survey on hair animation and rendering techniques in [WFK\*06]. The volumetric method for hair interactions presented in Section 3 belongs to the volumetric interactions approach: it provides a real-time alternative to fluid-like interactions when a coarser approximation is sufficient. Methods for improving the efficiency of collision detection and the realism of collision response in the interacting guide-strands approach are detailed in Sections 4 and 5.

### 3. A volumetric approach for real-time self-interactions

The work presented in this section was first introduced in [BMC05], as a side application of a method for handling

hair self-shadowing in real-time. We detail here the application of this approach to hair self-interactions.

#### 3.1. A volumetric structure for hair

An acceptable approximation of hair self-interaction consists of considering that internal collisions mainly result into the preservation of hair volume [LK01]. Starting from this assumption, hair density information is very useful: If the local density of hair is over a fixed threshold (corresponding to the maximum quantity of hair that can be contained within a cell), the hair strands should undergo external forces that spread them out.

Bertails *et al.* [BMC05] use a light-oriented voxel grid to store hair density values. This enables them to efficiently compute *both lighting and mechanical interactions* inside the hair volume in real-time. Though very simple, this method yields convincing interactive results for animated hair, is very simple to implement, efficient and can easily be parallelized to increase performance.

More precisely, the volumetric structure used is based on a 3D light-oriented density map, which combines an optimized volumetric representation of hair with a light-oriented partition of space. This voxel structure stores the local hair density in space, computed from the number of guide-strand segments within a given cell. It is used to approximate the light attenuation through each cell of the grid: since the cells are sorted along the light direction, computing the accumulated translucency for each cell through the hair volume becomes straightforward.

#### 3.2. Application to hair interaction

At each animation step, all guide-strand are moved to their new position and the density map is updated. Then, hair self-collisions are taken into account for the next simulation step by adding density-based interaction forces where needed: repulsive forces directed from the center to the border of a grid cell are generated. They are applied to each hair-guide element located in a cell whose density is over a threshold. This threshold value depends on the desired level of hair fuzziness.

Although this interaction method is extremely simple, it yields convincing results. In practice, it was tested with an accordingly simple, yet robust algorithm for animating the guide-strands: hair is composed of approximately a hundred wisps, each of which being simulated using three guide-strands modeled as chains of rigid links. The latter are animated using a fast and robust but non-accurate method [vO91]. The rendering technique is a hybrid between continuum and wisp-based methods: interpolation between the three guide-strands is used to generate a continuum of hair inside each deformable wisps. The overall method results into interactive hair animations that include

self-interactions as well as self-shadowing, and generate visually convincing hair volume (see Figure 2). Furthermore, with this technique, handling hair self-collisions only requires 2.5% of the whole processing time.



**Figure 2:** Interactive hair self-shadowing processed by accumulating transmittance values through a light-oriented voxel grid [BMC05]. (left) Animated smooth hair; (right) Animated curly hair.

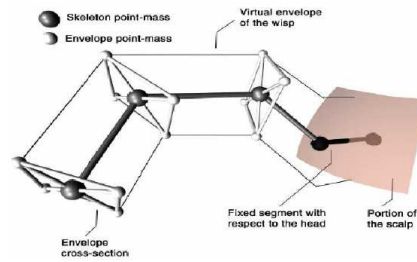
#### 4. Detecting guide-strand interactions

Volumetric methods as the simple solution presented above are not sufficient for generating high quality animation of non-smooth hair: two hair wisps of different orientations may cross each other during motion despite of the volumetric forces they undergo. Most hair animation methods have thus relied on the distance between pairs of guide-strands or on the penetration between wisps of hair defined around them for accurately detecting hair self-interactions. In this chapter, we call these more accurate approaches *guide-strand interactions*.

A naive implementation of guide-strand interactions would lead to  $O(n^2)$  tests, where  $n$  is the total number of guide-strand segments (or wisp segments) in the hair model. Following Plante [PCP01], most methods use a pre-detection based on a regular 3D grid data structure, built around the character and its hair, to quickly get rid of most non-intersecting cases. Each grid cell contains a list of hair-guide elements (or wisp segments) whose bounding box intersects the cell. At each animation step, the grid is used for quickly determining a shorter list of segments susceptible to intersect. A mailbox parameter indicates the last time step when a given pair of such segments has been tested, ensuring that each pair is tested only once. The 3D grid data structure can also be used for optimizing collision detection between hair and the character model: to achieve this, each cell also references the polygons of the character model that intersect it.

##### 4.1. Deformable versus cylindrical hair wisps

To account for the complex interactions observed in real hair during fast motion, Plante *et al.* represented hair using a fixed set of deformable, volumetric wisps [PCP01, PCP02].



**Figure 3:** Elements defining a deformable volumetric wisp [PCP01].

Each wisp is structured into three hierarchical layers: a skeleton curve (called here guide-strand) that defines its large-scale motion and deformation, a deformable volumetric envelope that coats the skeleton and accounts for the deformation due to hair interaction within a wisp, and a given number of hair strands distributed inside the wisp envelope and which are generated only at the rendering stage (see Figure 3). More precisely, the deformable sections that shape a wisp of hair around its guide-strand are animated using 4 1D damped springs, attempting to capture the way a wisp of hair deforms when it moves and most often comes back to its initial size at rest. The wisp volume was defined as a quadratic surface envelop controlled by these cross-sections.

Using such a complex deformable wisp model for the detection of guide-strand interactions proved very time consuming: more than 70% of the simulation time was used in collision detection between hair wisps, despite of the space grid used to accelerate the process. In total, without taking hair rendering into account, about 3 hours of computations were required, in 2001, to compute 3 seconds of animation.

Bertails *et al.* [BKCN03] introduced an adaptive animation control structure, called the *Adaptive Wisp Tree* (AWT), which enables the dynamic splitting and merging of hair wisps. The AWT depends on a full hierarchical structure for the hair, which can either be precomputed - for instance using a hierarchical hairstyle [KN02] - or computed on the fly. The AWT represents at each time step the wisps segments (or guide-strand segments) of the hierarchy that are actually simulated (called *active segments*). Considering that hair should always be more refined near the tips than near the roots, the AWT dynamically splits or merges hair wisps while always preserving a tree-like structure, in which the root coincides with the hair roots and the leaves stand for the hair tips.

In addition to limiting the number of active hair-wisp segments, one of the key benefits of the AWT for collision detection is that the splitting behavior of the wisps models their deformation: there is no need for the complex deformable wisp geometry used in [PCP01]. For collision processing,

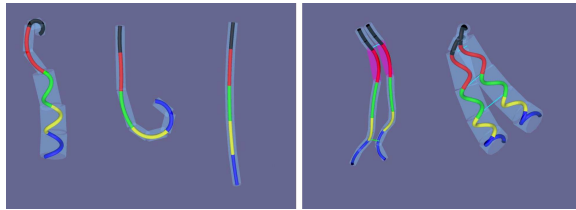


active wisp segments of the AWT are thus represented by cylinders, which greatly simplifies collision detection tests: detecting interactions simplifies into detecting the local minima of the distance between guide-strand and comparing its value to the sum of the wisp radii. With this method, ten seconds of animations could be computed, in 2003, in less than five minutes.

#### 4.2. Handling curly hair and exploiting temporal coherence

The Super-Helix model that was recently introduced [BAC\*06] is the first model that accurately simulates the dynamics of curly hair: unlike previous approaches, curly hair wisps are not modeled using a straight mass-spring skeleton around which wavy strands are drawn at the rendering stage, but are instead accurately modeled using wavy to fuzzy guide-strands, which have a piece-wise helical shape. Detecting interactions between such complex helical guide-strands is indeed more costly.

To handle collisions between hair clumps guided by Super-Helices in a both accurate and efficient way, our strategy is based on the two following ideas: 1) the use of *adaptive* cylindrical bounding envelopes around each hair wisp, whose number and size can automatically adapt during motion, depending on the geometry of the wisp, and 2) the *tracking of the closest points* between the skeletons (*i.e.*, the principal axes) of the bounding cylinders.



**Figure 4:** Left: The three different adaptive representations for the bounding volume of a wisp segment. Right: Tracking the pairs of closest points between the skeletons of guide volumes (for smooth and curly hair) [Ber06].

1. **Adaptive bounding envelopes:** the bounding volume of a helical element  $Q_i$  of the guide hair strand is composed of a single, large cylinder if the helix's spires are tight enough. In other cases (*i.e.* for straighter strands), we use one or two cylinders, oriented along the mean local tangent of the element, to approximate the volume of the wisp (see Figure 4).
2. **Tracking pairs of the closest points:** we adapted the algorithm of Raghupathi *et al.*, originally designed for detecting self-collisions in long and thin deformable objects [RCFC03], to the collision detection between guide hair volumes. Since guide hair volumes are composed of

a set of cylinders, the method amounts to computing minimal distances between pairs of segments (the principal axes of the cylinders), as in [RCFC03]. For each pair of guide-strands, we first initialize a closest point pair near the root. At each time step, each closest point pair is updated by letting the closest points slide along the associated wisp, from the positions they had at the last time step. They stop in a location that locally minimizes the distance between the two wisp volumes. When this distance is under a threshold, new pairs of points are created at both sides of the initial pair, to track the possible multiple local minima. When two closest point pairs slide to the same location, they are merged together. At each time step, because of temporal coherence, only very few of these pairs need to be moved, so advancing them is very fast. Each time the distance between two guide volumes is locally smaller than the sum of their radii, collision is detected.

This algorithm ensures that at least one pair of closest points is maintained between two guide volumes, while keeping the number of tracked pairs between guide volumes low (merging occurs when two different pairs slide towards the same place). The algorithm has thus a  $n^2$  complexity where  $n$  is the number of guide hair strands composing the hairstyle instead of the total number of segments composing hair, as it would be when using a naive algorithm.

The same adaptive wisp volumes and temporal coherence technique are used for detecting collisions between the hair and the body of the character. Distance tests are computed between segments and spheres, as the body is approximated by a unions of spheres. Using this technique, we obtained a total frame rate of only 3 seconds per frame for a dynamic hair style composed of a hundred of guide hair strands, including self-interactions and interactions with the body.

## 5. Response to guide-strand interactions

As already mentioned hair is a very soft and light material. Seen as a whole, it deforms rather than bouncing when it collides with a relatively rigid obstacle such as the character's body. Indeed, hair self-collisions should be very soft as well, and result into frictional rather than bouncing behaviors. Therefore, response to guide-strands interactions have been modeled using soft penalty forces together with friction forces.

### 5.1. Anisotropic response in wisp-based methods

As noted by Plante *et al.* [PCP01, PCP02], accounting for collisions between hair wisps is fairly different from modelling collisions between standard deformable bodies. Wisps are highly anisotropic, since they are just a virtual representation for a group of hair strands. While two perpendicular colliding wisps should be compressed in order to avoid intersection, interpenetration can be allowed between neigh-

bouring wisps moving roughly in the same plane. In consequence, the authors proposed an anisotropic model for the interactions between hair wisps: Wisps of similar orientations are mostly submitted to viscous friction and penetrate each other, whereas wisps of different orientations actually collide in a very dissipative way.



**Figure 5:** The layered wisp model [PCP01] (right) captures both continuities and discontinuities observed in real long hair motion (left).

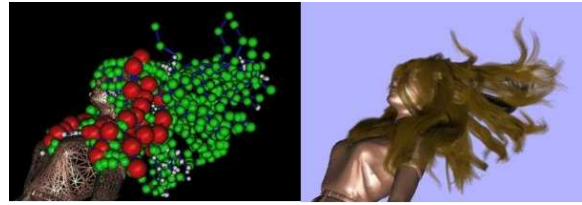
As illustrated in Figure 5, this approach yields convincing results, even for fast motions: the model adequately captures the discontinuities that can be observed in long, thick hair, preserves hair volume and prevents crossing between hair wisps. Nevertheless, the high number of contacts that needed to be computed between the different wisps at rest caused some noticeable artifacts such as instabilities when hair comes to rest.

The previous anisotropic collision response model was re-used and improved by the Adaptive Wisp Tree (AWT) method [BKCNO3]: an AWT implicitly models some of the mutual hair interactions, since neighboring wisps with similar motions merge, thus efficiently yet robustly mimicking the static friction in real hair. This merging behavior also avoids subsequent collision processing between these wisps, thus increasing efficiency as well as gaining stability from the reduced number of primitives. Typically, an AWT simulation starts with a reduced number of hair wisps. While the character moves, these wisps refine where and when needed (see Figure 6), to merge again as soon as they can. When the character is back at rest, the simulation eventually ends up a single large hair wisps. This totally avoids the local instabilities noted in previous approaches.

## 5.2. Setting up realistic penalty and friction forces

The recent work on Super-Helices tackled the problem of setting up more accurate response forces between interacting guide-strands [BAC\*06]. Interactions between guide-hairs, and between hair and external objects (such as the body) are performed through penalty forces which include a normal elastic response together with a tangential friction force.

As in [Dur04], the normal penalty force is stabilized thanks to a quadratic regularization for small penetrations.

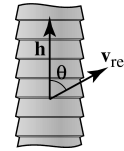


**Figure 6:** Left: The structure of an AWT at a given animation step. Most of the parent wisps (in red) have split into medium-size wisps (in green), which eventually have split into small ones (in white). Right: Rendering of the same frame [BKCNO3].

From a regularization depth  $\delta_{reg}$  (arbitrarily chosen), the normal reaction force  $\mathbf{R}_N$  exerted between the two closest points of interacting guide-strands is computed as follows:

$$\begin{cases} \text{if } (gap \leq 0) & \mathbf{R}_N = \mathbf{0} \\ \text{if } (0 \leq gap \leq \delta_{reg}) & \mathbf{R}_N = \frac{k_c gap^2}{2\delta_{reg}} \mathbf{n}_c \\ \text{else} & \mathbf{R}_N = k_c (gap - \frac{\delta_{reg}}{2}) \mathbf{n}_c \end{cases}$$

where  $\mathbf{n}_c$  is the unitary vector giving the direction of collision (calculated as the cross product of the vectors defining the two closest segments), and  $k_c$  an arbitrary constant value.



**Figure 7:** Angle  $\theta$  between the fiber orientation and its relative velocity w.r.t the external object in contact with the fiber.

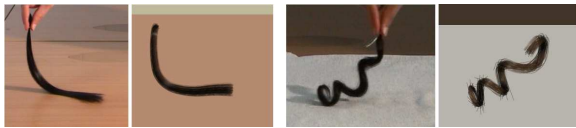
To simulated friction between wisps in contact or friction with an obstacle, the method extends viscous friction law in [CK05], defined as :

$$\mathbf{R}_T = -\nu (\mathbf{v}_{rel} - (\mathbf{v}_{rel} \cdot \mathbf{n}_c) \mathbf{n}_c)$$

To account for the oriented scales covering individual hair fibers, the friction coefficient  $\nu$  is multiplied by a sine function to account for the orientation of hair fibers with respect to their sliding motion over the external object:  $\nu = \nu_0 (1 + \sin(\theta/2))$ , where angle  $\theta$  is defined in Figure 7.

The parameters of interaction forces, as well as the other parameters of the Super-Helices model, can be set up using the actual study of real wisps of hair: The friction parameter  $\nu_0$  between hair and a given material is directly adjusted from real observations of sliding contacts between the hair clump and the material.

As Figures 8 and 9 show, the Super-Helices model results in realistic simulations which can be compared side by side with videos of real hair in motion.



**Figure 8:** Validation of the friction model in [BAC\*06] on a sliding motion of a smooth (left) and curly (right) hair clump over different kinds of material (left: wooden surface, right: cotton).



**Figure 9:** Comparison between a real full head of hair and the model based on interacting Super-Helices [BAC\*06], on a head shaking motion (straight and clumpy hair type).

## 6. Conclusion

As we have shown, processing hair interactions requires a dedicated set of methods, due to the very specific nature of the hair material. Impressive advances were made in the last six years, from the first models able to handle hair self-collisions to efficient, robust and even partly validated methods. This chapter has detailed several specific solutions that range from the use of a volumetric approach when a very quick solution is required to realistic models that still keep the computational load to an acceptable rate.

In spite of all these advances, there still remains very challenging issues in the modeling of hair self-interactions: these interactions are indeed the origin of the complex collective behavior of hair. Especially they cause hair to group into clusters during motion; this phenomenon has never been accounted before (except in very simplified models, such as the AWT), as previous models usually assume that hair granularity is fixed by the number of simulated guide-strands. Moreover, hair interactions vary a lot according to external conditions such as moisture (wet hair being the extreme case), combing, or the use of cosmetic products. Lastly, hair triboelectricity has never been modelled in an accurate way.

Future research should include attempts to make volumetric methods such as the one presented in section 3 more accurate at low cost, by taking local hair directional distribution into account while setting up the response force. The approaches that seek for realism should probably extract the internal damping inside a hair wisp from the preliminary study of hair chunks actually modeled using a full set of interacting hair strands. This study should also bring more accurate criteria for splitting a wisp into sub-wisps or merging them, and could help characterizing the number of hair guides re-

quired according to the natural curliness and smoothness of a given hair type.

## Acknowledgments

The authors would like to thank all the collaborators who have worked with them on hair animation in the past few years, and more specifically Eric Plante, Pierre Poulin, Tae-Yong Kim, Ulrich Neumann, Clément Ménéier, Basile Audoly, Bernard Querleux, Jean-Luc Lévêque and Frédéric Leroy.

## References

- [AUK92] ANJO K., USAMI Y., KURIHARA T.: A simple method for extracting the natural beauty of hair. In *Proceedings of ACM SIGGRAPH 1992* (August 1992), Computer Graphics Proceedings, Annual Conference Series, pp. 111–120.
- [BAC\*06] BERTAILS F., AUDOLY B., CANI M.-P., QUERLEUX B., LEROY F., LÉVÊQUE J.-L.: Super-helices for predicting the dynamics of natural hair. In *ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference)* (August 2006).
- [BCN03] BANDO Y., CHEN B.-Y., NISHITA T.: Animating hair with loosely connected particles. *Computer Graphics Forum* 22, 3 (2003), 411–418. Proceedings of Eurographics'03.
- [Ber06] BERTAILS F.: *Simulation de chevelures naturelles*. PhD thesis, Institut National Polytechnique de Grenoble, 2006.
- [BKCN03] BERTAILS F., KIM T.-Y., CANI M.-P., NEUMANN U.: Adaptive wisp tree - a multiresolution control structure for simulating dynamic clustering in hair motion. In *ACM SIGGRAPH Symposium on Computer Animation* (July 2003), pp. 207–213.
- [BMC05] BERTAILS F., MÉNIER C., CANI M.-P.: A practical self-shadowing algorithm for interactive hair animation. In *Graphics Interface* (May 2005). Graphics Interface'05.
- [CCK05] CHOE B., CHOI M., KO H.-S.: Simulating complex hair with robust collision handling. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), pp. 153–160.
- [CJY02] CHANG J. T., JIN J., YU Y.: A practical model for hair mutual interactions. In *ACM SIGGRAPH Symposium on Computer Animation* (July 2002), pp. 73–80.
- [CK05] CHOE B., KO H.-S.: A statistical wisp model and pseudo-physical approaches for interactive hairstyle generation. *IEEE Transactions on Visualization and Computer Graphics* 11, 2 (March 2005).
- [CSDI99] CHEN L., SAEYOR S., DOHI H., ISHIZUKA M.: A system of 3d hairstyle synthesis based on the wisp model. *The Visual Computer* 15, 4 (1999), 159–170.
- [DKT93] DALDEGAN A., THALMANN N. M., KURIHARA T., THALMANN D.: An integrated system for modeling, animating and rendering hair. *Computer Graphics Forum* 12, 3 (1993), 211–221.
- [Dur04] DURVILLE D.: Modelling of contact-friction interactions in entangled fibrous materials. In *Procs of the Sixth World Congress on Computational Mechanics (WCCM VI)* (September 2004).

- [HMT01] HADAP S., MAGNENAT-THALMANN N.: Modeling dynamic hair as a continuum. *Computer Graphics Forum* 20, 3 (2001), 329–338. Proceedings of Eurographics'01.
- [KH01] KOH C., HUANG Z.: A simple physics model to animate human hair modeled in 2D strips in real time. In *Computer Animation and Simulation '01* (Sept. 2001), pp. 127–138.
- [KN00] KIM T.-Y., NEUMANN U.: A thin shell volume for modeling human hair. In *Computer Animation 2000* (2000), IEEE Computer Society, pp. 121–128.
- [KN02] KIM T.-Y., NEUMANN U.: Interactive multiresolution hair modeling and editing. *ACM Transactions on Graphics* 21, 3 (July 2002), 620–629. Proceedings of ACM SIGGRAPH 2002.
- [LK01] LEE D.-W., KO H.-S.: Natural hairstyle modeling and animation. *Graphical Models* 63, 2 (March 2001), 67–85.
- [PCP01] PLANTE E., CANI M.-P., POULIN P.: A layered wisp model for simulating interactions inside long hair. In *EG workshop of Animation and Simulation* (sep 2001).
- [PCP02] PLANTE E., CANI M.-P., POULIN P.: Capturing the complexity of hair motion. *Graphical Models (Academic press)* 64, 1 (january 2002), 40–58.
- [RCFC03] RAGHUPATHI L., CANTIN V., FAURE F., CANI M.-P.: Real-time simulation of self-collisions for virtual intestinal surgery. In *Proceedings of the International Symposium on Surgery Simulation and Soft Tissue Modeling* (2003), Ayache N., Delingette H., (Eds.), no. 2673 in Lecture Notes in Computer Science, Springer-Verlag, pp. 15–26.
- [RCT91] ROSENBLUM R., CARLSON W., TRIPP E.: Simulating the structure and dynamics of human hair: Modeling, rendering, and animation. *The Journal of Visualization and Computer Animation* 2, 4 (1991), 141–148.
- [Rob94] ROBBINS C. R.: *Chemical and Physical Behavior of Human Hair*, third ed. Springer-Verlag, New York, 1994.
- [vO91] VAN OVERVELD K.: An iterative approach to dynamic simulation of 3-d rigid-body motions for real-time interactive computer animation. *The Visual Computer* 7 (– 1991), 29–38.
- [WFK\*06] WARD K., F.BERTAILS, KIM T.-Y., MARSCHNER S., CANI M.-P., LIN M.: A survey on hair modeling: Styling, simulation and rendering. *IEEE Transaction on Visualization and Computer Graphics* (2006). To appear.
- [WL03] WARD K., LIN M. C.: Adaptive grouping and subdivision for simulating hair dynamics. In *Pacific Graphics Conference on Computer Graphics and Applications* (October 2003), pp. 234–243.
- [Zvi86] ZVIAK C.: *The Science of Hair Care*. Marcel Dekker, 1986.

# Collision Detection on Deformable Surfaces

Pascal Volino and Nadia Magnenat-Thalmann

MIRALab, University of Geneva - CH-1211 Geneva, Switzerland

## Abstract

*Deformable objects cover a large range of applications in computer graphics and simulation, ranking from modeling techniques of curved shapes to mechanical simulation of cloth or soft volumes. Efficient collision detection is used in all these applications for ensuring consistent design and simulation.*

## 1. Collision Detection Strategies

Unlike rigid bodies, deformable objects have evolving shapes. In most cases, this implies that their surface are curved. Adequate modeling techniques are needed to describe these objects. Among the most popular, polygonal meshes and implicit surfaces (for example metaballs) are used. Whereas surfaces are usually described as polygonal meshes (usually flat polygons such as triangles or quadrangles, but possibly also curved patches such as Bezier patches or subdivision surfaces), volumes are most of the time also described by their bounding surfaces. Collision detection is usually performed on these surfaces.

The usual complexity of collision detection processes result from the large number of primitives that describe these surfaces. Most of collision detection applications need to compute which polygons of large meshes do actually collide. In most of the cases also, these meshes are animated (through user interaction or simulation processes) and collision detection has to be involved at each steps of these animations for ensuring immediate and continuous feedback to the animation control.

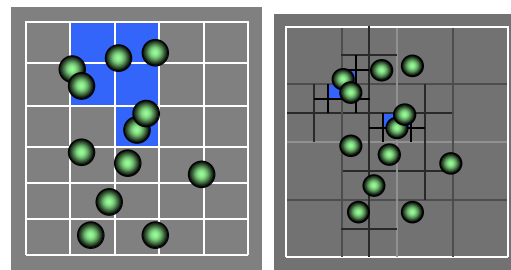
This creates a particular context that collision detection has to take advantage for optimal performance.

### 1.1. Hierarchy Structure

Most of efficient collision detection algorithms take advantage of a hierarchical decomposition of the complex scheme. This allows to avoid the quadratic time of testing extensively collisions between all possible couples of primitives.

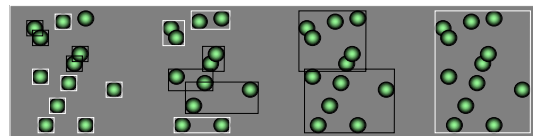
There are two major ways of constructing such hierarchies:

- \* Space subdivision schemes, where the space is divided in a hierarchical structure. These are typically octree methods. Using such structure, a reduced number geographical neighbors of a given primitive are found in  $\log(n)$  time (the depth of a hierarchy separating geographically  $n$  primitives) and tested for collisions against it.



**Figure 1:** Space subdivision methods, flat (left) or hierarchical (right) rely on a partition of space into regions containing primitives.

- \* Object subdivision schemes, where the primitives of the object are grouped in a hierarchical structure. These are typically methods based on bounding volume hierarchies. Using such structure, large bunches of primitives may be discarded in  $\log(n)$  time (the depth of a well-constructed hierarchy tree of  $n$  primitives) through simple techniques such as bounding-volume evaluations.



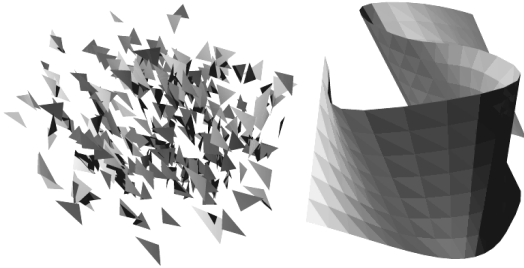
**Figure 2:** Object subdivision methods rely on a subdivision of the scene into groups of primitives.

Other methods, such as voxel methods, projection methods or render overlap methods, usually belong to non-hierarchical space subdivision schemes.

Any of these methods may be used in the context of deformable objects. However, maximum efficiency is obtained by taking advantage of all invariants that could reduce the amount of time spent in computation. In the context of deformable surfaces which use extensively discretized surface animations (polygonal meshes or patches animated through motion of their control points), the major invariant to take advantage of is *the local invariance of the mesh topology*.

Unlike polygon soups, where the primitives are totally independent one from another, the primitives of a polygonal

mesh maintain a constant adjacency structure which defines, in a local state, some constant geographic properties between these primitives. Hence, adjacent elements of a polygonal mesh have similar positions whatever the motion of the surface, during all the animation.



**Figure 3:** A polygon soup, and a polygonal mesh which maintains local position constancy.

Considering that the topology of the mesh defines a constant native geographical neighborhood of the primitives of the mesh (which are usually non-colliding primitives), a good idea is to define the collision detection hierarchy based on this criteria. On the other hand, any topologically unrelated primitives ("far away" from each other on the mesh topology) which come to be geographically close to each other are very likely to be colliding.

This is why object-based hierarchies are very likely to be the best paradigm for defining an efficient collision detection algorithm for animated meshes. Compared to space subdivision methods, the main benefits are the following:

- \* In most cases, they can work on a constant hierarchy structure, which does not need to be updated along the animation.
- \* The topology of the hierarchy reflects the adjacency of the surface regions described in it. Adjacency information may thus be used in various optimizations, such as the approach for optimizing self-collision detection, as discussed later.

## 2. Bounding Volume Hierarchies

Hierarchical collision detection heavily relies on bounding volumes. Performance depends on:

- \* How tight they enclose the object part corresponding to their hierarchy level, for avoiding at best false positives in the collision detection tests.
- \* How efficiently they can be computed from a primitive.
- \* How efficiently they can be combined, for propagation up in the hierarchy.
- \* How efficiently they can be updated if the object is animated.
- \* How efficiently a collision test can be performed between two volumes.

The adequate bounding volume is chosen so as to offer the best compromise between these factors, and this is quite dependent on the simulation context (kind and number of object primitives, kind of animation...).

Among popular choices,

- \* Bounding spheres or ellipsoids.
- \* Bounding boxes or Discrete Orientation Polytopes.

### 2.1. Choosing the Suitable Bounding Volume Scheme

The major choice is to be set between three types of bounding volumes:

- \* Axis-independent volumes, such as spheres.
- \* Axis-aligned volumes which are defined in absolute world coordinates.
- \* Object-oriented volumes which are defined in local object coordinates.

This choice is particularly important when working with animated objects. In that context, the most important issue is to reduce the time taken for updating the bounding volume hierarchy for each step of the animation.

#### 2.1.1. Object-Oriented Volumes

The first idea is to attempt to skip this recomputation whenever possible. This is possible when working with rigid objects, which only have rigid motion in the scene (translation and rotation). In this case, rather than defining the volumes in world coordinates, attaching them to object coordinates removes the need of updating them.

Another motivation for using axis-oriented volume is the optimization of the bounding tightness. Hence, if the object is flat or elongated, there are large benefits in using an adequately aligned volume that fits the shape tightly.

The difficulty is however moved to another place in the collision detection process: Combining and comparing bounding volumes defined in different coordinate systems. While the combination process is usually performed once for all in the hierarchy describing rigid objects, testing for collisions between bounding volumes require coordinate transformation computations that may take significant computation resources. Some schemes also expand the volumes in order to compensate the change of axis rotation, at the expense of bounding tightness. This difficulty can also be avoided by using axis-independent volumes (spheres), but these are very inefficient in bounding tightness (specially for flat or elongated objects).

#### 2.1.2. Axis-Aligned Volumes

The other idea is to make perform extensively the recomputation of the bounding volume hierarchy at each position change, with operations on bounding volumes made as simple and efficient as possible. This is actually the best approach for deformable objects, as there is no way to efficiently exploit shape invariance.

The most popular choice for this are axis-aligned bounding boxes. They are essentially defined by the minimum and maximum coordinates of the enclosed objects, which can be computed very easily. Combining bounding boxes is also trivial, and collision test between two boxes is simply evaluated by testing min-max overlap along all coordinates.

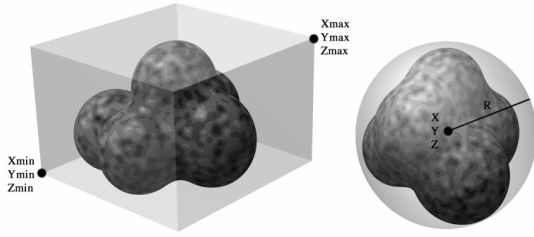


Figure 4: A bounding box and a bounding sphere.

One of the biggest issues with axis-aligned bounding boxes is that they do not always offer good bounding tightness, particularly when objects are flat or elongated along a diagonal direction. Of course, reverting to their object-oriented variant would void the benefits of fast collision tests. The other solution is to use a generalization of bounding boxes along more directions than the native axes alone.

## 2.2. Beyond Bounding Boxes: Discrete Orientation Polytopes

Bounding boxes are based on min-max interval computations on the directions defined by the natural world coordinate axes. While this is sufficient for defining bounding volumes, why not adding more directions? It's like "cutting away" a bounding volume along particular directions so as to obtain tighter bounding volumes.

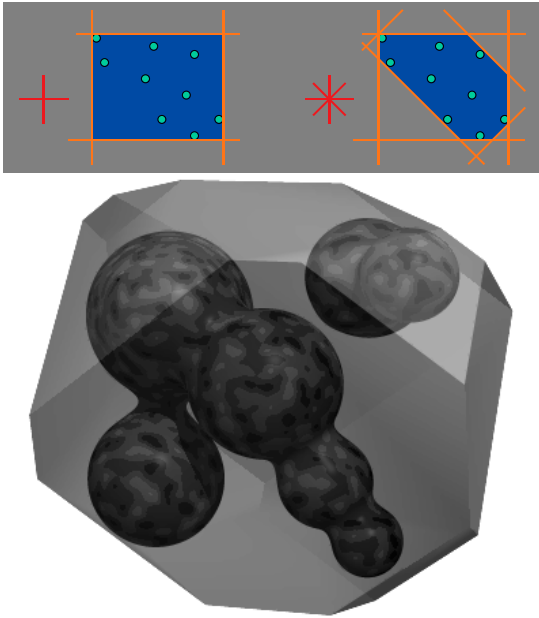


Figure 5: Discrete Orientation Polytopes are generalizations of bounding boxes along arbitrary directions.

Mathematically, given a set of directions  $\mathbf{D}_i$ , a Discrete Orientation Polytope (DOP) a set of vertices  $\mathbf{V}_k$  is defined by two vectors  $\mathbf{M}_i$  and  $\mathbf{N}_i$  such as:

$$\mathbf{M}_i = \min_k(\mathbf{D}_i \cdot \mathbf{V}_k) \quad \text{and} \quad \mathbf{N}_i = \max_k(\mathbf{D}_i \cdot \mathbf{V}_k)$$

The union of two DOPs is computed as follows:

$$\mathbf{M}_i = \min(\mathbf{M}_{i_1}, \mathbf{M}_{i_2}) \quad \text{and} \quad \mathbf{N}_i = \max(\mathbf{N}_{i_1}, \mathbf{N}_{i_2})$$

Two DOPs do not intersect if the following condition is met for at least one value of  $i$ :

$$\mathbf{M}_{i_1} > \mathbf{N}_{i_2} \quad \text{or} \quad \mathbf{M}_{i_2} > \mathbf{N}_{i_1}$$

The adequate set of directions to be considered should describe a sampling of the direction space as uniform as possible. Of course, this sampling is point-symmetric, since a direction vector also represents its opposite direction.

In 3D, the easiest approach is to construct a set of directions starting from the cube (standard bounding box, which is a DOP of 6 directions):

$$\mathbf{D}_0=(1,0,0) \quad \mathbf{D}_1=(0,1,0) \quad \mathbf{D}_3=(0,0,1)$$

... and add the cube diagonals (directions to its vertices):

$$\mathbf{D}_4=(\sqrt{3},\sqrt{3},\sqrt{3}) \quad \mathbf{D}_5=(\sqrt{3},-\sqrt{3},-\sqrt{3}) \quad \mathbf{D}_6=(-\sqrt{3},\sqrt{3},-\sqrt{3}) \\ \mathbf{D}_7=(-\sqrt{3},-\sqrt{3},\sqrt{3})$$

... for obtaining a DOP describing 14 directions. 12 additional directions can be added by using the square diagonals (directions to its edge centers):

$$\mathbf{D}_8=(0,\sqrt{2},\sqrt{2}) \quad \mathbf{D}_9=(0,\sqrt{2},-\sqrt{2}) \quad \mathbf{D}_{10}=(\sqrt{2},0,\sqrt{2}) \\ \mathbf{D}_{11}=(\sqrt{2},0,-\sqrt{2})$$

Note that for this set, it is not actually necessary to normalize the direction vectors, relieving the necessity of performing multiplications for computing a DOP enclosing a set of vertices. However, normalized direction vectors offer good evaluation of the size of the DOP through the min-max interval width in each direction, as well as a simple expansion scheme for distance-based collision detection.

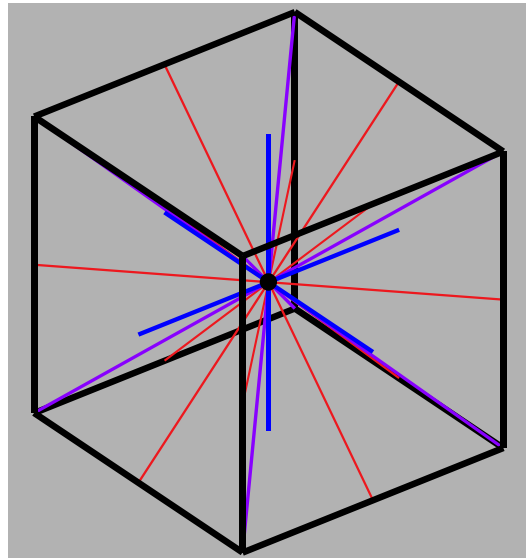


Figure 6: The 26 direction set defined by a cube.

When more directions are needed, it is also possible to construct a set from a dodecahedron (12 directions), adding to it directions to its vertices (20 additional directions) and

to its edge centers (30 additional directions). The benefit of this set is a better distribution of directions. However, multiplications cannot be avoided for computing the DOP of a set of vertices.

Ultimately, with a huge number of directions, DOP tend to get the shape of the convex hull of the enclosed objects.

The choice to be made is actually to find the optimal number of directions defining the DOP. In one hand, the more directions, the tighter the DOP fits to the convex hull of the object (or to the object itself if it is convex). In the other hand, the more directions, the more computation is needed for computing the DOPs and evaluating their intersections. The best choice is actually dependent on the shape of the objects (flat or elongated objects will get more benefits from tight DOPs than round or concave ones), and also the extra cost of performing explicit collision detection on object marked as colliding by rough quickly-evaluated DOPs (for instance, it takes much more time evaluating collisions between curved patches than flat polygons). The only way of finding the best compromise is to carry out experimental benchmarks on various examples typical of the wanted simulation context.

### 3. Collision Detection on Polygonal Meshes

Polygonal meshes are the most popular way of describing deformable objects. They may either represent the surface object itself (for example, cloth), or the boundary of a volume object. Collision detection is usually carried out by finding which of the polygons of the meshes are actually colliding.



Figure 7: Objects animated and simulated as polygonal meshes.

Given the considerations discussed above, the typical best choices for detecting collisions on animated polygonal meshes are the follows:

- \* Use of a hierarchical bounding volume scheme constructed on the objects.
- \* Use of efficient axis-aligned bounding volumes, such as DOPs.

Choosing a constant hierarchy built on the mesh seems to be quite a good assumption, as for typical objects, the distance on the mesh is quite a good evaluation of the native distance that should separate features of the mesh, and bounding volumes enclosing a small region of the mesh are likely to remain small for any usual deformation.

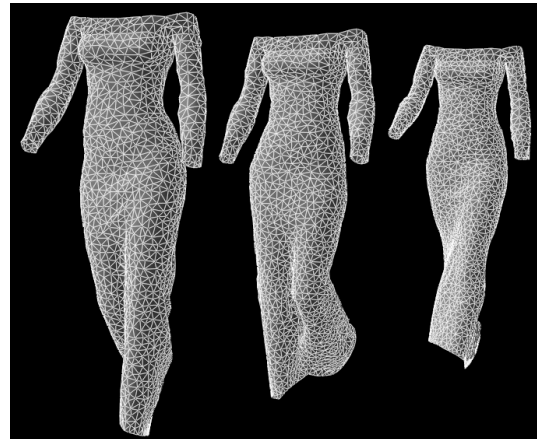


Figure 8: Moving polygonal meshes describing the cloth surface.

#### 1.2. Building Mesh Hierarchies

The performance of the collision detection algorithm is highly dependent on how well-constructed the hierarchy tree is.

The tree should be well conditioned, meaning:

- \* Each node should have at maximum  $O(1)$  children.
- \* The tree should be balanced, so the tree should have  $O(\log n)$  maximum depth.

Another essential quality of the tree is to offer minimal bounding volumes for each tree node. This means that the surface elements represented by a tree node should have maximum vicinity. Thus, it is important to build the hierarchy consistently to the topology of the mesh, which is a good evaluation on the relative positioning of the mesh elements.

One efficient solution is to build the hierarchy tree using an ascending process. First the leaf nodes of the tree are constructed, corresponding to all the individual polygons of the surface. Then, an upper layer of the hierarchy is built by grouping two or three nodes in a common parent node. The tree is built level by level until only one element remains, which is the root node of the tree.

The grouping can be performed by a region-merging algorithm. Initially, each surface polygon is assigned a unique region ID, which identifies a group of polygons. During the grouping process, candidate edges that separate two different groups (two polygons having different ID) are considered. One of these edges is then selected, and all the polygons of one group (usually the smallest) are assigned the ID of the polygons of the other group. This algorithm generates groups of connected regions, within each of which all the polygons are connected by at least one edge. A counter should also be included in each group, to keep track of the number of subgroups that have been merged in the group. It can be used to limit the number children a group has. This algorithm is in fact very close to automatic labyrinth-generation algorithms which are based on the same region-merging scheme.

It is important to determine efficiently which nodes to group in order to create the parent node. First, only adjacent



surface regions should be connected. Secondly, the regions generated should be as “well shaped” as possible, i.e. closer to a disk than a elongated or sprawling branched structure. The resulting bounding boxes will therefore be as compact as possible, whatever the result of any reasonable 3D deformation of the surface.

A good way to characterize “well-shaped” polygons is to compare its contour length to its surface area. The algorithm can construct groups by maximizing the “shape factor”  $\sqrt{\text{SurfaceArea}} / \text{ContourLength}$ . At a given hierarchy level, this ratio is computed for any group that can be generated by the removal of an edge. The potential groups are then sorted by this criteria, and new groups are then constructed wherever possible, according to this sorting. Between two hierarchy levels, the algorithm first tries to merge groups into pairs, and then merges the remaining groups into these new groups.

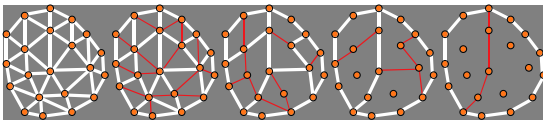


Figure 9: Building a mesh hierarchy

Though this proposed grouping process does not necessarily yield hierarchy regions that globally optimize the shape factors, the results obtained are however quite acceptable for our application. More precisely, the self-collision detection algorithm remains efficient as long as the bounding volumes of non-adjacent hierarchy groups (that do not share at least a common vertex) do not intersect. Using the proposed algorithm, this feature is verified almost everywhere in usual polygonal meshes.

This algorithm should be implemented to build a hierarchy on any polygonal mesh that will be involved in collision detection. This computation should only be performed as preprocessing, and does not have to be performed for each collision detection when the surfaces have moved.

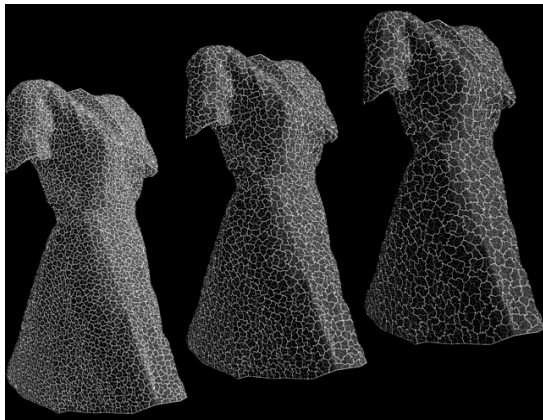


Figure 10: Hierarchisation of a 50 000 triangle mesh: Levels 5 to 7.

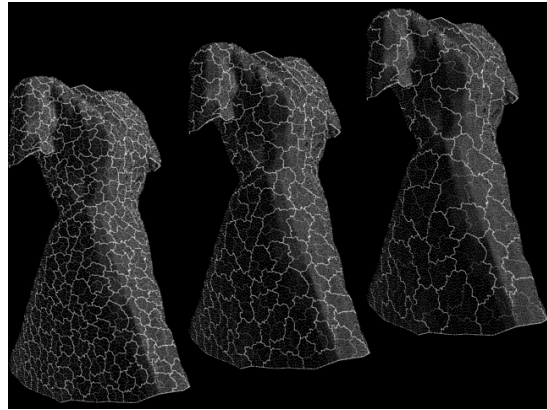


Figure 11: Hierarchisation of a 50 000 triangle mesh: Levels 8 to 10.

### 3. Self-Collision Detection

There are very few algorithmic differences between self-collision detection within one object and collision detection between two separate objects. Ultimately, a hierarchical algorithm will end up splitting a single object into separate pieces, and perform usual collision detection between these pieces.

There is actually a major performance issue related to self-collision detection, related to adjacent elements actually seen as “colliding” by usual bounding-volume tests.

#### 3.1. Why Self-Collision Detection is so Inefficient

Self-collision detection pertains to collisions between elements of the same surface. Of course, neighboring elements of the same surface are naturally in contact to each other. Any collision detection algorithm is designed to detect geometric contact between elements, and thus will be misled by these adjacent elements, and will consider them as colliding elements.

The number of adjacencies is usually proportional to the total number of elements in one surface. In a triangular mesh, the adjacency number is roughly 1.5 times the total number of triangles for common-edge adjacencies and 6 times the total number of triangles for common-vertex adjacencies.

The time spent detecting collisions is proportional to the number of colliding elements multiplied by the logarithm of the total number of elements. Typically, the number of self-colliding elements is very small compared to the total number of elements and often null if no collisions occur. Thus, “detecting” all the adjacencies as if they were collisions is a great waste of time, particularly in situations with very few collisions.

For efficiency, an algorithm should be designed to ignore all collisions that in fact are only element adjacencies.

#### 3.2. Optimizing Self-Collision Detection with Curvature Evaluation

The curvature criteria is expressed as follows:

A flat surface cannot have self-collisions. On the other hand, if a surface has self-collisions, it must be bent enough to form a loop.

Is there a way to formalize this intuition? Curvature appears to be the key for achieving efficiency in self-collision detection.

When there are self-collisions on a surface, at any geometrical intersection point, the surface appears twice. For obtaining such configuration, the surface has to be bent enough to form a loop.



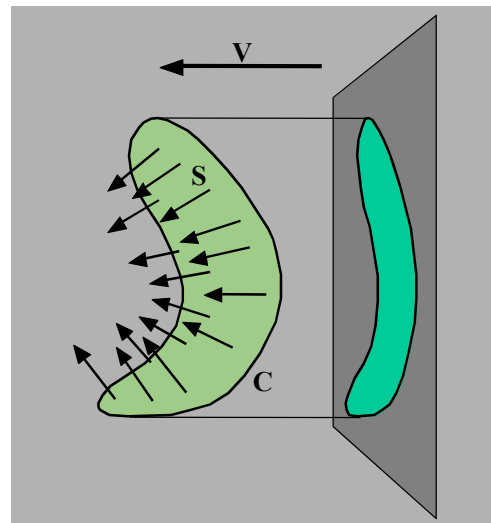
**Figure 12:** Self-collisions only occur if the surface is bent enough for creating a loop.

This condition cannot be met if there exists a direction for which the orthogonal projection of the surface does not exhibit folds. If the surface is continuous, this means that all the normals of the surface have a dot product of constant sign with that direction vector.

We also have to consider self-collisions that may occur because of the shape of the surface boundary. This may also exhibit loops even if the surface is almost flat, when the surface contour itself exhibits a loop causing self-intersection. An additional test has therefore to be done on the surface contour. Having found a projection direction for the surface curvature, a sufficient criteria for non-intersection is that the projection of the surface contour along this direction should not self-intersect.

Thus, a surface does not have self-collisions if the following criteria are met:

- \* Let **S** be a continuous surface in Euclidean space, delimited by one contour **C**.
- if
- There exists a vector **V** for which  $\mathbf{N} \cdot \mathbf{V} > 0$  at (almost) every point of **S** (**N** being the normal vector of the surface at the considered point)
- and
- The projection of **C** on a plane orthogonal to **V** along the direction of **V** has no self-intersections
- then
- There are no self-collisions on the surface **S**.



**Figure 13:** The curvature criteria.

Such criteria allow us to efficiently discard, from the detection process, “almost flat” surface regions that will not exhibit internal self-collisions. The union of two adjacent surface regions may also be “almost flat” so we need not detect collisions between these two regions. In particular, adjacent elements need not be checked for collisions. Implemented efficiently, this criterion will allow us to deal with the major cause of inefficiency of self-collision detection.

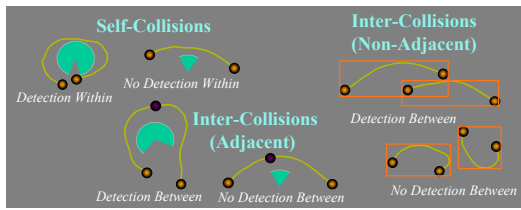
It is now important to adapt the hierarchical collision detection scheme to include this criteria. As discussed, the general hierarchical collision detection algorithm works with bounding boxes. There are two main processes:

- \* For detecting collisions within a surface region, collisions are detected within and between all the children of its corresponding hierarchy tree node.
- \* For detecting collisions between two surface regions, collisions are detected between the respective children nodes.

Collisions between two nodes may be efficiently detected using a bounding-box evaluation: if the bounding boxes do not intersect, there are no intersections. However, in the standard hierarchical algorithm, there are no bounding box techniques for self-collision detection within one node. Replacing the bounding box test by a curvature test can overcome this limitation.

For the self-collision stage, how to integrate curvature considerations is clear: Collisions should be detected within one node only if the corresponding surface does not match the curvature criteria.

When detecting collisions between two nodes, we should consider two cases: Dealing with two adjacent surfaces, collisions should be detected between the nodes only if the corresponding surface union does not verify the curvature criteria (obviously, the bounding boxes will always intersect, so the curvature criteria acts as a good replacement test). Dealing with non adjacent surfaces, the standard bounding box criteria should be used.



**Figure 14:** Using curvature or bounding volumes: The different cases.

Thus, the curvature criteria is incorporated into the hierarchical algorithm by replacing the bounding volume test by a curvature test within surface regions or between adjacent surface regions.

Given a curved surface, we need to determine the existence of, and find, a compatible direction vector that has positive dot product with all normal vectors of the surface. In our discrete model of the surface, that means that this vector should have positive dot product with the normals of all the surface polygons.

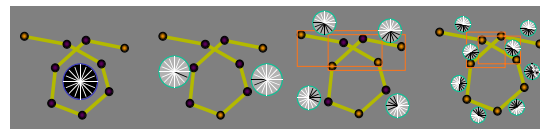
Similar to what was done for the bounding boxes, “direction boxes” that contain the allowable directions will be propagated up to the hierarchical parents.

Assimilating the direction space to a sphere, the allowable direction box of a flat mesh polygon is a half sphere. When we consider two polygons, the globally allowable direction is the common part of the corresponding two sphere halves. This gives us the mechanism that should be propagated upwards in the tree hierarchy. The resulting direction box at the root of the tree may be empty, and in this case no suitable direction can be found for the whole surface. If not empty, any vector within the direction box is suitable.

Unlike volumes boxes that can be represented efficiently using space coordinates, there is no easy way to describe our direction boxes exactly. One solution is to use “direction cones” defined by a direction and an aperture angle. The bounding tightness is however very limited and combining two cones is not an efficient operation.

That difficulty can also be addressed by building a discrete set of direction vectors that will represent our direction space. The same set of sample directions can be used as the ones used for defining the DOPs used as bounding volumes: Using the 26 directions toward the face centers, vertices and edge centers of a cube, the angle accuracy is around 25°. The more directions used, the more accurate are the direction evaluations, at the expense of computation time.

During the update process of the bounding volumes in the hierarchy tree, the direction boxes are updated as well. For each polygon of the mesh is computed the set of direction from the sample set that have positive positive dot product with the normal of the polygon. The result is stored in an array of boolean. Propagation up in the tree is done by combining these arrays with element-wise boolean AND operations. Then for any collision test, the low curvature criteria is met if TRUE elements remains in the array, and the corresponding directions are the directions that meet the dot product requirements for all the corresponding surface region.



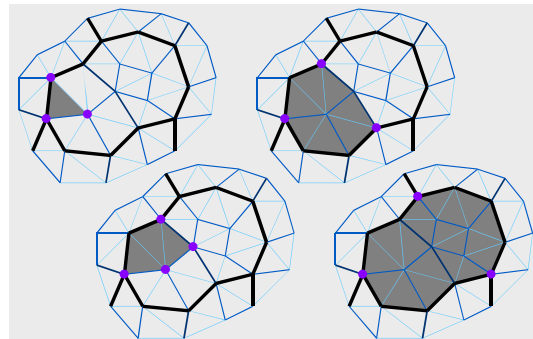
**Figure:** Combining curvature boxes and bounding volumes during the detection process.

Another major problem is the adjacency test: Given two arbitrary nodes in the hierarchy, are the corresponding surfaces adjacent? (Do they have at least one common vertex?)

This represents the major difficulty of our algorithm. This adjacency should be detected efficiently (the computation complexity should be  $O(\log n)$ ), and the extra storage in the tree data structure should be reasonable (the extra information for each node should be  $O(1)$ ).

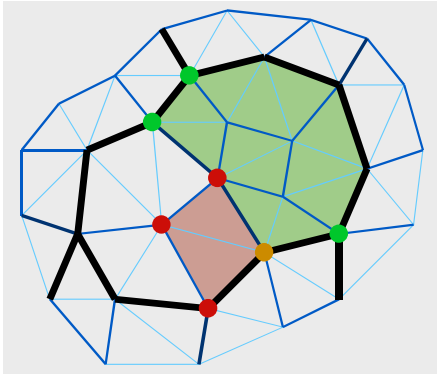
For each hierarchical node, a list of vertices should be defined, these being the vertices surrounding the corresponding surface region. Several circular lists should be considered when dealing with non-connected surfaces or surfaces having holes. Obviously, if two nodes are adjacent, they have at least one of these vertices in common.

Not all the boundary vertices should be stored, but only the vertices that separate two different surface regions among those of the highest hierarchy level that also do not include the region being considered. “Outside” is considered as a particular surface region. Whatever the node level and the total number of polygons surrounding this surface region, the number of stored vertices is approximately constant, and for usual meshes hardly exceeds six. As this adjacency information only depends on the mesh topology, this stage is usually performed once in the preprocessing stage.



**Figure 15:** Storing region boundary vertices in the hierarchy tree: The number remains roughly constant whatever the level in the hierarchy.

Adjacency testing is then performed easily: Two hierarchy nodes are adjacent if and only if they have at least one common vertex among those stored for these two nodes. This test is performed in  $O(1)$ .



**Figure 16:** Two adjacent (point-connected) surface regions of the hierarchy have at least one common stored vertex.

The boundary shape of the surface may also be the cause of self-collisions. In most cases, this happens when an elongated strip is fold so as to produce a cone-shaped surface. Such collisions are also very likely to happen around sharp concavities of the surface contour, where minimal folds can also produce self-collisions.

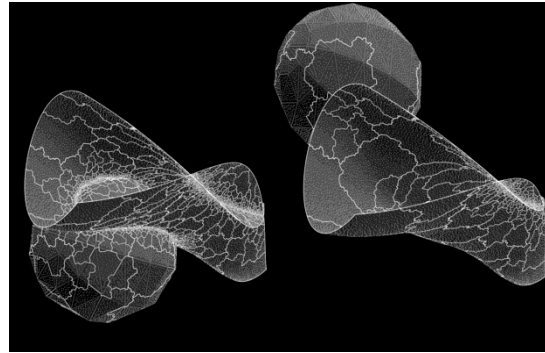
The most systematic solution would be to construct, for any surface region fulfilling the surface normal criteria, the 2D projection of the surface boundary on a plane orthogonal to one of the found directions. Despite flatness, collision detection should be carried out within or between children containing boundary intersections. This 2D collision detection process can also be based on a similar kind of curvature optimization. This would however require a significant amount of extra computation, along the data required for managing a contour-based hierarchies.

Practical test have however shown that these tests are rarely significant in most "real-world" situations involving for example garment simulation or deformation of soft objects. These tests may however improve detection of long objects with large curvature deformations (simulation of long ribbons) or surfaces with complex non-convex contours (cuts, holes).

These marginal situations may be addressed using various low-cost approximate techniques. The simplest method is to "expand" the direction boxes with a certain angle for mesh elements that are located on the boundary of the surface. This angle may also be increased for elements adjacent to non-convex boundary locations. The larger the angle, the most systematic the collision detection is, at the expense of computation time.

### 3.3. Efficiency

The main interest of this algorithm is its efficiency in detecting self-collisions: Hierarchy regions that are not curved enough to contain self-collisions are efficiently omitted from the detection process. The following figure shows the regions considered when performing collision detection between two objects, as well as self-collision detection within these objects also. As a result, the algorithm efficiently focuses on the intersecting parts of the surfaces.

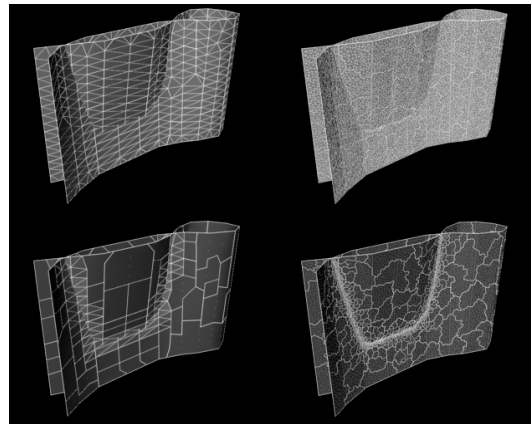


**Figure 17:** Collision detection is focused on the colliding parts of the surface.

The execution time required for performing collision detection is subdivided as follows:

- \* Update of the bounding volumes of the hierarchy tree.
- \* Update of the direction boxes of the hierarchy tree, if self-collision detection is required.
- \* Running the collision detection algorithm on the hierarchy tree.

The time required for the two first steps is proportional to the number of mesh elements. The involved computations are quite reduced for mesh elements (evaluation of min-max of linear combination of vertex coordinates and normal computation of polygons usually also required for other purposes (mechanics, rendering...), and dot product with a set of directions). Their propagation along the hierarchy tree is also trivial (min-max interval merges, boolean operations). These linear-time computations only add a small overhead to the global simulation process.



**Figure 18:** During the detection process, surface regions considered for the detection are similar in size and number whatever the discretization.

The proposed scheme also behaves very well for highly discretized surfaces, as extra discretization usually yields flatter surfaces relatively to the size of the mesh elements. Hence, unless being near a collision area, the area of the surface regions considered during the detection process always have similar sizes whatever their discretization.

The major benefit of the curvature-based hierarchical collision is that full performance of hierarchical bounding-volume collision detection is preserved for self-collision detection, as the computation is not crippled by detecting all the "colliding" adjacent mesh elements of the surface.

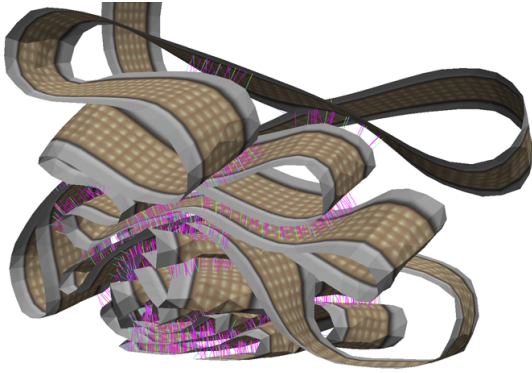


Figure 19: Cloth simulation is heavily relying on self-collision detection.

#### 4. Collision Response

Once the colliding polygons of the mesh have been extracted, they need to be expressed as geometric information that carries some meaning on how the surfaces are colliding. Usually, collisions may be sorted out as **intersections**, where two surface elements interpenetrate each other, and **proximities**, where two surface elements are separated by a distance below a given threshold, usually representing the "thickness" of the surface. In the case of collisions between polygonal meshes, we can identify the following cases:

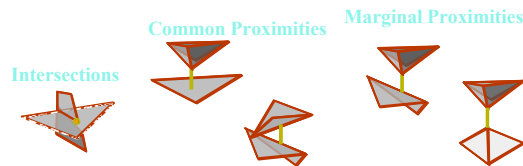


Figure 20: Collision configurations in a polygonal mesh.

Collision response intends to enforce the fact that real surfaces cannot cross each other. It may either handle intersections usually by backtracking the motion leading to the surface crossing and integrating the collision effect, and proximities by maintaining a minimum separation distance between the surfaces. In either case, the collision effect is usually applied on the mesh vertices of the colliding elements, which carries the geometrical information of the mesh.

##### 4.1. Collision Response Schemes

Collision response has to reproduce reaction and friction effects through adequate action in the ongoing mechanical simulation. Its integration in the mechanical simulation system goes through alteration of the mechanical quantities from the value they would have without the collision effects. There are two main ways for handling collision response:

\* **Mechanical response**, where the collision reaction is simulated by forces or by force adjustments which reproduce the contact effect.

\* **Geometrical response**, where the collision reaction is simulated by direct corrections on the positions and velocities of the objects.

The mechanical approach is the most formal way of dealing with the problem. The forces or energetic contributions generated from the response can directly be integrated into the mechanical model and simulated. As all the effects are taken into account in the same computation step, the resulting simulation produces an animation where collision response and other mechanical forces add their effects in a compatible way. Reaction is typically modeled by designing a collision penalty force which will repulse the colliding objects from each other and prevent them from intersecting. The repulsion force function is usually designed as a continuous function of the collision distance, and as a piecewise function using simple linear or polynomial intervals. Designing the optimal shape is difficult, because of all these compromises which depend on the actual mechanical context of the simulation. The biggest issue is to model in a robust way geometrical contact (very small collision distance), in which collision response forces only act in a very small range when considered at the macroscopic scale. This implies the use of very strong and rapidly evolving reaction forces, which are difficult to simulate numerically, since a suitable numerical process should discretize the collision contact duration into timesteps that are numerous enough for an accurate reproduction of the collision effects and which cause problem with the usual simulation timesteps which are usually too large.

The geometrical approach aims to reproduce directly the effects of collision response on the geometrical state of the objects without making use of mechanical forces, and thus in a process separated from the mechanical simulation. The advantages are obvious: Geometrical constraints are directly enforced by a geometrical algorithm, and the simulation process is relieved from high intensity and highly discontinuous forces or other mechanical parameters, making it faster and more efficient. This drawback however results from this separation: As collision response changes the geometrical state of the objects separately from the mechanical process, nothing ensures the compatibility of this deformation to a correct variation of the mechanical state that would normally result from it. Furthermore, there is no compatible "additivity" of geometrical variations as there is for forces and energy contributions. The resulting collision effects may be incompatible with mechanics, but also between several interacting collisions. All these issues have to be addressed for providing a collision response model that provides acceptable and steady responses between all the frames of an animation.

Collision effects are decomposed into **reaction effects** (normal components), which are the obvious forces preventing the object penetrating into each other, and **friction effects** (tangential components), which model additional forces that oppose the sliding of objects. The most common friction model is the solid Coulombian friction, where friction forces opposing the motion do not exceed reaction forces times a friction coefficient.

#### Bibliography

G. BERGEN, Efficient Collision Detection of Complex Deformable Models Using AABB Trees, *Journal of Graphics Tools*, 2(4), pp 1-13, 1997.

- G. BERGEN, *Collision Detection in Interactive 3D Environments*, Morgan Kaufmann Publishers, 2004.
- G. BRADSHAW, C. O'SULLIVAN, Sphere-Tree Construction Using Dynamic Medial Axis Approximation, *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp 33–40, 2002.
- R. BRIDSON, R. FEDKIW, J. ANDERSON, Robust Treatment of Collisions, Contact and Friction for Cloth Animation, *ACM Transaction on Graphics*, Proceedings of ACM SIGGRAPH, 21(3), pp 594–603, 2002.
- J.D. COHEN, M.C. LIN, D. MANOCHA, M.K. PONAMGI, I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments, *Symposium of Interactive 3D Graphics*, pp 189–196, 1995.
- S.A. EHMANN, M.C. LIN, Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition. Proceedings of Eurographics, pp 500–510, 2001.
- C. ERICSON, *Real-Time Collision Detection*,. Morgan Kaufmann Publishers, 2004.
- K. FISHER., B. GARTNER, The Smallest Enclosing Ball of Balls: Combinatorial Structure and Algorithms, *Symposium on Computational Geometry*, pp 292–301, 2003.
- K. FUJIMURA, H. TORIYA, K. YAMAGUSHI, T.L. KUNII, Octree Algorithms for Solid Modeling, Computer Graphics, Theory and Applications, *Proceedings of InterGraphics '83*, Springer-Verlag, pp 96–110, 1983.
- F. GANOVELLI, J. DINGLIANA, C. O'SULLIVAN, Buckettree: Improving Collision Detection Between Deformable Objects, Proceedings of Spring Conference on Computer Graphics, 2000.
- T. GIANG, C. O'SULLIVAN, Closest Feature Maps for Time-Critical Collision Handling, *Workshop on Virtual Reality Interaction and Physical Simulation*, 2005.
- S. GOTTSCHALK, M.C. LIN, D. MANOCHA, OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, *Computer Graphics*, Proceedings of ACM SIGGRAPH, Addison-Wesley, pp 171–180, 1996.
- M. HELD, J.T. KLOSOWSKI, J.S.B. MITCHELL, Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs, *7th Canadian Conference on Computational Geometry*, 1995.
- P.M. HUBBARD, Collision Detection for Interactive Graphics Applications, *IEEE Transactions on Visualization and Computer Graphics*, 1(3), pp 218–230, 1995.
- P.M. HUBBARD, Approximating Polyhedra with Spheres for Time-Critical Collision Detection, *ACM Transactions on Graphics*, 15(3) pp 179–210, 1996.
- D.L. JAMES, D.K. PAI, BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models, *ACM Transactions on Graphics*, 23(3), pp 393–398, 2004.
- P. JIMENEZ, F. THOMAS, C. TORRAS, 3D Collision Detection: A Survey, *Computer and Graphics*, 25(2), pp 269–285, 2001.
- J.T. KLOSOWSKI, M. HELD, J.S.B. MITCHELL, Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, *IEEE transactions on Visualization and Computer Graphics*, 4(1), pp 21–36, 1997.
- T. LARSSON, T. AKENINE-MOLLER, Collision Detection for Continuously Deforming Bodies, *Proceedings of Eurographics*, pp 325–333, 2001.
- T. LARSSON, T. AKENINE-MOLLER, A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection, *Workshop on Virtual Reality Interactions and Physical Simulations*, pp 91–100, 2005.
- M.C. LIN, J.F. CANNY, Efficient Collision Detection for Animation, *Eurographics Workshop on Animation and Simulation*, 1992.
- M.C. LIN, S. GOTTSCHALK, Collision Detection Between Geometric Models: A Survey, *Proceedings of the IMA Conference on Mathematics of Surfaces*, 1998.
- J. LOMBARDO, M.P. CANI, F. NEYRET, Real-Time Collision Detection for Virtual Surgery, *Proceedings of Computer Animation*, IEEE Press, pp 82–91, 1999.
- J. MEZGER, S. KIMMERLE, O. ETZMUSS, Hierarchical Techniques in Collision Detection for Cloth Animation, *Journal of WSCG*, 11(2), pp 322–329, 2003.
- C. O'SULLIVAN.,J. DINGLIANA, Real-time Collision Detection and Response Using Sphere-Trees, *Spring Conference on Computer Graphics*, pp 83–92, 1999.
- I.J. PALMER, R.L. GRIMSDALE, Collision Detection for Animation using Sphere-Trees, *Computer Graphics Forum*, 14, pp 105–116, 1995.
- X. PROVOT, Collision and Self-Collision Handling in Cloth Models Dedicated to Design Garments, *Proceedings of Graphics Interface*, pp 177–189, 1997.
- S. REDON., A. KHEDDAR, S. COQUILLARD, Fast Continuous Collision Detection between Rigid Bodies, *Computer Graphics Forum* (Proceedings of Eurographics), 21(3), pp 279–279, 2002.
- A. SMITH, Y. KITAMURA, H. TAKEMURA, F. KISHINO, A Simple and Efficient Method for Accurate Collision Detection among Deformable Polyhedra, *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp 136–145, 1995.
- M. TESCHNER, S. KIMMERLE, B. HEIDELBERGER, G. ZACHMANN, L. RAGHUPATHI., A. FUHRMANN, M.P. CANI, F. FAURE,N. MAGNENAT-THALMANN, W. STRASSER, Collision Detection for Deformable Objects. *Computer Graphics Forum*, 24(1), pp 61–81, 2005.
- G. VANDENBERGEN, Efficient Collision Detection of Complex Deformable Models using AABB Trees, *Journal of Graphics Tools*, 2(4), pp 1–14, 1997.
- P. VOLINO, N. MAGNENAT-THALMANN, Efficient Self-Collision Detection on Smoothly Discretised Surface

Animation Using Geometrical Shape Regularity, *Computer Graphics Forum* (Proceedings of Eurographics), Blackwell Publishers, 13(3), pp 155-166, 1994.

R.C. WEBB, M.A. GIGANTE, Using Dynamic Bounding Volume Hierarchies to improve Efficiency of Rigid Body Simulations, *Communicating with Virtual Worlds* (Proceedings of CGI'92), pp 825-841, 1992.

G. ZACHMANN, Rapid Collision Detection by Dynamically Aligned DOP-Trees, *Proceedings of IEEE Virtual Reality Annual International Symposium*, pp 90-97, 1998.

# Continuous Collision Detection and Handling for Rigid and Articulated Bodies

Stephane Redon<sup>†</sup>

i3D - GRAVIR - INRIA Rhone-Alpes - France

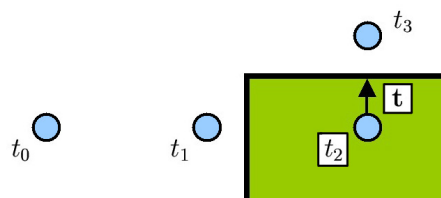
## Abstract

*In these notes, we present an overview of our recent work on continuous collision detection and handling methods, which guarantee consistent simulations by computing the time of first contact and the contact state for colliding objects. We describe techniques to perform continuous collision detection for rigid and articulated bodies. The time-parameterized equations for continuous collision detection between rigid triangle primitives are presented and methods are described to solve them efficiently. Continuous overlap tests between hierarchies of bounding volumes, which help achieve efficient collision detection for complex models, are presented as well. In a second part, we briefly discuss ways to compute a collision response based on the contact information provided by the continuous collision detection solver. Especially, we describe how to couple the collision detection and dynamics modules. Finally, we present some recent applications and extensions of continuous collision detection to motion planning and six degree-of-freedom haptic display of rigid bodies.*

## 1. Introduction

Collision detection methods can roughly be split into two categories. Most well-known collision detection methods are *discrete*: they sample the objects' trajectories at discrete times and report *interpenetrations* only. Discrete collision detection methods, whereas generally simpler to implement and used very frequently in dynamics simulators, may cause various problems. Besides the lack of physical realism resulting from the penetration, these methods can miss collisions when objects are too thin or too fast. Whereas an adaptive timestep and predictive methods can be used to correct this problem in *offline* applications, it may not be adapted in interactive applications when a relatively high and constant framerate is required.

A second problem due to the discretization of trajectories is that *backtracking* methods must be used to compute the time of first contact, which is often required in constraint-based dynamics simulation methods [FMM77, MW88, Bar90]. These backtracking methods consist in looking for the time of first contact through a recursive method. Assume that the current time interval is  $[t_n, t_{n+1}]$ . Essentially,

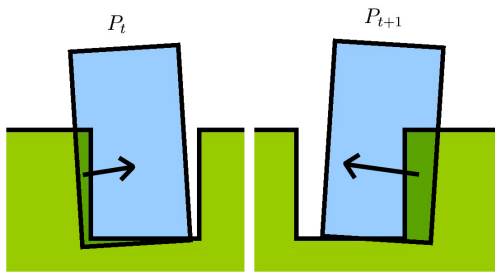


**Figure 1:** Allowing interpenetration between objects can lead to incoherences in the simulation. In this example, the object that penetrated at time  $t_2$  pops out from the wrong side of the obstacle.

one time of first contact  $t_e$  is *estimated* in this interval (for example, by taking the intermediate instant  $\frac{t_n+t_{n+1}}{2}$ , or by a linear or quadratic interpolation method depending on objects' velocities and accelerations). Objects' positions are then computed at this instant and an interpenetration detection is performed again. Depending on whether the objects interpenetrate or not, the algorithm decides that the first time of collision is in  $[t_n, t_e]$  or  $[t_e, t_{n+1}]$ , respectively, and loops on

<sup>†</sup> stephane.redon@inria.fr





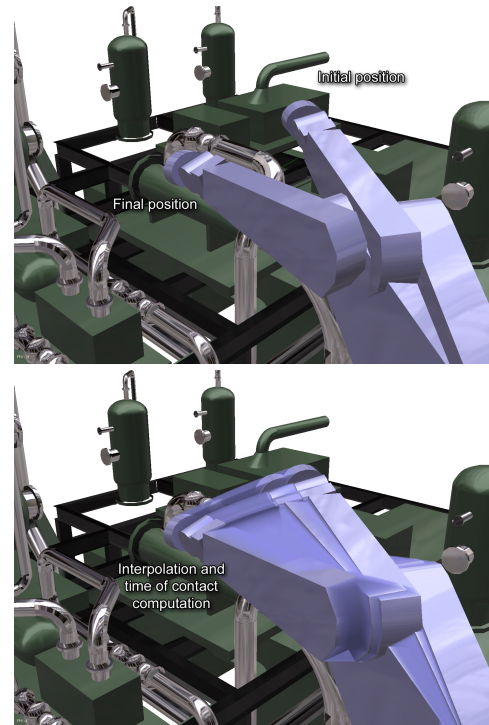
**Figure 2:** Interpenetrations between objects may yield unstable simulations.

this new interval. The process stops when the amount of interpenetration is smaller than a predetermined threshold.

Such backtracking methods can have a high computational cost (which may be difficult to predict) when objects are complex or when they have interpenetrated much. Besides, since backtracking is only performed when an interpenetration has been detected, non-connected objects, or even non-convex objects, can enter a configuration from which they could not get out (consider, for example, the case of two torii which, at one frame, would not be interlocking and, at the next one, would be).

One way to avoid using backtracking methods is to allow interpenetrations between objects, which can be difficult to justify from a physical point of view, and to determine the amount of interpenetration. This problem, however, is extremely difficult for general (non-convex) objects, and the best present results do not take the trajectory of the object into account when determining this interpenetration amount [KOLM02]. Thus, in the case depicted in Figure 1, the mobile point is inside the obstacle at time  $t_2$ , but at this time the smallest interpenetration is represented by the vector  $\mathbf{t}$ , which leads to take out the mobile point by the top of the obstacle as time  $t_3$ . Let us note, however, that the amount of interpenetration can be used to speed up backtracking methods by leading to a better estimation of the time of first contact  $t_e$ .

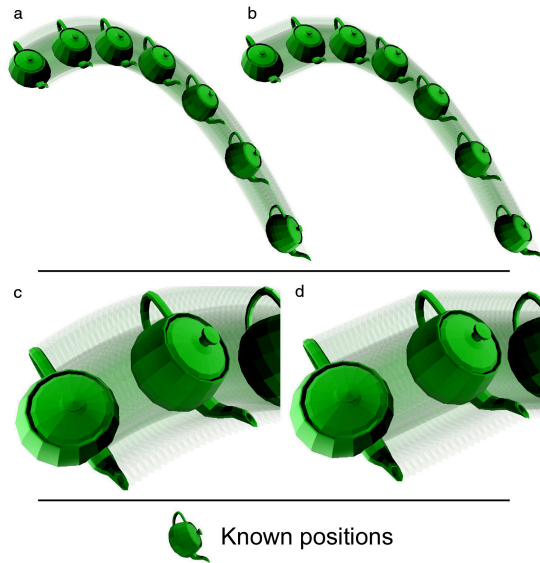
Finally, the interpenetration of objects can be a cause of instability in the dynamics simulation. What is not a problem in a purely virtual simulation (*i.e.* non-interactive) has much more importance when a haptic device is used to interact with the virtual environment. For example, if one refers to the classic benchmark used to evaluate interaction methods, the *peg-in-a-hole* test depicted in Figure 2, it is clear that a first interpenetration of the peg on one side of the hole at time  $t$  (position  $P_t$ ) creates a large force to remove the interpenetration. This force often leads to a greater interpenetration on the opposite side of the hole at the next instant  $t + 1$  (position  $P_{t+1}$ ), which creates a greater reaction force than the



**Figure 3:** Benefit of using a continuous collision detection method for an articulated body. The upper half of the figure shows two successive configurations of a Puma robot which do not penetrate the environment. The lower half shows a linear interpolation of the configurations and an intermediate configuration corresponding to the first time of contact between the robot and the environment.

previous. This oscillation, by amplifying itself, leads to an unstable simulation [GMELM00].

We present an overview of some of our recent work on *continuous* collision detection methods, which guarantee consistent simulations by computing the time of first contact and the contact state for colliding objects. We describe techniques to perform continuous collision detection for rigid and articulated bodies. The time-parameterized equations for continuous collision detection between rigid triangle primitives are presented and methods are described to solve them efficiently. Continuous overlap tests between hierarchies of bounding volumes, which help achieve efficient collision detection for complex models, are presented as well. Figure 3 shows an example of the benefit of using a continuous collision detection method for an articulated body. The upper half of the figure shows two successive configurations of a Puma robot which do not penetrate the environment. The lower half shows a linear interpolation of the configurations and an intermediate configuration corresponding to the first time of contact between the robot and the environment.



**Figure 4:** Use of an arbitrary in-between motion to interpolate the known successive positions of the teapot.

## 2. Arbitrary in-between motions

Most of the time, the actual motion of the objects is not available. Two major reasons are:

**Sampling interface** When one or more of the objects are acted upon through a user interface (e.g. a simple 2d mouse, a joystick, or a full-body motion capture system), the user actions are sent at discrete times only. As a consequence, the user actions between these discrete instants are lost.

**Discretized dynamics formulation** When the objects take part in a dynamics simulation, the dynamics equations have to be discretized in order to be solved (e.g. through an Euler or Runge-Kutta scheme). The positions, velocities and accelerations of the objects are computed at discrete times only (recall moreover that the discretization includes approximations, so that even the positions, velocities and accelerations computed at discrete times are approximations).

In order to prevent any interpenetration of the objects, we thus need to provide a continuous motion with which we will perform collision detection. Precisely, we are going to use an *arbitrary in-between motion*, which must satisfy several requirements:

**Interpolation** The in-between motion must at least interpolate positions. Higher order interpolations can be used depending on the application.

**Continuity** The interpolation must be at least  $C^0$ . The motions we are going to use in these notes will actually be  $C^\infty$ .

**Rigidity** The in-between motion needs to preserve the rigidity of the links. For consistency reasons, we cannot use a straight segment interpolation for object vertices when the object rotates.

**Application-dependent constraints** Depending on the application, some supplementary constraints might have to be satisfied by the in-between motion. In robotics applications, for example, some links might have a pre-defined special type of motion (e.g. screw motion). The arbitrary in-between motion chosen for the application needs to be able to produce these constrained motions.

Provided these requirements are satisfied, we can *arbitrarily* choose an in-between motion. The goal is to determine an arbitrary in-between motion which makes it efficient to perform the various steps in the continuous collision detection algorithm.

To better visualize the way the arbitrary motion is used, let's consider the case of an interactive dynamics simulator which computes the position of a teapot and render a new frame at a fixed rate of 30 frames per second, for example. Between two frames, the motion of the teapot is not visible by the user and is replaced by an arbitrary in-between motion. Since the real positions are respected at the successive discrete instants, only the *local* motion of the teapot is modified, and its *global* motion is preserved, as shown in Figure 4. In this example, the position of the teapot is known at seven discrete instants  $t_0, \dots, t_6$ . In the left part ((a), zoomed in (c)), the real motion of the teapot between these instants is represented in transparency. In the right part ((b), zoomed in (d)), the known positions are preserved, but the real motion of the object between two successive known positions has been replaced by an arbitrary in-between motion, represented in transparency. The general aspect of the teapot trajectory is preserved.

It should be clear, now, that using an arbitrary in-between motion is more or less equivalent to the underlying principle of every integration scheme used to solve the dynamics differential equations: discretizing the differential equations amounts to make finite-order assumptions on velocities and accelerations between successive discrete instants.

Let us now formalize the constraints imposed on the arbitrary in-between motion. To do this, let  $\mathbf{P}_R(t)$  denote the  $4 \times 4$  matrix describing the *real* position of the object during the time interval  $[t_n, t_{n+1}]$ . Recall that this matrix allows us to compute the real (homogeneous) coordinates  $\mathbf{x}_R(t)$  of a point of the object in the global frame from its (homogeneous) coordinates  $\mathbf{x}_o$  in the local frame of the object:

$$\mathbf{x}_R(t) = \mathbf{P}_R(t)\mathbf{x}_o. \quad (1)$$

Vectors  $\mathbf{x}_R(t)$  and  $\mathbf{x}_o$  are homogeneous vectors in  $\mathbb{R}^4$ , for which the last coordinate is the real number 1.

Let's denote now the object's position matrix when using the *arbitrary* motion, during the same time interval  $[t_n, t_{n+1}]$

by  $\mathbf{P}_A(t)$ . Similarly, the arbitrary coordinates  $\mathbf{x}_A(t)$  of a point of the object in the global frame are obtained from its coordinates in the local frame of the object:

$$\mathbf{x}_A(t) = \mathbf{P}_A(t)\mathbf{x}_o \quad (2)$$

The three constraints can be formalized simply:

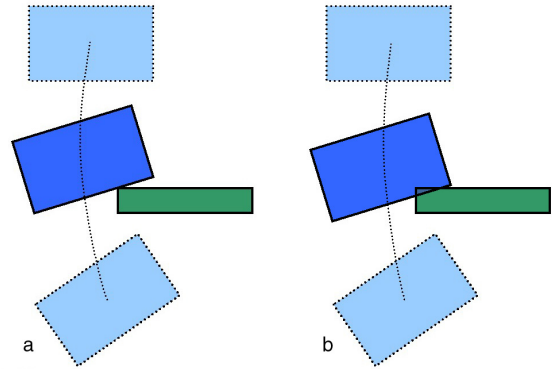
- the interpolation constraint merely imposes that  $\mathbf{P}_A(t_n) = \mathbf{P}_R(t_n)$ , as well as  $\mathbf{P}_A(t_{n+1}) = \mathbf{P}_R(t_{n+1})$ ,
- the continuity constraint imposes that the function  $t \mapsto \mathbf{P}_A(t)$  is continuous on the interval  $[t_n, t_{n+1}]$ , that is, that the components of  $\mathbf{R}_A(t)$  and  $\mathbf{T}_A(t)$  are continuous functions of time on this interval.
- the rigidity constraint imposes that the matrix  $\mathbf{P}_A(t)$  is a position matrix at every time  $t$  between  $t_n$  and  $t_{n+1}$ . In other words, it must not include deformation terms (scaling terms, for example), and must be the combination of a rotation matrix and a translation vector, according to the classic form of a homogeneous position matrix:

$$\mathbf{P}_A(t) = \begin{pmatrix} \mathbf{R}_A(t) & \mathbf{T}_A(t) \\ \mathbf{0} & 1 \end{pmatrix}, \quad (3)$$

Of course, a fourth constraint is implicit: the arbitrary in-between motion should be close to the real motion. It would thus be desirable to define a measure allowing to evaluate the difference between the real object motion and the arbitrary one, used to detect collisions between the successive discrete instants. However, we noticed that this real motion is rarely available. For this reason, we suggest using the motion defined by the position and velocity determined by the dynamics calculator as a reference. Thus, we saw that for each successive discrete instant the dynamics calculator computes objects' positions and velocities from those of the previous discrete instant and possibly, for higher-order integration schemes (as the fourth-order Runge-Kutta integration scheme), of those established to certain intermediate instants. It is possible, as is done in a way by the dynamics calculator, to assume that the object's rotational and translation velocities are constant during the timestep. A natural arbitrary motion is then a motion whose velocities remain close to these reference velocities.

Although it is possible to define this last constraint more rigorously, for example by defining the maximal error on the position of points of the object resulting from the use of an arbitrary motion, we noticed empirically that the arbitrary motions described in these notes are always sufficiently natural so that the user is not surprised by the position of the object that he manipulates at the time of the collision (as one can expect when examining Figure 4).

Let's note that replacing the objects' motions by arbitrary ones between two successive discrete instants  $t_n$  and  $t_{n+1}$  has a consequence on the simulation only if a collision occurs between these two instants. If no collision is detected during the in-between time interval, the objects are placed to the final positions computed by the dynamics calculator.



**Figure 5:** To avoid interpenetrations, it is necessary to compute the objects' positions at the instant of collision from the in-between motion used for the detection of collisions, and not from the interpolating motion computed by the dynamics equations.

However, if a collision between two objects is detected at time  $t_c$ , it is necessary to use the arbitrary motions to compute the positions of *all* the objects at time  $t_c$ , since these motions have been used for the detection of collisions.

Otherwise, some interpenetrations could occur, as shown in Figure 5. In (a), a collision has been detected at time  $t_c$  while using the arbitrary in-between motion. In (b), the dynamics calculator has been used to compute the real position of the object at time  $t_c$ , which results in an interpenetration. For the same reason, when a collision is detected, the arbitrary motion must also be used for objects that did not enter in contact with another object. To compute their real positions at time  $t_c$  using the dynamics calculator could induce interpenetrations, since these positions have not been used for the detection of collisions.

Consequently, the use of an arbitrary in-between motion to detect collisions perturbs the course of the simulation. It is indeed very unlikely that the real object motion and the arbitrary in-between motion would lead to detect collisions at the same instants and at the same places. It is not even guaranteed that a collision which occurs between two objects when one of the two motions (real or arbitrary) is used would also occur when the other motion is used. This is the price we have to pay to perform continuous collision detection when the actual object motion is not known. This allows, however, to continuously detect collisions very efficiently, while preserving the benefits of a continuous method that would use the real object motion. Indeed, with this method, objects are permanently in a consistent state: no interpenetration is possible and no collision can be missed.

Besides, for some more rigorous physical applications, it should be sufficient to reduce the length of the timestep, thanks to the continuity and interpolation constraints im-

posed on the arbitrary motion. Indeed, the instants between which the motion of the object is replaced are not necessarily instants to which objects are displayed. Let's assume, for example, that the object moves very fast on the time interval  $[t_n, t_{n+1}]$ . In order to reduce the error between the real object motion and the arbitrary one used for continuous collision detection, it can be preferred to divide the time interval in two smaller intervals  $[t_n, t_i]$  and  $[t_i, t_{n+1}]$ , where  $t_i = \frac{t_n + t_{n+1}}{2}$ . At the intermediate time  $t_i$ , the position of the object is evaluated by the dynamics simulator and is used to replace the real object motion by *two* successive arbitrary motions, on  $[t_n, t_i]$  first then on  $[t_i, t_{n+1}]$ . This ensures that the intermediate position to the time  $t_i$ , at least, is the one computed by the dynamics simulator.

Thus, the error created by the use of an arbitrary in-between motion relies upon the same approximation principle than the one which prevails when discretizing the dynamics equations. Overall, we believe that this approximation problem is largely compensated by the benefits provided by a continuous collision detection method.

To conclude this section, let's summarize the (simple) idea behind the use of an arbitrary in-between motion: *since the real object motion between any two successive discrete instants cannot be used to continuously detect collisions, it is replaced by an arbitrarily fixed in-between motion, which must satisfy three constraints: this arbitrary motion must interpolate in a continuous and rigid way the object's configurations between successive discrete instants. Among the arbitrary motions which satisfy these constraints, we choose one which allows us to perform the various steps of the continuous collision detection algorithm very efficiently.*

## 2.1. Rigid bodies

Let us now describe two possible arbitrary in-between motions for rigid bodies. Again, recall that we want to choose a simple motion.

### 2.1.1. Constant-velocity translation and rotation

One possibility is to assume that the rigid motion over the timestep is a constant-velocity one, composed of a translation along a fixed direction, and a rotation along a fixed (potentially distinct) direction.

Let the 3-dimensional vector  $\mathbf{c}^0$  and the  $3 \times 3$  matrix  $\mathbf{R}^0$  denote the position and orientation of the rigid body in the world frame at the beginning of the (normalized) time interval  $[0, 1]$ . Let  $\mathbf{s}$  denote the total translation during the timestep, and let  $\omega$  and  $\mathbf{u}$  respectively denote the total rotation angle and the rotation axis. For a given time step,  $\mathbf{c}^0$ ,  $\mathbf{R}^0$ ,  $\omega$ ,  $\mathbf{u}$  and  $\mathbf{s}$  are constants.

The position of the rigid body at a given time  $t$  in  $[0, 1]$  is thus:

$$\mathbf{T}(t) = \mathbf{c}^0 + t\mathbf{s}, \quad (4)$$

The orientation of the rigid body is:

$$\mathbf{R}(t) = \cos(\omega t) \cdot \mathbf{A} + \sin(\omega t) \cdot \mathbf{B} + \mathbf{C}, \quad (5)$$

where  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  are  $3 \times 3$  constant matrices which are computed at the beginning of the time step:

$$\begin{aligned} \mathbf{A} &= \mathbf{R}^0 - \mathbf{u} \cdot \mathbf{u}^T \cdot \mathbf{R}^0 \\ \mathbf{B} &= \mathbf{u}^* \cdot \mathbf{R}^0 \\ \mathbf{C} &= \mathbf{u} \cdot \mathbf{u}^T \cdot \mathbf{R}^0 \end{aligned} \quad (6)$$

where  $\mathbf{u}^*$  denotes the  $3 \times 3$  matrix such as  $\mathbf{u}^* \mathbf{x} = \mathbf{u} \times \mathbf{x}$  for every three-dimensional vector  $\mathbf{x}$ . If  $\mathbf{u} = (u^x, u^y, u^z)^T$ , then:

$$\mathbf{u}^* = \begin{pmatrix} 0 & -u^z & u^y \\ u^z & 0 & -u^x \\ -u^y & u^x & 0 \end{pmatrix} \quad (7)$$

Consequently, the motion of the rigid body is described by the following  $4 \times 4$  homogeneous matrix:

$$\mathbf{P}(t) = \begin{pmatrix} \mathbf{R}(t) & \mathbf{T}(t) \\ \mathbf{0} & 1 \end{pmatrix}, \quad (8)$$

in the world frame.

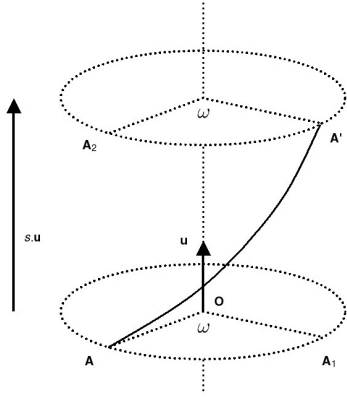
The motion parameters  $\mathbf{s}$ ,  $\mathbf{u}$  and  $\omega$  are easy to compute. Assume  $\mathbf{c}^0$  and  $\mathbf{c}^1$  (resp.  $\mathbf{R}^0$  and  $\mathbf{R}^1$ ) are the initial and final positions (resp. orientations) of the rigid body in the world frame. Then  $\mathbf{s} = \mathbf{c}^1 - \mathbf{c}^0$ , and  $(\mathbf{u}, \omega)$  is the rotation extracted from the rotation matrix  $\mathbf{R}^1(\mathbf{R}^0)^T$ .

### 2.1.2. Screw motions

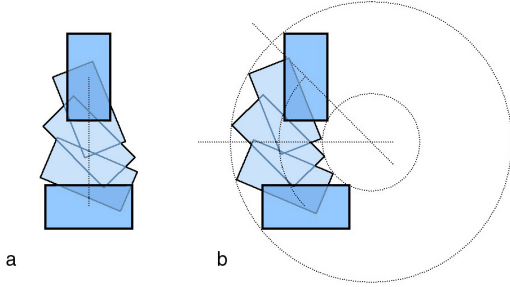
An even simpler motion can be used, for which the rotation axis and the translation have the same direction. Such a motion is called a *screw motion*.

Precisely, a screw motion  $\mathcal{V}(\omega, s, \mathbf{O}, \mathbf{u})$  is the commutative composition of a rotation and a translation along the same axis. The real parameters  $\omega$  and  $s$  (now a real number) respectively denote the total amount of rotation and the total amount of translation in the transformation,  $\mathbf{O}$  is a point on the the screw motion axis, and  $\mathbf{u}$  is a unit vector describing the axis orientation. Note that the total translation is now  $\mathbf{s} = s \cdot \mathbf{u}$ . A screw motion is depicted in Figure 6. In this example, the screw motion transforms the point  $\mathbf{A}$  into  $\mathbf{A}'$ . Depending on whether the rotation or the translation is applied first to the point  $\mathbf{A}$ , the intermediate point is respectively  $\mathbf{A}_1$  or  $\mathbf{A}_2$ .

The benefit of using screw motions comes from the fact that they allow us to interpolate any two rigid positions with less parameters, and thus reduce the computational cost of evaluating the motion matrix. Whatever the object positions at times  $t_n$  and  $t_{n+1}$ , Chasles' theorem states that there exists an unique screw motion which transforms the initial position (*i.e.* at time  $t_n$ ) into the final position (*i.e.* at time  $t_{n+1}$ ) (when  $\mathbf{O}$  on the screw motion axis is fixed, and when  $\omega$  is required to be positive [Cha1831]).



**Figure 6:** A screw motion is the commutative composition of a rotation and a translation of same axes.



**Figure 7:** Using a screw motion to replace the real object motion. a: the real object motion is a pure translation at constant velocity (from top to bottom) combined to a rotation at constant velocity around the object's center of mass. b: the real object motion has been replaced by the equivalent (and unique) screw motion with positive angle. For applications which require a very large rotation angle over the time interval  $[0, 1]$ , it might be advisable to subdivide the time interval into several smaller ones.

In theory, using a screw motion to interpolate two successive positions could lead to a non natural in-between motion. In Figure 7, the real object motion (on the left) has been replaced by the equivalent screw motion with positive angle (on the right). For applications which require a very large rotation angle over the time interval  $[0, 1]$ , it might be advisable to subdivide the time interval into several smaller ones.

We can now build a general class of screw motion-based arbitrary in-between motions. Assume, without loss of generality, that the current time interval is the interval  $[0, 1]$ . In order to get a rigid and continuous motion that interpolates the initial and final positions, it is sufficient to make the parameters  $\omega$  and  $s$  vary continuously. This can be achieved by choosing two functions  $a : \mathbb{R}^2 \times [0, 1] \rightarrow \mathbb{R}$  and  $b :$

$\mathbb{R}^2 \times [0, 1] \rightarrow \mathbb{R}$  such as, for all pair  $(\omega, s)$  in  $\mathbb{R}^2$ , the functions

$$a_{\omega,s} : \begin{cases} [0, 1] \rightarrow \mathbb{R} \\ t \mapsto \omega(t) = a(\omega, s, t) \end{cases} \quad (9)$$

$$b_{\omega,s} : \begin{cases} [0, 1] \rightarrow \mathbb{R} \\ t \mapsto s(t) = b(\omega, s, t) \end{cases} \quad (10)$$

are  $C^1$ , monotonous, and respect the interpolation constraint, i.e.  $a_{\omega,s}(0) = b_{\omega,s}(0) = 0$ , and  $a_{\omega,s}(1) = \omega$  and  $b_{\omega,s}(1) = s$ .

The class of screw motion-based arbitrary in-between motions has the form:

$$\mathcal{M} : \begin{cases} [0, 1] \times \mathbb{R}^3 \rightarrow \mathbb{R}^3 \\ (t, A) \mapsto A(t) = \mathcal{V}(a_{\omega,s}(t), b_{\omega,s}(t), O, \vec{u})(A_0) \end{cases} \quad (11)$$

where  $A_0$  is a point of the object at time 0 and  $A(t)$  the same point during the arbitrary in-between motion. It is worth noticing that the two functions  $a$  and  $b$  depend on the screw motion parameters only, and not on the object shape or part. This guarantees that all points of the object have the same rigid motion. Besides, thanks to the conditions imposed on the functions  $a_{\omega,s}$  and  $b_{\omega,s}$ , arbitrary motions of form (11) are truly rigid, continuous and interpolating.

A motion in the class (11) can be expressed simply in matrix form. Define first a *screw motion frame* as a frame in which the  $Oz$  axis is the screw motion axis. Because of axial symmetry, there exists an infinity of such frames, and it is sufficient to choose one of them. In one of these frames, the screw motion can be expressed simply:

$$\mathbf{V}(t) = \begin{pmatrix} \cos(a_{\omega,s}(t)) & -\sin(a_{\omega,s}(t)) & 0 & 0 \\ \sin(a_{\omega,s}(t)) & \cos(a_{\omega,s}(t)) & 0 & 0 \\ 0 & 0 & 1 & b_{\omega,s}(t) \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (12)$$

for  $t \in [0, 1]$ . In the global frame, the screw motion is then:

$$\mathbf{S}(t) = \mathbf{P}_V^{-1} \mathbf{V}(t) \mathbf{P}_V \quad (13)$$

where  $\mathbf{V}(t)$  is the screw motion with  $Oz$  axis,  $\mathbf{P}_V$  is the transformation matrix from the global frame to the screw motion frame, and  $\mathbf{P}_V^{-1}$  is the inverse of  $\mathbf{P}_V$ .

Thanks to the expression of the screw motion in the global frame (13), it is possible to get the coordinates of any object point  $\mathbf{x}(t)$  during the arbitrary in-between motion:

$$\mathbf{x}(t) = \mathbf{P}(t) \mathbf{x}_0 = \mathbf{P}_V^{-1} \mathbf{V}(t) \mathbf{P}_V \mathbf{P}_0 \mathbf{x}_0 \quad (14)$$

where  $\mathbf{x}_0$  denotes the point coordinates in the object frame, and  $\mathbf{P}_0$  is the objects's position matrix at time 0. The object's position matrix during the arbitrary motion is  $\mathbf{P}(t)$ .

## 2.2. Articulated bodies

An articulated body is defined as a set of rigid bodies, or *links*, connected by bilateral constraints. Assume there is no loop in the articulated body. It is simple to define an arbitrary

in-between motion: it suffices to express the motion of each link in the reference frame of its parent link, and not in the world frame. The motion of the root link of the articulated model is still expressed in the world frame.

Assume, for the sake of simplicity of notation, that the parent of link  $i$  is  $i - 1$ . The index denoting the world frame is 0. Let  $\mathbf{P}_i^{i-1}(t)$  denote the position matrix of link  $i$  in the reference frame of its parent link  $i - 1$ . Then the matrix

$$\mathbf{P}_i^0(t) = \mathbf{P}_1^0(t) \cdot \mathbf{P}_2^1(t) \dots \mathbf{P}_i^{i-1}(t) \quad (15)$$

describes the motion of link  $i$  in the world frame.

### 3. Interval arithmetic

A simple way to robustly perform the computations involved in the various steps of a continuous collision detection algorithm is to use interval arithmetic.

Interval arithmetic consists in computing with intervals instead of numbers. Several good introductions to interval arithmetic can be found for example in [Moo62, Sny92, Kea96]. As is well known, the definition of a closed real interval  $[a, b]$  is:

$$I = [a, b] = \{x \in \mathbb{R}, a \leq x \leq b\} \quad (16)$$

This definition can be generalized to vectors. A vector interval is simply a vector whose components are intervals:

$$I_n = [a_1, b_1] \times \dots \times [a_n, b_n] \quad (17)$$

$$= \{\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n, a_i \leq x_i \leq b_i \quad \forall i, 1 \leq i \leq n\} \quad (18)$$

In  $\mathbb{I}\mathbb{R}^3$ , for example, a simple alternate notation can be:

$$\begin{pmatrix} [x_l, x_u] \\ [y_l, y_u] \\ [z_l, z_u] \end{pmatrix} \quad (19)$$

The set of intervals is denoted  $\mathbb{I}\mathbb{R}$ , while the set of vector intervals is denoted  $\mathbb{I}\mathbb{R}^n$ .

Basic operations can be transposed to intervals:

$$\begin{aligned} [a, b] + [c, d] &= [a + c, b + d] \\ [a, b] - [c, d] &= [a - d, b - c] \\ [a, b] \times [c, d] &= [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \\ 1/[a, b] &= [1/b, 1/a] \quad \text{if } a > 0 \text{ or } b < 0 \\ [a, b]/[c, d] &= [a, b] \times (1/[c, d]) \quad \text{if } c > 0 \text{ or } d < 0 \\ [a, b] \leq [c, d] &\quad \text{if } b \leq c \end{aligned} \quad (20)$$

Elementary operations in  $\mathbb{I}\mathbb{R}^n$  are performed component-wise. Operations between real numbers and real intervals can be performed by identifying  $\mathbb{R}$  and the set of “point” intervals  $\{[x, x], x \in \mathbb{R}\}$ .

Interval arithmetic can be used to bound a function over

an interval very easily, provided the analytic expression of the function is known, and provided we can bound easily the sub-expressions in the function.

An example will make this clear. Assume we want to bound the function  $t \mapsto \sqrt{3} \cos(t) + \sin(t)$  over the time interval  $[0, \pi/2]$ . This function is very similar to the ones we obtain when we plug the arbitrary in-between motions described above in the continuous collision detection equations.

Being able to bound the sine and cosine sub-expressions is all that is required to bound this function. We know that:

$$t \in \left[0, \frac{\pi}{2}\right] \Rightarrow \begin{cases} \cos(t) \in [0, 1] \\ \sin(t) \in [0, 1] \end{cases}$$

Note that this is not deduced from the elementary interval operations, but has to be known. This is what is meant by “we can bound easily the sub-expressions in the function”. From now on, however, we only need to use the elementary interval operations to provide some bounds on the function. Since, by definition,

$$\sqrt{3} \in [\sqrt{3}, \sqrt{3}],$$

and

$$\cos(t) \in [0, 1], \forall t \in \left[0, \frac{\pi}{2}\right],$$

we determine that

$$\sqrt{3} \cos(t) \in [\sqrt{3}, \sqrt{3}] \times [0, 1] = [0, \sqrt{3}], \forall t \in \left[0, \frac{\pi}{2}\right],$$

by performing a simple interval multiplication.

Similarly, using the interval addition, we know that

$$\sqrt{3} \cos(t) + \sin(t) \in [0, \sqrt{3}] + [0, 1] = [0, \sqrt{3} + 1],$$

for all  $t$  in  $\left[0, \frac{\pi}{2}\right]$ , and we have thus bounded the function.

Note that the bounds we have obtained are not exact, since the tightest bounding interval is actually  $[1, 2]$ . In this example, the reason for the looseness of the bounds is that the sine function is increasing while the cosine function is decreasing. Provided we know exact bounds on these sub-expressions, however, it can be shown that the bounds on the function tend to be exact when the size of the time interval tends towards zero.

Exact bounds on the sub-expressions we encounter in these notes are actually very easy to obtain. For example, since the cosine function is decreasing over  $[0, \pi/2]$ , we know that

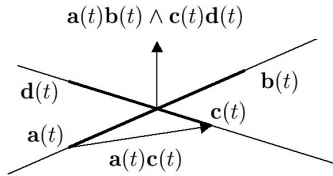
$$a, b \in \left[0, \frac{\pi}{2}\right], a < b \Rightarrow \cos(t) \in [\cos(b), \cos(a)], \forall t \in [a, b].$$

The power of interval arithmetic for our purpose comes from the fact that the interval operations can be simply and efficiently implemented. We refer the reader to [Red04a] for some C++ code for elementary interval arithmetic classes and related code bits.

#### 4. Elementary continuous collision detection

Continuous collision detection methods for polyhedral objects must only detect three types of contact. Indeed, all contacts between two polyhedral objects  $A$  and  $B$  include at least one of these three *elementary* contact types:

- an edge of  $A$  contacts an edge of  $B$ .
- a vertex of  $A$  contacts a face of  $B$ .
- a face of  $A$  contacts a point of  $B$ .

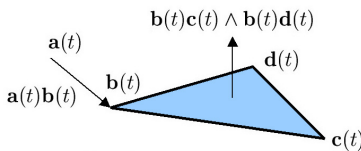


**Figure 8:** Collision detection between two edges.

These contact types are easily expressed geometrically. For the edge/edge case, it suffices to detect a collision between the lines containing the edges. If  $\mathbf{a}(t)\mathbf{b}(t)$  is the first edge and  $\mathbf{c}(t)\mathbf{d}(t)$  is the second edge, then the lines intersect when:

$$\mathbf{a}(t)\mathbf{c}(t) \cdot (\mathbf{a}(t)\mathbf{b}(t) \wedge \mathbf{c}(t)\mathbf{d}(t)) = 0, \quad (21)$$

*i.e.* when the vector  $\mathbf{a}(t)\mathbf{c}(t)$  is in the plane defined by the two edges (cf. Figure 8). Once an intersection has been detected at some instant between the two lines, we check whether it belongs to the edges or, equivalently, if the edges intersect *at that time* (and not only the supporting lines. This can be robustly performed thanks to a discrete edge/edge proximity test (in general, due to finite precision computations, the edges do not *exactly* touch at the collision time). We then keep the earliest valid collision. The contact time is the earliest valid collision time. The contact position is the position of the vertex at that time, and the contact normal is the (normalized) cross-product of the edges at that time.



**Figure 9:** Collision detection between a point and a face.

For the vertex/face and face/vertex, a collision is first detected between the point and the plane containing the face.

If  $\mathbf{a}(t)$  is the point and  $\mathbf{b}(t)\mathbf{c}(t)\mathbf{d}(t)$  is the triangle, a collision occurs when:

$$\mathbf{a}(t)\mathbf{b}(t) \cdot (\mathbf{b}(t)\mathbf{c}(t) \wedge \mathbf{b}(t)\mathbf{d}(t)) = 0, \quad (22)$$

that is when the vector  $\mathbf{a}(t)\mathbf{b}(t)$  is in the vector plane defined by the face normal  $\mathbf{b}(t)\mathbf{c}(t) \wedge \mathbf{b}(t)\mathbf{d}(t)$  (cf. Figure 9). When such a collision is detected, we check whether the point belongs to the face *at that time*. This can be robustly performed thanks to a vertex/triangle proximity test (in general, due to finite precision computations, the vertex is not *exactly* in the plane at the collision time). We then keep the earliest valid collision. The contact time is the earliest valid collision time. The contact position is the position of the vertex at that time, and the contact normal is the normal to the triangle at that time.

In practice, interval arithmetic can be used to solve equations (21) and (22). Formally, these equations have the form

$$f(t) = 0, t \in [0, 1],$$

and we want to determine the smallest root  $t_c$ . Assume we are able to bound the function  $f$  over the time interval  $[0, 1]$ . If these bounds do not contain zero, meaning that the function is strictly positive or strictly negative over the time interval  $[0, 1]$ , then  $f$  cannot have any root in  $[0, 1]$ .

However, if these bounds *do* contain zero, then the function  $f$  *might* have a root in  $[0, 1]$  (might only, if the bounds are not tight or if the function is not continuous. Of course, the functions involved in these notes are all continuous). In this case, we refine the time interval and repeat the process: we bound the function  $f$  on the time intervals  $[0, 1/2]$  and  $[1/2, 1]$ , and we examine these bounds (first  $[0, 1/2]$  and then  $[1/2, 1]$ , since we are looking for the *earliest* collision). This process is recursively performed until the examined bounds do not contain zero (meaning that the function does not have any root on the time sub-interval), or until the size of the examined time sub-interval is smaller than a user-defined threshold (which characterizes the precision of the collision detection).

This binary subdivision process can be easily implemented. The bounds on the function  $f$  are computed using interval arithmetic, as explained in Section 3, by first computing bounds on the motion matrices (and interval-multiplying them, for articulated bodies), and then on the vertices involved in the collision detection equation using interval matrix-vector multiplications [Red04a].

It should now be clear why the choice of the arbitrary in-between motion has a huge impact on the overall efficiency of the continuous collision detection algorithm. The arbitrary in-between motion is going to be evaluated several times whenever some bounds on a continuous collision detection function are needed. If acceptable in the application, it can be advised, for example, to use an in-between motion which reduces the collision detection equations to polynomial equations [Can86, RKC00].

## 5. Continuous overlap tests between bounding volumes

In order to avoid performing all possible elementary tests for any object pair, many collision detection algorithms rely on *bounding-volume hierarchies*. Basically, if two objects are enclosed in bounding volumes which don't overlap, then it is known for sure that they don't collide. Hierarchies of bounding volumes are used to recursively perform such overlap tests which can conservatively cull away large parts of the objects when testing for a collision. Typical bounding volumes used for collision detection include spheres [Qui94, Hub95, RKK97, BS02], axis-aligned bounding boxes (AABBs) [VDB98], oriented bounding boxes (OBBs) [Got99, GLM96], *k*-dops [KHMSZ98] and DOP-trees [Zac98].

Since we want to perform continuous collision detection between objects, we need to design a continuous overlap test between two bounding volumes. Precisely, we need to determine whether two bounding volumes will overlap during the timestep.

What makes the task easier is that it is not necessary to perform an *exact* test. We need to be sure that we do not miss an overlap between two bounding volumes during the timestep, but it is acceptable to declare that two bounding volumes overlap when they don't. The error will be captured later by smaller bounding volumes in the hierarchy, or ultimately by the elementary continuous collision detection tests. Such a test is called *conservative*.

In the following, we describe continuous overlap tests between spheres, axis-aligned bounding boxes, and oriented bounding boxes.

### 5.1. Spheres

Assume spheres are used as bounding volumes. Let  $\mathbf{c}_1$  and  $\mathbf{c}_2$  denote the centers of the spheres, and let  $r_1$  and  $r_2$  denote the radii of the spheres.

The spheres overlap if and only if the distance between their centers is smaller than the sum of their radii:

$$\|\mathbf{c}_1 - \mathbf{c}_2\| \leq r_1 + r_2,$$

or, equivalently, if and only if

$$(\mathbf{c}_2 - \mathbf{c}_1)^2 \leq (r_1 + r_2)^2. \quad (23)$$

Using interval arithmetic, we can design a conservative test to bound the left member of inequality (23) on any time interval  $I$ . If the lower bound of the left member is greater than the right member, then we know for sure that the distance between the centers is greater than the sum of the radii during the whole time interval  $I$ , in which case it is safe to declare that the spheres won't overlap during this time interval.

If the lower bound of the left member is smaller than the

right member, however, there might be an overlap during the time interval, and we conservatively declare so.

The bounds on the left member are obtained by first bounding the coordinates of the center of the spheres, and then performing the interval evaluation of the function in the left member.

### 5.2. Axis-aligned bounding boxes

Axis-aligned bounding volumes are typically recomputed at the beginning of each time step in discrete methods. Assuming these boxes are attached to the bodies during the timestep, they simply become *oriented* bounding boxes, as they lose their axis-aligned characteristic during the objects' motions. Consequently, the appropriate continuous overlap test in that case is the one between two oriented bounding boxes, described in the next section.

However, it is simple to obtain axis-aligned boxes which bound an object during a whole time interval. Indeed, any three-dimensional vector interval is actually an axis-aligned bounding box. Assume we determine bounds on a vertex motion during a time interval. By definition, since the bounds are computed coordinate per coordinate, **we have actually obtained an axis-aligned box which bounds the moving vertex during the whole time interval**. We can thus easily determine AABBs which bound the moving object during the whole time interval.

Note that the AABBs can be obtained from a simplified version of the object geometry, provided this simplified version contains the original object. Assume, for example, that the object is included in a sphere. Using interval arithmetic, the bounds on the coordinates of the center of the sphere can be obtained easily. These bounds are in fact an AABB which encloses the moving center of the sphere during the time interval. Enlarging this AABB by the radius of the sphere results in an AABB which contains the sphere, and thus the object, during the whole time interval.

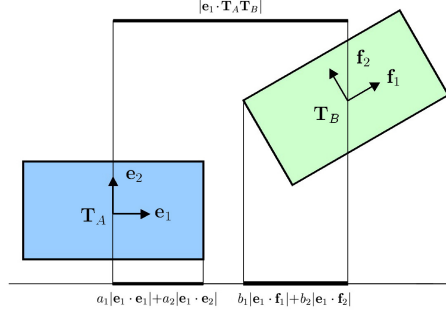
More generally, assume the object is enclosed in the convex hull of a set of points. An AABB can be obtained for each of these points using interval arithmetic. An AABB which contains all these AABBs is guaranteed to contain the object during the whole time interval.

When these dynamically generated AABBs have been computed, the traditional discrete AABB/AABB test can be used to conservatively determine whether the objects are going to overlap or not during the time interval.

### 5.3. Oriented bounding boxes

Let us now proceed to the case of oriented bounding boxes (OBBs). For a rigid object, a hierarchy of OBBs can be computed offline. Similarly, for an articulated model composed of rigid links, a hierarchy of OBBs can be computed offline for each link.





**Figure 10:** The axis  $\mathbf{e}_1$  separates the two oriented bounding boxes since, in the axis direction, the projected distance between the centers of the boxes  $|\mathbf{e}_1 \cdot \mathbf{T}_A \mathbf{T}_B|$  is larger than the sum of the projected radii of the boxes,  $(a_1|\mathbf{e}_1 \cdot \mathbf{e}_1| + a_2|\mathbf{e}_1 \cdot \mathbf{e}_2|) + (b_1|\mathbf{e}_1 \cdot \mathbf{f}_1| + b_2|\mathbf{e}_1 \cdot \mathbf{f}_2|)$ .

The goal is thus to (conservatively) determine whether the boxes are going to overlap during the time interval.

A well-known overlap test for oriented bounding boxes is the one which relies upon the separating axis theorem [GLM96]. Lets 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  $a_3$ . Similarly, assume 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: an axis  $\mathbf{a}$  separates the OBBs if and only if:

$$|\mathbf{a} \cdot \mathbf{T}_A \mathbf{T}_B| > \sum_{i=1}^3 a_i |\mathbf{a} \cdot \mathbf{e}_i| + \sum_{i=1}^3 b_i |\mathbf{a} \cdot \mathbf{f}_i|. \quad (24)$$

This test is performed for 15 axes at most. The sufficient set of fifteen axes is:

$$\{\mathbf{e}_i, \mathbf{f}_j, \mathbf{e}_i \wedge \mathbf{f}_j, 1 \leq i \leq 3, 1 \leq j \leq 3\} \quad (25)$$

Intuitively, the left member inequality (24) is the projected distance between the two centers of the boxes in the direction of  $\mathbf{a}$ , and the right member is the sum of the projected radii of the boxes, in the same direction (cf. Figure 10).

We can extend the discrete OBB/OBB overlap test to the continuous domain using interval arithmetic [RKC02b]. Since each member of inequality (24) is a function of time depending on the specific arbitrary in-between motion, we can use interval arithmetic to bound both members over a time interval  $I$ . When the lower bound on the left member is larger than the upper bound on the right member, the axis  $\mathbf{a}$  separates the boxes during the entire time interval  $I$ , and the pair of boxes is discarded, since we know for sure that the boxes will not overlap during the time interval.

As before, once the bounds on the elements involved in

the test have been computed, the bounds on the two members are determined easily (C++ code available in [Red04a]).

#### 5.4. Remarks

Again, we have used interval arithmetic to perform the continuous tests. As opposed to what happens with the elementary tests, though, the interval computations which occur during the continuous overlap tests between bounding volumes are generally performed once only, over the full time interval  $[0, 1]$ : the bounds on the functions involved in the tests are computed once and for all on the time interval  $[0, 1]$  and these bounds are used to conservatively determine the overlap status of the bounding volumes during this time interval. This comes from the fact that we do not really need to know *when* the bounding volumes will begin to overlap (although that might be a useful information), but only *if* they are going to overlap during the given time interval.

However, we have noted in Section 3 that the bounds obtained using interval arithmetic are generally not tight. This is especially the case when the total amount of rotation is very large over one single time step (interval arithmetic produces tight bounds when the motion is a pure linear translation thanks to the monotony of the functions involved). In order to reduce the conservativeness of the test, which would lead to declare that the bounding volumes overlap too often and would make us loose the benefit of using bounding-volume hierarchies, it is best to subdivide the time interval one or several times when the total amount of rotation is very large. The cost of replacing the single test by several tests on smaller time sub-intervals is usually compensated by the early culling, which prevents the need to unnecessarily go further down the hierarchies of bounding volumes.

For articulated bodies, an intermediate culling step can be added in order to prevent the increased conservativeness of interval arithmetic when the depth of the articulated model increases [RKLMO4].

## 6. Handling constraints

Once the contact information has been computed for the current timestep, the dynamics solver needs to be able to determine an appropriate response. In an event-driven multi-body dynamics simulation, two different problems have to be solved:

**Collision response problem** When a collision occurs, the simulator must determine the objects' *post-impact velocities* from their pre-impact velocities and their dynamic properties.

**Constraint collision problem** When the collision problem has been solved, the simulator has to compute the objects' *constrained accelerations* from their unconstrained accelerations (the ones the objects would have if there weren't any constraints acting on them) and the geometrical constraints imposed by the transient contacts (those for which the relative velocity is zero, *i.e.* those which have a non-zero duration).

Numerous approaches have been suggested to solve both problems and traditional algorithms include penalty methods [MW88, KZ90], impulse-based methods [MC95], constraint-based methods [Lot84, Bar94, Bar95, AP97, MS01], time-stepping methods [ST96, APS99], and iterative methods [GBF03, KEP05]. Penalty methods are generally used when no precise contact information is available, as they compute the contact forces from the amount of interpenetration between contacting objects. Continuous collision detection naturally provides all the necessary contact information: the contact position, the contacting elements and the contact normal. Consequently, simulation methods which make use of this contact information are a natural choice.

Most constraint-based methods formulate both dynamics problems as a linear complementarity problem (LCP) in the *contact-space*, which relates contact forces and accelerations. For example, in the frictionless constrained motion problem, the LCP expresses the relation between the normal contact forces and the relative normal accelerations at the contact points. If  $\mathbf{f}$  and  $\mathbf{a}_{cp}$  are two vectors in  $\mathbb{R}^m$  describing the normal contact forces (in a frictionless system, the contact forces are normal to the contact plane) and the normal relative accelerations to the  $m$  contact points, then there exists an  $m \times m$  matrix  $\mathbf{A}$  and a vector  $\mathbf{b}$  in  $\mathbb{R}^m$  such as:

$$\begin{cases} \mathbf{a}_{cp} = \mathbf{A}\mathbf{f} + \mathbf{b} \\ \mathbf{a}_{cp} \geq 0 \\ \mathbf{f} \geq 0 \\ \mathbf{a}_{cp}^T \mathbf{f} = 0 \end{cases} \quad (26)$$

The complementarity relation  $\mathbf{a}_{cp}^T \mathbf{f} = 0$  expresses the fact that, for each contact point, either the relative normal acceleration is non-zero (the contact breaks) and then the normal contact force is zero, or the contact force is non-zero (objects remain in contact) and therefore the relative normal acceleration is zero.

The matrix  $\mathbf{A}$  is traditionally computed from the constraints created by the contact points [Bar94, RK97]. Let  $i$  and  $j$  denote two contacting objects.  $I$  is a contact point,  $\mathbf{n}$  is the contact normal directed from  $j$  to  $i$ . Depending on the object it belongs to,  $I$  is denoted  $I_i$  or  $I_j$ .

Note that this distinction is necessary to establish the constraint equations. Although these two points are identical in theory, it is not the case in practice, for example because of the finite precision of the computations. Moreover, we will see in the next section that the method we use to combine the continuous collision detection algorithms to the dynamics algorithms requires us to maintain a distinction between  $I_i$  et  $I_j$  which become, in practice, the closest points belonging to the polyhedral primitives. In a vertex/face case, for example,  $I_i$  is the vertex and  $I_j$  is the point in the face which is the closest to  $I_i$ .

With this notation, the *non-penetration constraint* on the relative normal acceleration of  $I$  is [Bar94]:

$$(\mathbf{a}_i(I_i) - \mathbf{a}_j(I_j)) \cdot \mathbf{n} + 2 \cdot (\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)) \cdot \frac{d\mathbf{n}}{dt} \geq 0 \quad (27)$$

Similarly, a *collision response constraint* on the relative normal velocity can be derived when a collision occurs, to solve the collision response problem:

$$(\mathbf{v}_i^+(I_i) - \mathbf{v}_j^+(I_j)) \cdot \mathbf{n} \geq -e(\mathbf{v}_i^-(I_i) - \mathbf{v}_j^-(I_j)) \cdot \mathbf{n} \quad (28)$$

where  $e$  is the restitution coefficient at the contact point.

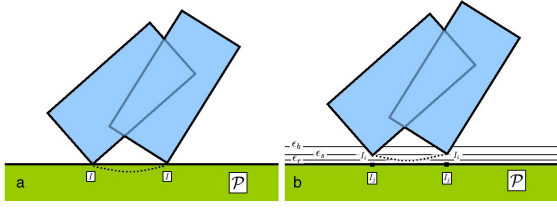
We have shown in [RKC02a] that for frictionless systems Gauss' least constraint principle provides a *motion-space* formulation of both dynamics problems, through a projection problem which relates the object's accelerations or velocities and the contact forces. Although the two formulations are mathematically equivalent, the motion-space formulation presents several algorithmic benefits: it is better conditioned, always sparse, requires less memory and allows to avoid some redundant computations performed by an algorithm operating in the contact-space. An experimental comparison has suggested that an algorithm operating in the motion-space takes advantage of sparsity to perform increasingly better than a contact-space algorithm as the average number of contact points per object increases. As a result, our system uses a motion-space algorithm based upon Wilhelmssen's projection algorithm [Wil76].

Whatever dynamics solver is used in combination with the continuous collision detection module, however, some care has to be taken in order to couple both modules. One possible solution is presented in the next section.

## 7. Combining continuous collision detection and response

Because we use an arbitrary in-between motion to interpolate successive objects' positions, we need to permanently maintain a small *security distance* between contacting objects. Most of the time, indeed, the arbitrarily chosen in-between motion does not satisfy the dynamics constraints during the successive time intervals.

Consider for example the case of a rectangular box in transient contact with a plane surface, as visible in Figure 11.a. In this example, the contact point  $I$  should be permanently on the contact plane  $\mathcal{P}$  during the considered time interval  $[t_n, t_{n+1}]$ . However, the real object motion during this time interval is replaced by an arbitrarily fixed motion which, although it preserves the object's positions at times  $t_n$  and  $t_{n+1}$ , doesn't guarantee that the point  $I$  remains on the plane  $\mathcal{P}$  during the time interval. Depending on the initial and final positions of the box, which impose the in-between motion used for the collision detection, this point  $I$  might indeed violate the non-penetration constraint *immediately after*  $t_n$ , leading to the detection of a collision at time  $t_n$ . In order to



**Figure 11:** a: the arbitrary in-between motion used for the collision detection doesn't satisfy the non-penetration constraint during the time interval  $[t_n, t_{n+1}]$ . b: contacting objects are maintained slightly distant from each other (from [Red04a]).

avoid that the simulator remains blocked at time  $t_n$  or, generally, that it detects these collisions artificially created by the use of an arbitrary in-between motion too often, we introduce a security distance  $\epsilon_s$  between contacting objects, as shown in Figure 11.b. As a result, we consider that each contact point which has been determined by the continuous collision detection module is characterized by two points  $I_i$  and  $I_j$  which are the closest points on the polyhedral primitives. These closest points are easily updated whenever the geometrical constraints are modified, as long as a contact is considered to be active.

In order to maintain the security distance between contacting objects over time, the constraints imposed to accelerations or velocities are modified slightly. For example, when the local distance between the two objects (i.e. the one separating the two points  $I_i$  and  $I_j$  which characterize the contact point) is smaller than the security distance  $\epsilon_s$ , the non-penetration constraint (27) becomes:

$$\begin{aligned} & \mathbf{a}_i(I_i) - \mathbf{a}_j(I_j) \cdot \mathbf{n} + \\ & 2 \cdot (\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)) \cdot \frac{d\mathbf{n}}{dt} \geq K + k(\epsilon_s - d) \end{aligned} \quad (29)$$

where  $k$  is a coupling constant, and  $d$  is the distance between the points  $I_i$  and  $I_j$ .

Two other values,  $\epsilon_b$  and  $\epsilon_r$ , are used in combination with the security distance  $\epsilon_s$ . The first one,  $\epsilon_b$ , larger than  $\epsilon_s$ , is the value beyond which the simulator declares that the contact between the two objects is broken (thus removing the need to update  $I_i$  and  $I_j$ ). The second,  $\epsilon_r$ , smaller than  $\epsilon_s$ , is an alert value below which the simulator stops the simulation time and enters a *repositioning cycle*, because it didn't manage to maintain the objects at a sufficient distance from each other during the simulation. This repositioning cycle, similar to the one introduced by Baraff in [Bar95], is performed between two successive frames by computing the smallest objects' displacements which satisfy the *repositioning constraints*. The repositioning constraints are similar to collision response constraints (28) which become:

$$(\mathbf{v}_i(I_i) - \mathbf{v}_j(I_j)) \cdot \mathbf{n} \geq \epsilon_s - d. \quad (30)$$

Once these velocities are computed, they are used to move objects (while detecting collisions). As long as there exists a contact for which the local distance is below the repositioning distance  $\epsilon_r$ , a new repositioning step is done. We have observed that repositioning cycles are generally very short, nearly imperceptible by the user, and don't hinder the progression of the interactive simulation.

Let's note that the repositioning problem is very similar to the collision response problem, since it consists in computing the smallest possible velocities, in the velocity-space, among those which satisfy the repositioning constraints.

Let's note finally that repositioning objects doesn't correspond to a physical phenomenon, and can add energy in the system. In order to avoid this, it may be useful to slightly decrease the objects' velocities after a repositioning step. Note however that this problem doesn't occur in a first-order (quasi-static) simulation, since objects' velocities are zeroed at each frame. More generally, the repositioning problem is similar to a constraint stabilization problem, for which a classic solution is the one proposed by Baumgarte [Bau72]. In this case, however, constraints are unilateral ones. Moreover, the more difficult problem of unilateral constraint stabilization is greatly facilitated by the continuous collision detection.

## 8. Continuous collision detection for motion planning and haptics

In this section, we demonstrate how continuous collision detection has been used to design novel algorithms for motion planning of articulated bodies and six degree-of-freedom haptic display of rigid bodies.

### 8.1. Motion planning

We have recently presented an efficient and practical local planning method in contact-space, i.e. a planning method which uses continuous collision detection in order to sample the surface of the C-obstacles (the obstacles in configuration space) [RL05]. Some of the most well-known motion planning methods rely on *probabilistic roadmaps* (PRMs) [KL94, KSLO96, Ove92, OS94]. The simplest PRM algorithms generate a set of configuration in the free space. The roadmap graph is used to generate a path between the start and goal configurations. However, the efficiency of these planners can degrade in configurations containing narrow passages or cluttered environments. In such cases, a significant fraction of randomly generated configurations are not in the free space. Moreover, it is difficult to generate a sufficient number of samples in the narrow passages or *connect* all the nodes in the free space using local planning methods.

Traditionally, connecting two sampled configuration in C-space has often been performed using either discrete collision detection methods, which sample a path connecting

the two nodes (most often a straight line) with some finite resolution, or exact collision checking [Can86, SSL02]. Given the contact information provided by a continuous collision detection module, this local planning step can be enhanced in the following ways:

**Contact-space sampling:** Since continuous collision detection computes the time of impact as well as the contact features (contact position and contact normal), it is possible to efficiently and robustly sample the contact space, and not only the free space.

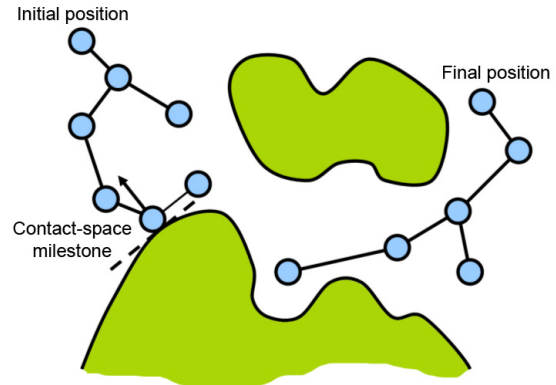
**Constrained sampling:** Given the precise contact information provided by the continuous collision detection algorithm, a piecewise-linear characterization of the local tangent space of C-Obstacles around the nodes in the contact space can be determined. This piecewise-linear characterization can be used to constrain the search of a new node when expanding the roadmap from a node belonging to the contact space.

Figure 12 demonstrates these extensions in a two-dimensional configuration space. A *contact-space milestone*, i.e. a configuration in which the robot is in contact with the environment, has been found by the continuous collision detection module. The corresponding contact information (especially, the contact position and the contact normal), is used to constrain the search of a valid neighboring configuration, in order to expand the roadmap. Using this strategy, we have been able to observe up to 70 times performance improvement when adding our contact-space sampling and constrained sampling methods to a freely available planner (the Stanford MPK planner) in preliminary benchmarks involving cluttered and narrow passages [RL05].

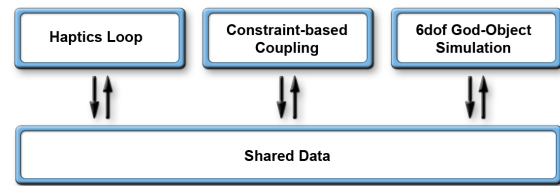
## 8.2. Six degree-of-freedom haptic display of rigid bodies

Our continuous collision detection solver has recently been used to help design a novel algorithm six degree-of-freedom haptic display of rigid bodies [ORC05] (cf. Figure 13). The algorithm combines continuous collision detection and constraint-based quasi-statics to generalize the well-known god-object method for haptic exploration of a rigid body with a three degree-of-freedom haptic device [ZS95]. The new algorithm preserves the desirable properties of the original god-object method:

- Thanks to the continuous collision detection module, the god-object never penetrates the environment obstacles, which is known to improve the perceived stiffness of the haptic feedback [SBB96].
- The constraint-based forces applied to the user are always orthogonal to the constraints, and do not suffer from the artifacts typically encountered in previous methods (e.g. forces felt at a distance, sticking, artificial friction). The



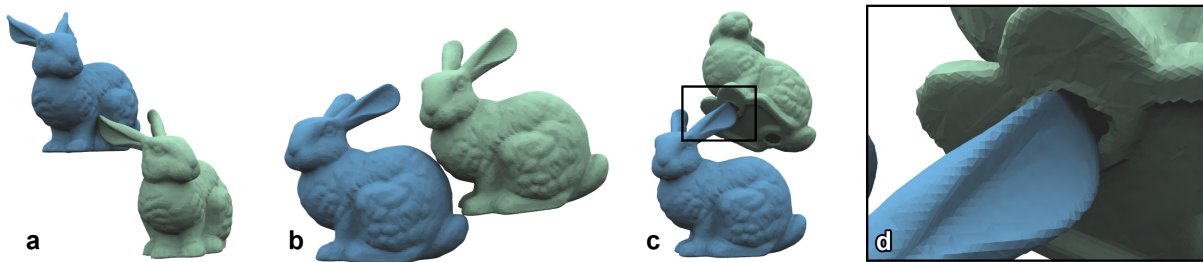
**Figure 12: Local planning in the contact-space.** Continuous collision detection allows to extend a probabilistic motion planner by allowing efficient and robust sampling of the contact-space (i.e. the surface of the obstacles in the configuration space). Furthermore, the contact information can be used to constrain the search of valid neighboring nodes, when expanding the roadmap. This has been shown to greatly improve the performance of a planner in the presence of narrow passages, by allowing the robot to slide on the surface of the obstacles (cf. [RL05]).



**Figure 14: Schematic representation of our method.** Our method for haptic display of six degree-of-freedom manipulation of rigid bodies is divided in three asynchronous blocks (from [ORC05]).

new method contributes to enhance the realism of the haptic feedback, as it has been shown that an incorrect force direction perturbs the perceived orientation of the haptic surface [SPBS00].

Our algorithm is divided in three asynchronous loops: (a) the god-object simulation loop, which updates the configuration of the god-object based on the configuration of the haptic device and the environment obstacles; (b) the constraint-based coupling loop, which determines the constraint-based force applied to the user based on the configurations of the god-object and the haptic device, as well as the current set of contact points and normals; (c) the haptics loop, which controls an impedance-like haptic device which reads the force that has to be applied to the user and writes the current configuration of the haptic device (cf. Figure 14). The separation into asynchronous processes allows us to satisfy the different



**Figure 13: Haptic interaction with Stanford Bunnies.** The user manipulates the green bunny. **a:** the ear of the green bunny slides in a ridge of the blue bunny. **b:** continuous collision detection and constraint-based quasi-statics allows the manipulated object to precisely contact and slide on the obstacles. **c-d:** our method provides the user with the ability to precisely feel the contact between pairs of triangles, resulting in highly detailed haptic display of contacting rigid bodies (from [ORC05]).

update rates required by the haptic and the visual displays, and go around the potentially lower update rates of the collision detection module (since the complexity of collision detection is *output-dependent*).

This novel approach has been implemented on a 3.2 GHz Xeon biprocessor and successfully tested on complex benchmarks. The constraint-based force applied to the user, which handles any number of simultaneous contact points, is typically computed within a few microseconds, while the update of the configuration of the rigid god-object is performed within a few milliseconds for rigid bodies containing up to tens of thousands of triangles. For more details, we refer the reader to [ORC05].

## 9. Conclusion

Because they seem able to alleviate some of the shortcomings of traditional (discrete) collision detection methods, continuous collision detection methods have recently generated a greater interest, and research, with applications in several domains. Some examples include graphics, virtual reality, CAD/CAM, and games (e.g. [VDB05]). A freely available software library ("Bullet", with source code) maintained and developed by E. Coumans and several developers provides continuous collision detection modules, and a corresponding discussion forum (<http://www.continuousphysics.com/Bullet>) allows researchers and developers to discuss and share information about continuous collision detection and physics. Furthermore, several commercial physics libraries are beginning to offer some form of continuous collision detection (e.g. Ageia<sup>TM</sup>, Havok<sup>TM</sup>, etc.).

In order to accompany these exciting developments, some more work is needed so that continuous collision detection can be performed in combination with some of the most recent simulation paradigms (e.g. time-stepping methods, iterative solvers [GBF03], etc.) and handle e.g. numerous bodies simultaneously. Other active research topics include specific optimizations or extensions for new object classes (e.g. sub-

division surfaces, closed manifolds, etc.), continuous collision detection methods for augmented reality, novel resolution methods, etc. We believe that the need for robust physics libraries (for example in games) should stimulate further research in those areas.

## References

- [AP97] M. Anitescu and F. A. Potra. Formulating Dynamic Multi-Rigid-Body Contact Problems with Friction as Solvable Linear Complementarity Problems. *Nonlinear Dynam.* 14 (1997), no. 3, 231–247.
- [APS99] M. Anitescu, F. A. Potra and D. E. Stewart. Time-stepping for Three-dimensional Rigid Body Dynamics. *Computational Modeling of Contact and Friction. Comput. Methods Appl. Mech. Engrg.* 177 (1999), no. 3-4, 183–197.
- [Bar90] D. Baraff. Curved Surfaces and Coherence for Non-penetrating Rigid Body Simulations. In *Computer Graphics (Proc.SIGGRAPH)*, volume 24, pages 19-28. ACM, August 1990.
- [Bar94] D. Baraff. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *SIGGRAPH 94 Conference Proceedings, Annual Conference Series*, pp 23-34. ACM SIGGRAPH, Addison Wesley, 1994.
- [Bar95] D. Baraff. Interactive Simulation of Solid Rigid Bodies. *IEEE Computer Graphics and Applications*, 15(3), pp 63-75, May 1995.
- [Bau72] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Comp. Meth. in Appl. Mech. and Eng.*, 1:1-16, 1972.
- [BS02] G. Bradshaw and C. O'Sullivan. Sphere-Tree Construction using Dynamic Medial Axis Approximation. In *Proceedings of ACM Symposium on Computer Animation 2002*.
- [BW97] D. Baraff and A. Witkin. *Partitioned Dynamics*, Technical Report CMU-RI-TR-97-33, Robotics Institute, Carnegie Mellon University, 1997.

- [Cam90] S. A. Cameron. collision detection by four-dimensional intersection testing. *IEEE Trans. Robotics and Automation*, 6, 3 (June 1990), pp 291-302.
- [Can86] J. F. Canny. *collision detection for moving polyhedra*. *IEEE Trans. Patt. Anal. Mach. Intell.* 8,2 (March 1986), pp 200-209.
- [Cha1831] M. Chasles. Note sur les Propriétés Générales du Système de Deux Corps Semblables Entre Eux, Placés d'une Manière Quelconque Dans l'Espace; et sur le Déplacement Fini, ou Infiniment Petit d'un Corps Solide Libre. *Bulletin des Sciences Mathématiques de Ferussac*, XIV, pp. 321-336. 1831.
- [CSB95] 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.
- [Duf92] T. Duff. *Interval Arithmetic and Recursive Sub-division for Implicit Functions and Constructive Solid Geometry*. *Computer Graphics*, 26(2), July 1992, pp. 131-138.
- [FMM77] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations*, Prentice Hall, Inc., Englewood Cliffs, 1977.
- [GRLM03] N. Govindaraju, S. Redon, Ming C. Lin and Dinesh Manocha. CULLIDE: Interactive Collision Detection between Complex Models in Large Environments using Graphics Hardware. *ACM SIGGRAPH/ Eurographics Graphics Hardware Proceedings*, 2003.
- [Got99] S. Gottschalk. *collision queries using oriented bounding boxes*. PhD Thesis. 1999.
- [GLM96] 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.
- [GMELM00] 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.
- [GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics*, 22(3), pp. 871-878, 2003. (SIGGRAPH 2003).
- [Har99] Michael Hardt. *Multibody Dynamical Algorithms, Numerical Optimal Control, with Detailed Studies in the Control of Jet Engine Compressors and Biped Walking* Department of Electrical and Computer Engineering (Intelligent Systems, Robotics, and Control) University of California San Diego, June 1999.
- [Hub95] P. M. Hubbard. *collision detection for interactive graphics applications*. Ph.D. Thesis, April 1995.
- [JTT01] P. Jiménez, F. Thomas and C. Torras. 3D collision detection: a survey. *Computers and Graphics*, 25 (2), pp. 269-285, (April 2001), Pergamon Press / Elsevier Science.
- [KEP05] D. M. Kaufman, T. E. and D. K. Pai. Fast Frictional Dynamics for Rigid Bodies. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3), August 2005.
- [KL94] L. Kavraki and J. C. Latombe. Randomized pre-processing of configuration space for fast path planning. *IEEE Conference on Robotics and Automation*, pages 2138–2145, 1994.
- [KSLO96] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580, 1996.
- [Kea96] R. B. Kearfott. Interval Computations: Introduction, Uses, and Resources, *Euromath Bulletin* 2 (1), pp. 95-112 (1996).
- [KZ90] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. In *Computer Graphics (Proc. SIGGRAPH)*, volume 24, pages 29-38. *ACM*, August 1990.
- [KOLM02] Young J. Kim, Miguel A. Otaduy, Ming C. Lin, Dinesh Manocha. Fast Penetration Depth Computation for Physically-based Animation. *ACM Symposium on Computer Animation*, July 21-22, 2002.
- [KR03] B. Kim and J. Rossignac. Collision Prediction for Polyhedra under Screw Motions. *ACM Symposium in Solid Modeling and Applications*, pp. 4-10, June 2003.
- [KHMSZ98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral and 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.
- [LKCGC01] A. Lécuyer, A. Kheddar, S. Coquillart, L. Graux, and P. Coiffet, A Haptic Prototype for the Simulations of Aeronautics Mounting/Unmounting Operations. *IEEE Int. Workshop on Robot-Human Interactive Communication*, Bordeaux and Paris, France, 2001.
- [LG98] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *IMA Conference on Mathematics of Surfaces (San Diego (CA), May 1998)*, vol. 1, pp. 602–608.
- [Lot84] P. Lötstedt. Numerical simulation of time-dependent contact friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, vol. 5, no. 2, pp. 370- 393, 1984.
- [MS01] V. J. Milenkovic and H.Schmidl. Optimization Based Animation. *SIGGRAPH 2001*
- [MC95] B. Mirtich and J. Canny. Impulse-based Simulation of Rigid Bodies. In *Proceedings of Symposium on Interactive 3D Graphics*, 1995.

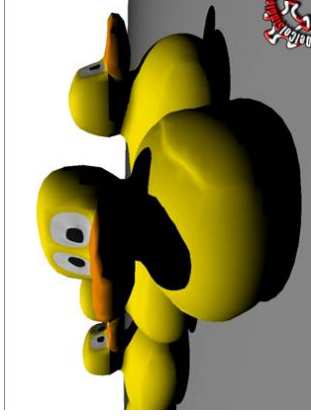
- [Moo62] R. E. Moore. Interval analysis and automatic error analysis in digital computation. PhD Thesis, Stanford University, October 1962.
- [MW88] M. Moore and J. Wilhelms. collision Detection and Response for Computer Animation. In *Computers Graphics (Proceedings of SIGGRAPH 88)*, Annual Conference Series, pp 289-298. ACM SIGGRAPH, August 1988.
- [ORC05] M. Ortega, S. Redon and S. Coquillart. A Six Degree-of-Freedom God-Object Method for Haptic Display of Rigid Bodies. In *Proceedings of IEEE International Conference on Virtual Reality*, 2006.
- [ODGK03] O'Sullivan, C. Dingliana, J., Giang, T. Kaiser. Evaluating the Visual Fidelity of Physically Based Animations. M.K. *ACM Transactions on Graphics*. 22(3), *Proceedings of SIGGRAPH 2003*, July 2003.
- [Ove92] M. H. Overmars. A random approach to motion planning. Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, October 1992.
- [OS94] M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Algorithmic Foundations of Robotics*. A. K. Peters, Wellesley, MA, 1995.
- [Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pp 3324-3329, 1994.
- [Red04a] S. Redon. Fast Continuous Collision Detection and Handling for Desktop Virtual Prototyping. To appear in *Virtual Reality Journal* (Springer Verlag).
- [Red04b] S. Redon. Continuous Collision Detection for Rigid and Articulated Bodies. *ACM SIGGRAPH Course Notes*, 2004.
- [RKC00] 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.
- [RKC01] S. Redon, A. Kheddar and S. Coquillart. CONTACT: arbitrary in-between motions for continuous collision detection. In *Proceedings of IEEE ROMAN'2001*, Sep. 2001.
- [RKC02a] 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.
- [RKC02b] S. Redon, A. Kheddar and S. Coquillart. Fast Continuous collision Detection between Rigid Bodies. In *Proceedings of Eurographics 2002*. September 2002.
- [RKC02c] S. Redon, A. Kheddar and S. Coquillart. Hierarchical Back-Face Culling for collision Detection. In *proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. September 2002.
- [RKLM04] S. Redon, Young J. Kim, Ming C. Lin and Dinesh Manocha. Fast Continuous Collision Detection for Articulated Models. In *Proceedings of IEEE Solid Modeling 2004*.
- [RKLMT04] S. Redon, Young J. Kim, Ming C. Lin, Dinesh Manocha and Jim Templeman. Interactive and Continuous Collision Detection for Avatars in Virtual Environment. In *Proceedings of IEEE International Conference on Virtual Reality 2004*.
- [RL05] S. Redon and M. C. Lin. Practical Local Planning in the Contact Space. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2005.
- [RK97] D. Ruspini and O. Khatib. Collision/Contact Models for the Dynamic Simulation of Complex Environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems: IROS'97*.
- [RKK97] D. C. Ruspini, K. Kolarov and O. Khatib. The Haptic Display of Complex Graphical Environments. *Computer Graphics Proceedings, SIGGRAPH 97* pp 345-52
- [SPBS00] W. L. Sachtler, M. R. Pendexter, J. Biggs, and M. A. Srinivasan. Haptically perceived orientation of a planar surface is altered by tangential forces. *Fifth Phantom User's Group Workshop*, 2000.
- [SSL05] F. Schwarzer, M. Saha, and J.-C. Latombe. Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. *IEEE Tr. on Robotics and Automation*, June 2005.
- [SSL02] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In *Workshop on Algorithmic Foundations of Robotics (WAFR)*, Dec. 2002.
- [SIS96] Yair Shapira, Moshe Israeli, Avram Sidi. Towards Automatic Multigrid Algorithms for SPD, Nonsymmetric and Indefinite Problems. *Journal on Scientific Computing* Volume 17, Number 2 pp. 439-453, 1996.
- [Sny92] J. Snyder. Interval analysis for Computer Graphics. *Computer Graphics*, 26(2), pages 121-130, July 1992.
- [SWFCB93] 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.
- [SBB96] M. A. Srinivasan, G. L. Beauregard, and D. L. Brock. The impact of visual information on the haptic perception of stiffness in virtual environments. *ASME Winter Annual Meeting*, November 1996.
- [ST96] D. E. Stewart and J. C. Trinkle. An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic collisions and Coulomb Friction. *International Journal of Numerical Methods in Engineering*, 39:2673-2691, 1996.

- [TSHBS03] N. Tarrin, S. Coquillart, S. Hasegawa, L. Bouguila, M. Sato. The Stringed Haptic Workbench: a New Haptic Workbench Solution. In proceedings of Eurographics, September 2003.
- [VDB98] G. Van den Bergen. *Efficient collision detection of complex deformable models using AABB trees*. Journal of Graphics Tools, 2(4):1-14, 1997.
- [VDB05] G. Van den Bergen. Continuous Collision Detection of General Convex Objects Under Translation. Game Developer Conference, 2005.
- [VHBZ90] 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.
- [Wil76] 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.
- [Zac98] Gabriel Zachmann. Rapid Collision Detection by Dynamically Aligned DOP-Trees. Proc. of IEEE Virtual Reality Annual International Symposium. VRAIS '98.
- [ZS95] C. B. Zilles and J. K. Salisbury. A constraint-based god-object method for haptic display. International Conference on Intelligent Robots and Systems, Proceedings, August 1995.



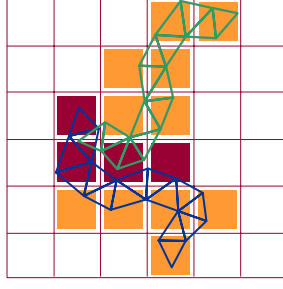


## Collision Detection for Volumetric Deformable Objects



## Spatial Partitioning - Idea

- space is divided up into cells
- object primitives are placed into cells
- object primitives within the same cell are checked for collision
- pairs of primitives that do not share the same cell are not tested (trivial reject)

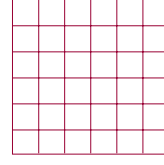


## Outline

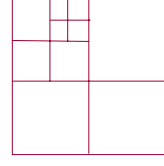
- introduction to spatial data structures
- binary space partitioning trees
- voxel grids
- spatial subdivision with graphics hardware



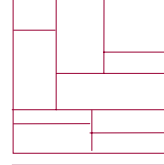
## Spatial Data Structures



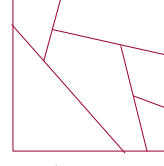
voxel grid



octree



k-d tree



BSP-tree

- cells maintain references to primitives intersecting the cell
- information is updated for each object transformation
- octree, k-d tree, and BSP-tree are object-dependent
- voxel grid is object-independent





## Voxel Grid

- space partitioning into (uniform) rectangular, axis-aligned cells
- primitives per cell are found by
  - scan conversion of primitives to the grid or
  - scan conversion of AABBs of the primitives
- fast cell access
- optimal cell size?
  - large cells increase the number of primitives per cell
  - small cells cause spreading of primitives to a large number of cells
- less efficient in case of non-uniform primitive distribution



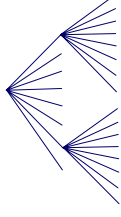
Freiburg University - Computer Science Department - Computer Graphics



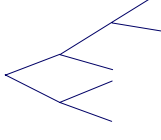
## Octree and k-d Tree

- hierarchical structures
- space partitioning into rectangular, axis-aligned cells
- root node corresponds to AABB of an object
- internal nodes represent subdivisions of the AABB
- leaves represent cells which maintain primitive lists

octree



k-dimensional binary tree



Freiburg University - Computer Science Department - Computer Graphics



## Octree and k-d Tree

- uniform or non-uniform subdivision
- adaptive to local distribution of primitives
  - large cells in case of low density of primitives
  - small cells in case of high density
- dynamic update
  - cells with many primitives can be subdivided
  - cells with less primitives can be merged



Freiburg University - Computer Science Department - Computer Graphics



## Outline

- introduction to spatial data structures
- **binary space partitioning trees**
- voxel grids

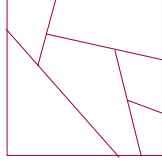


Freiburg University - Computer Science Department - Computer Graphics



## BSP Tree

- binary space partitioning tree
- hierarchical structure
- space is subdivided by means of arbitrarily oriented planes
- generalized k-d tree
- space partitioning into convex cells
- discrete-orientation BSP trees DOBSP (finite set of plane orientations)

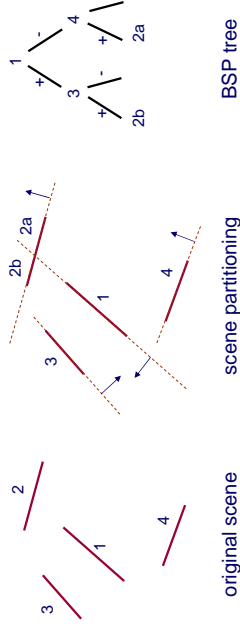


Freiburg University - Computer Science Department - Computer Graphics



## BSP Tree for Rendering

- [Henry Fuchs et al. 1980] proposed a visible surface algorithm using a pre-computed BSP

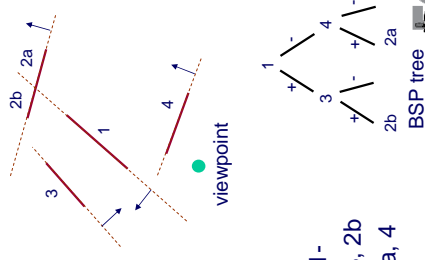


Freiburg University - Computer Science Department - Computer Graphics



## BSP Tree for Rendering

- for a given viewpoint
  - render far branch
  - render root (node) polygon
  - render near branch
- recursively applied to sub-trees
- back to front rendering
- example: viewpoint is in 1- rendering of 1+, 1, 1-
- rule recursively applied to 1+ and 1-
- viewpoint is in 3+ -> rendering of 3, 2b
- viewpoint is in 4- -> rendering of 2a, 4

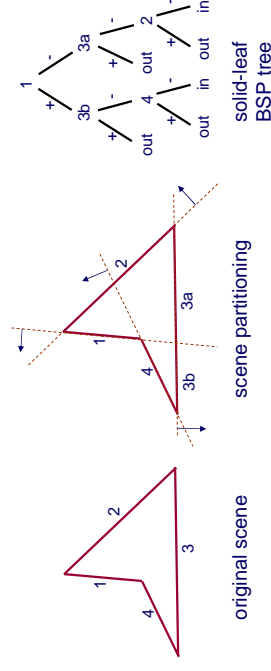


Freiburg University - Computer Science Department - Computer Graphics



## BSP Tree for Collision Detection

- BSP trees can be used for inside / outside classification of closed polygons

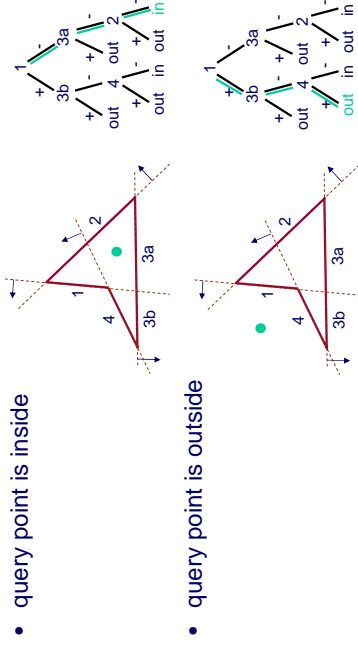


Freiburg University - Computer Science Department - Computer Graphics





## Collision Query



- query point is inside
- query point is outside



## BSP Tree Construction

- keep the number of nodes small
- keep the number of levels small
- introduce arbitrary support planes (especially in case of convex objects, where all polygon faces are in the same half-space with respect to a given face)



## Outline

- introduction to spatial data structures
- binary space partitioning trees
- voxel grids



## Related Approaches

- [Levinthal 1966]
  - 3D grid ("cubing")
  - analysis of molecular structures
  - neighborhood search to compute atom interaction
- [Rabin 1976]
  - 3D grid + hashing
  - finding closest pairs
- [Turk 1989, 1990]
  - rigid collision detection
  - 3D grid + hashing



Cyrus Levinthal, MIT



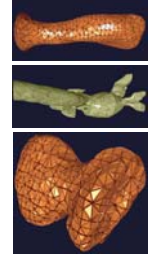


# Deformable Collision Detection

- [Teschner, Heidelberger et al. 2003]
  - collisions and self-collisions for deformable tetrahedral meshes
  - uniform 3D grid
  - non-uniform distribution of object primitives → hashing
  - no explicit 3D data structure
  - analysis of optimal cell size



Epidaure, INRIA

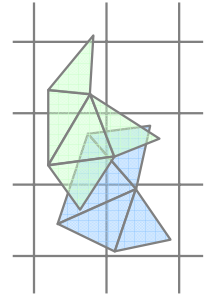


NCCR Co-Me



# Algorithm - Setup

implicit uniform grid:



hash function:

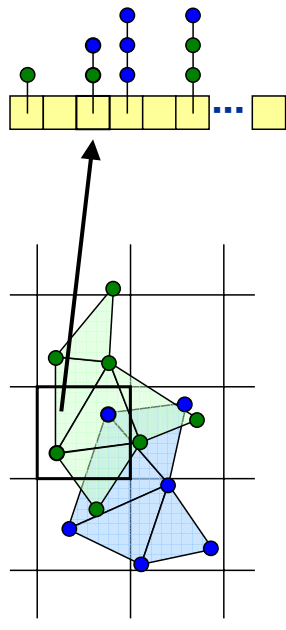
$$H(\text{cell}) \rightarrow \text{hash table index}$$

hash table:



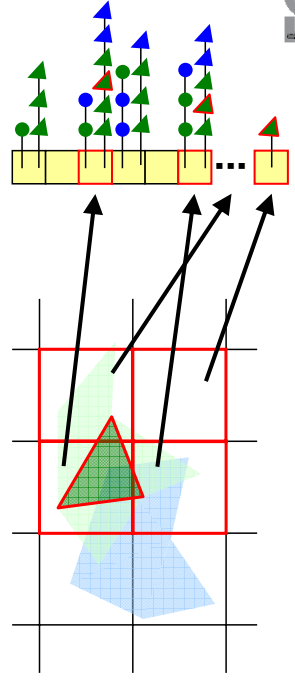
# Algorithm – Stage 1

- all vertices are hashed according to their cell:



# Algorithm – Stage 2

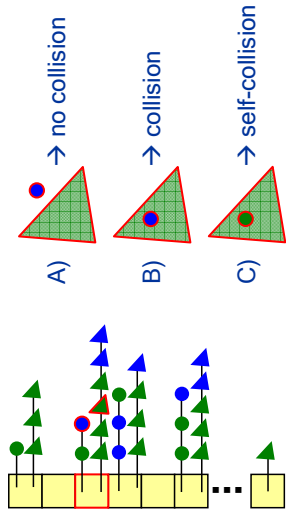
- all tetrahedrons are hashed according to the cells touched by their bounding box



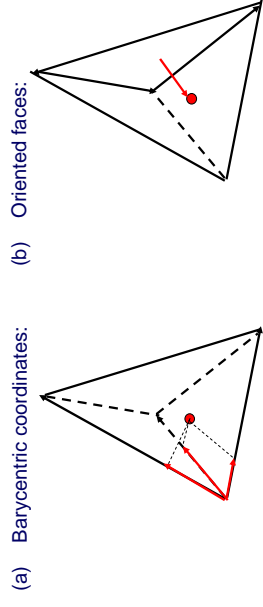
## Algorithm – Stage 3



- vertices and tetrahedrons in the same hash table entry are tested for intersection:



## Vertex-in-Tetrahedron Test



→ Barycentric coordinates faster



## Algorithm – Summary



- stages:
  - hash all vertices
  - hash all tetrahedrons
  - intersection test within each hash table entry

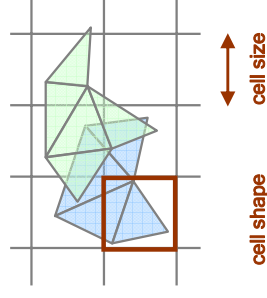
- parameters:
  - grid cell size
  - grid cell shape
  - hash table size
  - hash function



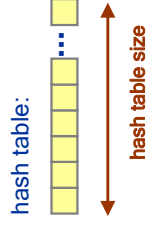
## Algorithm - Parameters



implicit uniform grid:



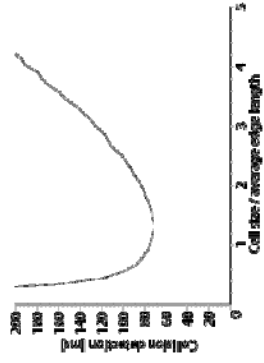
hash function:  
 $H(\text{cell}) \rightarrow \text{hash table index}$



## Grid Cell Size



- [Bentley et al. 1977] suggest a cell size equal to the size of the bounding box of an object primitive
- [Teschner, Heidelberger et al. 2003]



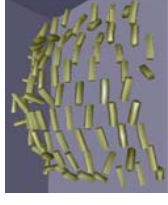
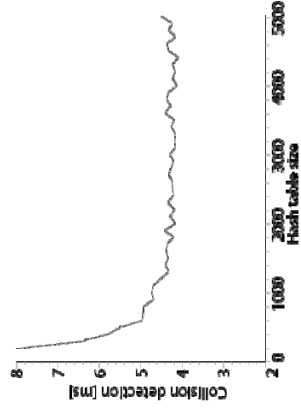
test scenario



## Hash Table Size



- larger hash table reduces hash collisions



test scenario



## Hash Function



$$H(i, j, k) := (i \cdot p_1 \text{ xor } j \cdot p_2 \text{ xor } k \cdot p_3) \bmod n$$

$i, j, k$  : cell coordinates

$p_1, p_2, p_3$  : large primes

$n$  : hash table size



test scenarios



## Performance

- [Teschner, Heidelberger et al. 2003] collision and self-collision detection

objects	tetras	vertices	max time [ms]
100	1000	1200	6
8	4000	1936	15
20	10000	4840	34
2	20514	5898	72
100	50000	24200	174

Pentium 4, 1.8GHz

## Uniform Voxel Grids



- collision and self-collision detection of tetrahedral meshes
- no explicit spatial partitioning (AABB and cells are not explicitly represented)
- hash map
- performance dependent on number of object primitives
- performance independent of number of objects
- algorithm can work with various object primitives



## Uniform Voxel Grids



- simple and efficient technique
- especially interesting for deformable, n-body, and self-collision detection
- in case of non-uniform or sparse spatial distribution of object primitives, hashing is a good choice
- parameters have to be investigated




## References



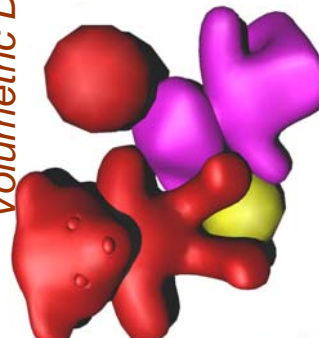
- C. Levinthal, "Molecular model-building by computer," *Scientific American*, pp. 42-52, June 1966.
- J. L. Bentley, D. F. Stanat, E. H. Williams, "The complexity of fixed-radius near neighbor searching," *Inf. Process. Letters*, vol. 6, 209-212, 1977.
- G. Turk, "Interactive collision detection for molecular graphics," TR90-014, University of North Carolina at Chapel Hill, 1990.
- S. Banti, D. Thalmann, "An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies," *Proc. of Eurographics*, pp. 259-270, 1995.
- A. Gregory, M. Lin, S. Gottschalk, R. Taylor, "H-COLLIDE: A framework for fast and accurate collision detection for haptic interaction," TR98-032, University of North Carolina at Chapel Hill, 1998.
- S. Melax, "Dynamic plane shifting BSP traversal," *Proc. Graphics Interface*, pp. 213-220, 2000.
- M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proc. Vision, Modeling, Visualization VMV'03*, pp. 47-54, Nov 2003.
- G. van den Bergen, "Collision Detection in Interactive 3D Environments," Elsevier, Amsterdam, ISBN: 1-55860-801-X, 2004.









# Collision Response for Volumetric Deformable Objects

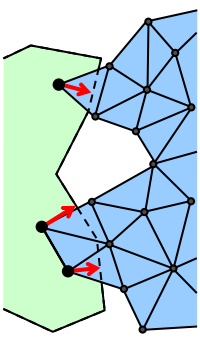


Computer Graphics  
Freiburg University





## Motivation

- compute consistent penetration depth information for all intersecting points of a tetrahedral mesh
- can be used to compute penalty forces which provide realistic collision response for deformable tetrahedral meshes

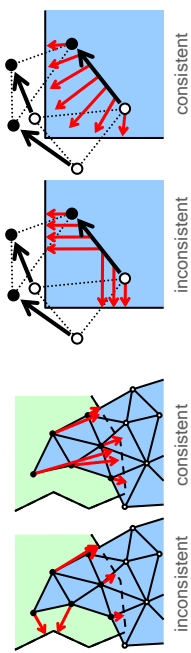


Freiburg University - Computer Science Department - Computer Graphics




## Challenges

- inconsistent penetration depth information due to discrete simulation steps and object discretization
- inconsistent penetration depth results in oscillation artifacts or non-realistic collision response

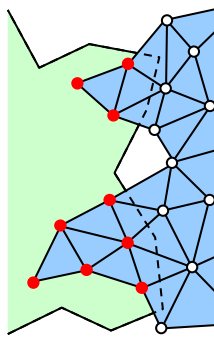


Freiburg University - Computer Science Department - Computer Graphics



## Algorithm – Stage 1

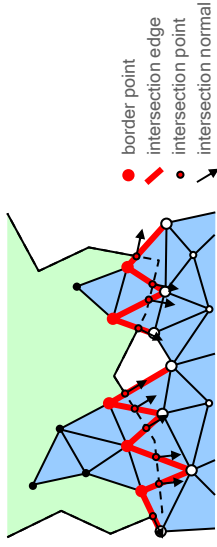
- object points are classified as colliding or non-colliding points → slides on spatial hashing



Freiburg University - Computer Science Department - Computer Graphics

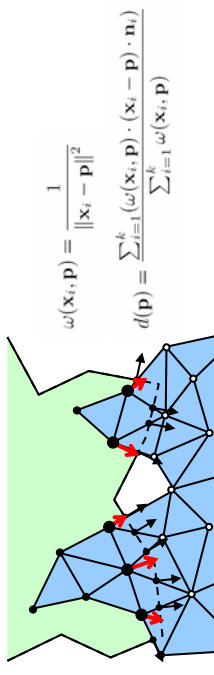
## Algorithm – Stage 2

- border points, intersecting edges, and intersection points are detected → extension of spatial hashing



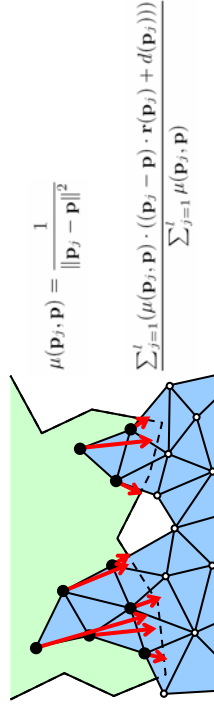
## Algorithm – Stage 3

- penetration depth  $d(\mathbf{p})$  of a border point  $\mathbf{p}$  is approximated using the adjacent intersection points  $\mathbf{x}_i$  and normals  $\mathbf{n}_i$



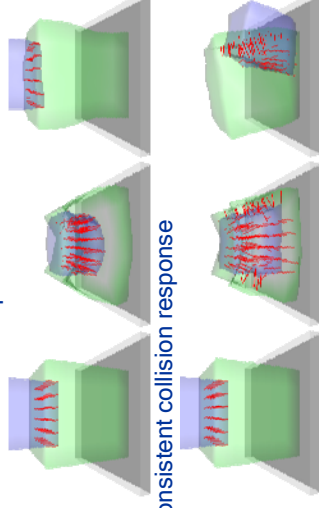
## Algorithm – Stage 4

- consistent penetration depth information at points  $\mathbf{p}_j$  is propagated to other colliding points  $\mathbf{p}$



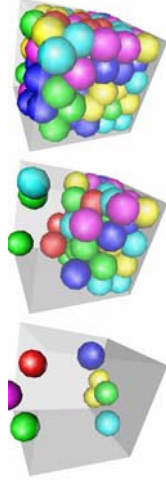
## Results

- consistent collision response
- inconsistent collision response



## Summary

- consistent penetration depth information in case of
  - discrete object representation
  - discrete time simulation
- addresses the problem of discontinuities in magnitude and direction of the penetration depth
- provides realistic penalty-based collision response



Freiburg University - Computer Science Department - Computer Graphics



## References

- B. Heidelberger, M. Teschner et al., "Consistent Penetration Depth Estimation for Deformable Collision Response," *Proc. VMV*, Stanford, USA, 2004.



Freiburg University - Computer Science Department - Computer Graphics





# Collision Detection on Point Clouds

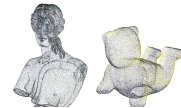
Gabriel Zachmann  
Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)

EG '06, Sep 2006, Vienna, Austria



## Motivation

- Renaissance of points as object representation because of 3D scanners



- Goal:
  - Fast collision detection between 2 given point clouds
  - No polygonal reconstruction

Abstract Point Clouds Conclusion



## Contents

1. Definition of surfaces over point clouds (PCs)
2. Hierarchical PC collision detection
3. Intersection detection at leaf level (stochastic)
4. Kinetization of BVHs (PC hierarchies)

Abstract Point Clouds Conclusion

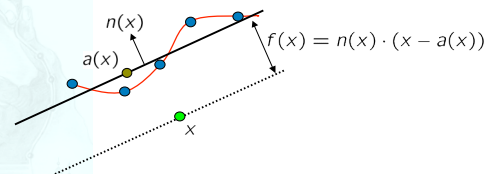


## Definition of the surface

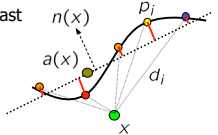
- Idea: assume "distance function"  $f$  from surface, then surface  $S$  is

$$S = \{x \in \mathbb{R}^3 \mid f(x) = 0\}$$

- "Distance" function  $f$ :

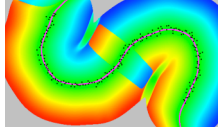
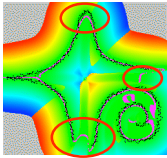


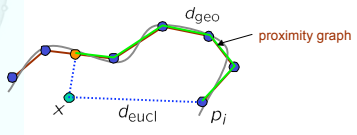
Abstract Point Clouds Conclusion

- Definition of  $n$  by Weighted Least Squares:
 

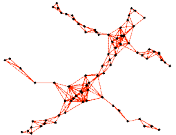
$$\mathcal{A}(x) \in \text{Par}(\mathcal{A}) \Leftrightarrow \sum_{n, |n|=1} \sum_{i=1}^n (n \cdot A_i)^T(x) - (x - P)^2 \cdot A(x) \in \mathbb{R}^3 \times n$$
- Weighting function (kernel):
 
$$\theta(d_i) = e^{-d_i^2/h^2}$$

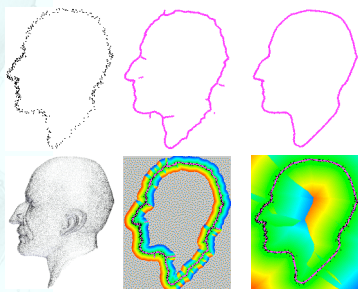
$d_i$  = "distance" between  $p_i$  and  $x$

- Visualization  $f(x)$  using Euclidean distance:
 
- Problems:
 

- Cause and solution:
 
- Which neighborhood graph?
 

→ k-SIG (sphere-of-influence graph)



- Result:
 

### Point Cloud Hierarchy

- Build BVH according to some local criterion (e.g., volume of child BVs)
- Construct subsampling and sphere covering at inner nodes
- Efficient storage

### Sphere covering

- Observation: surface stays (usually) within set of convex hulls  $C$

- Randomised technique:
  - Basic operation: construct random point inside  $\cup C^i$
  - Draw set of samples from orig. point set for sphere centers
  - Compute common radius like Monte-Carlo integration

### Automatic bandwidth detection

- Which bandwidth  $h$  in kernel  $\theta(d_i) = e^{-d_i^2/h^2}$  ?
  - Too small  $\rightarrow$  noisy surface
  - Too large  $\rightarrow$  no details
- Introduce "Sampling Radius" of a point cloud:
 

Given point cloud  $P$  in BV  $A$ ,  $P' \subseteq P$ .  
 Sampling radius  $r(P')$  := smallest radius, so that spheres about  $P'$  with this radius cover surface  $S_p$ , defined by  $P$ , within  $A$ .

- Compute bandwidth  $h$  from  $r(P)$ :
 
$$h(P) = \frac{\eta r(P)}{\sqrt{-\log \theta_\epsilon}}$$

$\eta$  = sampling-independent bandwidth,  
 $\theta_\epsilon$  = small threshold (e.g., machine precision)  
 $r(P)$  = sampling radius of (sub-) point cloud
- Estimate  $r(P)$  :
 
$$r(P') \approx \sqrt{|P|/|P'|} \cdot r(P)$$

### Geometric Proximity Graph

- Here: Sphere-of-Influence graph

- Estimate  $r(P)$  by edge lengths
- Better yet: make bandwidth  $h$  local, i.e.,  $h(x)$ 
  - Replace  $h(P)$  or  $h(P')$  by  $h(x;P)$  and  $h(x;P')$
  - From local neighborhood in graph

Abstract2    Point Clouds    Conclusion

### Result

# points: 148,000 ... # points: 89,106 ... # points: 35,700 ... # points: 62,299 ... # points: 35,556 ... # points: 137,125 ... # points: 157,115

Abstract3    Point Clouds    Conclusion

### Coll.Det. of PCs using Stochastic Sampling

- Given two point clouds A and B (or subsets thereof), construct a sampling of
 
$$\mathcal{Z} = \{x \mid f_A(x) = f_B(x) = 0\}$$
- Overall method:
 

```

      graph LR
      A[A, B] --> B["(p1, p2) ∈ A on different sides of B"]
      B --> C["Approx. intersection points"]
      C --> D["Refined intersection point"]
      
```

Abstract2    Abstract3    Point Cloud Intersection

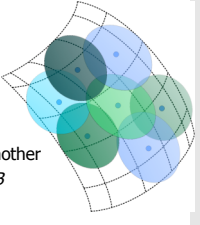
### Algorithm Overview

- Bracket intersections by pairs of points
- Find approximate intersection point (AIP) by interpolation search
- Refine AIP by (randomized) sampling

Abstract2    Abstract3    Point Cloud Intersection

## 1. Root Bracketing


- Goal:
  - The pairs should evenly sample the surface A
  - The two points should not be too far apart
  - Do it *without* explicit spatial data structure
- a) First step: draw number of points (to be one side of the root brackets)
- Thought experiment:
  - Assume surface is covered by  $p$  surfels.
  - Draw enough points from PC A so that each surfel is hit by at least one point
- b) Second step: for each point, try to find another point from A lying on the "other" side of B (completing the brackets)



Point Cloud Intersection

## a) drawing the points

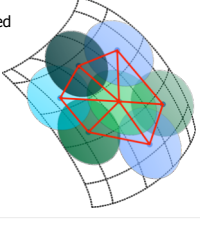
- Avoid spatial data structure (e.g., grid)
- Pursue probabilistic approach: occupy all  $p$  surfels with high probability
- Assumption: PC A is uniformly sampled
- Lemma [WSCG'05] →
  - draw  $O(p \ln p + c \cdot p)$  random and independent points from  $A \cap \text{Vol}(A \cap B)$ ,
  - then each surfel is hit by probability  $P_r = e^{-e^{-c}}$
- Depending on app: choose  $p$  constant or adapt  $p$  to sampling density



Point Cloud Intersection

## b) completing the brackets

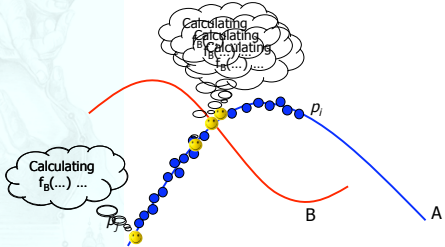
- Use  $f_B(p_i) \cdot f_B(p_j) < 0$  as an indicator
- Test only points  $p_j$  that
  - belong to the randomly chosen points
  - are close to each other
- Only very few points  $p_j$  need to be tested
- Solution: neighborhood graph (e.g., Sphere-of-Influence graph)
- Complexity of finding brackets:
  - $O(d \cdot p \log p)$
  - where  $d = \max.$  out-degree



Point Cloud Intersection

## 2. Interpolation Search

- Find  $\tilde{p} \in A$  along shortest path  $\overline{p_i \tilde{p}_j}$  in the geometric proximity graph, such that  $|f_B(\tilde{p})|$  is minimal.
- Utilize interpolation search →  $O(\log \log m)$ ,  $m = \#$  pts on path



Point Cloud Intersection



### 3. Precise Intersection Points

- Intercept Theorem (assuming surface is not curved):

$$\frac{d_1}{d_1 + d_2} = \frac{r_1}{\|\hat{p}_1 - \hat{p}_2\|}$$

$$x = \frac{1}{d_1 + d_2}(d_2 \cdot \hat{p}_1 + d_1 \cdot \hat{p}_2)$$

- True intersection point is in sphere  $S(x, r)$ , with  $r = \max(r_1, r_2)$
- Sample sphere by points on regular grid

Point Cloud Intersection

### Results

- Benchmarking old vs. new method

time / msec

distance (relative to bbox size)

28,000 points

Point Cloud Intersection

- Theoretical complexity:  $O(\log \log N)$ 
  - Assumptions:  $f(x)$  monotone along paths  $\hat{p}_i \hat{p}_j$ ; and, evenly sampled point cloud.
- Experimental complexity:

avg. time / ms.ec.

number of points per object

buddha

aphrodite

Point Cloud Intersection

### Conclusion of Stochastic Point Cloud CD

- Technique:
  - utilizes a proximity graph for collision detection and surface definition
  - needs no BV hierarchies and no spatial partitioning data structure
  - any BV hierarchy can be augmented by this technique to increase performance
- Runtime:
  - fast (approximate) collision detection
  - overall runtime:  $O(\log \log M)$  in average case
- Quality/resolution of output (intersection points) can be adjusted ( $\rightarrow$  "surfel" radius)

Point Cloud Intersection

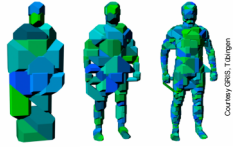
## Conclusions on Stochastic Approach

- Cannot be proven error-free
- Good for plausible & fast simulations
- Interesting alternative to BVHs in specific cases
- Can often be combined with BVHs
- Naturally yield time-critical collision detection

Navigation: Introduction | Abstract2 | Abstract3 | Conclusion | **Finis**

## Kinetic AABB Trees

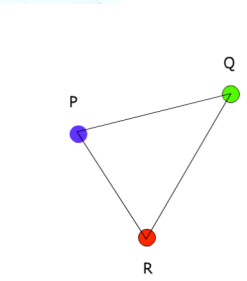
- Not just for collision detection (ray-tracing, occlusion culling, collision detection)



- Pre-processed hierarchy becomes invalid when object deforms  
→ BVH must be rebuilt or updated after deformations

Navigation: **Introduction** | Abstract2 | Abstract3 | Conclusion

## Brute Force Updates



Navigation: **Introduction** | Abstract2 | Abstract3 | Conclusion

## Our Approach

- Observation:
  - Motion in the physical world is normally continuous
  - Changes in the **combinatorial structure** of the BHVs occur only at **discrete time points**
- We store **only** the combinatorial structure of the BVH and use an event-based approach for updating (kinetization)

Navigation: Introduction | **Overview** | Abstract2 | Abstract3 | Conclusion

### Kinetic Updates

Max  
x Q  
y Q

Min  
x P  
y R

Overview Abstract2 Abstract3 Conclusion

### Advantages

- Fewer update operations
- Valid BVHs at every point in time
- Independent of query sampling frequency
- Can handle all kinds of objects
  - Polygon soups, point clouds, and NURBS models
- Can handle insertions/deletions during run-time
- Can handle all kinds of deformations
  - Only a flightplan is required for every vertex
  - These flightplans may change during simulation

Overview Abstract2 Abstract3 Conclusion

### Kinetic AABB Tree

- Kinetization of the AABB tree
- Pre-processing: Build the tree by any algorithm suitable for static AABB trees
  - It is only required that the height of the BVH is logarithmic
- Store with every node the indices of those points that determine the BV
- Initialize the event queue

Overview Abstract2 Kinetic AABB Abstract3 Conclusion

### Simulation Loop

```

while simulation runs
  determine time t of next rendering
  e ← min event in event queue
  while e.timestamp < t
    processEvent(e)
    e ← min event in event queue
  check for collisions (or cast ray, or ...)
  render scene
  
```

Overview Abstract2 Kinetic AABB Abstract3 Conclusion

## Analysis

- Theorem 1:**  
The kinetic AABB tree requires only  $O(n)$  space. Each vertex participates in only  $O(\log n)$  events. Consequently, the kinetic AABB tree can be updated in only  $O(\log n)$  time when an event occurs. Finally, it is a valid BVH at every point of time.
- Theorem 2:**  
Given  $n$  vertices, we assume that the number of intersections for each pair of vertex flightplans is bounded by a constant. Then, the total number of events is in  $O(n \log n)$ , i.e., it is independent of the number of in-between frames or queries.

Abstract2 Kinetic AABB Abstract3 Conclusion

## Results

- Total number of BV updates for complete animation sequence (note logarithmic scale):

Triangles of cloth	Kin-AABB	Bottom-Up
1684	~30	~5000
6891	~100	~15000
15503	~1000	~50000

Abstract2 Abstract3 Experiments Conclusion

- Average update time per frame:

Num in-between frames	Kin AABB	Bottom-Up
20	~3	~17
80	~1	~17
160	~1	~17
320	~1	~17

Abstract2 Abstract3 Experiments Conclusion

- Total time incl. collision detection time:

Triangles of shirt	AABB	Bottom-up
32,198	~10	~40
107,866	~15	~50
3,236,330	~25	~80

Abstract2 Abstract3 Experiments Conclusion

## Summary of kinetic AABB Tree

- Novel data structure for updating an AABB tree under deformation
- Efficiency due to event-based approach
- Theoretical analysis:
  - $O(n \log n)$  for the updates required to keep a BVH valid
- Up to 20 times faster than bottom-up updates in practically relevant scenarios

Abstract2 Abstract3 Conclusion End

## Future Work

- Use our kinetic Data Structures also for continuous collision detection
- Utilize our data structures for other kinds of motion
  - physical based simulations
  - other animation schemes
- Use our KDS for other applications like ray-tracing or occlusion culling

Abstract2 Abstract3 Conclusion End

## References

- Jan Klein and Gabriel Zachmann, "ADB-Trees: Controlling the Error of Time-Critical Collision Detection", *Proc. VMV '03*
- Jan Klein and Gabriel Zachmann, "Time Critical Collision Detection Using an Average Case Approach", *Proc. VRST '03*
- M.C. Lin and J.F. Canny "Efficient Collision Detection for Animation", *Eurographics Workshop on Animation and Simulation '92*
- Stephane Guy and Gilles Debunne, "Monte Carlo Collision Detection", *INRIA Technical Report RR-5136*, 2004
- Laks Raghupathi et al. "Real-time Collisions and Self-Collisions for Virtual Intestinal Surgery", *Surgical Simulation and Soft Tissue Modeling*, pp.38-46, Springer, 2003
- Laks Raghupathi et al. "An Intestinal Surgery Simulator: Real-Time Collision Processing and Visualization", *IEEE TVCG*, Vol. 10, No. 6,
- Stefan Kimmeler, Matthieu Nesme and Francois Faure, "Hierarchy Accelerated Stochastic Collision Detection", *Proc. VMV '04*
- Jan Klein and Gabriel Zachmann: "The Expected Running Time of Hierarchical Collision Detection", SIGGRAPH 2005, Poster
- Jan Klein and Gabriel Zachmann: "Interpolation Search for Point Cloud Intersection", *Proc. of WSCG 2005*

Abstract2 Abstract3 Conclusion Finis

- Jan Klein and Gabriel Zachmann: "Nice and Fast Implicit Surfaces over Noisy Point Clouds", SIGGRAPH 2004, Sketches and Applications
- Adams & Dutre: "Interactive boolean operations on surfel bounded solids", 2003
- Hubbard: "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", 1996
- Dingliana, O'Sullivan: "Graceful Degradation of Collision Handling in Physically Based Animation", 2000
- Adamson & Alexa: "Approximating and Intersecting Surfaces from Points", 2003
- Jan Klein and Gabriel Zachmann: "Point Cloud Collision Detection", *Proc. EUROGRAPHICS 2004*
- Otaduy & Lin: "Sensation Preserving Simplification for Haptic Rendering", 2003
- Otaduy & Lin: "CLOUDs: Dual Hierarchies for Multiresolution Collision Detection", *Symp. on Geometry Processing (2003)*
- Weller, Zachmann: "Kinetic Bounding Volume Hierarchies for Deformable Objects", *VRCA (2006)*

Abstract2 Abstract3 Conclusion Finis