Physically-based and Real-time Simulation of Brittle Fracture for Interactive Applications

Loeïz Glondu

Physically-based and Real-time Simulation of Brittle Fracture for Interactive Applications

THESE ENS Cachan
pour obtenir le grade de
DOCTEUR DE L'ENS CACHAN
Spécialité: informatique

présentée par
Loeïz GLONDU
ECOLE DOCTORALE : MATISSE
LABORATOIRE: IRISA

Thèse soutenue le 6 Novembre 2012 devant le jury compsé de :

Luc Bougé	Professeur à l'ENS Cachan – *Président*
François Faure	Professeur à l'Université Joseph Fourier, Grenoble – *Rapporteur*
Matthias Harders	Senior Researcher at ETH Zürich, Switzerland – *Rapporteur*
Miguel A. Otaduy	Associate Professor at URJC Madrid, Spain – *Examinateur*
Bogdan Bucur	Destruction Team Lead at Havok, Dublin, Ireland – *Examinateur*
Georges Dumont	Professeur à L'ENS Cachan – *Directeur de thèse*
Maud Marchal	Maître de conference à l'INSA de Rennes – *Encadrante*

Acknowledgments

Writing the acknowledgments section makes me realize that my Ph.D. is finished. I cannot believe that I started three years ago. At this time, I did not expect to live so many professional and personal experiences. Now, I can say that this Ph.D. was a wonderful experience, and I am filled of good souvenirs. The main reason for all these good moments is the people I was surrounded by. I am happy to be able to thank those people in the following paragraphs.

First of all, I am deeply grateful to my two supervisors Maud Marchal and Georges Dumont. They have proven to be understanding, patient and helpful at a level I did not expect. We shared a lot of good moments, in France but also in the United States, in Germany or even in The Netherlands. They also generously shared their professional experiences and expertise that made all this possible. Maud, Georges, your advices were always precious to me, and I would like to sincerely thank you for these three successful years.

I would also like to thank all the distinguished researchers that kindly accepted to be part of my Ph.D. committee: Pr. François Faure, Dr. Matthias Harders, Dr. Miguel A. Otaduy, and M. Bogdan Bucur. Thanks to all of them for reviewing my manuscript, providing helpful comments, and for their valuable presence during my defence.

During my Ph.D., I had the great opportunity to work at the University of Rey Juan Carlos in Madrid, where I have been warmly welcomed by Miguel A. Otaduy. My stay in Spain was filled of great moments. It has been a great pleasure to meet Sara with who I worked with, and to meet Loïc again. Miguel, Sara, Loïc, thank you for making my stay in Madrid an awesome souvenir.

This section would be incomplete without giving a special word to all my colleagues. Special thanks to Gabriel Cirio with who I spent a lot of great moments, here and abroad. Gabriel, you are one of the most generous and nice person I know, don't change! Special thanks to Vincent Alleaume, Florian Nouviale and Quentin Avril too, with who I spent most of my lunch times, and a great amount of good moments. But of course I would like to thank also all my other colleagues for their help and their support. If time went so fast during these three last years, it is also thanks to you. To avoid any jealousy (and because I'm too afraid to forget someone), I won't give any name here, but consider that you are concerned if you read this piece of text. I will miss your joy, your jokes, playing badminton with you, and your cakes :)

A special thank too to all the people exterior to the lab, and from who we drove fruitful and joyful collaborations.

Last but not least, I wish I could thank enough all my family, my main source of joy. You always encouraged me, never doubt that I could success, and you were always here when I needed to. I hope you understand that I would not have been able to do all this without you.

Contents

List of Figures

List of Tables

Introduction 1

Fracture is a natural phenomenon that can be observed in everyday life on window glass, grounds or buildings. Although easy to observe and omnipresent in our world, the fracture phenomena only began to be studied by physicists in the 20^{th} century. Nowadays, many fracture phenomena are well understood and have been mathematically modeled. Over the years, numerical simulations of fracture have also been proposed, with first applications in architecture and fabrication. More recently, in the computer graphics field, the animation of fracture became a key tool to enrich special effects such as explosions or shattering bodies, mostly for the movie and the video game industries. Despite all the advances made in the field, it is still challenging to simulate brittle fracture in real-time due to its intrinsic complexity. Thus, numerous simplifications and approximations are made in the existing approaches. If it was possible to perform a brittle fracture simulation based on physics, with verified results, at interactive rates, a whole new range of applications could come into being, such as more realistic interactive special effects, or virtual prototyping of fragile objects. This is the challenge that is taken up by this manuscript, entitled "Physically-based and Real-time Simulation of Brittle Fracture for Interactive Applications: Models, Interaction and Evaluation", where new models designed for interactive simulations of brittle fracture are presented, in conjunction with appropriate algorithms that ensure the interactivity of the simulation, as well as evaluations of the proposed models.

We start this introduction by defining the term "brittle fracture" (the core topic of interest of this manuscript). We follow by providing a brief history of fracture mechanics, and how it progressively began to be addressed in the computer graphics field. The challenges and the issues of simulating physically-based brittle fracture in real-time are then presented. Finally, we end the introduction with a presentation of the contributions brought by our works, and the organization of the manuscript.

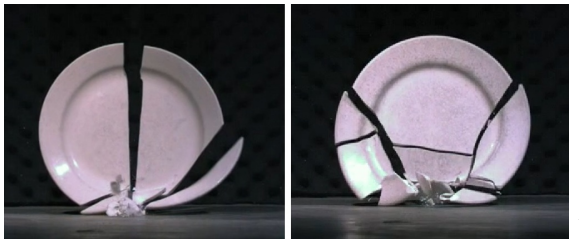
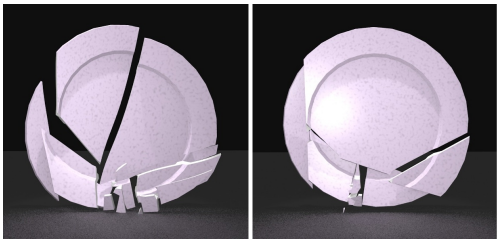
a.  b. 

Figure 1.1 – Examples of real and virtual brittle fracture. a. Real plates dropped from a height and breaking (images taken from [Zheng 10]). b. Simulation of a plate dropped from a height and breaking, using the fracture simulation method presented in this manuscript.

Brittle fracture: definition

The term "fracture" actually embeds several meanings in the literature, which are:

- **Breaking** (or shattering). Generally brief and violent separation of a body into two or more pieces due to an impact, or a strong and brief loading. *Examples*: a bone fracture, a plate dropped from a height as shown in Figure 1.1.
- **Crumbling**. Multiple and quasi-simultaneous separation of a body into many small parts, usually generated by rubbing an object on an other one. It can be considered a special case of breaking. *Examples*: a chunk of dry soil squeezed, a crumbly body rubbed on a rough surface, a chalk writing on a blackboard.
- **Cracking**. Formation of cracks inside or at the surface of a body, without necessarily separating them. *Examples*: surface cracks on a baked pottery, cracks on a dried ground.
- **Tearing**. Progressive separation of the matter due to tensile stress. This type of fracture is not studied in this manuscript. *Examples*: cloth tearing, progressive separation of a chewing gum.

The adjective *brittle* refers to stiff materials that undergo generally small and only elastic deformations before they fracture. By extension, the term “brittle fracture” refers to fractures that happen on brittle materials. Typical brittle materials are glass, ceramics, most concretes, or stone. One characteristic of brittle fracture is that the cracks propagations are very brief (cracks propagate at about 5100 meters per second in common glass), and cannot be observed by a human eye. This fast propagation is due to the stiffness and the elastic properties of brittle materials.

The other type of fracture is called *ductile* fracture. This kind of fracture, not addressed in this manuscript, involves elastic and plastic deformations. Therefore, it is usually modeled through different approaches. In this kind of materials, cracks propagate in different ways, and much more slowly than in its brittle counterpart.

Both brittle and ductile fracture have been widely studied in the past, but their different properties led to the use of different models and methods to simulate them. In this manuscript, we present contributions on the real-time simulation of **brittle** fracture. Our result can be used to simulate *breaking*, *cracking* or *crumbling* objects, as defined above. Before getting to a summary of our contributions, we present a brief history of the study of the fracture phenomena.

History: from fracture mechanics to computer graphics

The first experiments on brittle fracture date back to 1913, with the work of Charles E. Inglis on a cracked glass slab. The following years, in the context of the First World War, both brittle and ductile fracture phenomena received increasing attention, motivated by the observations of fractures on military vehicles such as tanks. Alan A. Griffith *et al.* were the first in the 1920’s to relate the amount of loading necessary to propagate a crack in brittle materials with the elastic properties of the material [Griffith 21]. They launched a new field of research called fracture mechanics. Although the original model developed by Griffith was correct for brittle materials such as glass, it neglected the plastic deformation occurring at the tip of the cracks, preventing the model to be used for ductile materials such as steel (see Figure 1.2). His model was then extended in the 1950’s by a group working under George R.

Irwin [Irwin 57] at the U.S. Naval Research Laboratory during the Second World War. The extended model of Irwin and colleagues is now used in several engineering fields to predict failures or to determine the sizes of beams and other structures in constructions.

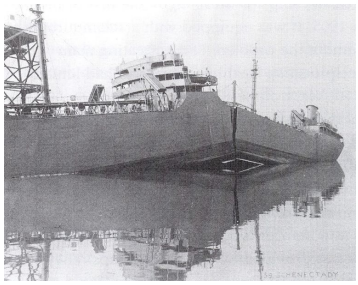


Figure 1.2 – The *Schenectady* tanker having its hull cracked almost in half while in harbor (1944).

In the computer graphics community, the first simulations of fracture were launched in the 1980's by Terzopoulos *et al.* [Terzopoulos 88], demonstrating to the community the potential use of physically-based simulation of deformation and fracture for animation. From this point, a variety of work has been presented to simulate surface cracks and other specific fracture cases. A notable work on fracture in computer graphics was presented in 1999 by O'Brien *et al.* [O'Brien 99], where the authors showed how conventional deformation methods are used to produce realistic fracture patterns.

Nowadays, fracture simulation methods are numerous, and are widely used to create special effects in movies and in video games (Figure 1.3). Large scale offline simulations of entire buildings being destroyed and earthquakes have been demonstrated in recent movies such as *Transformers 2* [Criswell 11], or *2012* [movie2012 09]. Impressive real-time simulations of brittle fracture have also been demonstrated in recent games such as *Battlefield 3* [Battlefield 12].

a.  b. 

c. 

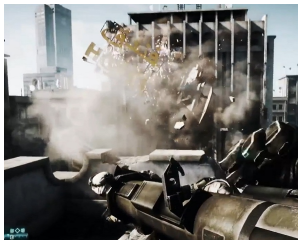
d. 

Figure 1.3 – Fracture images from movies and video games. **a**. A building being destroyed by an explosion in Transformers 2 [Criswell 11]. **b**. Earthquake simulation in 2012 [movie2012 09]. **c**. Shattering dishes in Ratatouille [Ratatouille 07]. **d**. A building being destroyed in Battlefield 3 [Battlefield 12].

Although a variety of methods are available today for the simulation of brittle fracture, the real-time approaches can still benefit from great improvements in term of efficiency and physical plausibility, as presented in the next paragraphs.

Real-time simulation of brittle fracture: challenges and issues

The first challenge is the modeling of fracture. Figure 1.4 shows examples of ceramic tiles that have been broken by dropping a metallic ball at their centers. To reproduce this phenomenon, the fracture simulation algorithm must support several features. First, the deformation of the material due to impacts should be simulated, since it is the deformation of the material that will cause the cracks to propagate. Brittle materials such as glass or ceramics are stiff (common glass has a Young's modulus of 90GPa), and small time steps (in the order of the microsecond) must be used to simulate a detailed deformation. However, the smaller the time step, the bigger the computational cost, which makes small time steps rarely compatible with real-time simulations. Therefore, there is no real-time method that proposes computing the dynamics of the deformations generated by impacts. Second, in order to reproduce the star crack pattern observed on the tile, the simulation model must handle non-constrained propagation of the cracks. Note that in Figure 1.4 all the cracks propagate from the location of the impact. However, existing real-time approaches do not allow free crack propagations, and cannot reproduce these patterns.

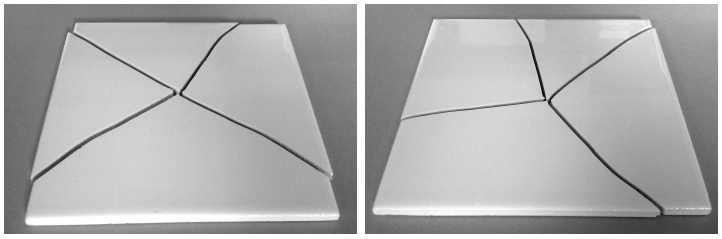


Figure 1.4 – Real ceramic tiles fractured by dropping a steel ball at their centers.

The second challenge concerns the efficiency of the simulations for interactive applications. Aiming at real-time performances allows the simulation to be used in any interactive application. Interactive applications are numerous and their uses go beyond video games. For instance, Virtual Reality aims at immersing a human operator into a virtual world, and all the studies and experiments performed in a Virtual Reality context use real-time methods to simulate the virtual world in which the human operator behaves. However, interactive applications provide multi-sensory feedbacks that require high frequency update of the state of the virtual world to be understandable by a human operator (up to 1Khz for modalities such as haptic rendering [Colgate 95]). This real-time constraint is the main reason why more and more efficient simulation methods are constantly developed, to allow more and more physically-based interactive simulations. Also, the dynamic fracture of the bodies present unique challenges for the handling of the collision detection between dynamically generated fragments. Indeed, in fracture simulations, new bodies with unpredictable shapes are created at run-time, which reduces the chances for efficient collision pruning.

Finally, the third challenge is about the physical plausibility of the simulations. Although the visual perception of the fracture realism is enough for animation in movies or games, the physical plausibility is a feature required for other applications such as virtual prototyping. However, it is hard to define a criterion of validity of the physical correctness of a fracture simulation. For example, the two tiles of Figure 1.4 have been broken with the same conditions but show different crack paths. New ways to evaluate the correctness of the fracture simulation should thus be defined. Finally, different aspects of the fracture simulation can be validated independently. For example, in an interactive breaking test simulation, one is interested in knowing whether or not the tested object will fracture, but not in how it will fracture. In a simulation of aging buildings, one will be more interested on how the material will break, and the resulting patterns.

Organization of the manuscript, and contributions

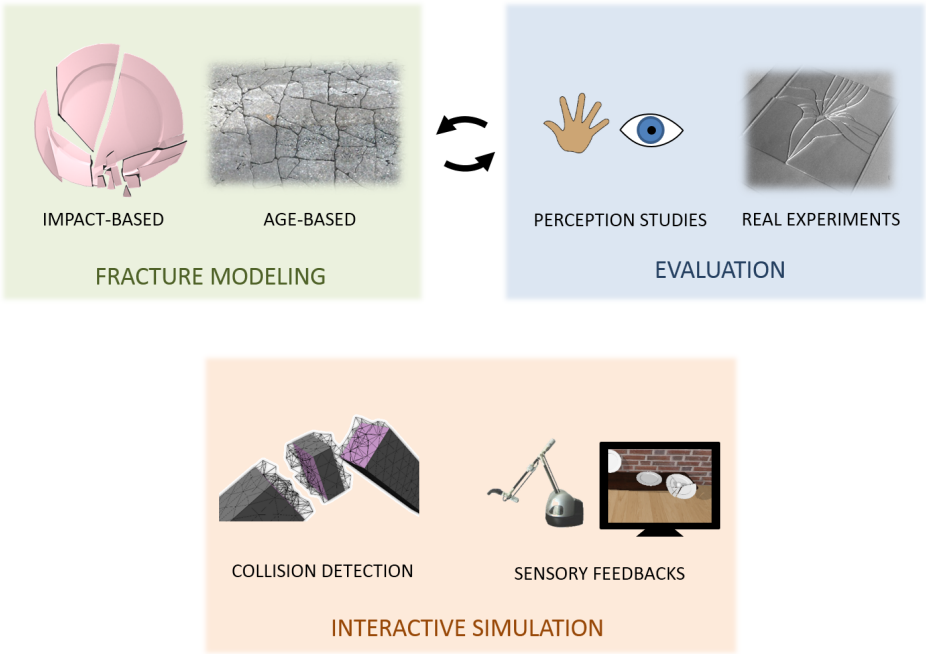


Figure 1.5 – Overview of the three main axes studied in this manuscript.

The manuscript is organized around three main axes illustrated in Figure 1.5:

- The **modeling of the fracture phenomena**: the study of numerical models needed to simulate breaking bodies. Two causes of fracture are studied: impact-based fractures (see Figure 1.1), that are caused by impact between colliding bodies, and age-based fractures (see Figure 1.6), that are caused by the modification of the materials properties over time. We propose a new model of the fracture state of the bodies, which is used to define an efficient method based on modal analysis for the simulation of impact-based brittle fracture. We also propose an efficient method that uses the same fracture state model for the simulation of age-based fracture. We focus on models that allow physically-based simulations to be compatible with interactive application. Therefore, our approach is designed for people interested either in the physical plausibility of the result to do *e.g.* virtual prototyping, or in the computational efficiency, or both of them.

- The **interactive simulation for fracture**: methods that allow simulating a virtual world composed of interacting breakable bodies. The interaction can be between the bodies of the virtual world, or an interaction with an external human user. The main problems addressed are the collision detection between the breaking bodies, and methods for interactions with haptic feedback. We propose a new collision detection algorithm design for real-time brittle fracture. We also demonstrate the first proof of concept of haptic interaction with fracturable objects, by providing new methods that take into account the specificities of both fracture simulation and haptic rendering.

- The **evaluation of the fracture models**: aiming at physically plausible simulations, means of validation and evaluation of the proposed models are studied. We present new ways to evaluate fracture simulations, and provide new fracture data from lab

experiments to the community. This enabled us to evaluate our method w.r.t. ground truth data, and calibrate it, to provide physically plausible and real-time fracture.

Each axis of study will be treated in a specific chapter of this manuscript.

The first chapter of this manuscript is a state of the art review of the literature on the brittle fracture simulation and its related topics. Since the simulation of fracture is linked to the simulation of the deformation, we present the general methods of physically-based simulation of deformation. After this background, we present the related work on brittle fracture simulation itself, focusing on the publications in the computer graphics field. We end the first chapter with a review of the different related collision detection mechanisms and methods used in real-time simulations, and on haptic interaction techniques.

The second chapter is focused on the models and methods to simulate impact-based and age-based fracturing. We start by presenting a new **model of the fracture state of a body**. We then show how we leverage this model to define **efficient crack propagation** and efficient surface sampling methods. We follow by presenting a new **approach based on modal analysis** to compute efficiently the deformations of impacted bodies, and decide where the cracks should propagate, and in which direction. We end this chapter with our contributions on the **efficient modeling of age-based fracture**.

Figure 1.6 – Examples of naturally observable fractures due to aging processes.

The third chapter deals with interactions with fracturable objects in virtual worlds. We begin the chapter with the proposition of a new approach for an **efficient collision detection method for fracturing bodies**. Based on a reconfigurable sphere tree and mesh-based distance field, we show how we obtain real-time simulation of brittle fracture due to our models. The second topic presented in this chapter is our contributions on haptic rendering for fracturing bodies. We start with a **comparison of common rigid body simulators** w.r.t haptic rendering, based on ill-conditioned scenarios and stress tests. Later, we introduce **new coupling schemes** to allow the haptic interaction in virtual worlds composed of a growing number of bodies.

The fourth chapter presents experiments and the development of a new metric for the validation of our simulation methods. We start with the presentation of a **subjective interpretation of the patterns obtained through a user study**. We propose analyzing statistically the geometry of the generated fragment to define a **new metric to compare obtained fracture patterns**. Finally, we present results and comparison of our simulation results with experiments on real ceramic tiles.

Background

2

Contents

2.1 Introduction

In this chapter, we review the previous work on fracture modeling, collision detection methods, and haptic interaction. Before presenting the previous work on fracture modeling, we provide the required background on general physical simulation, and on the simulation of deformation.

At the end of each section, we provide a summary of the previous work organized in a table. The tables rate the previous methods w.r.t. the main objectives of the manuscript: a real-time, physically-based fracture simulation for interactive application. Along the comparisons, we will use the following two criteria:

- **Efficiency**. Computational cost of the method. A method is twice more efficient than another method producing the same result if it computes this result in twice less computation time.
- **Physical conformity**. Differences or errors produced by a method w.r.t. the result obtained in the real world. When the physical conformity is valid for a certain range of application (*e.g.*, certain type of materials), it will be explicitly specified in the text.

2.2 Notations

Along this report, we will use the following notation conventions to facilitate the reading of mathematical expressions.

Mathematical representations

1. A **scalar** value will be represented in italic font. Example:

$$a = 0 \tag{2.1}$$

2. A **vector** value will be represented in bold font. By default, we use column vectors that are represented between square brackets. Examples:

$$\mathbf{v} = \begin{bmatrix} a & b & c \end{bmatrix}^T \quad \text{or} \quad \mathbf{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.2}$$

A vector or a matrix can be transposed using the superscript $\mathbf{v}^T$ as shown on equation (2.2), left. Vectors can be appended with semi-colon using the following notation:

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 \end{bmatrix}^T \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 & b_2 \end{bmatrix}^T \tag{2.3}$$

$$\mathbf{p} = \begin{bmatrix} \mathbf{a} \; ; \; \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & b_1 & b_2 \end{bmatrix}^T \tag{2.4}$$

3. A **matrix** is represented using upper case characters. As for the vector, we use square bracket to define matrices. Examples:

$$M = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \quad \text{or} \quad M = \begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{bmatrix} \tag{2.5}$$

In (2.5), right, note that we use three column vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ to build the matrix M. The identity matrix of size n will be noted I_n.

4. A mathematical **set** will be denoted using calligraphic upper case characters. Example:

$$\mathcal{H} = \{x_1; x_2; x_3\} \tag{2.6}$$

When possible, we use a subscript $x_i \in \mathcal{H}$ where i is an integer value to denote the elements of the same set $\mathcal{H}$. Horizontal bars $|\mathcal{H}| = n$ denote the cardinality (the numbers of elements) of the set $\mathcal{H}$.

If x_i represents an element indexed by i into an ordered set $\mathcal{H}$, then $\mathbf{v} = [x_i]$ represents the column vector built using all the elements of $\mathcal{H}$. Similarly, if the element x_{ij} is indexed with two values i and j, then $M = [x_{ij}]$ is the matrix built from the elements, i being the row index and j being the column index into the matrix.

Differentiation

If $\mathbf{q}(t, \mathbf{p})$ is a differentiable function of time and space, we use the dot notation:

$$\dot{\mathbf{q}} = \frac{\partial \mathbf{q}}{dt} \tag{2.7}$$

to denote the time derivative of the function, and we use the subscript notation:

$$\mathbf{q}_{,x} = \frac{\partial \mathbf{q}}{dx} \tag{2.8}$$

to denote a spatial derivative. The identifier after the coma represents the dimension which is used for the derivative.

Miscellaneous

Since we are in a 3-dimensional space context, vectors are elements of $\mathbb{R}^3$ and matrices elements of $\mathbb{R}^{3\times3}$ if no precision is given within the text.

We use the square bracket notation around a single vector to denote the "cross matrix" value of a vector $\mathbf{v} = [v_x\, v_y\, v_z]^T$ defined as:

$$[\mathbf{v}] = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix} \tag{2.9}$$

2.3 Fundamentals of physical simulation

Physical simulations aim at reproducing natural physical phenomena using numerical models. In the computer graphics field, physical simulations can be interpreted as numerical models of mechanics theory. As such, models used in numerical simulations have a lot in common with models defined in classical mechanics. However, the terminology used in physical simulation (and especially in computer graphics) can differ from the terminology used in classical mechanics. We define in this section the fundamental terminology and models used in physical simulation.

2.3.1 Model of a physical body

A physical body is a mechanical system defined by a state vector $\mathbf{q}$, representing its degrees of freedom. The state of the body is unique at a given time, and thus can be expressed as a function of time: $\mathbf{q}(t)$. For simplicity of reading, we simplify $\mathbf{q}(t)$ by writing it $\mathbf{q}$ when there is no ambiguity. The first time derivative of the state $\dot{\mathbf{q}}$ represents the velocity of the body, while the second time derivative $\ddot{\mathbf{q}}$ represents its acceleration.

In classical mechanics, the typical scenario consists in the study of the behavior of an isolated mechanical system. However, in computer graphics, virtual worlds are composed of several physical bodies that can interact with each other through contacts events, or other external events. If each body numbered i is defined by its corresponding state vector $\mathbf{q}_i$, the global state $\mathbf{q}$ of the world can be defined as the concatenation of all the state vectors of all the bodies:

$$\mathbf{q} = [\mathbf{q}_1; \mathbf{q}_2; \cdots ; \mathbf{q}_n] \tag{2.10}$$

where n is the number of physical bodies composing the virtual world. More information on the models of virtual worlds is given in section 2.6.

2.3.2 Stepping a discrete simulation forward in time

Simulating a physical body corresponds to compute its state $\mathbf{q}(t)$ over time. In the real world, the body takes an infinite number of identifiable states given any interval of time, *i.e.*, the state function $\mathbf{q}(t)$ is continuous. In contrast, in numerical simulations, the state of a body is only computed at discrete times. That is, the output of a numerical simulation is a sequence of states $\{\mathbf{q}(t_0); \mathbf{q}(t_1); \cdots ; \mathbf{q}(t_n)\}$, where t_i is the time for which the ith sample of the state $\mathbf{q}$ has been computed.

To simplify the writing in the following, we will denote by $\mathbf{q}^i$ the state $\mathbf{q}(t_i)$ at time t_i.

2.3.2.1 The differential system to be solved

In our context, the evolution of the state of a body is ruled by Newton's second law of motion:

$$M(\mathbf{q}, \dot{\mathbf{q}})\, \ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.11}$$

where $M(\mathbf{q}, \dot{\mathbf{q}})$ is the mass matrix of the system, and $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$ is the sum of forces applied on each of the degree of freedom represented in the state vector. This law of motion (2.11) relates the acceleration $\ddot{\mathbf{q}}$ of the state to the forces applied on it. The mass M plays the role of a ratio between the acceleration and the forces applied. The lower the mass, the higher the acceleration for the same force applied.

In practice, the force vector $\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$ can be complex and unpredictable (if an external unpredictable actor is allowed adding forces to the system). Therefore, analytical solutions for equation 2.11 are only available for simple textbook cases. When no analytical solution is available, a numerical integration is performed to approximate the exact solution.

2.3.2.2 Numerical integration

The goal of numerical integration is to compute the integral of a function $\mathbf{f}(t)$ with the following approximation:

$$\int_0^t \mathbf{f}(t)dt \approx \sum_{0<i<n} \mathbf{f}(t_i)\Delta t \tag{2.12}$$

The integration step Δt is a crucial parameter in the integration. The smaller it is, the better the accuracy of the integral, but the higher the computational cost. The choice of the integration step in a numerical simulation is not straight forward, and has great consequences on the outcome of the simulation.

In the case of physical simulation, we deal with a second-order differential equation, and a double integration is needed. The usual workaround consists in writing (2.11) down to a set of two coupled differential equations:

$$\dot{\mathbf{q}} = \mathbf{v} \tag{2.13}$$
$$\dot{\mathbf{v}} = M^{-1}\mathbf{f}(\mathbf{q}, \mathbf{v}) \tag{2.14}$$

where $\mathbf{v}$ is an intermediate variable representing the velocity of a body. Applying a numerical integration on this coupled system involves summing the acceleration $M^{-1}\mathbf{f}(\mathbf{q}, \mathbf{v})$ to obtain new velocities, and summing the velocities will yield to the new state. In computer graphics, various versions of integration have been studied, and we detail here three common ones: the Euler explicit integration, the Runge-Kutta steps, and the Euler implicit integration method. More information on integration methods used in Computer Graphics can be found in [Witkin 01b, Müller 08b].

Explicit Euler integration

Euler's explicit integration method is a simple way to integrate from state i at time t_i to state $i+1$ at time t_{i+1}:

$$\mathbf{q}^{i+1} = \mathbf{q}^i + \Delta t \mathbf{v}^i \tag{2.15}$$
$$\mathbf{v}^{i+1} = \mathbf{v}^i + \Delta t M^{-1}\mathbf{f}(\mathbf{q}^i, \mathbf{v}^i) \tag{2.16}$$

The new state $\mathbf{q}^{i+1}$ of the body is computed using its previous state $\mathbf{q}^i$ and the current velocity $\mathbf{v}^i$, while the new velocity is computed using the current force applied on the system and the mass of the body.

Due to the errors brought by this method along the integrations, the size of the time step Δt must be chosen carefully in order to avoid the system to become unstable. A slight modification of the explicit Euler integration method consists in using the new velocity $\mathbf{v}^{i+1}$ instead of the current one $\mathbf{v}^i$:

$$\mathbf{q}^{i+1} = \mathbf{q}^i + \Delta t \mathbf{v}^{i+1} \tag{2.17}$$
$$\mathbf{v}^{i+1} = \mathbf{v}^i + \Delta t M^{-1}\mathbf{f}(\mathbf{q}^i, \mathbf{v}^i) \tag{2.18}$$

In this case, an order is imposed on the resolution of the system. The new velocity must be computed before the new state. Although small, this modification can lead to great improvements on the stability of the simulation.

Runge-Kutta steps

In explicit Euler integration, the acceleration is evaluated once, at time t_i, and its value is integrated twice, leading to a possible great error if the acceleration function varies a lot between time t_i and time t_{i+1}. The Runge-Kutta method enables to reduce the integration error by evaluating several times the force function (giving several accelerations values) at intermediates points. We describe here the Runge-Kutta stepping of order 2 (also called the mid-point method), but the order can be chosen arbitrarily, and the higher the order of the Runge-Kutta step, the higher the precision of the integration (but also the higher the computational costs).

Let $\mathbf{q}^e$ be the state at time t_{i+1} computed using Euler's explicit integration (we thus have $\mathbf{q}^e = \mathbf{q}^i + \Delta t \mathbf{v}^i$). The Runge-Kutta methods propose to take into account the variation of the acceleration function (coming from the variation of the force function) by evaluating it at intermediate points. At order 2, a middle state $\mathbf{q}^{mid}$ is computed as:

$$\mathbf{q}^{mid} = \frac{\mathbf{q}^i + \mathbf{q}^e}{2} \tag{2.19}$$

$$\mathbf{v}^{mid} = \frac{\mathbf{v}^i + \mathbf{v}^e}{2} \tag{2.20}$$

The state $\mathbf{q}^{mid}$ is then used to compute the force for the integration:

$$\mathbf{q}^{i+1} = \mathbf{q}^i + \Delta t \mathbf{v}^i \tag{2.21}$$

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \Delta t M^{-1} \mathbf{f}(\mathbf{q}^{mid}, \mathbf{v}^{mid}) \tag{2.22}$$

The implicit Euler method

Simple Euler integration and Runge-Kutta steps are *explicit* methods because they are expressed on known quantities at time t_i to extrapolate the final value at time t_{i+1}. The main weakness of explicit methods is the inability to cope with stiff equations with reasonable integration step size Δt. Implicit methods propose tackling this problem by evaluating the derivative (the force function) at time t_{i+1}, namely using the state $\mathbf{q}^{i+1}$ and the velocity $\mathbf{v}^{i+1}$. For a good introduction and understanding of the issues of explicit and implicit Euler integration, we refer the reader to [Baraff 01, Baraff 98]. The most common implicit integration is the implicit Euler integration method, which proposes solving for the following system:

$$\mathbf{q}^{i+1} = \mathbf{q}^i + \Delta t \mathbf{v}^{i+1} \tag{2.23}$$

$$\mathbf{v}^{i+1} = \mathbf{v}^i + \Delta t M^{-1} \mathbf{f}(\mathbf{q}^{i+1}, \mathbf{v}^{i+1}) \tag{2.24}$$

In (2.24) the new velocity is computed from the acceleration at time t_{i+1} which is unknown since the state or velocity at t_{i+1} is not known. While in (2.23), the new velocity must be known in order to compute the new state at time t_{i+1}. This system expresses the fact that we want to find a point $\mathbf{q}^{i+1}$ that is such that if we would run the simulation backward (and using explicit integration), we would go back to point $\mathbf{q}^{i+1}$ after a time Δt (namely, from (2.23), we can write $\mathbf{q}^i = \mathbf{q}^{i+1} - \Delta t \mathbf{v}^{i+1}$).

Unless $\mathbf{f}(\mathbf{q}, \mathbf{v})$ happens to be a linear function of $\mathbf{q}$ and $\mathbf{v}$, equation (2.24) cannot be solved directly. Introducing two new variables $\Delta\mathbf{q} = \mathbf{q}^{i+1} - \mathbf{q}^i$ and $\Delta\mathbf{v} = \mathbf{v}^{i+1} - \mathbf{v}^i$, it is

possible to replace $\mathbf{f}(\mathbf{q}^{i+1}, \mathbf{v}^{i+1})$ by its first order approximation, using the first order Taylor expansion of the function:

$$\mathbf{f}(\mathbf{q}^{i+1}, \mathbf{v}^{i+1}) = \mathbf{f}(\mathbf{q}^i + \Delta\mathbf{q}, \mathbf{v}^i + \Delta\mathbf{v}) = \mathbf{f}(\mathbf{q}^i, \mathbf{v}^i) + \Delta\mathbf{q}\frac{\partial \mathbf{f}}{\partial \mathbf{q}} + \Delta\mathbf{v}\frac{\partial \mathbf{f}}{\partial \mathbf{v}} \tag{2.25}$$

Rewriting (2.24) with the first order approximation of the force yield:

$$\Delta\mathbf{v} = \Delta t M^{-1}\left(\mathbf{f}(\mathbf{q}^i, \mathbf{v}^i) + \Delta\mathbf{q}\frac{\partial \mathbf{f}}{\partial \mathbf{q}} + \Delta\mathbf{v}\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right) \tag{2.26}$$

Replacing $\Delta\mathbf{q}$ by $\Delta\mathbf{q} = h(\mathbf{v}^i + h\dot{\mathbf{v}}^{i+1})$ and rearranging gives:

$$\left(I - M^{-1}\Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{v}} - M^{-1}\Delta t^2\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right)\Delta\mathbf{v} = \Delta t M^{-1}\left(\mathbf{f}(\mathbf{q}^i, \mathbf{v}^i) + \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\mathbf{v}^i\right) \tag{2.27}$$

Using (2.27) one can solve for $\Delta\mathbf{v}$ using an appropriated linear solver such as conjugate gradient method [Shewchuk 94] or Gauss iterative method. Once $\Delta\mathbf{v}$ is known, $\mathbf{v}^{t+1}$ and $\mathbf{q}^{t+1}$ can be immediately computed using (2.24) and (2.23) respectively.

Stepping forward the simulation using implicit integration techniques is computationally more expensive, but allows performing a numerical simulation with much higher time steps without loosing stability. The extra cost coming from the solving of the linear system at each time step is often compensated by the gain obtained by taking bigger integration steps Δt. The improvement brought by implicit integration method can be explained through the fact that it is exploiting deeper order derivatives of the state function (through the derivative of the force function), allowing it to anticipate the future variations and use bigger integration steps.

2.3.3 Example: 3-D simulation of a rigid body

Rigid bodies are one of the simplest model for physical simulation. Their simplicity combined with their ability to reproduce fairly well the dynamics of stiff bodies made them very popular in the computer graphics community. Rigid bodies are also a good introductory example for physical simulation. We present here the classical model of rigid bodies, and its dynamic laws of motion.

Model of rigid body

A rigid body is a body that cannot deform (a body of infinite stiffness). With the assumption of perfect rigidity, the state $\mathbf{q} \in \mathbb{R}^6$ of a rigid body can be expressed using six scalars: 3 scalars for its translation, and 3 scalars for its rotation. If we express the rigid body as an infinite set of particles with position $\mathbf{x}_v$, and density ρ_v, a particular point $\mathbf{x}$ called center of mass can be defined for this body as:

$$\mathbf{x} = \frac{1}{m}\int_V \rho_v \mathbf{x}_v dv \tag{2.28}$$

Where V is the volume of the bodies and $m = \int_V \rho_v dv$ is the total mass of the body. The center of mass of a rigid body is a special point from which can be expressed the position, rotation and velocities of the body and greatly simplify the further computations. For instance, when a rigid body is freely spinning, the axis of rotation of this spinning motion will pass through its center of mass. In the following, we will use four variables to define the dynamic motion of the rigid body

- The position $\mathbf{x} \in \mathbb{R}^3$ of the body's center of mass (translation vector from the origin).
- The linear velocity $\mathbf{v} = \dot{\mathbf{x}}$ of the body.
- The orientation of the body $R \in \mathbb{R}^{3\times3}$, stored as an orientation matrix.
- The angular velocity $\boldsymbol{\omega} \in \mathbb{R}^3$.

Here, we use the matrix representation of the rotation, which is a common representation of orientation in computer graphics. However, numerous ways to represent a 3-D orientation have been studied, each one having it advantages and drawbacks.

Mass, forces and integration

To simulate the motion of the rigid body, we can use the concepts presented in the previous section. Namely, the acceleration $\ddot{\mathbf{q}}$ of the body follows the Newton's rule $M\ddot{\mathbf{q}} = \mathbf{f}$. However, we have still not defined the mass properties M of the body, the coefficient between the force applied on a body and its acceleration. For a rigid body, the mass information can be divided in two parts when studied at the center of mass of the body: a translational part, and a rotational part. For the translational part, the total mass m of the body can be used to link the acceleration and a force applied at the center of mass of the body:

$$m\ddot{\mathbf{x}} = \mathbf{f}_{trans} \tag{2.29}$$

where $\mathbf{f}_{trans} \in \mathbb{R}^3$ in a force applied at the center of mass $\mathbf{x}$ of the body. Applying integration techniques presented in the previous section, we can write the equation for the simulation of the translational motion:

$$\begin{aligned} \mathbf{x}^{i+1} &= \mathbf{x}^i + \Delta t \mathbf{v}^i & (2.30)\\ \mathbf{v}^{i+1} &= \mathbf{v}^i + \Delta t m^{-1} \mathbf{f}_{trans} & (2.31) \end{aligned}$$

The computation of the rotational motion of the body is a bit less straight forward. The resistance of the body to an angular force $\mathbf{f}_{rot}$ (called a *torque*) depends on the direction of $\mathbf{f}_{rot}$. Therefore, the mass equivalent to m for rotational motion cannot be described by a single scalar value. Instead, a tensor J called inertia matrix is used to represent the distribution of the mass around the center of mass. This tensor is computed as:

$$J = \sum_i -m_i[\mathbf{r}_i][\mathbf{r}_i] \tag{2.32}$$

where $\mathbf{r}_i = \mathbf{x}_i - \mathbf{x}$ denotes the displacement from the center of mass to a particle position $\mathbf{x}_i$. For convenience, we compute J_{init} the inertia matrix for an rotation $R = I_3$. Then, it can be shown that for any rotation matrix R, the current inertia matrix can be retrieved from J_{init} as:

$$J = RJ_{init}R^T \tag{2.33}$$

The dynamics of the rotational part of the body can now be written as:

$$\begin{aligned} R^{i+1} &= [\Delta t \boldsymbol{\omega}^i] R^i & (2.34)\\ \boldsymbol{\omega}^{i+1} &= \boldsymbol{\omega}^i + \Delta t J^{-1} \mathbf{f}_{rot} & (2.35) \end{aligned}$$

Note how the rotation matrix is updated in equation (2.34). A simple sum is no longer possible to update the rotation, and a specific update is needed for each manner of representing the rotation. In the case of rotation matrices, the angular velocity $\boldsymbol{\omega}$ turns each of the axis of the rotation matrix to get a new matrix $R_\omega = [\boldsymbol{\omega} \times \mathbf{r}_x \ \boldsymbol{\omega} \times \mathbf{r}_y \ \boldsymbol{\omega} \times \mathbf{r}_z]$. Factorizing, we get $\dot{R} = [\boldsymbol{\omega}]R$.

2.4 Physical simulation of deformation

Although rigid bodies are very useful to efficiently simulate the general motion of stiff bodies, they are unable to simulate the deformation of the bodies due to loading forces. Since the fracture phenomenon is intimately related to the deformation phenomenon, we present in this section the main models used in computer graphics to simulate deformation. In the next section, we will explain more formally the link between the deformation and the fracture.

2.4.1 Background on continuum mechanics

Continuum mechanics state as main hypothesis that the physical quantities within the matter of a solid is a continuous function of position and time. At a given time, the spatial occupation $\mathcal{X} \subset \mathbb{R}^3$ of a body is a connected subspace of $\mathbb{R}^3$ (in 3 dimensions). Physical quantities can be associated to each material point $\mathbf{x} \in \mathcal{X}$ of the body. In the Lagrangian approach, each point is tracked over time, and each physical quantity inside the continuum body can be expressed as a function of position and time. For example, if one is interested in modeling the propagation of the temperature inside a material using the continuum assumption, a temperature function $\tau(\mathbf{x}, t)$ can be defined to track the temperature of each material point over time. Thanks to this model, using a equations of dynamics for the temperature propagation involving spatial $\nabla\tau$ derivatives and temporal derivative $\ddot{\tau}$ could be possible using a common mathematical framework.

In order to measure the deformation, we define a function $\boldsymbol{\phi}(\mathbf{x}, t) \in \mathbb{R}$ that tracks the position of the material point $\mathbf{x}$ into a constant world frame coordinate at time t. At time $t = 0$, we consider that the body is at its rest state and does not store any internal potential energy. For ease of writing, we denote $\mathbf{x}^0$ the position $\boldsymbol{\phi}(\mathbf{x}, 0)$ of the material point $\mathbf{x}$ at time 0:

$$\mathbf{x}^0 := \boldsymbol{\phi}(\mathbf{x}, 0) \tag{2.36}$$

From the position of each point, it is possible to define a *displacement* field $U \subset \mathbb{R}^3 \times \mathbb{R}^3$, as:

$$\mathbf{u}(\mathbf{x}, t) = \boldsymbol{\phi}(\mathbf{x}, t) - \mathbf{x}^0 \tag{2.37}$$

The displacement of a point $\mathbf{x}$ represents the difference of position between its current position $\boldsymbol{\phi}(\mathbf{x}, t)$ at time t and its initial position $\mathbf{x}^0$. Working with the displacement function $\mathbf{u}$ instead of the position function $\boldsymbol{\phi}$ yield to some simplification in the writing of the following. Figure 2.1 illustrates the position and displacement functions on a body being deformed.

The position and displacement functions provide a basis to measure the deformation of the body, and applying theory of elasticity and plasticity on it.

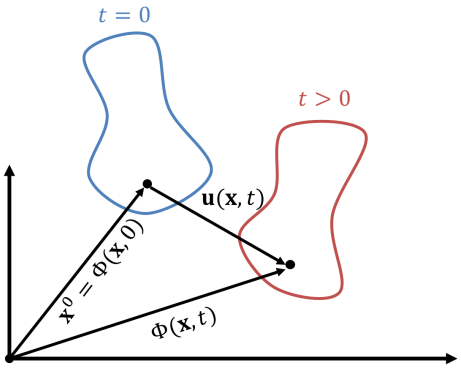


Figure 2.1 – A body in its initially undeformed state (in blue) at time $t = 0$, and its current deformed state (in red) at time $t > 0$. The world coordinates of the material point $\mathbf{x}$ (represented by the black dot) are defined by the function $\boldsymbol{\phi}$, while the function $\mathbf{u}(\mathbf{x}, t) = \boldsymbol{\phi}(\mathbf{x}, t) - \mathbf{x}^0$ gives the displacement of the point $\mathbf{x}$ from its initial position to its current one.

2.4.1.1 Measure of the deformation: the strain

In mechanics, the term that denotes the deformation of a body is the *strain*. Given a direction, the strain is a measure of a relative elongation of the material. It can be thought as a percentage of elongation of the material in a given direction. For example, a beam of initial length l_0 being uniformly stretched to a length l along the x axis coordinate will have a strain of $\Delta l / l_0$ ($\Delta l = l - l_0$) along this axis (see Figure 2.2).

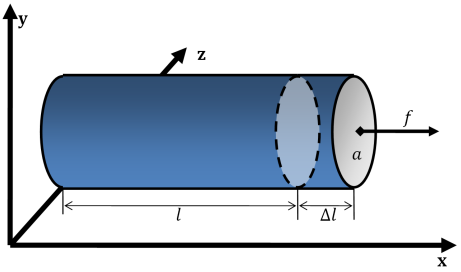


Figure 2.2 – Illustration 1-D of the strain. The beam with a rest length of l and a section area of a is stretched by a force f in the x axis direction. The variation of length is Δl. Under a uniform deformation, the stress of any point of the beam along the x axis is $\Delta l / l_0$.

However the strain of the points along another axis will be different (e.g. it is zero along the y axis). Therefore, a single scalar is not enough to characterize the strain information of a point. In three dimensions, the strain state $\varepsilon(\mathbf{x}, t)$ of a point at time t can be related to its displacement gradient $\nabla\mathbf{u}(\mathbf{x}, t)$ using a tensor of order 2:

$$\varepsilon_G(\mathbf{x}, t) = \tfrac{1}{2}\left(\nabla\mathbf{u}(\mathbf{x}, t) + [\nabla\mathbf{u}(\mathbf{x}, t)]^T + [\nabla\mathbf{u}(\mathbf{x}, t)]^T \nabla\mathbf{u}(\mathbf{x}, t)\right) \tag{2.38}$$

$$\varepsilon_C(\mathbf{x}, t) = \tfrac{1}{2}\left(\nabla\mathbf{u}(\mathbf{x}, t) + [\nabla\mathbf{u}(\mathbf{x}, t)]^T\right) \tag{2.39}$$

ε_G is the so-called Green-Lagrange non-linear tensor, and ε_C is its linearized version. Denoting the components of $\mathbf{u}(\mathbf{x}, t) = [u\ v\ w]^T$, the displacement gradient $\nabla\mathbf{u}(\mathbf{x}, t)$ is written as:

$$\nabla\mathbf{u}(\mathbf{x}, t) = \begin{bmatrix} u_{,x} & u_{,y} & u_{,z} \\ v_{,x} & v_{,y} & v_{,z} \\ w_{,x} & w_{,y} & w_{,z} \end{bmatrix} \tag{2.40}$$

2.4.1.2 Stress tensor and strain-stress relationship

The stress state $\sigma(\mathbf{x}, t)$ of a point gives an indirect information on internal forces acting on it. Taking the previous example of the beam, the force f per area a acting on points of the beam is linked to the strain (the relative elongation $\Delta l/l_0$) of the beam. Depending on the properties of the material of the beam, the function that links ε and σ can be various. If the material of the beam is Hookean, the stress/strain relationship is linear:

$$\frac{f}{a} = e\frac{\Delta l}{l_0} \tag{2.41}$$

Where f is the force acting on area a (the section of the beam) along the x axis (see figure 2.2). The scalar e linking the strain to the stress is called the Young's modulus and represents the stiffness of the material. More generally, for Hookean material, the stress is linked to the strain via a linear relation E:

$$\sigma(\mathbf{x}, t) = E\,\varepsilon_C(\mathbf{x}, t) \tag{2.42}$$

The relation between the stress and the strain is dependent on the material property, and is not necessarily linear. An example of non-linear model is the Saint Venant-Kirchhoff model where the stress, called second Piola-Kirchhoff stress, is defined using the non-linear Green strain ε_G as:

$$S = \lambda Trace(\varepsilon_G) I_3 + 2\mu\varepsilon_G \tag{2.43}$$

where λ and μ are the so-called Lamé constants, two properties defining the deformation behavior of the material. For more information on non-linear deformation, we refer the reader to [Bonet 97].

For a material point $\mathbf{x}$, the stress is a 3 by 3 symmetric matrix tensor:

$$\sigma(\mathbf{x}, t) = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{xy} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{xz} & \sigma_{yz} & \sigma_{zz} \end{bmatrix} \tag{2.44}$$

From this stress tensor, one can compute a force acting on an area a in a direction $\mathbf{n}$ as:

$$\frac{f}{a} = \sigma(\mathbf{x}, t) \cdot \mathbf{n} \tag{2.45}$$

By extension, from the stress tensor of a point $\mathbf{x}$, it is possible to compute a body force $\mathbf{f}_{stress}(\mathbf{x}, t)$ acting on this point as:

$$\mathbf{f}_{stress}(\mathbf{x}, t) = \nabla \cdot \sigma(\mathbf{x}, t) = \begin{bmatrix} \sigma_{xx,x} + \sigma_{xy,y} + \sigma_{xz,z} \\ \sigma_{xy,x} + \sigma_{yy,y} + \sigma_{yz,z} \\ \sigma_{xz,x} + \sigma_{yz,y} + \sigma_{zz,z} \end{bmatrix} \tag{2.46}$$

2.4.1.3 Dynamics of a deformable body

The strain field ϵ can be computed from the displacement field U with equation (2.38) or equation (2.39). Once we have ϵ, the stress field O is computed with a chosen constitutive law. With the stress field, a force field is computed thanks to (2.46). At this point, the fundamental law of dynamic $\mathbf{f} = m\ddot{\mathbf{x}}$ for each point $\mathbf{x}$ can be used. However, the mass m of

an infinitesimal material point is not defined. Instead, the density $\rho(\mathbf{x}, t)$ of the material at point $\mathbf{x}$ is used:

$$\rho(\mathbf{x}, t)\ddot{\mathbf{x}} = \mathbf{f}_{stress}(\mathbf{x}, t) \tag{2.47}$$

Although it is mathematically possible to model the deformation as (2.47), this equation has no general analytical solution. Indeed, depending on the initial boundary conditions imposed by the geometries of the bodies, the mathematical problem to solve has no analytical solution. In this case, numerical models based on a discretization of the body allow obtaining an arbitrary close approximate solution. A number of numerical models have been derived from continuum mechanics, based on spatial discretization of the studied bodies, and approximating the behavior of perfect continuum bodies, as presented in the following.

2.4.2 Models of deformable body for numerical simulations

In the computer graphics community, models for deformation has been introduced a bit more than twenty years ago by Terzopoulos and colleagues [Terzopoulos 87], [Terzopoulos 88] proposing methods to simulate elastic and plastic deformation, and fracture. We review here the most common models used in the literature, and complementary information on deformation modeling and simulation can be found into the surveys [Gibson 97] or [Nealen 06].

2.4.2.1 The Finite Element Method (FEM)

The key idea of the Finite Element Method (FEM) is to discretize the volume of the body into a set of nodes $\mathbf{x}_i$, and into a set of n elements $e_i (1 \leq i \leq n)$, which form a partition of the volume of the body (see Figure 2.3). Inside the elements, a mapping is used to interpolate the needed quantities such as strain and stress.

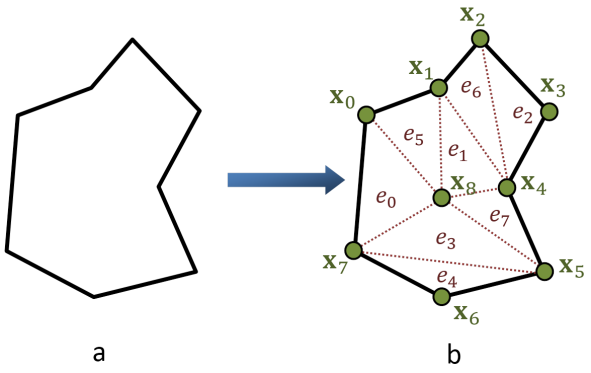


Figure 2.3 – A 2-dimension deformable body (**a**) and its decomposition (**b**) into arbitrary chosen elements e_i. Each node $\mathbf{x}_i$ can be shared by several elements.

As an example, let us choose a point $\mathbf{x} \in \mathbb{R}^3$ which is inside an element e, itself having four nodes of position $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$ (*e.g.* a 3-dimensional tetrahedron) in the undeformed state. We can express the vector $\mathbf{x}$ as a linear combination of the nodes coordinates using four barycentric coordinates:

$$\mathbf{x} = \mathbf{x}_1 b_1 + \mathbf{x}_2 b_2 + \mathbf{x}_3 b_3 + \mathbf{x}_4 b_4 = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 \end{bmatrix} \mathbf{b} = A\mathbf{b} \tag{2.48}$$

In the deformed state at time t, the same nodes have different coordinates, which are $\boldsymbol{\phi}(\mathbf{x}_1, t)$, $\boldsymbol{\phi}(\mathbf{x}_2, t)$, $\boldsymbol{\phi}(\mathbf{x}_3, t)$ and $\boldsymbol{\phi}(\mathbf{x}_4, t)$. In order to lighten notation, we will omit the time parameter using *e.g.* $\boldsymbol{\phi}(\mathbf{x}_1)$ instead of $\boldsymbol{\phi}(\mathbf{x}_1, t)$. The position of $\mathbf{x}$ in the deformed coordinates

is computed by interpolation using the same barycentric coordinates as in the undeformed state:

$$\boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x}_1)b_1 + \boldsymbol{\phi}(\mathbf{x}_2)b_2 + \boldsymbol{\phi}(\mathbf{x}_3)b_3 + b_4\boldsymbol{\phi}(\mathbf{x}_4) = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1) & \boldsymbol{\phi}(\mathbf{x}_2) & \boldsymbol{\phi}(\mathbf{x}_3) & \boldsymbol{\phi}(\mathbf{x}_4) \end{bmatrix} \mathbf{b} = B\mathbf{b} \tag{2.49}$$

From those two identities, we have $\mathbf{b} = A^{-1}\mathbf{x}$ and $\mathbf{b} = B^{-1}\boldsymbol{\phi}(\mathbf{x})$, yielding $B^{-1}\boldsymbol{\phi}(\mathbf{x}) = A^{-1}\mathbf{x}$, or:

$$\phi(\mathbf{x}) = BA^{-1}\mathbf{x} = P\mathbf{x} \tag{2.50}$$

Note that A^{-1} is constant over time since it is built from the undeformed position of the nodes. Since $P \in \mathbb{R}^{3\times3}$ is a linear mapping for position, the gradient of the displacement field is

$$\nabla\mathbf{u}(\mathbf{x}) = P - I \tag{2.51}$$

for any point $\mathbf{x}$ within the element. In other words, the gradient of the displacement field is constant within an element, yielding to a constant strain field and stress field per element. For convenience, we will denote by σ_e the stress tensor of an element e.

From the stress and strain, the potential elastic energy η of each point can be computed as:

$$\eta(\mathbf{x}, t) = \frac{1}{2}\mathrm{Tr}(\sigma(\mathbf{x}, t)\varepsilon(\mathbf{x}, t)) \tag{2.52}$$

From the partial derivative of the potential elastic energy integrated over the volume v_e of the element, we can compute the internal force $\mathbf{f}_i$ acting on the node i of this element with respect to the node current position $\boldsymbol{\phi}(\mathbf{x}_i)$ as:

$$\mathbf{f}_i = \frac{v_e}{2}\, A^{-1}\, G\, \sigma_e\, {A^{-1}}^T\, G^T\, \boldsymbol{\phi}(\mathbf{x}_i) \tag{2.53}$$

where the matrix A is defined in (2.48) as the matrix built using the undeformed coordinates of the nodes of the element, and G is defined as:

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.54}$$

An alternative approach to compute nodal forces called Finite Volume Method (FVM) [Teran 03] consists in using equation (2.45) to compute a force per face from the stress tensor σ_e of the element e. For example, a triangle face with vertices $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ will generate a force $\mathbf{f}_{face}$ for the surface of:

$$\mathbf{f}_{face} = \sigma_e \mathbf{n}_{face} \cdot a_{face} = \sigma_e \left[(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)\right] \tag{2.55}$$

where $\mathbf{n}_{face}$ is the vector normal to the face, and a_{face} is the area of the face. The force computed for each face is then distributed on all the nodes composing the face, as shown on figure 2.4.

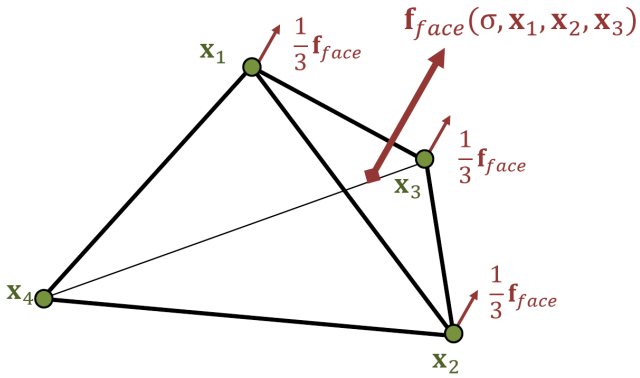


Figure 2.4 – Nodal force computation in the finite volume method. A force is determined for the face $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ thanks to stress value of the tetrahedron element, and the area and orientation of the face. The force $\mathbf{f}$ computed is distributed equally on the nodes $\mathbf{x}_i$ of the face. The final force for a node is the sum of all forces from faces it is adjacent with.

The stiffness matrix

If implicit integration methods presented in section 2.3.2.2 wants to be used with the finite element method, the jacobian of the nodal forces with respect to the position must be computed. This jacobian is stored in a matrix K called stiffness matrix, presented along this section.

Using either equation (2.53) or equation (2.55), it is possible to deduce the force acting on node i of an element e directly from the deformed positions of the nodes of the same element e. Therefore, let us write $\mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j)$ as the force acting on node i at its initial position, knowing the position $\mathbf{x}$ of the node j. Now let us define K_{ij}^0 as the gradient of $\mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j)$ with respect to the position $\mathbf{x}_j$ of the node j:

$$K_{ij}^0 = \frac{\partial \mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j)}{\partial \mathbf{x}_j} \tag{2.56}$$

We can now write the linear approximation of the values of the forces for a new position of node $\boldsymbol{\phi}(\mathbf{x}_j) = \mathbf{x}_j^0 + \mathbf{u}(\mathbf{x}_j)$ as:

$$\begin{aligned} \mathbf{f}(\mathbf{x}_i^0, \boldsymbol{\phi}(\mathbf{x}_j)) &= \mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j^0 + \mathbf{u}(\mathbf{x}_j)) && (2.57)\\ &= \mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j^0) + K_{ij}^0 \mathbf{u}(\mathbf{x}_j) + \mathcal{O}(|\mathbf{u}(\mathbf{x}_j)|^2) && (2.58)\\ &= K_{ij}^0 \mathbf{u}(\mathbf{x}_j) + \mathcal{O}(|\mathbf{u}(\mathbf{x}_j)|^2) && (2.59) \end{aligned}$$

Note that $\mathbf{f}(\mathbf{x}_i^0, \mathbf{x}_j^0) = \mathbf{0}$ in (2.58) since no force is generated from the undeformed state of the element. Equation (2.59) highlights that the first order approximation can be written in terms of the displacement field. In the case of a 3-dimensional tetrahedron, let us define $\mathbf{u}_e = [\mathbf{u}(\mathbf{x}_1)\, \mathbf{u}(\mathbf{x}_2)\, \mathbf{u}(\mathbf{x}_3)\, \mathbf{u}(\mathbf{x}_4)]^T \in \mathbb{R}^{12}$ as the concatenation of all the displacement vectors of the four nodes of the tetrahedron. Since the force acting on node i is the sum of all forces contribution of the elements of the nodes, we can obtain a first order approximation of the force $\mathbf{f}_i^0$ acting on node i using this gradient as:

$$\mathbf{f}_i^0 = \sum_{1 \le j \le 4} \mathbf{f}(\mathbf{x}_i^0, \boldsymbol{\phi}(\mathbf{x}_j)) = \sum_{1 \le j \le 4} K_{ij}^0 \mathbf{u}(\mathbf{x}_j) = K_i^0 \mathbf{u}_e \tag{2.60}$$

where $K_i^0 = [K_{i1}^0\, K_{i2}^0\, K_{i3}^0\, K_{i4}^0] \in \mathbb{R}^{3\times 12}$ is the matrix built from the four gradients affecting

the node i. Now, if we define $\mathbf{f}_e^0 = [\mathbf{f}_1\, \mathbf{f}_2\, \mathbf{f}_3\, \mathbf{f}_4]^T \in \mathbb{R}^{12}$ as the concatenation of the four force vectors of each node of the tetrahedron, we can write:

$$\mathbf{f}_e^0 = K_e^0 \mathbf{u}_e \tag{2.61}$$

where $K_e^0 = [K_1^0\, K_2^0\, K_3^0\, K_4^0] \in \mathbb{R}^{12\times 12}$ is built from the four K_i^0 matrices. The matrix K_e^0 is called the stiffness matrix of the element e.

Similarly, considering all the elements of a discretized deformable body, it is possible to assemble a global stiffness matrix K^0 from all the K_e^0 matrices of each element (since a node can belong to several elements, the assembling of the matrices K_e^0 has overlapping parts that are summed up to obtained K^0). Defining a global force vector $\mathbf{f}^0$ as the concatenation of all the forces $\mathbf{f}_e^0$ and a global displacement vector $\mathbf{u}$ as the concatenation of all the displacement vectors $\mathbf{u}_e$, we can write:

$$\mathbf{f}^0 = K^0 \mathbf{u} \tag{2.62}$$

The matrix K^0 is evaluated from the undeformed state at time t_0. In order to do the simulation, the matrix must be recomputed at each step, from the state at time t_i. Since the stiffness is computed from the current state of the body, it can be also expressed from its displacement. We write $K(\mathbf{u})$ the stiffness matrix computed for the state stored in the displacement vector $\mathbf{u}$. Therefore, equation (2.62) can be written in a general form as:

$$\mathbf{f}(\mathbf{u}) = K(\mathbf{u})\, \mathbf{u} \tag{2.63}$$

The evaluation of $K(\mathbf{u})$ for some configuration $\mathbf{u}$ can lead to numerical problem if some elements are ill-conditioned. Methods that compute robust nodal forces based on the deformation gradient have been proposed [Irving 04, Nesme 05], allowing recovering from inverted or ill-conditioned elements .

Computing the stiffness matrix can be computationally costly. If one is interested in modeling only small deformations, then the stiffness matrix can be precomputed only once at the rest state, and used unchanged for small deformed states. This technique is called linear FEM. Using linear FEM for big deformation lead to serious artifacts, as presented in Figure 2.6). A trade-off between linear FEM and classical FEM exists through the introduction of corotational formulation, as presented in the following.

Corotational formulations

It is possible to avoid artifacts of Figure 2.6 produced by linear FEM, by separating the rigid transformation from the non-rigid deformation when computing the forces acting at each node. This method consists in finding the rotational part of an element, and applying the force computed on the unrotated state. The principle of extracting the rotation part is called corotational formulation. In [Müller 02], a rotation information is retrieved at each node, and the authors mention that ghost forces appear due to the method used to retrieve the rotation matrices R_i for each node. The corotational method has then been extended to element-based (see *e.g.* [Etzmuβ 03, Parker 09]), in order to avoid artifacts of node-based formulations.

To understand how corotational formulation works, let us remind the element based computation of forces $\mathbf{f}_e$ form the displacement field $\mathbf{u}_e$:

$$\mathbf{f}_e = K_e \mathbf{u}_e \tag{2.64}$$

It is possible to find a 3×3 rotation matrix R_i which represents the rotational part of a deformed node i. Figure 2.5 shows a 2D example where a deformed triangle is unrotated. The nodal forces are computed using unrotated node positions $R_i^{-1}\boldsymbol{\phi}(\mathbf{x}_i)$ of each node. Most of the time, we choose to have the same rotation matrix for each node of the same element e. This enables to write an element-based rotation using a 12×12 matrix R_e, which is formed putting 16 (4 in the width, and 4 in the height) copies of the 3×3 rotation matrix R_i. The nodal force computation now becomes:

$$\mathbf{f}_e = R_e \, K_e \, R_e^{-1} \mathbf{u}_e \tag{2.65}$$

Namely, the displacement field is first unrotated. Local forces are then computed using the element stiffness matrix K_e, and the forces are transformed back into the rotated state of the deformed element.

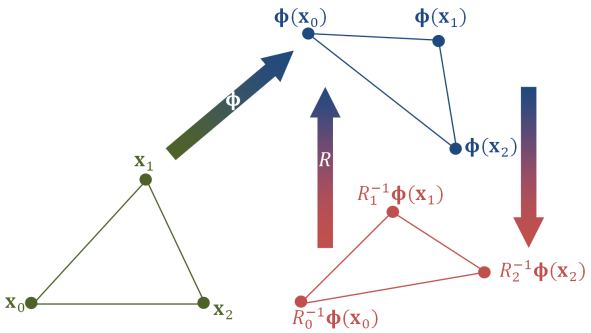


Figure 2.5 – Principle of the corotational formulation. A rotation matrix R_i is retrieved for each node (or for each element) in order to cancel the rigid rotation of the deformed body (in blue) into an unrotated state (in red). The nodal forces are computed from the unrotated state (canceling the artifacts of linear FEM) and are then transformed back into the rotated deformed state.

There are several ways to determine the rotation matrix R_e. It can be found from the transformation matrix P of the element defined in equation (2.50) using *e.g. Gram-Schmidt* method or *polar decomposition* [Müller 04a]. Corotational formulations enable fast and stable simulations of large deformations, without the artifacts of linear FEM presented in the previous section (see also Figure 2.6).

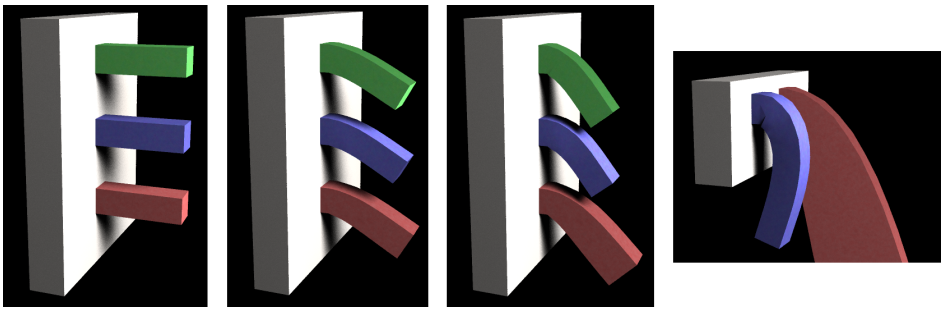


Figure 2.6 – Deformable beams attached to a vertical wall and deforming under gravity. Simulation using non-linear (green), corotational (blue) and linear (red) FEM. Using a longer bar, the non-linear FEM produces ill-conditioned elements for large deformations, the linear FEM shows a large drift, while the corotational FEM remains stable and produces a visually realistic deformation. Images taken from [Müller 02].

Introducing plastic deformation

So far, we related the force applied to a node directly from its displacement, *i.e.* from its deviation from its original position, as depicted on equation (2.62) or (2.63). This model corresponds to elastic deformation only (the higher the deformation, the higher the elastic force magnitudes). In order to take into account plastic deformation, it is possible to set a limit on the deformation of a node, and to set the current deformed state (or an interpolation between original and deformed states) as the new undeformed state. However, this approach has several immediate drawbacks. First, the new undeformed state can have some ill-conditioned elements, and second, since we have a new undeformed reference position, we must recompute the stiffness matrix K each time a plastic deformation occurs.

Müller and Gross [Müller 04a] propose an alternative way to avoid these problems. The reference initial undeformed state remains the same during all the simulation. From the difference between the reference state and the current state, a strain value ε^{total} is computed using equation (2.39). The total strain is actually the sum of the current plastic strain noted $\varepsilon^{plastic}$ and elastic strain noted $\varepsilon^{elastic}$. We now use the superscript "total" because we will differentiate elastic and plastic strain:

$$\varepsilon^{total} = \varepsilon^{plastic} + \varepsilon^{elastic} \tag{2.66}$$

Starting with an initial value of 0 for the plastic strain, the authors propose updating the strain values using the following rules:

1. Compute ε^{total} using (2.39)
2. $\varepsilon^{elastic} \leftarrow \varepsilon^{total} - \varepsilon^{plastic}$
3. **if** $||\varepsilon^{elastic}||_2 > c_{yield}$ **then** $\varepsilon^{plastic} += \delta t \times c_{creep} \times \varepsilon^{elastic}$
4. **if** $||\varepsilon^{plastic}||_2 > c_{max}$ **then** $\varepsilon^{plastic} \times = c_{max}/||\varepsilon^{plastic}||_2$

Three parameters which are c_{yield}, c_{creep} and c_{max} give control to respectively the yield point of the material, the velocity of the plastic absorption, and the maximum plastic deformation. In other words, if the elastic deformation is above c_{yield}, then the plastic strain stores at each time step some elastic deformation, with a limit c_{max} on the plastic deformation.

Keeping at all time a plastic strain and an elastic strain allows computing separately plastic and elastic forces that act on the reference body frame. This enables to work always on the same reference state without having to recompute the stiffness matrix.

2.4.2.2 Meshless models

In meshless models, a body is spatially discretized into a set $\mathcal{X}$ of points but, contrary to the finite element methods, there is no connectivity information on the points. In that context, the continuous quantities of the continuum at an arbitrary point $\mathbf{x}$ are retrieved using shape functions $\varphi_i(\mathbf{x})$ that enable to interpolate the quantity value at point $\mathbf{x}$ from the known values at nodes $\mathbf{x}_i$. Taking the example of the displacement, it is possible to retrieve the displacement $\mathbf{u}(\mathbf{x})$ of a point in function of the displacement of the discrete nodes as:

$$\mathbf{u}(\mathbf{x}) = \sum_{i \in \mathcal{X}} \varphi_i(\mathbf{x}) \mathbf{u}_i(\mathbf{x}) \tag{2.67}$$

In mesh free methods, the volume of the body is sparsely sampled, and each node can a priori act on each point of the continuum. Therefore, the shape function of node i is formed

through kernel functions $w(r,h)$, which is maximal when r (a distance value) is zero, and zero when $r > h$. More information and how $\varphi_i(\mathbf{x})$ is formed in meshless methods can be found in [Pauly 05]. An example of kernel function given in [Müller 04b] is:

$$w(r,h) = \left\{ \begin{array}{cc} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & \text{if } r < h \\ 0 & \text{otherwise} \end{array} \right\} \tag{2.68}$$

For convenience, we denote by $w_{ij} = w(|\mathbf{x}_j - \mathbf{x}_i|, h_i)$ the value of the kernel function using the distance between particles i and j. The value w_{ij} gives an indication on the power of interaction between node i and j (note that if $|\mathbf{x}_j - \mathbf{x}_i| > h_i$, then $w_{ij} = 0$, meaning that node i and j do not interact).

In [Müller 04b], Müller *et. al.* proposed a meshless method to simulate elastic and plastic deformation based on continuum mechanics. They propose using an estimation of the displacement gradient based on a least square minimization. Decomposing the displacement vector using $\mathbf{u}(\mathbf{x}_i) = [u_i \; v_i \; w_i]^T$, we can write a component wise minimization process. The principle is to minimize an error function weighted by the kernel function w_{ij} and given by:

$$e = \sum_{j \in \mathcal{X}} (\tilde{u}_j - u_i)^2 w_{ij} \tag{2.69}$$

where $\tilde{u}_j$ is a first order accurate predicted displacement based on the displacement gradient ∇u_i:

$$\tilde{u}_j = u_i + \nabla u_i (\mathbf{x}_j - \mathbf{x}_i) \tag{2.70}$$

The optimal solution for ∇u_i appears to have the following form:

$$\nabla u_i = A_i^{-1} \left(\sum j \in \mathcal{X} (u_j - u_i)(\mathbf{x}_j - \mathbf{x}_i) w_{ij} \right) \tag{2.71}$$

where A_i is the moment matrix around node i (that can be precomputed). The gradients for the two other components of $\mathbf{u}(\mathbf{x}_i)$ (v_i and w_i) can be computed similarly, and the threes gradients are assembled to obtain the Jacobian $\nabla \mathbf{u}$ of the displacement field. From $\nabla \mathbf{u}$, a strain and stress tensor can be computed. The forces applied at each node are deduced from the strain energy.

More recently, sparse meshless models have been proposed by Faure *et al.* [Faure 11] to accurately simulate deformation of heterogeneous bodies in real time, with a few control nodes. Their method relies on a volumetric map of the material properties and a number of frames to interpolate non-linearly the physical quantities inside the material, and perform an efficient and controllable simulation.

Shape matching

An alternative and non-physically motivated approach for simulating deformation has been proposed in [Müller 05]. The authors present a method that he calls *shape matching* to simulate the deformation of a body represented by a particle system. The main principle of shape matching is to make the undeformed state of the particle system to "fit" with the deformed particle system, in order to defined so-called goal positions from which forces are deduced on each particle to make the system retrieve its initial shape (see Figure 2.7).

A possible way to find the goal positions $\mathbf{g}_i$ is to minimize the sum of the squared distance of the goal position to the deformed position (least square minimization) as:

$$\sum_{i \in \mathcal{X}} m_i \left(\mathbf{g}_i - \boldsymbol{\phi}(\mathbf{x}_i) \right)^2 \tag{2.72}$$

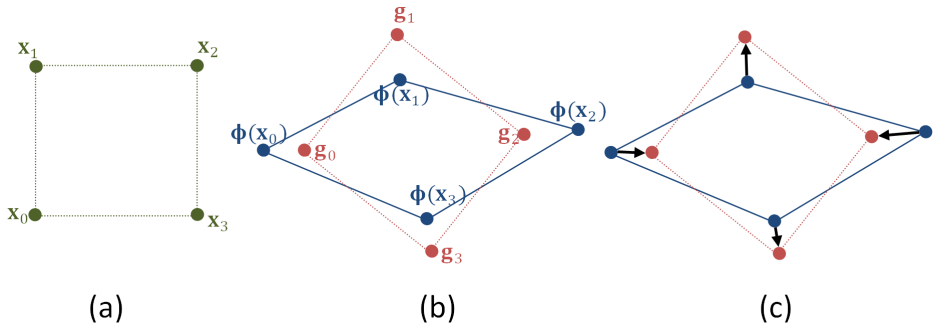


Figure 2.7 – Shape matching technique overview. (**a**): The undeformed state of the system of particles, defining the initial shape of a deformable body. (**b**): The system of particles has been deformed, and a rigid transformation is found that transform the $\mathbf{x}_i$ into the $\mathbf{g}_i$. The points $\mathbf{g}_i$ represent the best "fit" of the undeformed state into the deformed state. (**c**): From the positions $\mathbf{g}_i$, called the goal position, forces are applied on the deformed particles, so that the initial shape is retrieved.

under rigid transformation constraints on the $\mathbf{g}_i$. In (2.72), $\mathcal{X}$ is the set of particles, and the minimization is weighted using the mass m_i of each particle i. We express each goal position $\mathbf{g}_i$ in terms of the undeformed position $\mathbf{x}_i$ of the particle i, the center of mass $\mathbf{x}$ of the particle system, a translation $\mathbf{t}$ and a rotation R.

$$\mathbf{g}_i = R(\mathbf{x}_i - \mathbf{x}) + \mathbf{t} \tag{2.73}$$

First, the particle position $\mathbf{x}_i$ is rotated around the center of mass of the undeformed state, and then translated with $\mathbf{t}$. Substituting (2.73) into (2.72) gives the following expression to minimize:

$$\sum_{i \in \mathcal{X}} m_i \left[(R(\mathbf{x}_i - \mathbf{x}) + \mathbf{t}) - \boldsymbol{\phi}(\mathbf{x}_i) \right]^2 \tag{2.74}$$

The unknowns are the translation vector $\mathbf{t}$ and the rotation matrix R. The optimal translation $\mathbf{t}$ is the center of mass of the deformed position:

$$\mathbf{t} = \frac{\sum_{i \in \mathcal{X}} m_i \boldsymbol{\phi}(\mathbf{x}_i)}{m} \tag{2.75}$$

where $m = \sum_{i \in \mathcal{X}} m_i$ is the total mass of the body. The rotation matrix is not so straight forward to determine. The method proposed by Müller is to relax the problem, finding an optimal linear transformation matrix A that transforms the positions $\mathbf{x}_i - \mathbf{x}$ to fit with the positions $\boldsymbol{\phi}(\mathbf{x}_i) - \mathbf{t}$. From the optimal transformation A, the needed rotational part R can be extracted (*e.g.* with a polar decomposition). Once we know $\mathbf{t}$ and R, the goal positions $\mathbf{g}_i$ are computed using (2.73)

At each simulation step a force $\mathbf{f}_i$, which is proportional to $\boldsymbol{\phi}(\mathbf{x}_i) - \mathbf{g}_i$, is applied on each particle. To allow arbitrary linear deformations (rather than pure rotation and translation), the result can be extended using the following formulation for goal positions:

$$\mathbf{g}_i = (\beta A + (1 - \beta) R)(\mathbf{x}_i - \mathbf{x}) + \mathbf{t} \tag{2.76}$$

Where β is a control parameter that tells how fast the body will retrieve its initial shape. Similarly, it is possible to allow quadratic deformation by replacing the linear transformation matrix A with a more general 3×9 quadratic deformation matrix $\tilde{A}$, that has to be optimized with respect to (2.72).

Since the shape matching technique works directly on position of the particles, it is unconditionally stable. Moreover, the optimal solution for A or $\widetilde{A}$ are computationally cheap to compute, which makes the shape matching suitable for real time simulation.

2.4.2.3 Mass-spring models

Mass spring models are a simple way to simulate deformations of volumetric or thin bodies. The main idea is to spatially discretize the matter into a finite set $\mathcal{P}$ of n particles $i \in \mathcal{P}$ $(1 \leq i \leq n)$. Each particle i has a mass m_i and a position $\mathbf{x}_i \in \mathbb{R}^3$. The particles are connected with springs $s \in \mathcal{S}$ $(1 \leq s \leq m)$ ($\mathcal{S}$ being the set of m springs constituting the network). Each spring s has a stiffness k_s, a damping factor d_s, a rest length l_{0s}, and two more scalars representing the index of the two particles that are linked with this spring.

The particles are submitted to the classical dynamic equilibrium $\ddot{\mathbf{x}}_i = m_i \mathbf{f}_i$ where $\mathbf{f}_i \in \mathbb{R}^3$ is the total force acting on particle i. We can decompose $\mathbf{f}_i$ into $\mathbf{f}_i = \mathbf{f}_{exti} + \mathbf{f}_{spr_i}$ where $\mathbf{f}_{exti}$ are the external forces such as gravity or collision forces, and $\mathbf{f}_{spr_i}$ are internal forces of the body coming from spring network, both acting on particle i. If $\mathcal{J} \subset \mathbb{R}$ is the set of particle indexes which are neighbor of particle i (i.e. which has at least one spring linking with particle i), we can write:

$$\mathbf{f}_{spr_i} = \sum_{j \in \mathcal{J}} \left[\left(k_s(|\mathbf{x}_j - \mathbf{x}_i| - l_{0s}) \frac{|\mathbf{x}_j - \mathbf{x}_i|}{(\mathbf{x}_j - \mathbf{x}_i)} \right) - \left(d_s(\dot{\mathbf{x}}_j - \dot{\mathbf{x}}_i) \cdot \frac{|\mathbf{x}_j - \mathbf{x}_i|}{(\mathbf{x}_j - \mathbf{x}_i)} \right) \right] \tag{2.77}$$

Into the sum of (2.77), the first left term in parenthesis represents proportional force acting along the direction of the spring and proportional to the difference of length between the current length and rest length of the spring. The right term is the damping factor which is proportional to the velocity of the variation of the spring length. Finally, the scalar s is the index of the spring linking particle i to particle j.

Mass spring networks were the first models used to simulate deformation, and are widely used due to their simplicity and their cheap computation requirements [Gibson 97]. In computer graphics, mass spring network have been used for rope, cloth [Baraff 98], skin, and volumetric bodies deformation and fracture [Hirota 98]. Its main drawback comes from the fact that the behavior of the simulated body is dependent of the springs disposition, stiffness and damping factor. Also, maintaining volumetric constraints such as volume conservation cannot directly be handled by mass spring systems. Those drawbacks make spring network model very hard to use when one has to tune the network and the numerous coefficients until having the desired behavior.

2.4.3 Reduced deformable models

2.4.3.1 Introduction

Reduced deformable models purpose is to express the initial state $\mathbf{q} \in \mathbb{R}^n$ of a body using a reduced number of degrees of freedom (DOF) state $\mathbf{z} \in \mathbb{R}^m$ $(m < n)$ and a set of *modes of deformation* stored in a matrix $W \in \mathbb{R}^{n \times m}$, with the following relation:

$$\mathbf{q} = W\mathbf{z} \tag{2.78}$$

Each column of the $n \times m$ matrix W is a vector of n scalars representing a mode of deformation. Namely, it will represent the state taken by the body if the corresponding value

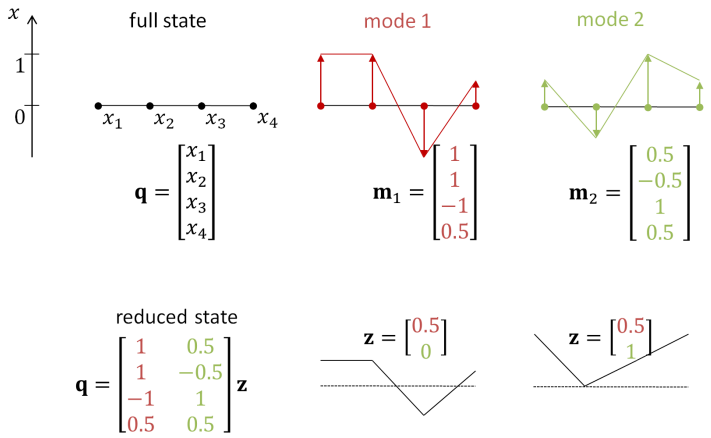


Figure 2.8 – Example of reduced deformable model on a small system. To be read left to right, top to bottom. A system with 4 DOFs is reduced in 2 DOFs with two modes of deformation (mode 1 in red, and mode 2 in green).

in the reduced state $\mathbf{z}$ is non-zero. The reduction of a small 4-DOFs system is illustrated on Figure 2.8. Complementary information on reduced models can be found in [Barbič 07a].

The immediate advantage in model reduction is the simplification of a system with a potentially large number of DOF to a system with a lower number of DOFs, leading to potentially great computational savings during the simulations. On the other hand, the range of deformation allowed using a reduced state is lower than the initial range of deformation allowed with the full dimension of the state $\mathbf{q}$.

The choice of the reduction basis W (*i.e.* the set of modes of deformation) depends on the targeted application and the amount of approximation tolerated. In the following, we present a commonly used reduction basis defined from the study of the dynamics of deformable bodies, namely the modal basis computed using modal analysis.

2.4.3.2 Modal analysis

The differential equation expressing the dynamics of a deformable body can be written in a linear version as:

$$K\mathbf{u} + D\dot{\mathbf{u}} + M\ddot{\mathbf{u}} = \mathbf{f} \tag{2.79}$$

where K, D and M are the stiffness, damping and inertia matrices of the system. The vectors $\mathbf{u}$ and $\mathbf{f}$ represent the displacement field and sum of external forces applied on the system. The main purpose of modal analysis is to diagonalize equation (2.79) to get a set of isolated differential equations, and get an analytical solution for this linearized system. To do so, the damping matrix D assumed to be expressed as a linear combination of the mass and the stiffness matrices (Rayleigh damping): $D = \alpha K + \beta M$. Diagonalizing the system can then be done by solving for the eigenvectors W_i and eigenvalues λ_i of the generalized eigen-problem $K\mathbf{x} + \boldsymbol{\lambda} M\mathbf{x} = 0$. The matrix W that embeds all the eigenvectors diagonalizes both K and M:

$$W^T K W = \Lambda \qquad W^T M W = ID \tag{2.80}$$

where $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ gathers all eigenvalues in one vector. Equation (2.79) can

now be transformed into :

$$\Lambda(\mathbf{z} + \alpha\dot{\mathbf{z}}) + (\beta\dot{\mathbf{z}} + \ddot{\mathbf{z}}) = \mathbf{g} \tag{2.81}$$

where $\mathbf{z} = W^{-1}\mathbf{u}$ is the vector of modal coordinates and $\mathbf{g} = W^T\mathbf{f}$ is the force vector transformed in the modal basis. Equation (2.81) represents a set of independent equations (one for each row i) for which we have an analytical solution of the form:

$$z_i(t) = c_{1i}e^{t\omega_i^+} + c_{2i}e^{t\omega_i^-} \tag{2.82}$$

where c_{1i} and c_{2i} are complex constants, and the ω_i are the angular frequencies of mode i (t is a scalar representing time). The response of a force $\mathbf{g}$ (expressed in modal coordinates) applied on the system for a duration Δt, is modeled through the constants c_{1i} and c_{2i} of each mode i with:

$$c_{1i} = \frac{2\Delta t\, g_i}{\omega_i^+ - \omega_i^-} \qquad c_{2i} = \frac{2\Delta t\, g_i}{\omega_i^- - \omega_i^+} \tag{2.83}$$

In computer graphics, the use of modal analysis has been first introduced by [Pentland 89] to provide fast computation of the dynamics of deformations for animation. Later, O'Brien *et. al.* leveraged modal analysis to propose efficient sound synthesis from rigid body simulations [O'Brien 02b]. Their work has been extended in [Hauser 03] to simulate reduced deformation with a user interaction. To do so, they present a way to include constraints in the simulation based on modal analysis. More recently, reduced models have been used to achieve better performances in haptics [Barbič 07b, Wan 11], and interactive applications [James 02, Huang 10]. Another limitation of modal analysis occurs when only a subset of the deformation modes is retained. Usually, the lower frequency modes that generate the more visible deformations are kept, and the higher frequency ones are ignored. Therefore, higher frequency phenomena such as local deformations are not well represented with modal analysis. To cope with this observation, Barbič *et al.* [Barbič 11] proposed a decomposition of the initial body into several domains. For each domain, a modal analysis is performed, and the domains interact through rigid links. This lead to a viable solution to maintain model reduction while being able to simulate local deformations.

2.4.4 Summary table

We present in table 2.1 a summary of the main methods available to simulate deformation. Each method is compared through criteria of efficiency and realism. The methods are compared w.r.t. five criteria. The efficiency and physical conformity are defined at the beginning of this chapter. In most brittle materials, the deformations that lead to the fracture are small. Therefore, the physical conformity criterion is considered on small deformation, while the "large deformation" criterion that defines how well the method keeps its physical conformity as the deformation increases. Finally, the "easy to tune" criterion defines the ease of tuning the parameters of the simulation to obtain the desired results. It is often linked to the nature of the parameters used. If the parameters are based on physical quantities, it is usually easier to tune them and to understand their influences on the simulation outcome.

On the efficiency criterion, modal analysis is at the first place. Indeed, as a result of modal analysis, analytical solution of the deformation over time is known, and no integration method is necessary. Also, a reduced number of modes can be considered as well as parallel computation for modal analysis, leading to very fast deformable simulations [James 02]. This

Method name	Efficiency[1]	Physical conformity	Large deformations	Easy to tune	Topological changes
non-linear FEM	■□□□□	■■■■□	■■■■■	■■■■□	■■■□□
linear FEM	■■■□□	■■■□□	■□□□□	■■■□□	■■■□□
linear FEM + corotational	■■■□□	■■■□□	■■■□□	■■■□□	■■□□□
mesh free [Müller 04b]	■■□□□	■■■□□	■■■■□	■■■□□	■■■■■
shape matching	■■■■■	■□□□□	■■■■□	■■□□□	■■■□□
mass-spring	■■■■□	■■□□□	■■■□□	■□□□□	■■■□□
modal analysis	■■■■■	■■■□□	■□□□□	■■■□□	■□□□□
modal analysis +2nd derivative	■■■■■	■■■□□	■■■□□	■■■□□	■□□□□

Table 2.1 – Comparison of deformable body simulation methods w.r.t. to efficiency of computation and realism. Each cell contains a mark that ranges from 0 (blank squares) to 5 (filled squares).

efficiency is the prize of being able to model accurately only small global deformations. Adding second order deformation mode [Barbič 07a] allows modeling large global deformations, but no large local deformations. Mass-spring systems are also efficient due to their simplicity. However, modeling accurately the observed deformation is a tedious task, and involves playing with numerous parameters. On the physical conformity criterion, the general FEM has the biggest score, enabling to express a very wide range of non-linear deformation, taking into account any constitutive law and changes of the material property over time. Meshless system can be a good trade-off for modeling large deformation and topological changes, since re-sampling can be efficiently performed to enrich information on the area of large strain. Finally, regarding the topological changes, structures that necessitate precomputations such as modal analysis are not efficient, since the precomputation has to be performed again at each topological change.

2.5 Physical simulation of brittle fracture

When a body deforms too much, it breaks. Although a bit oversimplified, this statement expresses well the intimate link between the simulation of fracture and the simulation of deformation. For this reason, each model used for the simulation of deformation has been extended to model fracture as well. Unfortunately, extending the deformation models to handle fracture is not always straightforward, and usually adds a non-negligible computation cost to the simulation, compromising their use for interactive applications. We review in this section the different methods proposed in the literature to simulate fracture. At the end of the section, we provide a summary table that compares each method w.r.t. the main objectives of the manuscript: a real-time, physically-based fracture simulation for interactive applications.

2.5.1 Geometrical approaches and cutting algorithms

Some authors treated fracture patterns taking a purely geometrical approach. The main principle of geometrical approaches is to design an algorithm that produces visually plausible

fracture patterns. In [Desbenoit 05], the authors presented an editor that enables to interactively map a fracture pattern on surface meshes. The main concern of their paper is the algorithm that enables to apply the fracture pattern on arbitrary meshes (see Figure 2.9).

Neff *et al.* [Neff 99] defined a recursive pattern generator to define 2D shards exploding. The underlying visual representations of fractured and deformed bodies are rarely used for physical simulation, and applying crack patterns to a visual mesh is also a field of research (see *e.g.* [Viet 06]).

Figure 2.9 – Artist-designed fracture patterns applied on an egg (left) and on a vase (right) [Desbenoit 05].

In general, when the crack surface is not represented by the mesh elements boundaries, remeshing or cutting algorithms are required to update the rendered mesh according to the crack surfaces. Algorithms that manage robust cuts of tetrahedra [Bielser 03, Sifakis 07] or first order cut of cubes [Pietroni 09, Dick 11] have been proposed. Most of these algorithms can be defined by a state machine that track the state of the edges of the elements (whether they are cut) to decide which surface triangles to produce. These algorithms are generally fast enough to be compatible with interactive simulations.

2.5.2 Mass-spring systems for fracture

The extension of mass spring systems (see section 2.4.2.3) for fracture is mainly performed by removing a spring linking two particles if its length, or its length variation exceeds a threshold representing the material resistance to fracture. Norton *et. al.* [Norton 91] proposed modeling the deformation of objects using mass spring network. A fracture is initiated into the network when two particles are separated by more than a maximum distance which is a parameter of the fracturing. In order to avoid strings of springs hanging, blocks of springs are removed at once during fracture.

Later Hirota *et. al.* [Hirota 98] proposed building a mass-spring network composed of two layers: a surface layer and a sublayer that enable them to measure shearing stress. The rest length of the spring of the surface layer is shrinked over time, allowing cracks to open and propagate. The authors demonstrate that different fracture patterns can be obtained by changing the topology of the network.

These two methods are based on the length deformation of the springs to decide where the springs should be cut, but no accurate information is obtained on the precise location of the beginning of the crack, or on the separation plane. Also, the crack pattern obtained depends on the underlying network topology, which can give control on the patterns, but become costly when one wants to generate high resolution crack lines.

2.5.3 Cohesive zone models

The main idea of the cohesive zone model is to partition the body into a set of volumetric pieces that are originally stuck together [Elices 02]. Then, when a loading force is applied on the body, the cohesive forces that hold the elements together are computed, and if the

cohesive force between two adjacent elements exceeds a threshold, the two elements are separated. Smith *et al.* [Smith 01] were the first to propose the cohesive zone model in the computer graphics literature. At each time step, they solve a linear system under constraints computing the cohesive forces magnitudes for each couple of linked tetrahedral elements. When the cohesive force is above a threshold, the link between the common face of the two tetrahedra centers is broken, and the neighboring faces resistance to fracture are weakened in order to give a higher chance for neighboring faces to propagate the cracks. The authors shown results obtained with this method by shattering brittle objects such as a glass table, glazed ceramic bowl and wine glasses. The main drawback of their approach is that the fracture lines are dependent of the tetrahedron mesh, and this can produce unrealistic fracture patterns.

The cohesive zone model is widely used in special effect for movies, because it provides believable results and is controllable by the artists [Zafar 11] (the shape of the elements as well as the weaknesses between elements can be designed manually).

2.5.4 Continuous approaches for fracture

Continuous mechanics provide a natural way to define fracture criteria, since it can provide a stress value for any point of a body.

2.5.4.1 Finite differences for fracture

Terzopoulos and Fleisher [Terzopoulos 88] brought the first attempt to model fracture in computer graphics. In their deformation simulations, the energy forces are approximated using finite differences. To model tearing, the authors proposed removing the link between two adjacent nodes whenever the distance between these two nodes exceeded a threshold. The results were demonstrated on a sheet of paper being torn.

2.5.4.2 Finite Element Method for fracture

Using the Finite Element Method (see section 2.4.2.1) has the advantage of being able to use fracture mechanics in a straightforward way to model fracture. To decide whether a crack should initiate or not, the stress value σ_e of each element e is exploited. Most often, the maximum principal stress magnitude (given by the maximum eignevalues λ and its associated unit-length eigenvector $\mathbf{n}$ of the stress tensor matrix σ_e) is compared to a threshold τ that represents the material resistance to fracture. If we have $\lambda > \tau$, a fracture occurs in the element e. Mainly based on $\mathbf{n}$ that represents the normal vector of the fracture plane, different strategies are adopted to decide how the body mesh and material properties are updated.

O'Brien *et. al.* [O'Brien 99] brought a new standard in graphical modeling and simulation of brittle fracture in 1999. Based on the FEM discretization (see section 2.4.2.1) of continuum mechanics, they use the principal stress values of the stress tensor to define what they call a separation tensor. To build the separation tensor, the stress tensor is first separated into tensile stress tensor (built from the eigenvectors of the stress tensors that have positive eigenvalues) and a compressive stress tensor (built from the eigenvectors of the stress tensors that have negative eigenvalues). Finally, the separation tensor is built from a balanced tensile and compressive forces (which are deduced from the tensile and compressive stress tensors).

The positive eigenvalues of the separation tensor are then compared to a threshold to decide whether a fracture should occur or not. The separation plane of the fracture is given by the normal plane to the corresponding eigenvector. The authors remesh the tetrahedron locally around the crack tip, enabling to build unconstrained crack paths as shown on Figure

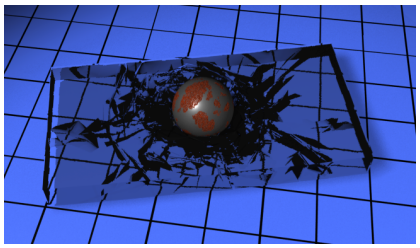


Figure 2.10 – A glass slab being broken by a metallic ball [O'Brien 99].

2.10. In order to avoid badly conditioned elements due to remeshing, the crack path is aligned to the elements boundaries when close to them. To reproduce the behavior of stiff materials without instability, small time steps and reduced stiffness were taken during the simulation. The time steps used are however incompatible with real-time simulation

In 2002, their work was extended to ductile fracture [O'Brien 02a]. To do so, the authors introduce a plastic strain that keeps track of the plastic deformations of the material. This tensor is updated at each step, in a similar way as the plastic deformation presented in section 2.4.2.1.

Figure 2.11 – Surface cracks pattern generated physically on the dragon [Iben 06].

In 2001, Müller *et. al.* [Müller 01] proposed a real time simulation of brittle fracture, based on a static analysis of the stress tensor around impacts. When an impact occurs between a fixed body and another one that can break, the part of the body which is far for the impact zone body is "anchored", and a static equilibrium is computed with the finite element method to estimate the deformation state around the impact zone. Fractures are then launched around the colliding point based on the principal stress values retrieved. The authors demonstrate the capabilities of their techniques by dropping various brittle objects of few hundreds of tetrahedra that break after impacts, all in real time. In their method, the fracture crack propagates only along the boundaries of the elements of the tetrahedral mesh, resulting into constrained crack lines.

Later, Müller and Gross [Müller 04a] applied stress tensor analysis with linear FEM discretization using corotational formulation to obtain fracture on stable and fast computation of stiff or large deformations. The accompanying video shows how it is possible to play with the simulation parameters such as Young's modulus, Poisson ratio, and fracture threshold, to simulate various fracturing phenomenon in real time, on both plastic and elastic deformations. For visual rendering, the triangle mesh and tetrahedron mesh are dissociated, and new triangle surfaces are created and cut dynamically. Again, the fracture patterns are dependent on the underlying tetrahedron mesh, which makes difficult the generation of high resolution crack lines.

In [Müller 04c], the authors propose retrieving data from a surface mesh: the deformation is computed from a regular grid that discretizes the volume of the meshes. The stress tensors

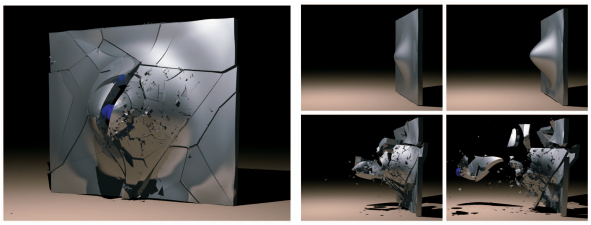


Figure 2.12 – An example of FEM-based fracture with plenty of small shards [Bao 07].

of the elements are analyzed to decide whether a fracture should open. After fracture, the graphical mesh is generated by adding interior triangles following the edges of the regular grid. This leads to a fast and stable simulation of deformation and fracture. Here the resolution size of the grid has a direct impact on the appearance of the fracture patterns and the computation times.

Oda and Chenney [Oda 05] proposed an adaptive and progressive refinement of the tetrahedron mesh around the potential impact zones, in order to circumvent the weird shapes that can take the fracture surfaces when they follow a predefined mesh.

FEM-based techniques have also been used to model surface crack patterns by evolving a stress field that can be heuristically driven for controllability [Iben 06].

In 2007, Bao *et. al.* [Bao 07] presented another use of Finite Element Method to simulate brittle and ductile fracture. They used velocity level stress quasi-static analysis that they call time averaged stress (*i.e.* stress integrated over a time step) principal axis to determine whether a fracture should occur. In order to generate various patterns, randomly seeded points define energy functions that influence the crack lines. In order to have accurate and mesh independent fracture patterns, the authors used the virtual node algorithm [Molino 04] that allows arbitrary fracture path and geometry through elements without generating ill-conditioned elements, as illustrated in Figure 2.12. They also developed a specific algorithm based on [Bridson 02] for treating contacts between small shards represented only by their triangle surface. Their algorithm demonstrated robustness and stability over the various examples provided but, even if no timing is mentioned, the complexity of their method seems not suited for real-time applications. A similar method has been used in the context of the simulation of the sound of breaking objects [Zheng 10]. The authors performed a quasi-static analysis of the deformation of the bodies during impact to localize the regions of high strain, and sample Voronoï cells centroids with a probability proportional to the strain. They used a remeshing technique to represent the cracks surfaces [Bielser 03].

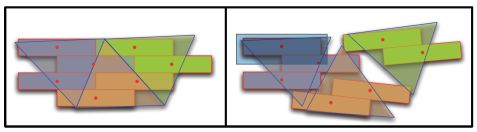


Figure 2.13 – Dissociation of visual rendering and physical model. Each tetrahedron contains visual elements whose centroids (red points) are inside. When the tetrahedra are separated during fracture (right), visual surfaces are created according to the visual element [Parker 09].

Parker and O'Brien [Parker 09] presented a combination of former work that lead to a fast and robust simulator of stiff or soft bodies with fracture. They modeled deformation using FEM as presented in [Müller 04a] and performed a stress analysis for fracture. They introduced a nice method called splinters that links graphical representation and tetrahedron for the FEM. When two tetrahedron separate, the graphical crack surface is defined by the splinters shape whose centroids are belonging to the separating elements (see figure 2.13).

Their method is optimized for parallel computation, and brought fast and believable results in their test cases.

2.5.4.3 XFEM: Extended Finite Element Method

Although well suited to model the deformation, the finite element method has some limitations to model cracking as presented in the previous section. A remeshing must be performed to model free crack propagation in a FEM framework, which can lead to numerical problems or inaccuracies. The main idea of the eXtended Finite Element Method (X-FEM) [Belytschko 99] is to express the material discontinuity directly into the shape functions of the elements, instead of explicitly model the boundaries of the discontinuities. To do so, discontinuous *enrichment* functions that define the stress at the crack tip are combined with the shape function of the elements. Therefore, the discontinuity due to the crack propagation can be modeled without remeshing. In order to keep track of the crack formation, each node belonging to a cracked element is augmented with an enrichment function [Moës 99]. The X-FEM method has been initially defined in 2-D, and Sukumar *et al.* [Sukumar 00] proposed an extension to 3-D cracks, and demonstrate the accuracy of the technique. Later, the method has been used in the context of interactive stable simulation [Jeřábková 09], where the authors demonstrated the efficiency of their method through a virtual surgery example. Kaufmann *et al.* [Kaufmann 09] presented a generalization of the enrichment functions into a regular grid that they named enrichment textures. They show how to represent arbitrary complete and partial cut in an element through these textures. Complementary information on the X-FEM and its application can be found in the survey of Abdelaziz *et al.* [Abdelaziz 08].

To our knowledge, no work has been presented on the use of the X-FEM for brittle fracture. Actually, this method tackles the problem of the representation of the discontinuity in an FEM framework, and is not specific to a particular type of fracture. Using X-FEM for real-time brittle fracture simulation could be doable, provided that the deformation of stiff bodies could be done in real-time. Also, if the discontinuity is modeled from the point of view of the physical simulation, it is not automatically represented on the mesh that is rendered on the screen. A few more steps are thus required to update the visual mesh according to the cracks.

2.5.5 Particle system for fracture

Particle systems without mesh information (see section 2.4.2.2) have also been extended to model fracture. The main advantage of meshless model for fracture is that when crack surfaces are created, new sampling point can be naturally added into the system. Pauly *et. al.* presented in [Pauly 05] how they modeled fracture phenomena of brittle or ductile object using a meshless particle system. To find the location of maximum stress within the body, they sample a random set a point within the body and keep the one on which the principal stress is the highest. If this maximum stress is above a threshold, a fracture is initiated. The authors propose a dynamic sampling of the cracks surfaces, an automatic re-sampling for deformation computation, a way to adapt the weight function between two nodes that are separated by a crack, and the handling of topological events (see Figure 2.14). Meshless methods present some issues for real-time methods, since the visual mesh to render has to be generated from the set of sampling points, and the number of points can grow in an unpredictable manner.

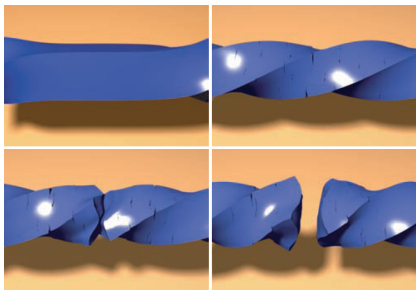


Figure 2.14 – Simulation of a ductile fracture using meshless deformation model [Pauly 05].

2.5.6 Summary table

We present in table 2.2 a summary of the state of the art methods available to simulate fracture. Each method is compared through criteria of efficiency and realism. The methods are compared w.r.t. five criteria. The efficiency and physical conformity are defined at the beginning of this chapter.

	Brittle fracture		Ductile fracture		
Method name	Efficiency	Physical conformity	Efficiency	Physical conformity	Dynamic fracture surfaces
mass-spring	■□□□□	■□□□□	■■■■□	■■□□□	NO
Cohesive zone model	■■■□□	■■■□□	-	-	NO
FEM (remeshing)	■□□□□	■■■■□	■■■□□	■■■■□	YES
FEM corotational	■□□□□	■■■□□	■■■■□	■■■□□	NO
static analysis	■■■□□	■■■□□	-	-	NO
static analysis + Voronoï	■■□□□	■■■□□	-	-	YES
mesh free	■□□□□	■■■□□	■■■□□	■■■□□	YES

Table 2.2 – Comparison of the different fracture methods w.r.t. to efficiency and realism.

Conventional methods (FEM, mass-spring and meshless deformation) are not efficient to simulate brittle fracture because tiny time steps must be used to capture the stiff deformations of the body, leading to high computational costs. To cope with this problem, a static analysis of the body can be performed to study its deformation at a desired time, and decide whether the fracture will occur. However, quasi-static approaches ignore the dynamic effects that play an important role on impact-based fractures. Some methods rely on element boundaries of the physical mesh to represent the fracture surface, leading to less realistic constrained crack paths. Finally, there is no method able to provide real-time and physically-based brittle fracture simulation with dynamic fracture surfaces.

2.6 Interactive simulation

Aiming at real time simulation of a particular phenomenon makes it eligible for being used in interactive applications. However, in most cases, the isolated simulation of a particular phenomenon has to be complemented with methods that allow interaction with it. Detecting the collisions with the concerned bodies, and react to these interactions is a needed feature for most interactive applications. In this section, we first present an overview of the available collision detection methods, discussing their applicability to brittle fracture. In a second part, we discuss the haptic interaction, a sensory feedback used during our project for evaluation and interactive applications.

2.6.1 Detecting collisions between the bodies

Collision detection is a necessary step to treat the different interactions that can occur between the bodies of a virtual world. In a three dimensional space, a *collision* occurs between a body A and body B if the volume of A overlaps with the volume of B. The first role of collision detection is to check whether bodies are colliding. If so, information such as the position of the overlapping part, the penetration depth or volume needs to be computed to *solve* the collisions. Solving the collision corresponds to apply forces, *impulses*, or *position correctors* that make the colliding bodies to separate, and avoid unrealistic overlapping. The solving methods are presented in Section 2.6.2.

The method used for checking whether a pair of body is colliding is dependent on the representation of the volume of the bodies. A wide variety of numerical structure has been studied in the literature. For complementary information, we refer the reader to general surveys on collision detection [Lin 98, Teschner 05].

The collision detection process is usually decomposed in 3 steps: a *broad-phase*, a *mid-phase*, and a *narrow-phase*. The broad-phase and mid-phase purposes are to quickly eliminate pairs of bodies that are not in collision using bounding volumes and other approximate structures. More information on broad-phase and mid-phase algorithms and performances is presented in [Avril 09]. In this manuscript, we focus only on the narrow-phase, responsible for checking the collision using the exact representation of the bodies.

2.6.1.1 Representation of the bodies for collision detection

Implicit shapes

Implicit shapes model the volume of a body with a mathematical surface of the form $\{\mathbf{x}|f(\mathbf{x}) = 0\}$ where f is a function defining the shape of the body. This representation is common in rigid body engines for shapes such as cubes, spheres, cones, cylinders, and capsules. These five simple shapes can be efficiently represented with implicit surfaces. Each of these shapes is parametrized with a few parameters, and algorithms checking the collision between a pair of two of these shapes are readily available. Handling collision for general implicit surfaces is more involving [Lin 95], and has no general exact solution.

Constructive Solid Geometry

Constructive Solid Geometry (CSG) purpose is to model a geometry using a set of primitive shapes, and binary operations such as unions, intersections, and differences [Hoffmann 89], as shown in Figure 2.15.

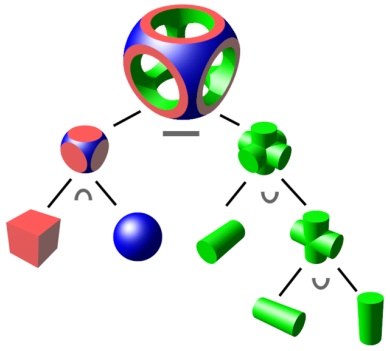


Figure 2.15 – Example of a CSG Solid constructed with the difference between an intersection and two unions of primitives shapes.

Mostly used for rendering purposes, CSG models have also received some attention in the collision detection field. However, it is not easy to check whether the intersection of two bodies represented by CSG is empty or not. Although some solutions consisting in approximating the CSG surfaces for collision detection have been studied, getting the general analytical solution to the collision test between CSG remains an unsolved problem [Keyser 97].

Distance fields

Distance Fields store in a grid distances to the surface of an object, and possibly the distance gradient. For rigid bodies, distance fields may be precomputed, hence the computation of penetration depth of a point inside a rigid body becomes trivial [Guendelman 03]. Adaptive distance fields [Frisken 00] store distances in an octree to reduce storage requirements. In some applications it is even sufficient to store information only near the surface of the object [McNeely 99]. Distance fields have also been used for deformable bodies by fast recomputation [Sud 06] or by approximating finite-element [Fisher 01] or modal deformations [Barbič 07a]. In various applications of computer animation, distance fields have been approximated using front propagation algorithms or graph-based distances [Steinemann 06].

Point shell

Point shell is a representation of the surface of the body into a set of discrete points. The immediate advantage of the point-based representation is that checking if a point is inside a geometry is often a cheap test. However, the point shell representation cannot be used alone, *i.e.* point-shell/point-shell collision checking is not possible. In the field of collision detection, point shell has been first used to detect efficiently the collisions between a moving body against a static environment [McNeely 99]. In this work, the environment was represented with a map of voxels. The algorithm has then been improved [Renz 01, Barbič 07b] to more stable or time-critical versions. The main drawback of point-shell based method is that some collisions may be missed since the surface is only represented at discrete locations.

Polygonal models

Another popular representation is the polygonal soup (often a triangle mesh) representing surface of the body. The triangle mesh has historically been used for rendering of the objects in a 3-D scene (for rendering purposes, representing only the surface of the mesh is sufficient). Although working with this structure for collision detection makes sense since it represents

the exact geometry that is displayed, checking naively whether two triangle meshes are overlapping has a complexity of $\mathcal{O}(n^2)$, n being the number of triangles of the meshes. In order to reduce this complexity, several additional structures have been proposed, as presented in the following section.

2.6.1.2 Algorithms and accelerating structures for polygonal models

Algorithms for collision detection between polygonal models

A wide variety of algorithms has been studied for collision detection between polygonal models. These algorithms could be classified in two categories: the algorithms for convex-convex collision detection, and the algorithms involving non-convex bodies.

For convex bodies, a well-known algorithm is the separating plane [Baraff 90, Chung 96] that exploits the temporal coherence and the convexity of the bodies to find a plane that separate a pair of objects, and decide whether they are colliding. Another popular approach for convex bodies is the so-called GJK algorithm [Gilbert 88]. This algorithm is based on the Minkowski difference between a simplex representation of the bodies. The authors show that checking whether two bodies are colliding is equivalent to check whether the origin is contained into the simplex obtained by the Minkowski difference between a pair of objects. Other approaches leveraging Voronoï decomposition of the space [Lin 91] and hierarchical decomposition of the polyhedron [Dobkin 90] have also been studied.

All the above algorithms leverage the convexity of the bodies to operate. Dealing directly with non-convex bodies is much more difficult, and a possible workaround is to decompose concave objects into sets of convex objects, and then apply the above algorithms for convex objects. Algorithms based on the sampling of the geometries of the objects into a regular grid (image-based methods) using rendering hardware can generally cope with both convex and non-convex objects [Teschner 05, Faure 08, Allard 10]. But these algorithms are approximate and are sensible to the projection planes chosen for the rendering. Another solution to deal with non-convex bodies is to use additional precomputed structures that allow accelerating the detection collision queries, as presented in the following paragraph.

Hierarchical structures for polygonal models

A common idea to accelerate the collision detection between polygonal models is to use hierarchical structures [Samet 95] to organize the primitives of the polygon. In the literature, hierarchies such as sphere-trees [Palmer 94, Weller 09], kd-trees [Klosowski 98] and octrees [Bandi 95] have been widely studied for detection collision. The main idea of these structures is to subdivide the polygonal mesh into regions that are organized into a hierarchy, as shown on Figure 2.16.

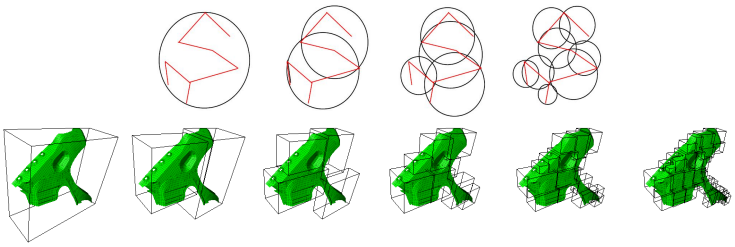


Figure 2.16 – Example of two acceleration structures commonly used in collision detection. **Top**: A sphere-tree built on a 2-D geometry. **Bottom**: An Axis-Aligned boxes tree on a 3-D geometry.

Hierarchical structures allow efficient overlapping checks between bodies. In order to check if two hierarchies are colliding, an efficient top-down search can be performed, pruning at an early stage the regions of the body that are not colliding. First, the two roots of the two hierarchies are checked, then the node at level one of the first hierarchy against root of the second hierarchy, etc. As long as a pair of node is colliding, the children of this node are visited to refine the query, until the leaves are reached. A more detailed explanation of the recursion steps on hierarchical structure for collision detection can be found [Teschner 05].

2.6.1.3 Collision detection desired queries

Even if the first purpose of collision detection is to check whether two objects are overlapping, this information is not enough to *solve* the collision. Indeed, the direction in which the objects should be pushed away to make the overlap vanish should be known as well. The position of the overlaps – the contact positions – are also needed to solve the collisions. Depending on the method used for collision solving, the needed information will vary. In general, a collision solver will need one or more of the following queries when treating a pair of bodies (A, B):

- **Check overlap**: Return whether the pair of bodies is colliding, *i.e.* if the intersection of their volume is not empty.
- **Get contact position**: For each overlapping area, return the position in world coordinate of a contact point representing the location where the pair of bodies touched for the first time (or an approximate position of this location).
- **Get penetration depth**: For each overlapping area, return the maximum distance from a point of body A inside the volume of body B to the surface of body B, or from a point of body B inside the volume of body A to the surface of body A.
- **Get penetration volume**: As an alternative to the penetration depth, some collision solver work with the penetration volume of each overlapping area rather than penetration depth.
- **Get distance to surface**: Return the distance of a point to the surface of a body. This query is mostly used by detection collision algorithms to accelerate the process by pruning non-colliding parts of the body.

2.6.1.4 Collision detection and haptic rendering for fracture simulation

Acceleration data structures for collision detection need to be updated or recomputed at fracture events, because precomputed distances or bounds are no longer valid or tight, and new surfaces need to be considered. Larsson and Akenine-Möller [Larsson 01] introduced the concept of selective restructuring of bounding volume hierarchies, according to fitting-quality metrics. Otaduy *et al.* [Otaduy 07] applied local restructuring operations to limit updates in progressive fracture. Recently, Heo *et al.* [Heo 10] have presented an algorithm that finds a good compromise between bounding volume restructuring and fast recomputation. All these approaches suffer two major limitations for simulations of brittle fracture. First, the quality of bounding volume hierarchies degrades immediately under brittle fracture, and full recomputations are needed. As a result, large computational spikes decrease the simulation cost at fracture events. Such spikes can be amortized in offline simulations, but not in hard real-time applications such as video games.

Due to the lack of real-time methods suitable for brittle fracture simulation and collision detection with fracturable bodies, there is to the best of our knowledge no previous work on haptic interaction with brittle fracture.

2.6.2 Handling contacts and constraints between the bodies

Bodies that are colliding will interact each other. From the collision detection step, we obtain a set $\mathcal{C}$ of contacts that contains information such as position or separating normal as presented in the previous section.

Solving for the collision can be performed at three different levels:

- The acceleration level – *forces*. Resolving contacts at the acceleration level consists in finding forces from the current state that counter the inter-penetrations when integrated and added as other forces into the simulator. Treating multi-contact systems at acceleration level is often computationally expensive, and some constraint based formulations do not always have a solution.
- The velocity level – *impulses*. At the velocity level, we abstract ourselves from the second order level of the system. The so-called *impulses* that have units of momentum (and can be thought as a constant force integrated over a period of time) and are applied directly to update velocity are used. System using only impulses should use a hybrid time stepping, where the velocity update and the position update are separated. Velocity level systems have several advantages. First, they are often more simple to express, computationally less expensive than acceleration equivalent formulations, and suffer less of solution non-existence [Stewart 00]. Also, when dealing with two rigid bodies inter-penetrating with a non-zero velocity, the only way to avoid inter-penetration is to introduce a discontinuity on its velocity, via impulses.
- The position level – *position correctors*. Müller proposed [Müller 07, Müller 08a] the position-based dynamics method that uses only the position state of the virtual world to solves the collisions. The inter-penetration resolution is performed by defining constraints on the position of the system, and to perform a mass-weighted projection of the position onto the constraint surface, using a Gauss-Seidel like solver.

We review in this section the main methods that have been used in the literature at each level of resolution.

2.6.2.1 Acceleration level methods

Penalty-based methods

Penalty-based methods are the simplest and the faster one for contact or collision. The main idea is to add penalty forces at contact points depending on the penetration depths, or any other energy function.

More generally, penalty methods are acceleration-level approaches whose main idea is to create a force from a non-penetration function. A non-penetration function $b(\mathbf{q})$ is a function defined for each contact point, and is such that its value is zero when the interpenetration distance at this contact point is also zero.

From this non-penetration function, an energy function $e(b)$ can be defined as:

$$e(b) = \frac{k}{2} b(\mathbf{q}) \cdot b(\mathbf{q}) \tag{2.84}$$

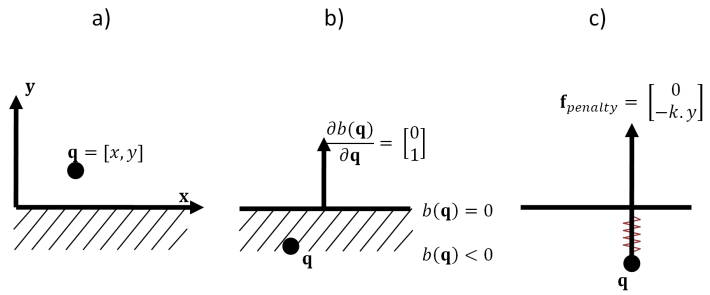


Figure 2.17 – (**a**): The state $\mathbf{q}$ is defined as a point in a 2-dimension plan. The point must not penetrate the ground surface represented by the x-axis. (**b**): The point has penetrated the ground, a non-penetration function $b(\mathbf{q}) = y$ is defined and will try to make the penetration vanish. (**c**): From the non-penetration function, a potential energy function is derived, and then a force along the surface normal. The function $b(\mathbf{q})$ generates spring-like forces to make the inter-penetration vanish.

where k is a stiffness constant.

From this potential energy, a force is defined as the opposite of its gradient as:

$$\mathbf{f}_{penalty} = -\frac{\partial e}{\partial \mathbf{q}} = -k \cdot b(\mathbf{q}) \frac{\partial b(\mathbf{q})}{\partial \mathbf{q}} \tag{2.85}$$

It is also possible to modify the force formulation (2.85) to include a damping term, which is dependent on the state velocity $\dot{\mathbf{q}}$, and uses a damping constant c, an analog to the stiffness constant k for damping forces. However, many non-penetration functions can be imagined. For example using a volume of penetration is often considered when modeling contact between deformable bodies, while penetration depth is usually used between rigid bodies. In that case, penalty forces act like springs that try to bring penetrating point back to the surface of the bodies.

Penalty-based methods are widely used for their simplicity and fast execution times, especially between deformable bodies [Spillmann 07]. However, they suffer from instability when the coefficient of stiffness k becomes big. Small time steps must be used to simulate stiff interactions, and it is impossible to completely avoid inter-penetration since reaction forces can occur only if the non-penetration function is non-zero. Also, the stiffness and the damping coefficient have no physical meaning which can make fine tuning of scenarios tedious.

Constraint-based methods

A different way to treat contact is to attempt to find a set of forces to apply at each contact point that prevent the bodies from penetrating each other. In static cases, this is equivalent to impose the acceleration toward the interior of the opposing body (the relative acceleration) at each contact point to be zero or positive. In multiple contact configurations, a force applied at any contact point will modify the relative acceleration of all the contact points. Therefore, a coupled system is formed, and solving for the zero or positive relative acceleration constraint at each contact is equivalent to solve for a Linear Complementary Problem (LCP) [Baraff 89, Witkin 90, Witkin 01a]. Solving for this LCP is a non-trivial problem, and adding friction into the acceleration-level system makes the problem potentially unsolvable [Baraff 91, Baraff 94]. As presented in the following, formulating the same kind of constraints at the velocity level solves this problem.

2.6.2.2 Velocity level methods

Impulses

Velocity-level methods work with *impulses*. An impulse $\mathbf{t}$ can be defined as:

$$\mathbf{t} = \int_{\Delta t} f \, dt \tag{2.86}$$

Impulses represent an immediate change on the momentum, and allow introducing discontinuity on velocities. Although using impulses breaks the differential equation defining the dynamics of the bodies, it is useful in the context of perfectly rigid bodies, where interpenetration could not be avoid without introducing immediate changes on the velocities. A comprehensive reference on the simulation of rigid bodies using impulses can be found in [Mirtich 96].

When two rigid bodies enter in collision with a certain relative velocity, it is possible to solve for an impulse that, once applied to the pair of bodies, prevent them from interpenetrating each other [Hahn 88]. Impulses have also been successfully applied for articulated bodies with rigid parts [Moore 88]. More recently, Guendelman *et al.* [Guendelman 03] and Erleben *et al.* [Erleben 07] have proven the use of impulse followed by stabilization steps to be viable for the simulation of stacking structure, without the resolution of LCP systems at each time step.

LCP formulations

For simultaneous contact configurations, one can solve for the impulses to be applied at each contact point so that the relative velocity is positive (at each contact point) once the impulses have been applied. Similar to the acceleration analog method, solving for the set of impulses that satisfy this constraint is equivalent to solve an LCP. As opposed to acceleration-level LCPs, velocity-level LCPs do not suffer from solution non-existence when incorporating friction. Stewart and Trinkle [Stewart 00] were the first to introduce a LCP system that uses a linearized version of the Coulomb cone friction, showing that their system always has a solution.

Other constraint-based systems

Other constraint-based formulations have been proposed in the literature to solve contacting systems cases. Milenkovic *et al.* [Milenkovic 01] express the dynamics of contacting rigid bodies into a convex Quadratic Problem (QP) to be solved under linear constraints.

Kaufman *et. al.* [Kaufman 05] proposed solving the inter-penetrations by projecting the global velocity state of the system onto a set a feasible velocities, which is dependent of the contact points positions and normal vectors. They also propose a way to solve the frictional problem with a second projection. To do so, they define a set of possible contributions of tangential forces on the global velocity. Then, they propose getting the frictional global impulse by projecting of the maximum opposing velocity onto the intersection between the feasible contribution of frictional impulses and the feasible velocities after contact. The final impulse after contact resolution is obtained as the sum of the normal and the frictional impulses. This projection method also works for multi-contact configuration, where the action of an impulse generates a change on the global state of the system. The authors also defined additional constraints that modify the set of feasible post velocities, to take into account momentum conservation and bouncing. They mention that the set still remains

convex for the optimization step (the projection). This double projection method leads to a fast simulator for rigid bodies as presented in the author's paper.

Kaufman *et.al.* extended their work in [Kaufman 08] by presenting a staggered projection method to solve with accuracy and stability a frictional dynamic problem in a multi-body system. The main idea of their algorithm is to define a pair of coupled projection (one for the inter-penetration problem, and one for the frictional problem) from which they extract a fixed point property that allows their algorithm to converge. The authors define a cone of the possible contributions of normal forces to the velocity, and a friction set from a linearization of the friction constraints. From this basis, the paper proposes an algorithm of "staggered" projections to solve for normal global impulses and frictional global impulse of a coupled minimization problems. At each iteration of their algorithm, two projections are performed and the solution obtained from an iteration is used at the next iteration to converge to a global solution to the simulation of multi body system with friction.

2.6.2.3 Position level methods

The projection level projection for collision solving purposes to directly project positions on constraint surfaces. Working directly on position has the advantage of giving more control on the simulation compared to second order actions produced by forces. Position level projection has been mostly used to constraint particle systems [Müller 07, Müller 08a]. Our particle system has a set $\mathcal{X}$ of particle denoted by their subscript i. Each particle $i \in \mathcal{X}$ has a position $\mathbf{x}_i$, a velocity $\mathbf{v}_i$ and a mass m_i.

At each simulation step, the velocity of all particles is updated using explicit Euler integration with a time step of Δt:

$$\mathbf{v}_i = \mathbf{v}_i + w_i \mathbf{f}_i \Delta t \tag{2.87}$$

where $w_i = 1/m_i$ and $\mathbf{f}_i$ is the external force acting on particle i for the current simulation step. Position predictors $\mathbf{p}_i$ are defined as the unconstrained position the particle i would have if integrated over a period Δt with its current velocity:

$$\mathbf{p}_i = \mathbf{x}_i + \mathbf{v}_i \Delta t \tag{2.88}$$

The constraints on the particle positions can now be defined. We denote by $\mathcal{D}$ the set of constraints. Each constraint $D_c \in \mathcal{D}$ is either bilateral or unilateral, and can be written as a function of the particles positions. A bilateral constraint D_c is valid if its scalar value is zero:

$$D_c(x_1, x_2, \ldots, x_{|\mathcal{X}|}) = 0 \tag{2.89}$$

Similarly, a unilateral constraint D_e is valid if its scalar value is above or equal to zero:

$$D_e(x_1, x_2, \ldots, x_{|\mathcal{X}|}) \geq 0 \tag{2.90}$$

The constraints are not necessarily functions of all the particles. For example, a distance constraint D_d that imposes particle 1 and particle 2 to stay at a distance d could be written $D_d(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1 - \mathbf{x}_2)^2 - d^2$.

After the velocity update and the position predictors (2.87) and (2.88), the particles are projected according to the constraint to have positions that satisfy bilateral and unilateral constraints. This modifies the position predictors $\mathbf{p}_i$. The velocity and the position of each particle are computed at the end of the simulation step as:

$$\mathbf{v}_i = \mathbf{p}_i - \mathbf{x}_i \quad (2.91)$$

$$\mathbf{x}_i = \mathbf{p}_i \quad (2.92)$$

After this update, the simulation step terminates, and the simulators goes back to (2.87) for another simulation step.

2.6.3 Handling physically-based interactions with haptic feedback

2.6.3.1 Introduction

Haptic interaction is a bidirectional interaction with a virtual world related to the sense of touch. This includes several components, which are:

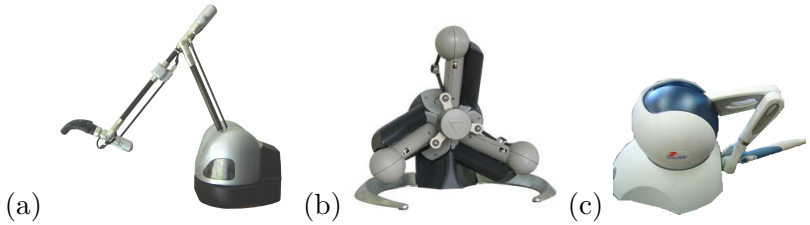
- **Force feedback**. When one wants to move an object, some forces may oppose that movement. These forces, which are usually felt at the user's articulations (wrist, elbow, shoulders, etc.) participate to force feedback. Force feedback rendering is performed through force feedback haptic devices, presented in section 2.6.3.2.
- **Tactile feedback**. While force feedback is measured thanks to sensors in human body articulation and muscles, tactile feedback is the restored feedback from a surface (such as rough, smoothness, ...). Such feedbacks are picked up by sensors under the skin, as the tip of fingers for example.
- **Thermal feedback**. Another aspect of haptic device can be the thermal exchanges between materials in the virtual environment. Simulating thermal interactions is far less common than force feedback.

In this paper, we will restrict ourselves on force feedback, which is the most common application of haptic rendering.

2.6.3.2 Haptic devices and control

Force feedback haptic devices are articulated robots containing electrical motors to provide a force feedback and sensors to measure its current state. The human user holds (usually with his hand) a specific point of the haptic device called the tip of the device. The haptic device is linked to a virtual body of the virtual world in such a way that the state of the linked body should be the same as the state of the tip of the device in the real world, and vice-versa.

Haptic devices are characterized by their number of Degrees Of Freedom on which they can apply forces (output DOFs) and the number of DOFs that is measured by sensors (input DOFs). Also, each device has a more or less big workspace, *i.e.* the tip of the device is free to move within a zone which is constrained by the length and the disposition of the device members (a good inventory of haptic devices with their characteristics can be found in [Andriot 07]). Depending on the strength of the device and the power of the motors, the device also has a working range of possible loadings. From those characteristics, a growing variety of haptic devices has been designed, going from small devices for precise manipulations to bigger device that allow having a workspace of a few meters and involving more consequents effort exchanged. Images of common haptic devices used during the PhD are shown in Figure 2.18.



(a) (b) (c)

Figure 2.18 – Example of haptic devices for force feedback haptic rendering. **a**: Virtuose 6-in/6-out DOFs arm (Haption, Soulge sur Ouette, France). **b**: Virtuose Desktop 6-in/6-out DOFs haptic device (Haption, Soulge sur Ouette, France). **c**: Phantom Omni 6-in/3-out DOFs haptic device (Sensable, Woburn, MA 01801, USA).

Haptic device control

On the point of view of the control of the device, two main categories of haptic device are available:

- **Impedance control**. Impedance haptic devices have sensors that measure the angle value of each articulation. The position of the tip of the device can then be retrieved from the morphology of the robot. Such devices output the position of the tip and take as input a force that must be applied at the tip of the device. Controlling such device is often not straight forward because it is more natural for a simulation to integrate forces and output positions. Using impedance control often involves performing *ad hoc* treatments on the coupled virtual object so that the simulation manages it correctly. However, most of haptic devices have a impedance control, mainly for technical reasons.
- **Admittance control**. Admittance haptic devices have the opposite behavior of impedance ones. They contain pressure sensors at the articulations from which forces applied by the user on the interface can be deduced. Such devices output the forces applied by the user on the tip of the device, and take as input a position (and an orientation for 6 DOFS haptic devices) that the tip must hold. Admittance devices are rarer because they are more expensive, technically harder to design and sensor of pressure add friction and inertia to the mechanism. However, admittance control interfaces naturally with numerical simulation: at each time step, the position of the linked object is ordered to the device, the user forces are retrieved and applied as external forces into the simulation for the next time step (see Figure 2.19).

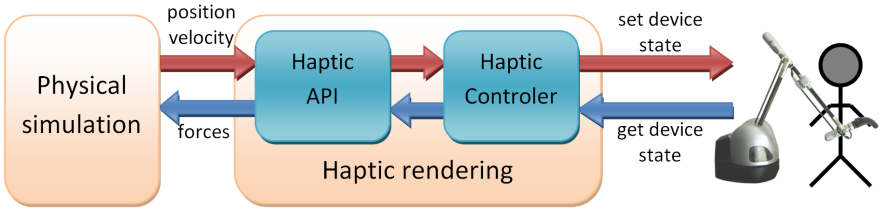


Figure 2.19 – Example of an admittance controlled device coupled with a physical simulation. The physical simulation provides successive states of the proxy which are sent to the haptic API. While the haptic API returns efforts values that are directly applied on the proxy into the virtual world.

The virtual coupling method

As highlighted before, even if admittance control of haptic device is more useful to couple with a numerical simulation, most of haptic devices are in an impedance mode, for cost and

complexity reasons. It is however possible to make them work as if they would be used in admittance mode. From the positions given by the device over time, one can estimate forces applied on it (*e.g.* by finite differences). Once we have the approximated forces, they can be applied onto the proxy and retrieve its position after a simulation step. From the difference of positions between the proxy and the haptic device, forces can be deduced and passed to the haptic device. This method is called a *virtual coupling*, or *god-object* method [Zilles 95], the god object being the object in the virtual world linked with the haptic device position, as illustrated in Figure 2.20.

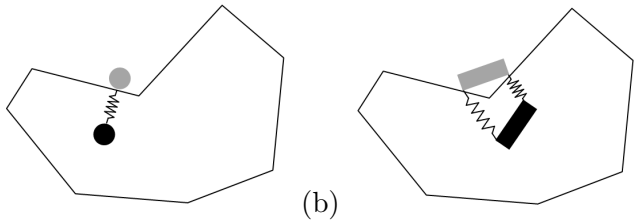


Figure 2.20 – Virtual coupling principle. The gray object is the god object (the object position in the virtual world), while the black body is the haptic device state. **(a)**: A damped spring model is used to minimize the error distance e in a mass point context. **(b)**: When the proxy is not a point, but a volumetric body, forces and torque must be computed using specific methods.

Virtual coupling methods allow dissociating the proxy state and the haptic device state, leading to serious advantages. First, admittance mode is emulated in this context, and haptic device controllers are sometimes able to emulate admittance mode, managing transparently the subservience of the impedance controlled device. This let the user the choice between impedance or emulated admittance control at the high level API programming. A user can be disoriented when seeing a proxy behind a wall, while he feels through the device that his hand or his finger is penetrating it. However, if the user applies a consequent effort on the tip on the device, his capacity to interpret the force feedback decreases.

2.6.3.3 Haptic rendering constraints

Haptic rendering is a quite recent modality of interaction, and still knows great development in research labs. Actually, haptic rendering is a particularly complex interaction that involves costly haptic devices. In this section, we discuss the constraints that haptic rendering imposes.

A bidirectional interaction

Haptic devices are material interfaces between our real world and a virtual world. In order to operate with a numerical simulation, a haptic device has its state sampled over time, which is also used to interact with the simulation. On the other side, the state of the proxy in the simulation is sent to the haptic controller which controls the motors of the haptic device to make the user feel a haptic feedback. We call the *synchronization* the discrete times at which the state of the device is sent to the simulation, and vice-versa (for simplicity, we consider that the synchronization is an instantaneous procedure).

Let us place ourselves in the case of admittance-emulated device. In that case, the synchronization procedure (from the point of view of the haptic rendering software) consists in fetching the haptic device returned forces (the position and velocity of its tip), and to order a goal state (a position and a velocity). However, between the discrete synchronizations, the two worlds evolve in a non-predictable way. Namely, the haptic device controller cannot know

in advance the next position and velocity which will be ordered at the next synchronization (even if it can try to guess it) due to the interactions of the proxy and the other objects of the virtual world. In the simulation point of view, the next forces that will be sent by the haptic devices are also unknown, since the human user which is free to move the tip of the device in a continuous and unpredictable way. This two-way interaction is a source of complexity of haptic rendering, and this constraint must be understood in order to design a stable haptic rendering algorithms. As discussed in the following, additional conditions must be satisfied in order to produce a realistic haptic feedback.

The frequency constraint

As highlighted in Figure 2.20, in order to manage the unavoidable errors difference between the current state of the haptic device and the state of the proxy, a virtual coupling is created. The simplest and most used virtual coupling is the damped spring model, which represents a proportional derivative control of the device, and of the proxy at the same time (the two entities are subject to different constraints, making them moving apart between synchronizations).

Let us consider for the moment that we are using a one degree of freedom output only haptic device. The current state of the device is represented by the scalars q_{device} and $\dot{q}_{device}$. The position of the proxy which is a goal position for the tip of the device is q_{goal}, and we also have $\dot{q}_{goal}$ giving the velocity of the proxy. In the case of a simple spring-damper virtual coupling, the force acting on the tip of the device is given by:

$$f(q_{device}, \dot{q}_{device}) = k(q_{goal} - q_{device}) - c(\dot{q}_{goal} - \dot{q}_{device}) \tag{2.93}$$

where k is the spring stiffness and c is the damping coefficient. In the following, we will denote by p the period of time in second between two synchronizations of the device ($1/p$ gives the haptic frequency). If the force $f(q_{device}, \dot{q}_{device})$ is applied constantly during the period p, then the position q_{device} will evolve following the rules of explicit Euler integration (see section 2.3.2.2). This integration is only conditionally stable, and high values of k impose small value for p to avoid the simulation to diverge. A way to improve the stability of the system is to use implicit Euler integration (see section 2.3.2.2) that imposes in this case the passivity of the system [Colgate 95] (*i.e.* the system only dissipates energy, and do not create any). This result can be extended to 6 DOFs input and output haptic devices (since the forces are still linear in the position and velocities). The forces Jacobian are simply the mass and inertia matrix of the proxy weighted by the stiffness k and damping c constants.

However, stability is only a necessary condition for a realistic haptic display. Another important parameter is the *frequency* $1/p$ of the haptic order. Actually, increasing the stiffness constant k of the coupling spring is not enough to increase the perceived stiffness of the force feedback. This can be simply explained noticing that using a fixed time period p, increasing k does not improve the stabilization time. Short stabilization times can only be obtained using high values for k and small values for p using spring damper model. Namely, in order to display stiff haptic effects such as impacts between two rigid bodies, the simulation must be performed at high frequencies to provide haptic orders at high frequencies. It is commonly admitted that for haptic interactions between deformable bodies, a frequency of 300Hz is suitable, while for high frequency phenomena such as impact between rigid bodies, frequencies as much as 1kHz are needed [Colgate 95].

Maintaining such high frequencies impose limitations on haptic rendering. In computer graphics community, real-time applications often target a frequency of 30 Hz to output on the screen. However, 30 Hz is clearly not enough for haptic rendering and the algorithms that

manage a real time visual display must be arranged in order to allow a haptic display. At the beginning of haptic rendering, simple virtual worlds containing a few degree of freedom and a few constraints were considered [Constantinescu 05]. The detection of collision between the proxy and the other objects is often the bottleneck of the haptic frequency, Gregory *et. al.* [Gregory 99] proposed an efficient collision detection algorithm is scenes involving only one movable object. Penalty and impulse-based method are often considered for collision response due to their fast execution time [Chang 97, Otaduy 06].

In [Barbič 07a], the authors present an adaptive time critical algorithm for detection collision and response between deformable bodies. Their algorithm is based on a voxel hierarchy and point sampling of the bodies. The hierarchy is browsed until the limit time has been reached for collision detection. In [Ruffaldi 08] is also presented a voxel hierarchy and collision response methods exploited for haptic rendering. However, single-threaded applications manage sequentially the simulation of the global virtual world, the visual display, the application events and the haptic rendering. Using this architecture, the haptic rendering frequency is prisoner of the time-consuming tasks such as global simulation or visual display. In order to free the dependency of the haptic rendering constraint, multi-rate architectures are used as presented in the next section.

2.6.3.4 Multirate simulation and intermediate models

In order to maintain high haptic frequencies, haptic rendering applications are often designed with parallel architecture containing at least two threads: one thread managing the physical simulation of the virtual world, the visual display and the application events, and a second thread which is dedicated to haptic rendering (see *e.g.* [Meseure 07] for a survey on multirate architectures). In the following, we call the first thread the *simulation thread*, and the second thread the *haptic thread*. Of course, it is possible to use more threads to increase to application overall timings, but we concentrate in this section on the dissociation between the haptic rendering and the other tasks of the application. The frequency requirements are different in the two threads. The simulation thread should run at least at visual display frequency (about 30Hz), while the haptic process runs at haptic frequencies, *e.g.* at 1kHz for displaying contacts between rigid bodies. Figure 2.21 shows an example of a parallel architecture using a simulation thread and a haptic thread running at different frequencies. The assumption of admittance-controlled devices is made in the following.

In order to communicate, the simulation and the haptic threads use a shared buffer which is updated by the simulation process at its rate. The haptic process uses the buffer data combined with the forces returned by the haptic device to deduce the next state of the proxy. The state of the proxy is also known by the simulation thread.

The shared buffer contains an intermediate and local representation of the virtual world around the proxy. The nature of the intermediate model can be various, and is such that the haptic thread is able to read it fast enough to provide the desired haptic update rate. In the following sections, we detail three kind of intermediate models.

Geometrical approach

The geometrical approach purpose is to extract locally around the proxy geometrical information of the virtual world. This geometrical information is exploited by the haptic thread that performs detection collision between the proxy and the local geometry (which is fast since the geometry extracted is not complex). The geometry extracted can be of different nature. Adachi *et. al.* [Adachi 95] proposed extracting a plane from the local geometry of a three DOFs proxy. Their result is extended in [Mark 96] where sets of plane can be used for

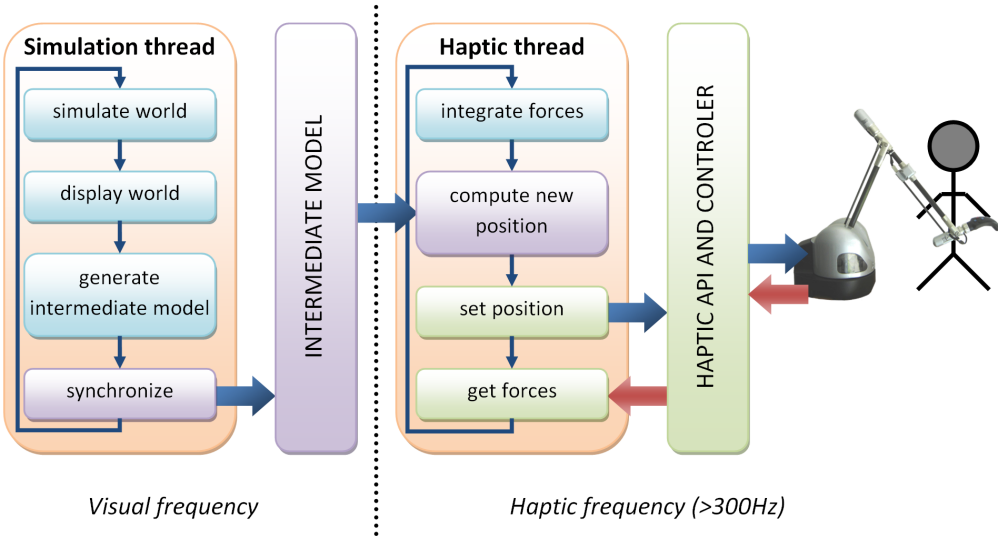


Figure 2.21 – A multirate architecture for haptic rendering. The simulation thread runs at a lower frequency, and updates at its rate a shared buffer containing an intermediate model. This intermediate model is used at higher frequencies by the haptic thread, enabling it to generate consistent haptic orders at haptic rate.

multi-contact and six DOFs haptic interactions. Parametrized surfaces have also been proposed [Balaniuk 99], avoiding the artifacts of smooth surfaces approximations by planes. A GPU-accelerated algorithm is proposed in [Mendoza 00] to extract all triangles that are near to the proxy and copied into the shared buffer as an intermediate representation. Otaduy *et. al.* [Otaduy 05] proposed a multi rate application sensation preserving algorithm for haptic rendering between complex meshes. The meshes are locally (at the collision points) simplified in a sensation preserving manner. Their result is extended in [Otaduy 06], where a collision response algorithm based on sampled point clustering is proposed.

The main limit of geometrical approach is the interaction between dynamic bodies. Actually, if only the proxy is able to move into an immobile world, good results are obtained. However, if bodies are moving in the virtual world, the geometry extracted is not valid during the haptic cycles, and when the buffer is updated, this creates artifacts that can be felt as vibrations in force feedback.

Force Jacobian approach

In the force Jacobian approach, the intermediate representation is a force derivative information [Picinbono 99]. Concretely, the buffer data is set of Jacobian matrix that represents the derivative of the force to apply with respect to a displacement of the tip of the device (this approach is designed for impedance control mode). For example, if we denote by $\mathbf{q}_{proxy}$ the position of the proxy in the virtual world, and by $\mathbf{q}_{device}$ the position of the device tip (we consider that the frame coordinate of the two worlds are consistent with each other), the displacement $\Delta\mathbf{q}$ of the tip of the device is given by $\Delta\mathbf{q} = \mathbf{q}_{device} - \mathbf{q}_{proxy}$. If $\Delta\mathbf{q} = 0$, the two states coincide, and no force is applied. It is possible to define a force function $\mathbf{f}(\mathbf{q})$ that links the position of the tip with a force to apply to correct its position. We have $\mathbf{f}(\mathbf{q}_{proxy}) = 0$. Using Taylor expansion, we can write:

$$\begin{aligned} \mathbf{f}(\mathbf{q}_{device}) &= \mathbf{f}(\mathbf{q}_{proxy} + \Delta\mathbf{q}) && (2.94)\\ &= \mathbf{f}(\mathbf{q}_{proxy}) + \nabla\mathbf{f}\Delta\mathbf{q} + \mathcal{O}(\Delta\mathbf{q}^2) && (2.95)\\ &\approx \nabla\mathbf{f}\Delta\mathbf{q} && (2.96) \end{aligned}$$

where $\nabla\mathbf{f}$ is the Jacobian of the force function with respect to the state vector $\mathbf{q}$. Using this Jacobian, it is extremely fast for the haptic process to deduce forces to apply at the tip of the device.

Multiresolution simulation

The principle of multiresolution approaches is to run two physical simulations in parallel. The simulation thread simulates the global virtual world at lower frequencies, while the haptic thread simulates a part of the world or the same world, but using lower resolution of models. Moreover, multiresolution methods are well suited for interaction with movable bodies.

Astley *et. al.* [Astley 97] proposed a multiresolution method for haptic interaction on deformable models. A lower resolution of the mesh of the deformable body is used around the interaction point, for the haptic thread. The full global resolution mesh is used in the simulation thread. The authors explain how the two parts of the mesh simulated at different rates can be connected. In [Cavusoglu 00], the authors proposed using a linear version of deformation method around the points of interest at different rate, while the simulation thread processes on a global mesh using non-linear methods for interaction.

2.6.4 Summary: interactive simulation and brittle fracture

In this section, we first reviewed the previous work in collision detection, a necessary step in the simulation of interactive virtual worlds. We saw that although various methods are available for collision detection, no method is directly applicable to brittle fracture. Indeed, collision detection methods between arbitrary geometries rely on data structures that are precomputed before the simulation. In the context of brittle fracture, a lot of new geometries are created at run-time, and during the same simulation frame. Therefore, precomputation or even fast computation of the proposed data structures cannot be done within a computation time compatible with interactive applications. The second topic reviewed in this section is the haptic feedback, a common interaction technique in virtual prototyping applications. Existing haptic frameworks [Ruspini 00, Luciano 05, Pocheville 04] for haptic rendering have their own collision detection system, and cannot be used directly for the haptic interaction with brittle fracture. Also, the high update haptic frequencies required by an haptic display highlights the need for more efficient models for brittle fracture, as well as adapted collision detection mechanisms.

2.7 Chapter conclusion

In this chapter, we first presented the background of the related work on physical simulation. We show that various methods are available that allow either fast or accurate deformations, or both fast and accurate, but modeling only small deformations. Then, we presented the methods of simulation of the fracture. Most of the proposed methods are extensions to the deformation models that localize the regions of higher stress to start and propagate fractures. We presented a summary table highlighting the pros and cons of each fracture simulation

method, with a focus on the computation time performances, a necessary feature for interactive applications. We showed that existing real-time approaches could be improved on two particular points. First, allowing unconstrained crack propagation, and second, simulating the dynamics effects of the impacts for fracture.

In a second part, we presented a brief review of the collision detection methods, and we noticed that the use of existing method for brittle fracture would not be straight forward, because of the new bodies with unpredictable geometries that are created at run-time, which preclude any precomputation. Finally, we presented the previous work on haptic rendering, highlighting the constraints brought by this bidirectional interaction, and its high frequency update required, making it a challenging modality of interaction.

New models for the real-time simulation of brittle fracture

3

Contents

Simulating brittle fracture in real-time on a physical basis has several challenges that have still not been addressed. First, existing real-time methods do not allow crack propagation in unconstrained directions, because of the computational and memory cost required by the current remeshing approaches. Also, the dynamic effects of impacts are not simulated in real-time, because stiff materials need small time steps to compute the propagation of the deformations from the impacted areas.

In this chapter, we first define a fracture state model that is designed to store the fracture information of the body. The design of this independent model allowed us to define an efficient propagation algorithm, with non-constrained crack direction, as well as an efficient meshing method for rendering. We then propose a simulation method based on modal analysis for the simulation of impact-based brittle fracture, allowing us to simulate the deformation of the brittle bodies in real-time. We also show how age-based cracking can be efficiently simulated using our model. We end this chapter with a demonstration of the use of a database of precomputed physical data for a possible handling of the newly created fragments.

3.1 Modeling the fracture state of a brittle body

In this section, we present a formal description of the fracture state of a body, suited for brittle fracture simulation. Later, we show how this model is leveraged for different real-time fracture simulations.

3.1.1 A new model based on volumetric meshes

As a deformable body has a rest state and a deformed state, a fracturable body will have an initial unfractured state, and a fracture state when at least one crack begins to propagate on it. The fracture state $\mathcal{F}$ of a body is a tuple containing information on where and how the body is fractured. Since fracture can occur inside the volume of the body, we rely on a volumetric mesh to store the fracture state information. The volumetric mesh is defined as:

$$\mathcal{M} = \{\mathcal{E}, \mathcal{N}, \mathcal{L}\} \tag{3.1}$$

where $\mathcal{E}$ is the set of elements that discretized the volume of the body, $\mathcal{N}$ is the set of nodes composing the mesh, and $\mathcal{L}$ the set of edges. In the following, we assume the elements to be tetrahedra, although our method could be generalized to other types of volumetric elements. From an element $e \in \mathcal{E}$, the four nodes composing this element can be accessed through the mapping $nodes : \mathcal{E} \longmapsto \mathcal{N}^4$, while the 6 edges composing this element can be accessed through the mapping $edges : \mathcal{E} \longmapsto \mathcal{L}^6$.

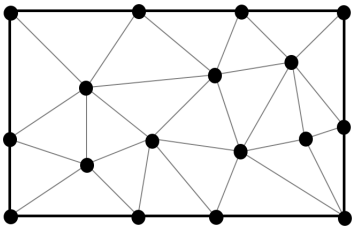


Figure 3.1 – Example of 2-D volumetric mesh. The set of elements $\mathcal{E}$ is represented by the gray contours, the set of nodes $\mathcal{N}$ is represented by the black dot, and the edges set $\mathcal{L}$ by the grey lines.

The key idea of our fracture state model is to use each of the structure composing the volumetric mesh to store a damaged state, the fracture surfaces, and a fragment list as presented in the following.

3.1.1.1 Modeling damage state with elements

By the term *damage*, we refer as elementary fracture, *i.e.* the fracture inside a single element of the volumetric mesh that holds the fracture state. In order to track the damage state of the body, we store in each element of its volumetric mesh a state that can take two distinct values: not damaged, or damaged. The damage state can be stored in a set $\mathcal{E}_{frac} \subset \mathcal{E}$ which is a subset of the element set $\mathcal{E}$ representing the set of damaged elements. Figure 3.2 illustrates the damage state set $\mathcal{E}_{frac}$ of a rectangular 2-D body.

Modeling the damage state of the body using the elements has several advantages: if a volumetric mesh is already used for the simulation of the deformation, it requires a few extra memory, and it is easy to update. Also, embedding the damage state into the volumetric mesh allows an efficient collision handling between cracks as presented in the following.

3.1.1.2 Modeling fracture surfaces with edges

Even if damage state is useful information for internal operations in the fracture process, it is not sufficient to produce a surface mesh for the rendering, or to identify the separated fragments. Therefore, the fracture surfaces are sampled on the edges of the volumetric mesh. To do so, each edge is associated with two structures: a cut state, and a cut position. The

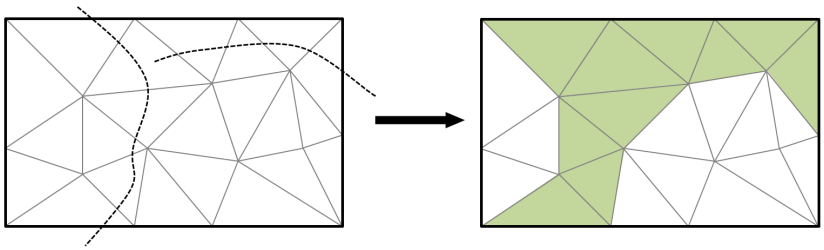


Figure 3.2 – Damage state stored in elements. A damage state that is associated to each element, taking two possible values: damage (white elements) or not damage (green elements). **Left**: A body is damaged by two cracks represented by the dotted lines. **Right**: The set of damaged elements corresponding the cracks is represented in green.

cut state stores whether an edge has been cut by a fracture surface, while the cut position is a scalar between 0 and 1 that stores the position of the cut on the edge. Figure 3.3 illustrates the sampling of the crack surface on the edges of the volumetric mesh.

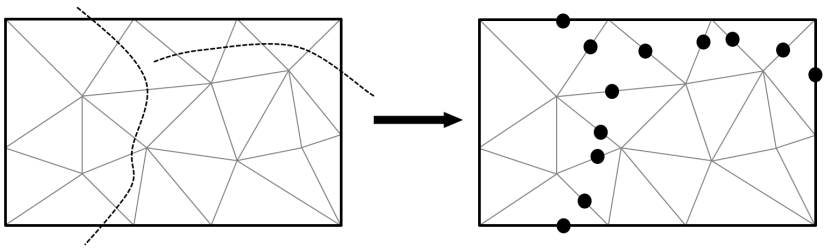


Figure 3.3 – Fracture surface position sampling on the edges of the volumetric mesh. To each edge of the mesh is associated a cut state (cut or not cut), and a scalar between 0 and 1 representing the cut position of the crack on this edge. **Left**: A body is damaged by two cracks represented by the dotted lines. **Right**: The cut positions of the cracks are sampled on the edges of the mesh, represented by the black dots.

The list of cut edges is stored in a set $\mathcal{L}_{frac}$, and the cut position is stored in the association $P_f : \mathcal{L}_{frac} \longmapsto [0:1]$.

3.1.1.3 Modeling fragments with nodes

If a crack separates a body (or a fragment) in two, two new fragments with independent rigid motions might be created. In order to define the fragments, each node is associated with a unique fragment identifier. The fragment identifier of each node is computed by flood filling once the set $\mathcal{L}_{frac}$ of cut edges is determined. At the end of the process, an association $N_c : \mathcal{N} \longmapsto \mathbb{N}$ linking each node to a fragment identifier is obtained.

After a new crack has propagated inside a fragment, the set $\mathcal{L}_{frac}$ of cut edges is updated. To update the fragment identifiers, a new fragment identifier f_{new} is chosen, and applied to an arbitrary node of the cut fragment. Then, this f_{new} is propagated recursively on the neighbors of this node with a condition. If the edge formed by two neighbors is cut, the propagation of the fragment identifier cannot occur. Figure 3.4 illustrates the computation of the fragment identifiers on a fractured body, while algorithm 1 present the flood filling algorithm used.

Algorithm 1 Flood filling algorithm for fragment identifiers

```
1: n ← unassigned node
2: Q =← {n}
3: f_new =← new fragment identifier
4: while Q ≠ ∅ do
5:    N_c ← N_c ∪ (n, f_new)
6:    for each n_e ∈ neighbors(n) do
7:       if n_e ∉ Q then
8:          e ← edge(n_e, n)
9:          if e ∉ L_frac then
10:            Q ← Q ∪ n_e
11:         end if
12:      end if
13:   end for
14:   Q = Q − n
15: end while
```

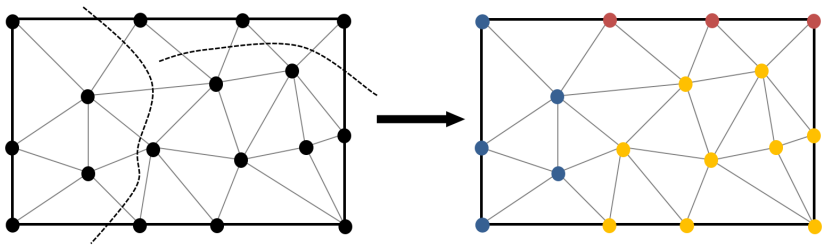


Figure 3.4 – Node fragment identifiers. Each node is associated to the fragment it belongs to. The connected set of nodes that have the same identifier form a fragment. **Left**: A body is damaged by two cracks represented by the dotted lines. **Right**: Resulting fragment identifiers on nodes (fragment identifiers are symbolized with colors)

3.1.2 Efficient generation of the surface meshes from the fracture state

The fracture state of the body efficiently stores the different information of fracture that separated the body into fragments. Although it cannot be used as is to be displayed using a traditional procedural rendering, a triangular surface mesh can be efficiently generated for visual rendering purposes. To generate the triangles corresponding to the fracture surfaces, we visit all the damaged elements in $\mathcal{E}_{frac}$. For each damaged element, we store in a list $\mathcal{L}$ the fragment identifier of each of its node. For tetrahedral elements, three different cases might arise. Either 1, 2 or 3 nodes of the element have a fragment identifier equal to the same fragment identifier r. For each configuration, we generate the appropriate surface triangles as shown in Figure 3.5. Repeating the process for each region $r \in \mathcal{L}$ allows generating all the faces of the fracture surface. Note that if the edges of the same tetrahedron are cut, the resulting meshing will produce an empty space at the center of the tetrahedron as shown in Figure 3.6. In this case, a new diamond-shaped fragment not connected to any node may be created to maintain the volume consistency.

To generate the triangles corresponding to the original surface, all the surface elements are visited. If the surface element is not damaged, each face of this element that is pointing to the outside generates one surface triangle. If the element is damaged, each of its face is treated independently. Two cases might arise for one damaged face and one fragment

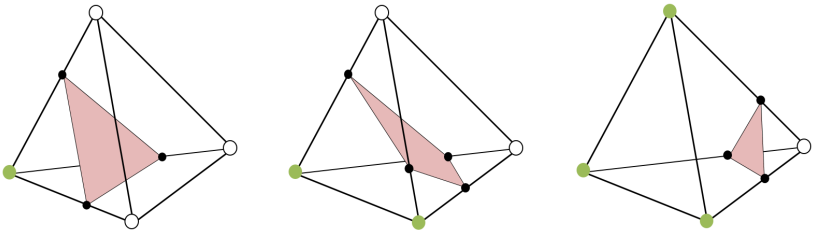


Figure 3.5 – Fracture surface meshing cases. The green nodes are nodes with fragment identifier r, while white nodes have a different fragment identifier. Black dots represent the cut position of the fracture surfaces on the edges, while red surfaces represent the geometry generated for rendering. **Left**: Only one node has a fragment identifier equal to r, a single triangle is generated. **Middle**: Two nodes have a fragment identifier equal to r, two triangles are generated. **Right**: Three nodes have a fragment identifier equal to r, a single triangle is generated.

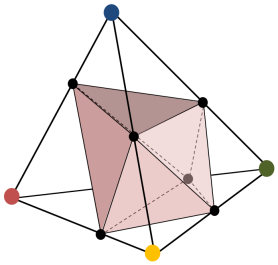


Figure 3.6 – Meshing of a fully damaged element.

identifier r: either 1 or 2 nodes of this face have the fragment identifier equal to r. For each configuration, we generate the appropriated triangles as shown in Figure 3.7

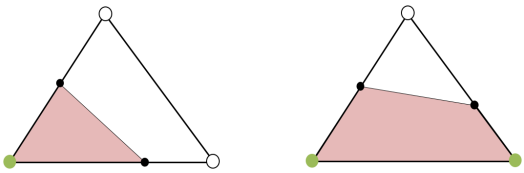


Figure 3.7 – Meshing of the damaged surface. The green nodes are nodes with fragment identifier r, while white nodes have a different fragment identifier. Black dots represent the cut position of the fracture surfaces on the edges, while red surfaces represent the geometry generated for rendering. **Left**: Only one node has a fragment identifier equal to r, a single triangle is generated. **Right**: Three nodes have a fragment identifier equal to r, two triangles are generated.

The global meshing process is summed up in Figure 3.8. The resolution of the final mesh to render is directly dependent on the resolution of the original volumetric mesh. However, sampling the surface on the edge allows obtaining a first order accurate representation of the fracture surfaces as presented in Figure 3.9.

3.2 Propagating cracks, or updating the fracture state model

Given the fracture state model defined in section 3.1, simulating the propagation of a crack corresponds to consistently update the set $\mathcal{E}_{frac}$ of damaged element, $\mathcal{L}_{frac}$ of cut edges, as well as the cut positions on the edges.

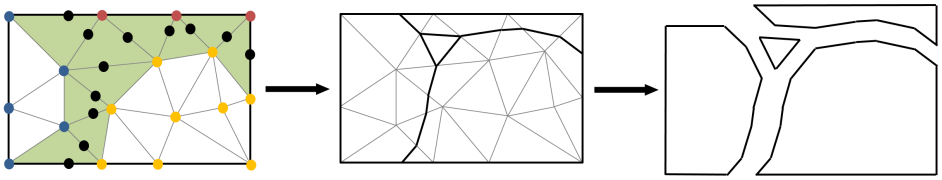


Figure 3.8 – Generation of the visual mesh from the fracture state. From the state of each element, the corresponding surfaces are generated.

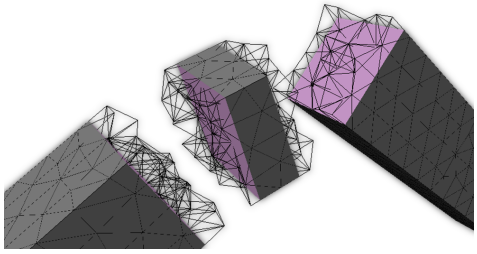


Figure 3.9 – Surface mesh and physical mesh. The colored sections represent a fracture that cut the body straight through the physical mesh. Although the visual mesh represents with fidelity the fracture surfaces computed, the underlying physical mesh (in wire frame) is not updated.

We first propose a flexible model of the fracture *surfaces*. Then, we present a crack propagation algorithm leveraging the fracture surface model as well as the fracture state model defined.

3.2.1 Fracture surface model

We propose defining the fracture surface using a general implicit surface $\mathcal{S}$:

$$\mathcal{S} = \{\mathbf{p} | s(\mathbf{p}) = 0\} \tag{3.2}$$

where $\mathbf{p} \in \mathbb{R}^3$. This definition allows a general and flexible way to model the fracture surfaces using mathematical expressions. In this manuscript, we used the following expression for $s(\mathbf{p})$ in (3.2):

$$s(\mathbf{p}) = bump\left(R^{-1}(\mathbf{p} - \mathbf{p}_0)\right) \tag{3.3}$$

where $R \in \mathbb{R}^{3\times3}$ is an orientation matrix that defines a global orientation of the fracture surface, $\mathbf{p}_0$ defines an offset, and $bump(a, b)$ is defined as follow with a 2-D simplex noise function [Perlin 02]:

$$p_y - noise(p_x, p_z) \tag{3.4}$$

An example of a surface texture obtained with a noisy fracture surface $\mathbf{p}_0$ is shown in Figure 3.10.

3.2.2 Propagating the crack through the mesh

Initial direction

The initial direction chosen by the crack to separate the material is the direction that maximizes the elastic energy dissipation [Cotterell 65]. The directions of principal stress satisfy

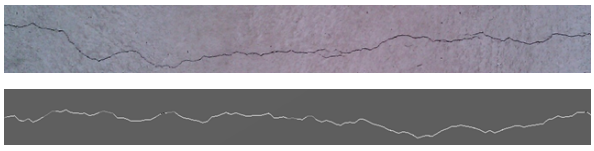


Figure 3.10 – Crack path and noise. Complex micro branching observed during crack propagation (top) can be reproduced with fidelity using a noise function (bottom). Similar results by full simulation would require expensive computation.

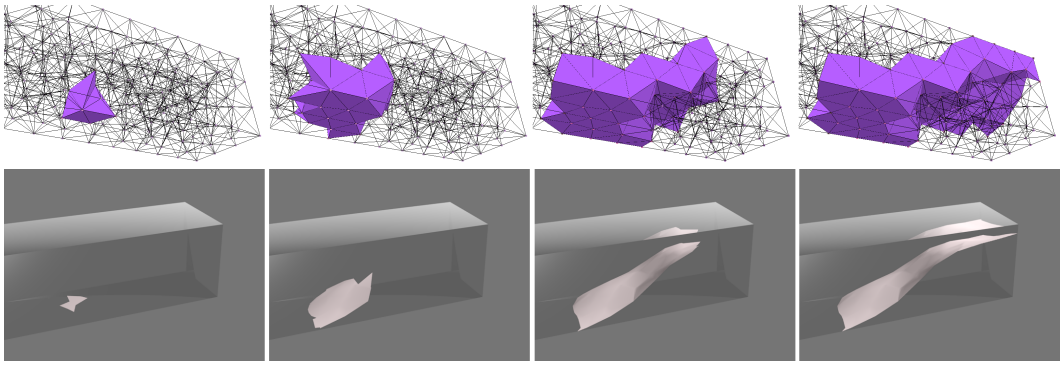


Figure 3.11 – Propagation of one crack in a coarse mesh. **Top line**: Representation of the current set $\mathcal{E}$ of elements cut by a fracture surface $\mathcal{S}$. Initially (top left), the set of element in composed of the element e_0 on which the fractured has been initiated. During the propagation, the neighbors of the set $\mathcal{E}$ that cross the surface $\mathcal{S}$ are added to $\mathcal{E}$ in a breadth-first search manner. **Bottom line**: representation of the actual fracture surface computed as the intersection between $\mathcal{S}$ and the volumes of $\mathcal{E}$. The surface is meshed with the edges of the mesh elements (see section 3.1.2).

this condition. Therefore, we choose to open the fracture in the direction $\mathbf{d}$ of the maximum principal stress of the element e_0 on which the fracture starts. The exact location of the beginning of the fracture is the center $\mathbf{c}_0$ of the element e_0. Therefore, the parameters of the fracture surface equation (3.3) are:

$$\mathbf{p}_0 = \mathbf{c}_0 \tag{3.5}$$

$$R = (\mathbf{d}_{\perp 1}\ \mathbf{d}\ \mathbf{d}_{\perp 2}) \tag{3.6}$$

where $\mathbf{d}_{\perp 1}$ and $\mathbf{d}_{\perp 2}$ are two arbitrary orthogonal vectors that are also orthogonal to $\mathbf{d}$.

Fracture Propagation

The key idea of our crack propagation algorithm is to use the implicit surface to visit the physical mesh elements. Starting from the initial element e_0, we visit the neighbors $neighbors(e_0)$ that are crossed by the implicit surface. An element is considered as crossed by the fracture surface if one of its node position $\mathbf{p}_i$ is on the negative side of the fracture surface ($s(\mathbf{p}_i) < 0$) and one of its node position $\mathbf{p}_j$ is on the positive side of the fracture surface, or on the fracture surface ($s(\mathbf{p}_j) >= 0$). We visit in a breadth-first fashion the neighbors of the crossed elements to form a set $\mathcal{E}$ of elements crossed by the fracture surface (see Figure 3.11).

Collisions between the cracks

A newly visited element cannot be added into the set $\mathcal{E}$ of cut elements if it is already marked as fractured. This simple rule allows handling the collisions between the arbitrarily complex implicit surfaces in an approximate manner using the physical mesh.

Algorithm summary

Algorithm 2 sums up our fracture propagation algorithm.

Algorithm 2 Propagation of one fracture surface

$\mathcal{S}$ = surface defined by equation (3.3)
$\mathcal{E}_{frac} = \{e_0\}$
$\mathcal{E}_{next} = \emptyset$
$\mathcal{E}_{tmp} = \{e_0\}$
$E_f = 0$
$E_s = 0$
while $\mathcal{E}_{tmp} \neq \emptyset$ **do**
  for all $e \in \mathcal{E}_{tmp}$ **do**
    $E_f = E_f + A_e.G_c$
    $E_s = E_s + A_e.\gamma.\eta_e$
    update $\mathcal{L}_{frac}$
    for all $n \in neighbors(e)$ **do**
      if n crossed by $surf(\mathcal{S})$ **and** $E_f < E_s$ **and** $n \notin \mathcal{E}_{frac}$ **then**
        $\mathcal{E}_{frac} = \mathcal{E}_{frac} \cup \{n\}$
        $\mathcal{E}_{next} = \mathcal{E}_{next} \cup \{n\}$
      end if
    end for
  end for
  $\mathcal{E}_{tmp} = \mathcal{E}_{next}$
  $\mathcal{E}_{next} = \emptyset$
end while

3.2.3 Energy stop condition

We adopt a macroscopic understanding of the brittle fracture propagation, and propose an energy-based criterion to stop the crack propagation. We express the fracture energy E_f (*i.e.* the amount of energy that has been used to propagate the crack) with a piecewise linear approximation:

$$E_f = \sum_{e \in \mathcal{E}} A_e.G_c \tag{3.7}$$

where A_e is the area of the fracture surface that crosses element e. G_c is the fracture toughness of the material, which expresses its resistance to fracture propagation. Similarly, we define E_s as the strain energy available from the non-fractured state of the body using:

$$E_s = \sum_{e \in \mathcal{E}} A_e.\gamma.\eta_e \tag{3.8}$$

where η_e is the strain energy density of element e, and γ is a constant factor that links E_f and E_s. The crack can propagate only if the available amount of energy permits it. When a new element e_i is visited, it can be added to the set $\mathcal{E}$ of fractured elements only if $E_f + A_{e_i}.G_c < E_s + A_{e_i}.\gamma.\eta_{e_i}$, *i.e.* if the new sum of fracture energy needed is under the sum of energy available. Figure 3.12 shows the influence of the fracture toughness G_c on the propagation of the fracture.

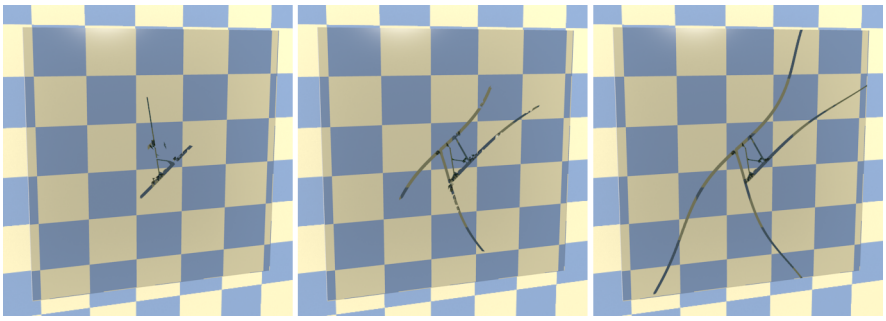


Figure 3.12 – Three glass slabs broken with different fracture toughness G_c (the fracture paths have been highlighted during the rendering). The left slab has a toughness G_c of 150 $J.m^{-2}$, the middle slab has a toughness of 90 $J.m^{-2}$, while the right slab has a toughness of 60 $J.m^{-2}$.

3.3 Modeling impact-based fractures

Impact-based fracture occurs when two stiff bodies enter in contact with a sufficient relative velocity to make at least one of them break. We use a third party rigid body simulator to simulate the contact between stiff bodies. Whenever a contact occurs between a pair of bodies, the deformations of the bodies is computed, and they eventually break. Compared to previous real-time approaches, we are able to simulate the dynamic effects of impacts such as inertia or damping thanks to the modal analysis approach.

3.3.1 Overview of the process

An overview of our brittle fracture simulation algorithm is presented in Figure 3.13.

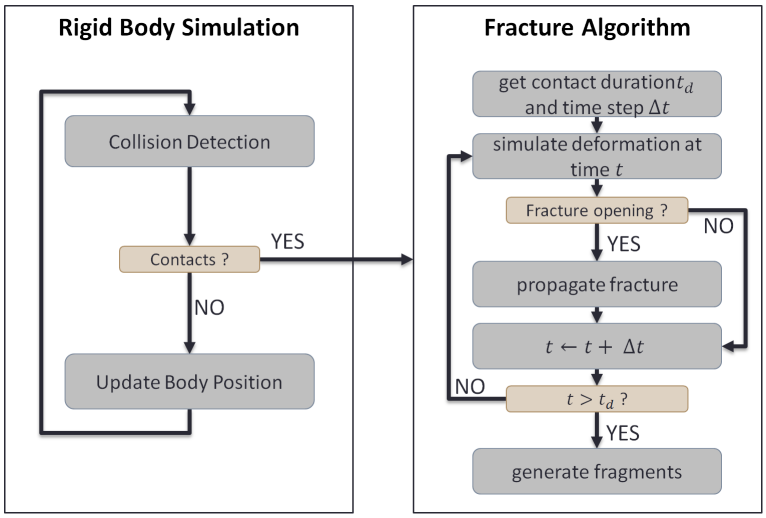


Figure 3.13 – Overview of our impact-base fracture algorithm for one body. At each contact event, the fracture algorithm is processed.

Once a contact is processed between a pair of body, their deformations due to this impact is computed. Since we need to simulate small deformation of stiff bodies, we propose using a modal analysis to simulate efficiently the deformations. Therefore, prior to the simulation, a modal analysis is performed on each body that can fracture (see section 2.4.3.2 for more details on modal analysis). Because the contact duration influences the fracture simulation (see

Figure 3.14), we present a way to estimate it in section 3.3.2. We also choose an appropriate time step to capture the main deformations of the body due to the impact, as explained in section 3.3.3. Section 3.3.4 details the contact force model that we defined to simulate contact forces between stiff bodies. Finally, section 3.3.5 summarizes the whole process.

3.3.2 Estimation of the contact duration

Our contact duration estimation is based on the Hertz's model formulation of sphere-sphere contacts [Johnson 87]:

$$t_d = c. \left(\frac{m^2}{E^2 r} . \frac{1}{v_{rel}} \right)^{1/5} \tag{3.9}$$

This formulation relates the contact duration t_d and the mass m, Young's modulus E, radius r of the spheres, as well as their velocity of approach v_{rel} at the time of impact (c being a constant scalar). The stiffer the body, the shorter the contact duration, while the heavier the body, the longer the contact duration. This property is useful in the context of brittle fracture, since the stiffness and mass of the bodies should modify the fracture simulation outcomes. However, the formulation in equation (3.9) is written for sphere-sphere impacts, and does not take into account the geometry of the body, nor the position of the impact on the body. Therefore, we propose an extension of this model based on modal analysis. We substitute the ratio $m^2/E^2 r$ of equation (3.9) by the mass/elasticity ratio of the mode of deformation which is the most excited by the impact. We choose the most excited mode because this the one that will involve the greatest displacements during the impact, as shown in Figure 3.14. Equation (3.9) becomes:

$$t_d = c. \left(\left(\frac{2\pi}{Im(\omega_{max})} \right)^2 . \frac{1}{v_{rel}} \right)^{1/5} \tag{3.10}$$

where $Im(\omega_{max})$ ($Im(x)$ is the imaginary part of the complex number x) is the natural frequency of the mode max, the most excited mode.

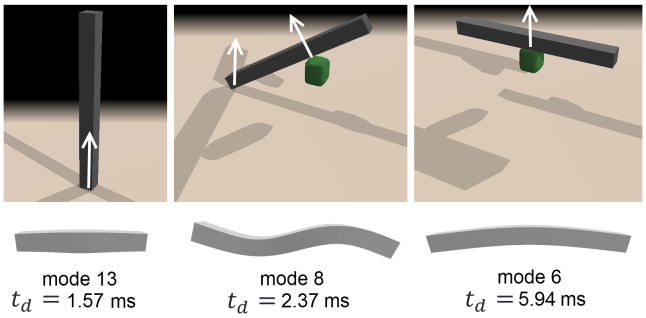


Figure 3.14 – Contact duration on a rod falling in different configurations. Depending on its initial position of the rod, the most excited mode is different, and so is the estimated contact duration.

The frequencies $Im(\omega_i)$ of each mode depend only on the stiffness (*i.e.* the material elastic properties), the mass, and the geometry of the body, keeping the same properties as the first formulation of contact duration. Note that if the mode i is critically damped, this approximation loses its sense. In practice, we select the most excited mode among those that are not critically damped. Also, since contacts involve at least two bodies, we compute contact duration estimation for each of the bodies involved in the contact, and retain the

largest duration. Finally, if the body is colliding at several locations at the same time, we choose the contact that has the smallest duration. The advantages of our contact duration on the simulation are highlighted in Figure 3.15.

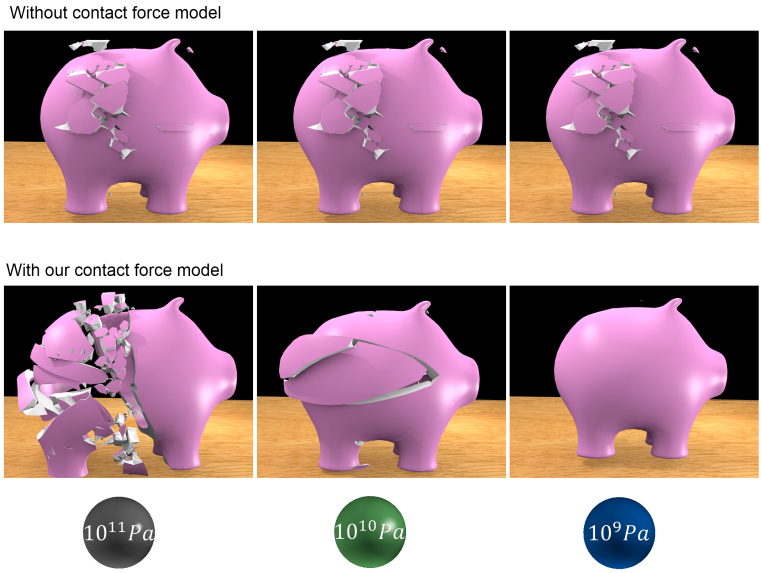


Figure 3.15 – Advantages of our contact duration estimation method. A piggy bank is hit by balls of different stiffness. **Top**: no contact force model is applied. We use a quasi-static approach to model a single deformation state of the piggy bank from the rigid body impulse. The fracture is not influenced by the ball stiffness. **Bottom**: the contact duration approximation of our contact force model leads to higher local stresses when the ball has a higher stiffness. With a softer ball, the contact will last longer and the energy of the ball will be damped without generating any fracture. In that case, the fracture is influenced by the ball stiffness as expected.

3.3.3 Simulation Time Step

To determine an appropriate time step for the simulation of the deformations during the contact, we use the Nyquist-Shannon sampling theorem. However, instead of taking the highest frequency of all retained modes, we take the highest frequency $Im(\omega_{high})$ of the non-critically-damped mode among the modes that have been excited by the contact impulse. Thus, the time step Δt used in our simulation is:

$$\Delta t = \frac{4\pi}{Im(\omega_{high})} \tag{3.11}$$

3.3.4 Contact Force Model

Following the Hertz model of impacts, we use a sinusoidal function $f(t)$ to model contact forces:

$$f(t) = a.\sin(u.t) \tag{3.12}$$

where a is the amplitude, u a frequency, and $f(t)$ is the magnitude of the contact force. The frequency u is computed with the contact duration (3.10) as $u = \pi/t_d$.

Force amplitude

The rigid body simulator computes contact impulses that prevent bodies from inter-penetrating. A contact impulse ϕ can be interpreted as the integral of a lasting contact force over time [Hahn 88]. In our case, we interpret it as the integral of the contact force of equation (3.12) over the duration of contact t_d:

$$\phi = \int_0^{t_d} f(t).dt = \int_0^{t_d} a.\sin(u.t) = a.\frac{1-\cos(u.t_d)}{u} \tag{3.13}$$

The amplitude a of the contact force is deduced from this integral as $a = (\phi.u)/(1 - \cos(u.t_d))$.

Figure 3.16 shows an example of the deformations of a slab due to a contact force.

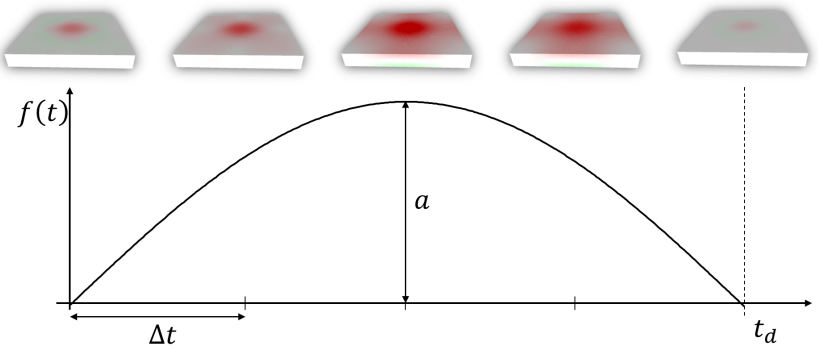


Figure 3.16 – Our contact force model applied on a stiff slab (color intensity is proportional to the maximum principal tensile stress). In order to model contact forces, a sinusoidal force $f(t)$ is applied at discrete times using a time step Δt for a duration of contact t_d. The parameters Δt, t_d and the amplitude a are determined thanks to modal analysis and rigid body simulation.

3.3.5 Fracture criterion

At each single or multiple contact event from the rigid body simulator, we analytically compute the damped deformation wave that occurs due to the contact force, using modal analysis. From the displacement vector computed for a body, we compute the Green-Lagrange strain ϵ_e tensor for each element e, and deduce the stress σ_e of the elements using a Hookean model of elasticity: $\sigma_e = C\epsilon_e$ ($C \in \mathbb{R}^{6\times6}$ being the linear constitutive law). If the maximum principal stress of an element exceeds a threshold R_c, a fracture is initiated (Rankine hypothesis [Gross 06]). To model the inhomogeneity of the material, we randomly sample "weakness" points into the bodies from a density of imperfection. Fractures are initiated at the location of the weak point which is the closest to the element that has the highest principal stress. The whole process of fracture initiation is summed up in Algorithm 3.

3.3.6 Results

3.3.6.1 Computation time performances

Configuration Our fracture simulation method has been implemented in C++ on a laptop with 4 GB of RAM, and an Intel®Core™2 Extreme (2.3 GHz, we only use one thread). Our GPU implementation has been tested on an NVidia Quadro FX 3700M. We tested

Algorithm 3 Fracture initiation test for one body $\mathcal{B}$

1: $\mathcal{M} \leftarrow modal\_analysis(\mathcal{B})$ {§ 2.4.3.2 }
2: $\mathbf{b} \leftarrow$ contact position
3: $\phi \leftarrow$ contact impulse magnitude
4: $t \leftarrow 0$
5: $t_d \leftarrow contact\_duration(\mathbf{b}, \mathcal{M}, \mathcal{B})$ {§ 3.3.2 }
6: $\Delta t \leftarrow time\_step(\mathbf{b}, \mathcal{M}, \mathcal{B})$ {§ 3.3.3 }
7: **for** $t < t_d$ **do**
8: $t \leftarrow t + \Delta t$
9: $f(t) \leftarrow$ contact force at t {§ 3.3.4 }
10: $modal\_deformation(f(t), \mathcal{M}, \mathcal{B})$ {§ 3.3.4 }
11: **if** $\exists$ element e with maximum principal tensile stress $> R_c$ **then**
12: propagate fracture at e {§ 3.2 }
13: **end if**
14: **end for**

our method with Havok Physics [Havok] and NVidia PhysX [PhysX] for the rigid body simulation [Glondu 10].

GPU Implementation

Thanks to modal analysis, the fracture initiation step can be fully parallelized [Che 06]. We implemented a parallel version of this step, using GPU hardware to accelerate the computations. All modes of deformations are transferred once onto the GPU global memory, avoiding expensive GPU/CPU memory transfers. In a first sub-step, a list of deformation states (the successive deformations during time of contact) is generated using one thread per degree of freedom of the initial mesh. The deformation states are stored on the GPU global memory, and never transferred to CPU memory. Then, these states are used in a second sub-step to compute a list of the maximum principal stresses of each element using one thread per element. The lists of principal stresses are finally transferred back to CPU memory, and exploited to apply fracture criterion and initiate fractures. Our parallel implementation of the fracture initiation step gives up to 14 times speed up on our configuration.

Computation Time Results

Table 3.1 summarizes test cases parameters and results. The fracture test initiation scales with the mesh complexity thanks to the introduction of the density of weak points. Two meshes of different resolution but with the same volume and the same density of weak points will have similar computation time (see last line of Table 3.1). The bottleneck of our method is the fracture initiation phase (about 75% of the computation time), where the deformations are computed to initiate fracture. The complexity of our fracture propagation algorithm is linear in the number of elements of the physical mesh that are crossed by the fracture surface. The complexity of the fragment generation algorithm is linear in the number of nodes of the mesh to be separated. The linear complexity of our global system enables us to fracture bodies composed of more than 175K elements in about 50 milliseconds.

3.3.6.2 Tests and Scenarios

We demonstrated the main features of our approach through various scenarios.

Body	Figure	Complexity		Physical parameters				
		nodes	tets	α	β	$E(Pa)$	$G_c(J/m^2)$	R_c
glass	3.18	7912	24410	1.10^{-9}	5	200.10^9	150	4.10^9
piggy bank	3.18	5992	20777	1.10^{-8}	3	200.10^9	500	5.10^8
piggy bank	3.15	5992	20777	1.10^{-8}	3	200.10^9	500	5.10^8
slab low	-	644	1740	1.10^{-10}	5	100.10^9	100	10^9
slab moderate	-	2761	8190	1.10^{-10}	5	100.10^9	100	10^9
slab high	3.12	12017	39901	1.10^{-10}	5	100.10^9	$60-150$	10^9
slab v. high	-	50476	175095	1.10^{-10}	10	100.10^9	100	10^9
bunny	3.19	5089	18767	2.10^{-9}	5	1.10^9	100	5.10^8

Body	Figure	Timings (ms)			
		initiation	propagation	meshing	total
glass	3.18	34 / 3.5	5.1	1.3	9.9
piggy bank	3.18	66 / 8.3	6.6	6	20.9
piggy bank	3.15	150 / 15	12.3	6.4	33.7
slab low	-	12 / 1.7	0.8	0.6	3.1
slab moderate	-	17 / 3.9	2.4	0.37	6.67
slab high	3.12	25 / 7	6	17	30
slab v. high	-	26 / 7	25	21	53
bunny	3.19	42 / 3.2	6	7	16.2

Table 3.1 – Parameters and timings for each scenario. The values after the slash in the initiation timings column are the timings obtained with GPU accelerations. Total times are computed considering GPU timings. The values α and β are the Rayleigh damping coefficients, and the number of retained modes is one hundred.

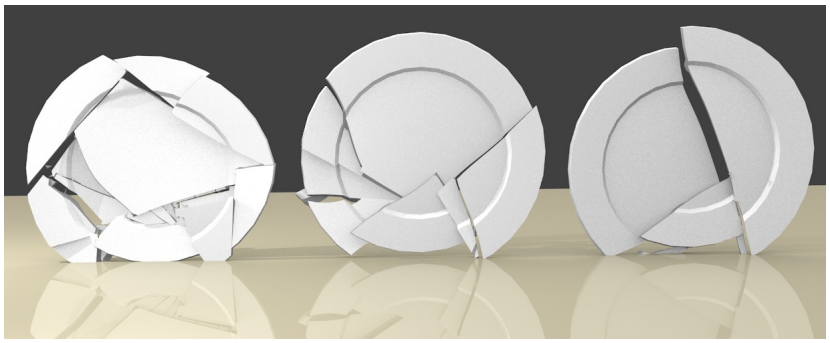


Figure 3.17 – Effect of damping on the fracture simulation. Our contact force model with modal analysis enables to model the inertia and damping of the plate. The left plate has the lowest damping value $\alpha = 0$, letting high frequency modes to propagate and generate many small fragments. The right plate has the highest damping value $\alpha = 10^{-6}$, generating less fragments.

Compared to previous real-time approaches, we simulate damping and contact properties through our contact force model and modal analysis. These phenomena have consequences on brittle fracture, as illustrated in Figure 3.15 (the stiffness of the projectile changes the contact properties) and in Figure 3.17 (different damping values lead to different fracture paths).

As shown in Figure 3.12, the fractures can propagate in any direction, and the fracture patterns are not guided by the elements boundaries of any mesh. Moreover, our method can model physically-based partial internal fractures as shown in Figures 3.18 and 3.12. Finally,

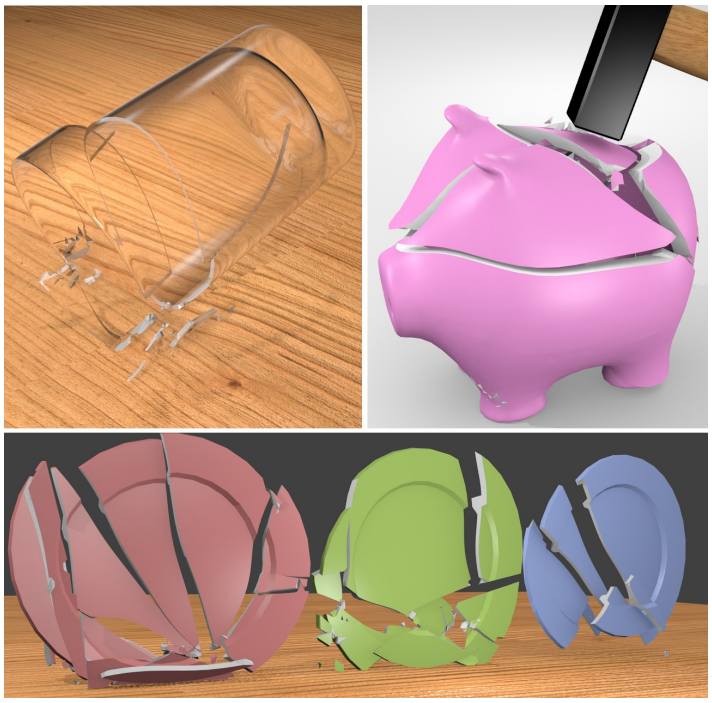


Figure 3.18 – Examples of brittle fracture simulations with our approach: **Left** a water glass breaks (with partial fractures) at the second bounce, **Right** a piggy bank smashed by a hammer, **Bottom** three plates dropped with different material properties.

our algorithms are valid for convex, concave, thin or thick bodies. Figure 3.19 shows the propagation of the fracture into a filled bunny broken at interactive rate.

Figure 3.19 – Propagation of the fracture into a thick filled body. Front, back and side view of a filled chocolate Stanford bunny thrown on a wall.

3.3.7 Discussion and conclusion

Limitations and perspectives for modal analysis

An hypothesis of our work is that we keep the initial modal analysis to simulate the deformation of the body, even if one or more fractures have started to open. We believe that this hypothesis is reasonable for several reasons. First, if one is interested only on a fracture test (*i.e.* checking whether the material will break), this hypothesis has no importance. Also, the deformations propagate during cracking, but it is not clear how. The solution we propose is a trade-off between the realism and the computation cost of the simulation.

Another issue of our model is that modal analysis is not well suited to simulate local deformations on big structures that have many branching parts. A solution would be to use domain decomposition [Barbič 11] to separate the initial body into smaller domains, and keep the advantages of speed and parallelization. Moreover, modal analysis can be extended in several ways (using *e.g.* modal derivatives [Barbič 07a] or [Huang 10]) to handle larger deformations. We plan to include this work in our framework to extend our method to ductile fracture, or fracture of softer bodies.

Recursive fracture for the fragments

An interesting property of our method is that the fracture propagation and the fragment generation algorithms can recursively be applied on fragments. However, the fracture initiation step cannot be performed since the modal deformation basis cannot be pre-computed for the fragments. One possibility to extend this method is to use modal proxies (or modal impostures) as suggested in [Pentland 89] and exploited in [Zheng 10] for sound generation. The main principle is to find a shape (proxy) in a database (for which modal analysis has already been performed) that fits well with a fragment, scale it and its modes, and link the nodes of the fragment to the elements of the proxy. Results combining this approach and our fracturing method can be found in section 3.5. Another possibility is to use the same deformation modes to generate the deformation on the fragments. When an impulse (computed with the mass of the fragment) is detected on a fragment, the modal DOFs are set as if the whole body was impacted, but only the elements composing the fragment are checked for fracture. This solution has no physical justification, but is acceptable if the visual effect only is desired. Finally, another solution would be to do a classical FEM simulation on the fragments. We already have the stiffness matrices for each element, and a quasi-static equilibrium can be computed as in [Müller 01] to initiate the fractures, while the propagation and fragment generation step can be applied recursively.

Treating resting contacts

In our approach, we are also able to treat resting contacts cases with a quasi-static approach solved with modal analysis. Indeed, after the modal analysis treatment, we compute a basis that diagonalizes the stiffness matrix. Therefore, computing the reduced deformation that solves the quasi-static equilibrium is computationally cheap. This provides an efficient and plausible test for resting contact cases.

A framework for brittle fracture simulation

We developed a complementary framework for fracture brittle simulation, in order to evaluate and compare brittle fracture simulation methods. It embeds features for quick set-up of various scenarios. It also provides tools for real-time visualizations of the influences of the different material parameters on the deformations of the bodies and their potential fractures.

3.4 Modeling cracking due to aging

Impacts are not the only cause of the fracture of brittle materials. In natural environments and in cities, cracked stones or buildings are observed. These cracking phenomena are due to the modification of the material properties over time, and to external loading forces. In this section, we show how the age-based fracture phenomenon can be reproduced using our

fracture state model, and an appropriated aging algorithm. We also show how to optimize the set of parameters of the simulation w.r.t. a desired pattern definition.

3.4.1 Aging process overview

To simulate crack opening due to age, we use a stress map that evolves over time as proposed in [Iben 06]. The map is initialized at the beginning of the simulation: the internal stress σ_e of each element e is initialized with σ_e^0. At each time step, the stress is updated with a specified stress increase rate $d\sigma_e$ that represents the variation of the stress over time. If the maximum principal tensile stress of an element e exceeds its resistance threshold R_{c_e}, a fracture is opened at the center of the element e, in a direction orthogonal to the direction of this maximum principal stress. The stress increase rate $d\sigma_e$ will be optimized since we do not have any a priori knowledge of the value of this parameter in the exemplar images. The resistance threshold R_{c_e} is different in each element, due to the presence of material flaws. The randomness of the parameter is taken into account in the optimization process by the use of an *a priori* normal distribution with a mean value R_{c_m} and a variance $R_{c_{var}}$.

The crack initiation algorithm is summarized below:

Algorithm 4 Stress evolution and crack opening.

```
1:  t = 0
2:  while t < age do
3:    for all e ∈ E do
4:      σ_e ← σ_e + dσ_e · Δt
5:      s ← max_principal_stress(σ_e)
6:      d ← dir_principal_stress(σ_e)
7:      if s > R_{c_e} then
8:        propagate_crack_at(e, d)
9:      end if
10:   end for
11:   t = t +Δt
12: end while
```

External forces

To take into account external loading forces $\mathbf{f}$, we compute the static equilibrium of the simulated body w.r.t. $\mathbf{f}$:

$$K\mathbf{u} = \mathbf{f} \ , \tag{3.14}$$

where the unknown $\mathbf{u}$ is the displacement of the degrees of freedom of the body due to the force vector $\mathbf{f}$. From this equilibrium, we compute the initial stress value σ_e^0 for each element using a Hookean law of elasticity. To improve the efficiency of the computation of the static equilibrium, we perform a modal analysis on the body in a precomputation step. This gives us a new coordinate system Λ that diagonalizes the stiffness matrix K and in which equation (3.14) can be solved directly. The effect of a loading force on the simulation is illustrated in Figure 3.20.

Stress relaxation

When a crack propagates, it alleviates the stress around the crack path. Indeed, the opening of the crack involves a displacement at the surface that relaxes the surrounding strain and

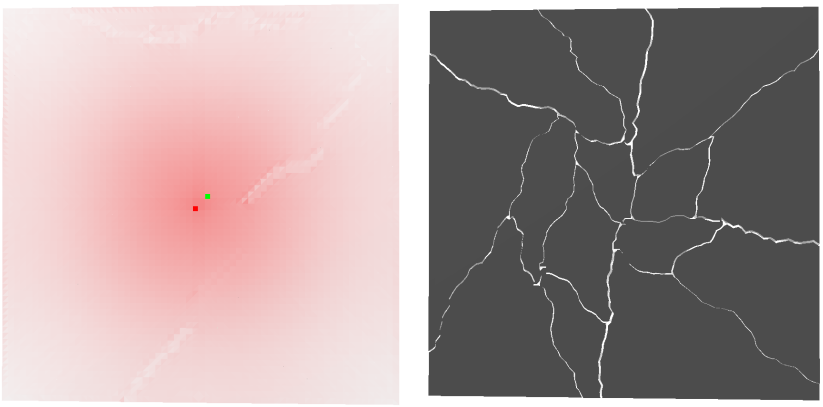


Figure 3.20 – Effect of a loading force on the simulation. A constant loading force is applied from the middle of the tile, leading to higher stress around the middle. Left: stress field computed from the loading force with quasi-static analysis (the intensity of the red value is proportional to the maximum principal stress). Right: A higher concentration of fragments is observed around the middle of the tile.

stress. We extend this method by adding a stress relaxation step in the simulation. Compared to previous work, rather than computing this relaxation by a direct simulation or using *e.g.*, virtual displacements [Iben 06], we propose a new method that uses an approximate but more efficient simulation of the stress relaxation.

We use a relaxation tensor $\sigma_{relax}(\mathbf{d})$ in a direction $\mathbf{d}$, computed as $\sigma_{relax}(\mathbf{d}) = \mathbf{d}\mathbf{d}^T/|\mathbf{d}|$. The relaxation has $|\mathbf{d}|$ as its only non-zero eigenvalue. Subtracting this tensor from a stress tensor σ_e alleviates it in the direction of $\mathbf{d}$.

The direction $\mathbf{d}$ of the relaxation for an element e is chosen to be the direction orthogonal to both the closest surface normal and the crack propagation direction. Points closer to the crack path will be relaxed more. For efficiency of the simulation, we choose to model this phenomenon using the following polynomial kernel $\phi(d)$, inspired by [Müller 04a]:

$$\phi(d) = max(0, \frac{1}{r_{relax}^6}(r_{relax}^2 - d^2)^3). \tag{3.15}$$

The kernel is equal to one if the distance d is zero, and zero if d is greater than r_{relax}. Therefore, the value $\sigma_{e_{relaxed}}$ of the stress of element e after relaxation is computed as:

$$\sigma_{e_{relaxed}} = \sigma_e - v_{relax}.\phi(d).\sigma_{relax}(\mathbf{d}) \ , \tag{3.16}$$

where d is the distance of the center of element e to the fracture surface, and v_{relax} is a new parameter denoting the stress relaxation rate around the crack. The introduction of the parameters v_{relax} and r_{relax} allow finer control of the shapes of the generated fragments, and thus will be determined in the optimization process.

Because we do not want to transform the tensile stress into compressive stress, we check the stress value $s = \mathbf{d}^T\sigma_{e_{relaxed}}\mathbf{d}$ in the direction $\mathbf{d}$. If $s < 0$, the relaxation produced compressive stress in the direction $\mathbf{d}$. In this case, we clamp the value to 0 by adding the value $-s * \sigma_{e_{relaxed}}$ to the stress tensor of the element.

In some patterns, privileged directions are not observed. In that case, we apply a relaxation of the stress in all the directions, and Eq. (3.16) becomes:

$$\sigma_{e_{relaxed}} = \sigma_e - v_{relax}.\phi(d).I_d \ , \tag{3.17}$$

where I_d is the identity stress tensor that alleviates the stress in all the directions. The effect of the relaxation method on the aspect of the fracture patterns is illustrated in Fig. 3.21.

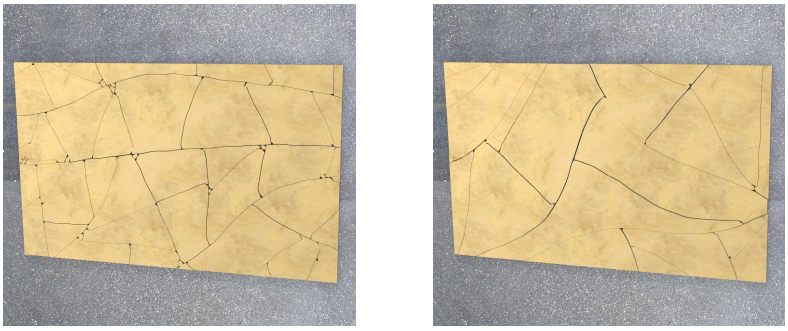


Figure 3.21 – Illustration of the two relaxation methods used. Left: the fracture has a preferred orientation. The stress relaxation of equation (3.16) is used. Right: the pattern does not have a preferred orientation. The stress relaxation of equation (3.17) is used.

3.4.2 Results and conclusion on the age-based fracture simulation

Figure 3.23 shows an example of the state of a road experiencing a loading force at four different ages. This example takes approximately 250ms to compute on a common laptop. It illustrates well our stress relaxation method that generates privileged cracking direction. Figure 3.22 show a bathroom scene were objects have been fractured using our aging algorithm.

Figure 3.22 – Bathroom scene with several broken objects.

Evaluation of the subjective perception of the obtained fracture patterns are presented in Chapter 5

3.5 Managing the deformation of the fragments: a database approach

In the context of interactive applications, the physical data cannot always be precomputed at run-time, especially when new objects are dynamically added to the scene as in fracture scenarios. A solution to this problem is to store precomputed data into a database, and use them on objects for which no physical information is provided. In [Zheng 10], the authors use

Figure 3.23 – From left to right and top to bottom: Road model showing fracture propagation, simulating the course of time.

a database containing "sound proxies" in order to accelerate the sound synthesis of shattering bodies. They proposed a method to find ellipsoids that match with their fragments. In order to extend this principle to have databases containing any variety of shape, it is possible to leverage the widely studied field of similarity search in databases of 3-D objects. We propose to use a database approach to provide a way to compute the deformation of fragments generated our fracturing algorithm.

Similarity search in database of 3-D objects retrieval purpose is to find objects of a database that are similar to an input query object (see [Bustos 05] for a survey on the topic). It is often performed by extracting features from the geometry of the query object and comparing these features to the entry of the database. The features extracted from the mesh of the body are called *feature vector* or *descriptor* of the object. The descriptors extracted from the object geometry can be built from their surface mesh, volumetric mesh, or from 2-D images of the object [Heczko 02]. In the literature, various kind of features are extracted from the objects, such as boxes [Paquet 00], spherical harmonics [Vranic 01b], Reeb graphs [Hilaga 01], moments of inertia [Bustos 05] or statistical information extracted from voxel representations [Vranic 01a] or surface distribution [Osada 02].

The descriptors can be classified on their efficiency (how long it takes to build it, and how long it takes to find the best match based on the similarity) and on their accuracy (how the defined similarity is relevant w.r.t. the targeted application). There is currently no method capable of capturing the local geometry features of the objects and performing a similarity search at interactive rates. In our context, efficiency is a crucial criterion, as we target interactive applications. We adapted voxel-based representations and moment-based methods to our needs to provide descriptors that propose a good trade-off between efficiency and accuracy.

3.5.1 Precomputed Shape Database

This section presents how the database is built offline in a first part, and how the online similarity search is performed is this database in a second part.

3.5.1.1 Creating the Database

We propose a process of creation of the database in five steps: (1) the generation of the surface mesh, (2) the computation of their volumetric meshes, (3) a normalization step, (4) the computation of the descriptors, and (5) the addition of physical data. These steps are detailed along this section.

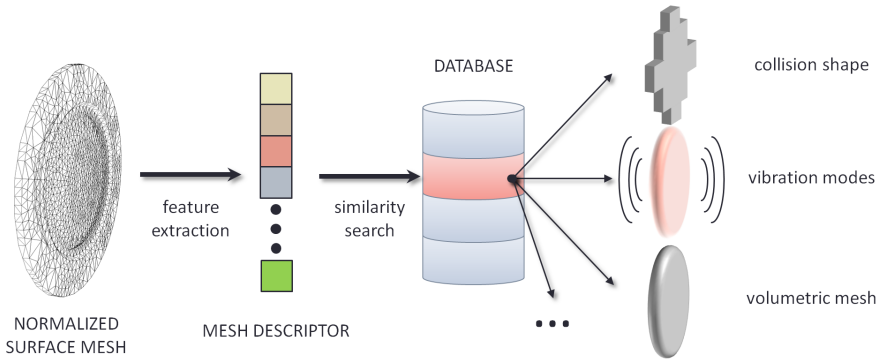


Figure 3.24 – Overview of the similarity search process. Features are extracted from the normalized input surface mesh to obtain the mesh descriptor. This descriptor is compared to the descriptor of each database entry to find the best match. Each database entry contains physical data such as a collision shape for collision detection module, the lowest vibrations modes of the body or a volumetric mesh.

Choice and Generation of Surface Meshes

The choice of the type of shapes that are stored in the database mainly depends on the targeted application. We arbitrarily selected the six main generic shapes that are shown in Figure 3.25.

From each main shape, we generate a set of surface meshes by scaling the initial shape along two or three axis depending on their symmetrical properties. In our example, about $4,500$ surface meshes have been created from the six main shapes. Each surface mesh follows the four following treatments to add an entry in the database.

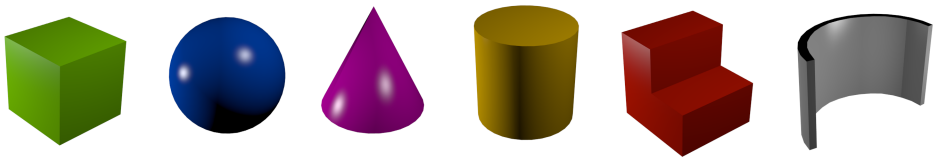


Figure 3.25 – The six main shapes used to generate our database. Each main shape generates hundreds of derived shapes by changing their size parameters.

Computation of the Volumetric Mesh

In order to compute the volume, mass and inertia matrices of the surface mesh, we first compute the volumetric mesh from the surface mesh. In that purpose, we apply a conforming Delaunay tetrahedralization using TetGen software [TetGen] to the surface mesh (TetGen is also able to rearrange the surface points to reinforce the quality of the mesh, or to apply a tetrahedron shape constraints). Once the tetrahedral mesh is built, we compute its center of mass, volume, and principal rotation axis (from an eigen-decomposition of its inertia matrix).

Normalization of the Shape

We normalize the surface mesh so that the extracted descriptors are translation, rotation and scaling-invariant (see Figure 3.26). We transform the input surface mesh to obtain a volume of $1\,m^3$, so that its center of mass coincides with the origin of the coordinate system, and so that its principal rotation axis is aligned with the axis of the coordinate system. We chose to

align the x-axis with the principal rotation axis that has the lowest inertia, and to align the y-axis with the second principal rotation axis (the direction of the third axis is arbitrarily determined on whether a left-handed or right-handed coordinate system is used).

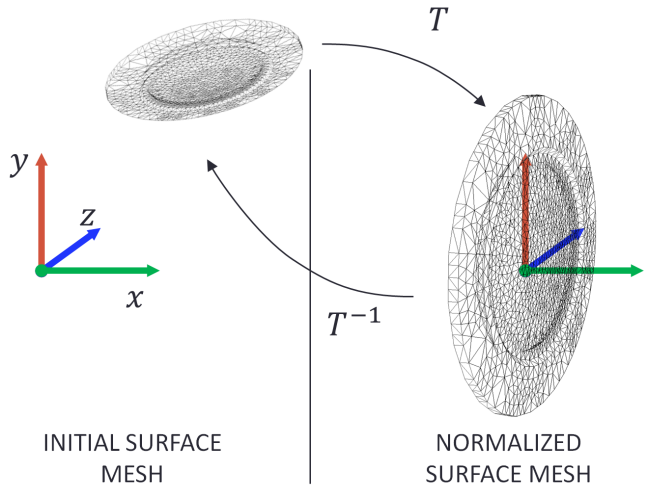


Figure 3.26 – Normalization of a surface mesh. After normalization, the center of mass coincides with the origin, the principal rotation axes coincide with the axes of the coordinate system, and the volume of the mesh is equal to 1. The affine transformation applied for the normalization is stored in a matrix $T \in \mathbb{R}^{4\times4}$.

Creation of the Descriptor

Once the surface mesh is normalized, we extract a descriptor $\mathbf{d} \in \mathbb{R}^n$. The size n and value of the descriptor depends on the type of descriptor used. The extraction of the descriptor is explained in section 3.5.2. When the descriptor is created, a new entry labeled with $\mathbf{d}$ is created into the database.

Adding Physical Data

To each entry of the database labeled by the value of a mesh descriptor $\mathbf{d}$, we associate as many physical data as needed by the simulation algorithms. For example, each entry can contain one or more of the follow data (see also Figure 3.13):

- A normalized tetrahedral volumetric mesh, that represents the shape of the object.
- The n first modes of deformation (computed by performing a modal analysis with either LAPACK [LAPACK] or ARPACK [ARPACK] software on the volumetric mesh). These deformation modes of a body enable to efficiently compute its small deformations due to input impulses [O'Brien 02b].
- Precomputed radiation data for sound generation [Gumerov 04, James 06]. These models can be used in association with the deformation modes to efficiently generate the sound of the objects.
- A collision shape structure: a compound shape built *e.g.* with spheres, cube and cylinders for rigid body simulators.

3.5.1.2 Searching into the Database

When a new object represented by its surface mesh is added to the scene, a similarity search is performed in the database to retrieve the needed physical data for simulation. In other words, given a query input mesh, we want to find in real-time the entry into the database which is the most similar to the input mesh. To perform the similarity search, the input mesh is first normalized as explained in section 3.5.1.1 to ensure translation, rotation and scale invariance. A descriptor $\mathbf{d}_{in}$ is then extracted from the normalized mesh. Finally, we select the entry e_i associated with its descriptor $\mathbf{d}_i$ into the database for which the similarity value $v(\mathbf{d}_{in}, \mathbf{d}_i)$ is the lowest (see Figure 3.24):

$$e_i = \underset{i}{\operatorname{argmin}} \left(v(\mathbf{d}_{in}, \mathbf{d}_i) \right) \tag{3.18}$$

In practice, the evaluation of $v(\mathbf{d}_{in}, \mathbf{d}_i)$ depends of the descriptor type and dimensionality. Moreover, it is often not necessary to check all the entries of the database to find the best match. Depending on the descriptor used, the database can be organized to optimize the similarity search, which are interesting properties w.r.t. real-time needs. These aspects are discussed in the following section.

3.5.2 Mesh Descriptors

We detail in this section the different descriptors used for the search request in the object database. In our context, an object descriptor $\mathbf{d} \in \mathbb{R}^n$ is a scalar vector of dimension n that is built from its surface mesh. The dimension as well as the content of the descriptor depends on the features extracted from the initial mesh. The real-time requirements impose several constraints on the choice of the descriptor. First, the descriptor must be efficiently built. Moreover, the dimension n of the descriptor should not be excessive, in order to avoid long search time w.r.t. the real-time update frequency. Finally, we preferred descriptors that enable to build partial ordered sets, to have more efficient search algorithms. With regards to these constraints, we retained three descriptors: a moment-based descriptor, a voxel-based descriptor and an improved voxel-based descriptor. Each descriptor is discussed along this section.

3.5.2.1 Moment-based Descriptor

Moment-based descriptor relies on the mass distribution of the material around axis. We compute the moment of inertia of the object around specific axes (we chose the three orthogonal axes aligned with the system of coordinates). The moment of inertia I around an axis $\mathbf{u}$ is computed by integrating the mass of the object around the axis $\mathbf{u}$ and over the volume V of the object:

$$I = \int_V dist(\mathbf{p}, \mathbf{u})^2 \cdot \rho(\mathbf{p}) \, dV(\mathbf{p}) \tag{3.19}$$

where $\mathbf{p}$ is a position into the volume, $\rho(\mathbf{p})$ is the material density at $\mathbf{p}$ and $dist(\mathbf{p}, \mathbf{u})$ is the distance between position $\mathbf{p}$ and axis $\mathbf{u}$. In our context, objects are defined with discrete geometry, and the moment of inertia is computed considering discrete positions $\mathbf{p}_i$ and masses m_i:

$$I = \sum_{i<n} dist(\mathbf{p}_i, \mathbf{u})^2 m_i \tag{3.20}$$

where n is the number of considered point-masses. In the case of surface meshes, we spread the mass of the object into its mesh nodes for the moment of inertia computation. If a tetrahedral volumetric mesh is available, it is possible to compute the moment of inertia using centers of tetrahedral elements as $\mathbf{p}_i$ and we use the volume of the element and its density to compute the associated m_i in equation (3.20).

We retain in the descriptor $\mathbf{d}$ the three moments of inertia computed around x-axis, y-axis and z-axis, sorting the values ascendantly. The descriptors are the sorted in the database, enabling dichotomous efficient search.

Similarity Computation

The similarity v between two moment descriptors $\mathbf{d}_1$ and $\mathbf{d}_2$ is computed using the square of the Euclidean distance between the two descriptor vectors:

$$v = \|\mathbf{d}_2 - \mathbf{d}_1\|^2 \tag{3.21}$$

3.5.2.2 Voxel-based Descriptor

Our version of voxel-based descriptors relies on a uniform grid that stores the spatial distribution of the surface of the object. We center a cubic uniform grid at the object center of mass, and size the grid so that its side is equal to the largest side of the bounding box of the input object. Each cell of the grid is marked with 0 if it contains no node of the object mesh, and 1 otherwise (see Figure 3.27(b)). The number of cells in the grid is a parameter of the descriptor that has consequences on the search results. The voxel-based descriptor $\mathbf{d}$ is built by writing the 3-D grid in a linear way.

Similarity Computation

The similarity v between two voxel-based descriptors is a value representing the number of cells that have a different value:

$$v = \frac{1}{n_{filled}} \sum_{i \in dim(\mathbf{d})} dif(d_{1i}, d_{2i}) \tag{3.22}$$

where n_{filled} is the total number of non-empty cells in both $\mathbf{d}_1$ and $\mathbf{d}_2$, and $dif(a, b)$ is 0 if a and b are equals, 1 otherwise. The normalization of the distance using $1/n_{filled}$ ensures that $0 \leq v \leq 1$, and that the objects that contain a large number of non-empty cells do not reduce artificially the distance between the descriptors.

3.5.2.3 Improved Voxel-based Descriptor

We propose a modified version of the voxel-based descriptor that avoids most non-identical meshes to have the same descriptor. The improved voxel-based descriptor contains the information of the voxel-based descriptor, but we add to each cell three scalars that represent the location of the center of mass of all the nodes that belong to this cell. This position gives an information on the distribution of the points inside the cell (see Figure 3.27(c)).

Similarity Computation

The similarity v between two improved voxel-based descriptors is computed using the Euclidean distance between the centers of mass of the cells. This distance is normalized using

the length l_{diag} of the diagonal of the cubic cell. If one cell of a descriptor is empty while the same cell of the other descriptor is not, the distance is set to 1:

$$v = \frac{1}{n_{filled} + n_{eq}} \cdot \sum_{i \in dim(\mathbf{d})} \left(dif(d_{1i}, d_{2i}) + (1 - dif(d_{1i}, d_{2i})) \times \frac{\|\mathbf{c}_{1i} - \mathbf{c}_{2i}\|}{l_{diag}} \right) \tag{3.23}$$

where n_{eq} is the number of common cells between $\mathbf{d}_1$ and $\mathbf{d}_2$ that are both equal to 1, $\mathbf{c}_i$ is the position of the computed center of mass of cell i.

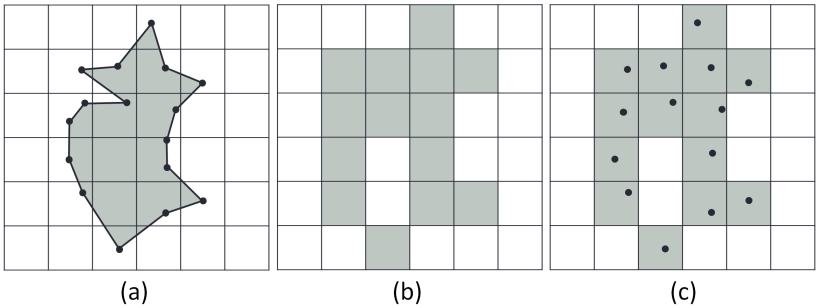


Figure 3.27 – Voxel-based descriptors used in our system. **(a)**: normalized surface mesh and its bounding grid. The center of mass of the mesh coincides with the center of the grid. **(b)**: Simple voxel-based descriptor that stores the cells of the grid containing at least one node of the surface mesh. **(c)**: Improved voxel-based descriptor that stores additionally the position of the center of mass of the nodes belonging to the cell.

3.5.3 Adaptation to Physical Simulation

Scaling the Physical Data

The physical data contained in the database are all normalized. In order to use them for physical simulation, there are two possibilities: (1) make a copy of the physical data and transform it so that it fits with the query object, or (2) transform the input data of the physical simulation so that it fits with the normalized object. The solution (1) consumes more memory, but it can be applied for most physical data (such as a stiffness matrix or deformation modes). Solution (2) can be more efficient, but can be applied only if the physical data properties scale linearly with a scale of the geometry of the object.

For example, the deformation modes, stiffness matrix and the collision data can be transformed using the inverse transformation T^{-1} of the normalization (see Figure 3.26). The geometric scale of the body is used to update the physical values (see [Zheng 10] for a derivation of how to scale stiffness matrix and mode of deformation). The volumetric mesh can be used to find a correspondence between a position $\mathbf{p} \in \mathbb{R}^3$ and the closest vertex of the mesh to this position (for collision and force application purposes). In this case, the mesh is neither scaled nor copied, but the input position $\mathbf{p}$ is transformed into the normalized coordinates using $\mathbf{p}' = T\mathbf{p}$. Then, the new position $\mathbf{p}'$ is used to find the closest vertex of the mesh.

Linking the Physical Data with the Virtual Objects

When simulating the deformation of the objects, we need to propagate the deformation stored into the physical data to the initial surface mesh. To do so, we express each Degree Of

Freedom (DOF) f_i of the initial surface mesh from a weighted sum of the degree of freedom of the physical mesh:

$$f_i = \mathbf{b}.\mathbf{q} \tag{3.24}$$

where $\mathbf{b}$ is a vector of weights (we impose $\sum_{i \in dim(\mathbf{q})} b_i = 1$), and $\mathbf{q}$ is the physical state of the body. In practice, we choose to find for each node of the surface mesh the element of the volumetric mesh that contains it. Then, we link the node of the surface mesh and the node of the element using barycentric coordinates (see Figure 3.28).

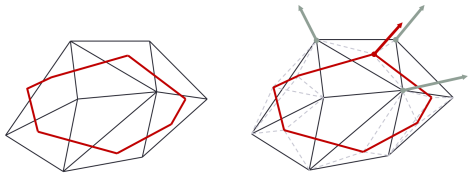


Figure 3.28 – Link between the surface mesh and physical data. The red mesh represents the initial object surface mesh. The black mesh is the best match from the database, transformed to fit with the surface mesh. Right: the surface mesh DOFs are linked to the physical mesh DOFs using the nodes of the element that contains the node of the surface mesh (represented with gray dotted lines). The red arrow is the displacement of the surface mesh node computed from the displacement of the physical mesh.

Database Search Optimization

Depending on the descriptor used, several optimizations can be applied to accelerate the database requests. For voxel-based descriptors, we set up an optimization based on the number of empty cells. We gather the descriptors that have the same number of empty cells. When a request is performed, the number of empty cells of the input descriptor is used to access directly the list of descriptors that have the same number of empty cells. This optimization enables to save computation time for each request. However, comparing the query object with the objects that have exactly the same number of empty cells in their descriptor is too restrictive, as we can miss the best entry in terms of equation (3.18). In practice, we take all the descriptors in the descriptors that have a number of empty cells n_e in the range $[n_e - r, n_e + r]$, where r is a manually set parameter (setting r to 6 in our test scenarios led to get always the best entry).

Summary of the Method

The overview of our method is presented in algorithm 5. This algorithm describes how the physical data is retrieved from the input surface mesh $\mathcal{S}$ using the database noted $\mathcal{D}$.

3.5.4 Results

Configuration

Our simulation method has been implemented in C++ on a laptop with 4 GB of RAM, and an Intel®Core™2 Extreme. The size of the grid for both voxel-based and improved voxel-based descriptors has been experimentally set to $10 \times 10 \times 10$.

Algorithm 5 Find the best physical data from a surface mesh

Require: $\mathcal{S}$: input surface mesh
Require: $\mathcal{D}$: shape database
1: T = normalize_transform($\mathcal{S}$)
2: $\mathcal{S}'$ = transform($\mathcal{S}$, T)
3: $\mathbf{d}$ = descriptor_from($\mathcal{S}'$)
4: e_i = best_entry($\mathbf{d}$, $\mathcal{D}$) // eqn (3.18)
5: $\mathcal{C}$ = collision_volume(e_i)
6: $\mathcal{M}$ = mode_deformation(e_i)
7: $\mathcal{C}$ = transform($\mathcal{C}$, T^{-1})
8: $\mathcal{M}$ = transform($\mathcal{M}$, T^{-1})
9: scale $\mathcal{M}$
10: link $\mathcal{M}$ and $\mathcal{C}$ to $\mathcal{S}$

Scenarios and Computation Time Performances

Figure 3.30 show a scenario where a plate is broken, and each fragment is assigned with a physical data. We show an example of scenario where a rigid bodies simulation is augmented to incorporate physically-based deformations computed with modal analysis in Figure 3.32. The improved voxel descriptor gives the best results thanks to the mass distribution information within each cell of their grid. On the opposite side, the moment-based descriptor gives less relevant results, due to a less efficient separation of the information of the mass distribution, and a lower dimensionality.

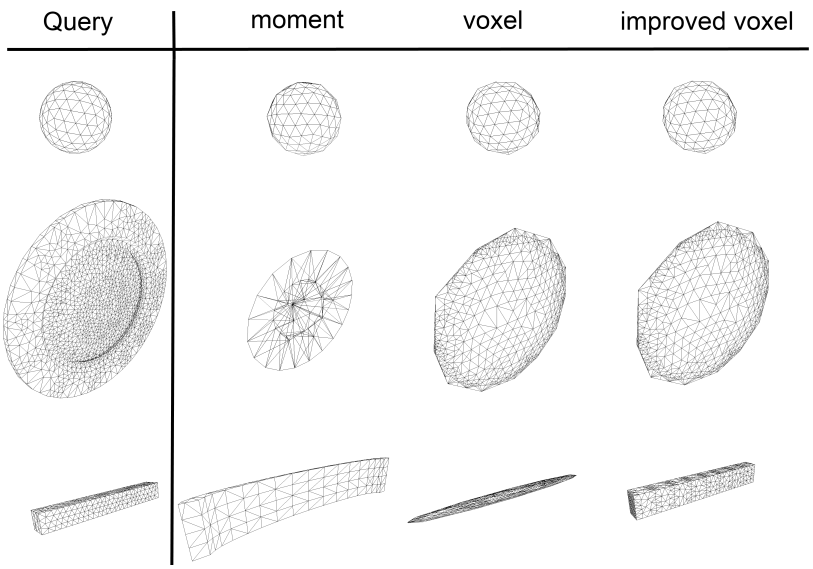


Figure 3.29 – Comparison of similarity search methods. The moment-based method gives more naive results, while the improved voxel-based method finds visually similar objects for the ball, the plate and the beam.

The timings for the scenario of Figure 3.31 shows that our solution is satisfying for interactive application. Indeed, the optimized version of the improved voxel based descriptor need a total of $4ms$ of processing time per fragment in average (the similarity search complexity is independent of the complexity of the input mesh).

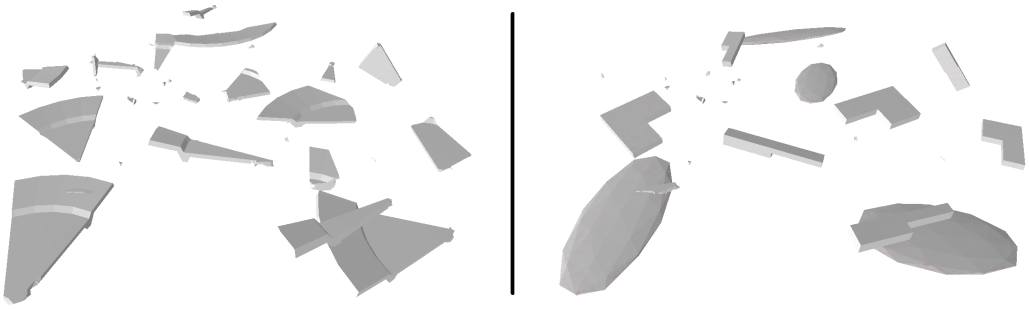


Figure 3.30 – Fragments associated to physical meshes in a fracture scenario. **Left**: surface meshes representing the fragments. **Right**: physical meshes associated to the same fragments.

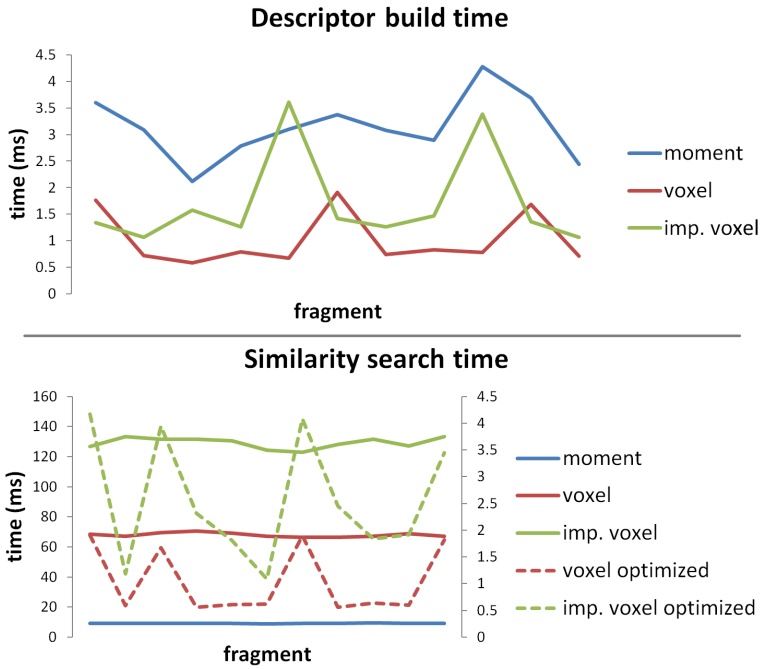


Figure 3.31 – Computation times for descriptor build and similarity search measured from scenario of Figure 3.30. The solid lines are relative to the left axis, while the dotted lines are relative to the right axis.

3.5.5 Discussion and conclusion

Physical Realism

The main limitation of our approach concerns the physical correctness that is achieved by using an approximated geometry. Indeed, the precomputed database contains a limited number of shapes, and complex concave objects have a small chance to be associated with acceptable data. It would be however possible to enrich the database with the objects during a specific scenario in an offline mode. This would result in a specific scenario-based database, and limit the physical error. Future work will concern a measurement of the physical correctness of applying approximated geometry on specific scenarios. Also, it would be necessary to establish a metric to measure the performances of the similarity search algorithm.

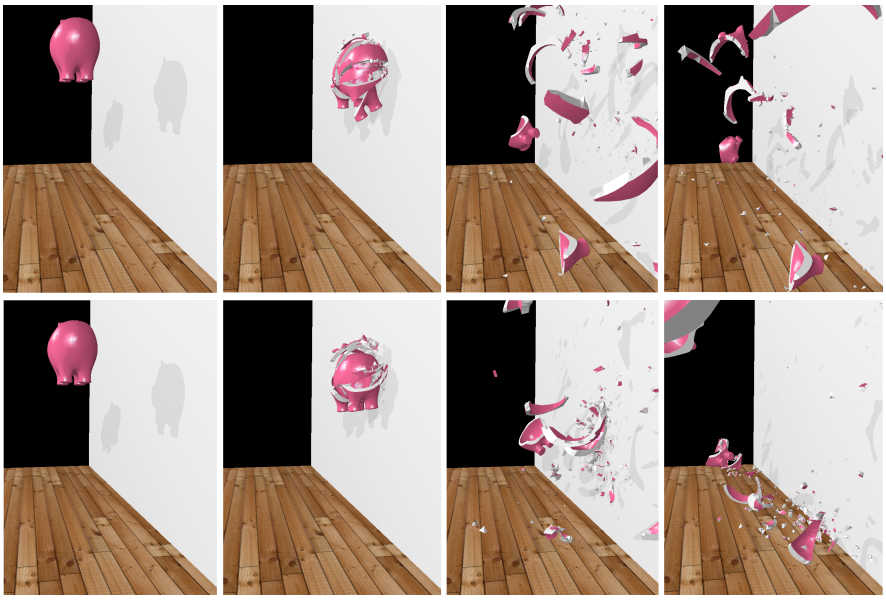


Figure 3.32 – Snapshots of a simulation of brittle fracture with modal analysis performed without our method (top), and with our method (bottom). Note the recursive fracture of the fragments generated from the first impact on the bottom animation that are not computed in the top animation.

Physical Data and Scaling

As highlighted in section 3.5.3, the physical data of the database should be scaled to fit with the size of the non-normalized query object. Even if most of physical data can be scaled with an homogeneous scale of the geometry of the body, our method cannot be applied with precomputed data that cannot scale with the geometry of the body.

Database Management

Currently, the database is entirely loaded into the RAM in order to provide a quick access to the data. However, increasing the number of objects in the database and/or the quantity of physical data would consume too much memory to be stored in the RAM. A solution could be to store only the objects descriptor in the RAM, and access the physical data from a massive storage device only when it is needed.

We have presented a method for real-time physically-based simulation of objects for which no physical data have been defined. Our method relies on a carefully generated database that stores all needed physical data, for a wide set of shapes. Each entry of the database contains a set of physical data for physical simulation. We also presented and compared three similarity search, and demonstrated their efficiency through scenarios of interactive physical simulation. We finally introduced an approach to scale and link the physical data stored in the database to the virtual objects. Our results show that the precomputed shape database proposes an interesting alternative solution for physical simulation in interactive applications.

3.6 Chapter conclusion

In this chapter, we presented models that allowed us to tackle the main limitations of existing real-time approaches. To allow free crack propagation, we designed an independent

fracture state model that stores the fracture state of bodies into formally defined mathematical structures. A crack propagation algorithm based on an implicit surface definition of the fracture surfaces has been proposed. The algorithm has a linear complexity in the number of cut tetrahedra, and features energy stop criterion that allows partial fracturing to occur. We also presented a meshing algorithm that generates efficiently the surface meshes of the fragments from the fracture state model.

In a second part, we proposed simulating the deformations of the bodies due to impacts using a modal analysis precomputed on the bodies. This modal analysis allowed us to approximate the duration of the contact as well as to choose an adaptive time step for the simulation. This leads to a real-time simulation of impact-based fracturing that takes into account the dynamic effects of the simulation. Results show that our method is capable of simulating inertia and damping effects that influence the fracture simulation, which was not previously possible to simulate in real-time.

We then proposed an adaptation of existing aging algorithm to simulate interactive age-based fracturing using our fracture state model. We introduced an approximate stress relaxation technique that allows simulating stress relaxation around cracks fast enough to provide interactive simulations.

One limitation of the modal analysis approach is the handling of fragments, for which no precomputation is possible, and the modal analysis cannot be applied of them. We addressed this problem proposing to store in a database precomputed physical data for a wide variety of shapes, and to associate the appropriate physical data at run-time for the generated fragments. Results showed that this approach generates approximate deformations that are mechanically not correct, but visually plausible.

Our model for brittle fracture has demonstrated fast and realistic results. To use this model in an interactive world composed of several fracturable bodies, new methods that handle the interactions between the bodies are needed, as presented in the next chapter.

New methods for interactive brittle fracture simulation

4

Contents

4.1 Introduction

While the previous chapter described new models for the simulation of brittle fracture, this chapter focuses on how a virtual world can be assembled with bodies described by our model of fracture. To this goal, several issues may be considered. First, the interaction between the bodies of the virtual world. Namely, to prevent bodies from overlapping each other, a collision detection step followed by a collision solving step must be performed in order to check whether bodies are overlapping, and if so, apply forces or impulses to solve for these overlaps. Second, the interactions between the human operator and the virtual world. We focus on the haptic interaction with a virtual world composed of fracturable bodies that has not been possible so far, and that presents new opportunities for interactive applications.

Collision detection and haptic interaction have been widely studied in the past. However, in the context of fracture simulation, unique challenges must be faced. In this chapter, we first present a new approach for an efficient collision detection adapted to brittle fracture. This work has been done in collaboration with the University of Rey Juan Carlos, in the team led by Miguel A. Otaduy. In a second part, we demonstrate the applicability of our method to the interactive simulation of fracture with a stable haptic interaction.

4.2 Efficient collision handling for brittle fracture

Collision detection (presented in section 2.6.1) and collision or *contact* solving (presented in section 2.6.2) have been widely studied in the literature, and a lot of efficient methods can be used out of the box in many scenarios. However, the case of fracturing bodies presents unique challenges. Indeed, new fragments with not necessarily convex shapes are created within one simulation step, preventing any precomputation, common in the collision detection field. Also, at the fracture instant, the fragments of the same body are all close to each other and share close and parallel surfaces that reduce the chances for a good pruning.

We present novel algorithms and data structures for collision detection in real-time brittle fracture simulations. We build on a combination of well-known efficient data structures, namely distance fields and sphere trees, making our algorithm easy to integrate on existing simulation engines. We propose novel methods to construct these data structures, such that they can be efficiently updated upon fracture events and integrated in a simple yet effective self-adapting contact selection algorithm. Altogether, we drastically reduce the cost of both collision detection and collision response.

4.2.1 Overview of the collision detection algorithm

We execute collision detection tests between pairs of objects A and B, which may be either original unfractured objects or fragments resulting from fracture events. Without loss of generality, let us refer to them as fragments. We augment each fragment A with two data structures for collision detection: a distance field $D(A)$ and a sphere tree $S(A)$. Section 4.2.2 and Section 4.2.3 describe the contents, construction and update of our novel *fragment distance field* and *fracturable adaptive sphere tree* data structures.

When broad-phase collision culling returns the pair (A, B) as potentially colliding, we query both $S(A)$ against $D(B)$ and $S(B)$ against $D(A)$. The result of a query $(S(A), D(B))$ is a set of contact constraints C, each defined by a tuple $(\mathbf{c}, d, \mathbf{n})$. $\mathbf{c} \in A$ is a contact point, d is the closest distance from $\mathbf{c}$ to the surface of B, and $\mathbf{n}$ is the contact normal. If $\mathbf{c}$ is inside B, d is negative and represents the penetration depth. After collision detection, we feed the complete set of contact constraints to a constraint-based contact solver with a velocity-level LCP (with friction), plus constraint drift correction. In our examples, we have used the off-the-shelf contact solver built in Havok Physics.

In our algorithm, a query $(S(A), D(B))$ builds on three elementary queries involving nodes of the sphere tree $S(A)$. Each node is represented by its center point $\mathbf{p} \in A$ and a radius r. Then, the three elementary queries are:

- insideTest$(\mathbf{p}, D(B))$: it determines whether $\mathbf{p}$ is inside or outside B.
- penetration$(\mathbf{p}, D(B))$: when $\mathbf{p}$ is inside, it computes the penetration depth from $\mathbf{p}$ to the surface of B, as well as a penetration direction $\mathbf{n}$.
- sphereTest$(\mathbf{p}, r, D(B))$: when $\mathbf{p}$ is outside, it performs a conservative test for intersection between B and the sphere of radius r centered at $\mathbf{p}$.

These three elementary queries will be described in detail in Section 4.2.2. In all our descriptions, we assume that the point $\mathbf{p}$ has been transformed to the local reference system of fragment B.

Our collision detection algorithm, outlined in Algorithm 6, traverses a sphere tree $S(A)$ in a breadth-first manner, and prunes branches that are completely outside the other fragment

Algorithm 6 Query sphere tree S against distance field D

```
1: INPUT: S, D
2: OUTPUT: Set of contacts C
3: Q = {root(S)}
4: while Q ≠ ∅ do
5:    node ← pop_front(Q)
6:    inside ← insideTest(node.p, D)
7:    if inside then
8:       (d, n) ← penetration(node.p, D)
9:       C = C ∪ (node.p, d, n)
10:      if sufficientContacts(C) then
11:         STOP
12:      end if
13:   else
14:      intersects ← sphereTest(node.p, r, D)
15:   end if
16:   if inside OR intersects then
17:      Q = Q ∪ children(node)
18:   end if
19: end while
```

B. Pruning is efficiently executed by comparing the radius of a sphere and the distance from its center to the surface of B. The algorithm can be easily modified to allow for a contact tolerance ϵ. A contact constraint is added to C if the distance is shorter than ϵ, and the query descends to the children if the distance is shorter than $r - \epsilon$.

We augment the basic collision detection algorithm with *self-adapting contact selection*. As described in Section 4.2.3, we construct the sphere tree in a way that allows adaptive contact selection by simple breadth-first tree traversal, defining a contact constraint whenever we encounter a sphere whose center is inside fragment B, until a sufficient number of constraints is reached.

4.2.2 Fragment Distance Field

Given a volumetric meshing of an object A, computed as a preprocess, we propose a fragment distance field data structure that is efficiently stored and updated even upon multiple fractures of the object. This data structure stores an approximate interior distance field of all fragments created by the fractures, using a precomputed volumetric mesh, without any remeshing. Moreover, we exploit the connectivity of the mesh to compute approximate distances in a very fast manner using a front propagation approach.

In this section, we first describe the distance field data structure and its run-time update, and then we describe how it is used to answer the three elementary queries outlined in the previous section.

4.2.2.1 Mesh-Based Interior Distance

Our distance field data structure is motivated by features of fracture simulation algorithms. First, the fragments resulting from a brittle fracture define an exact partition of the original object. Therefore, each point of the original object needs to store only one interior distance value even after multiple fractures. And second, popular approaches for fracture simulation

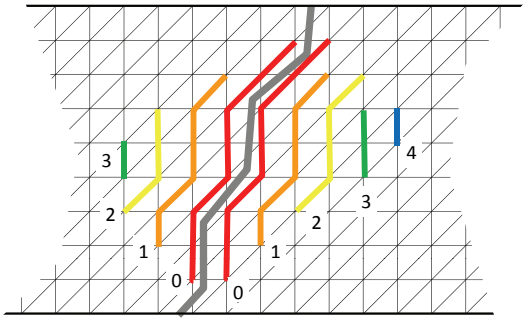


Figure 4.1 – Illustration of the front propagation algorithm for interior distance field computation.

use a volumetric mesh to compute an elastic deformation field and guide the propagation of crack surfaces [O'Brien 99, Müller 01]. In our fracture state model, the resolution of newly created fragments is limited, forcing them to include one node of the original mesh. We exploit this feature and store one interior distance value at each node of the original volumetric mesh.

In our implementation, we leverage the tetrahedral mesh used for fracture to store the fragment distance field. Specifically, each node of the mesh stores:

- f: an identifier of the fragment that contains the node.
- d: a value that approximates the shortest distance to the surface of fragment f.
- $\mathbf{n}$: a unit vector that approximates the direction from the node to the closest surface of f.

As a preprocess, we initialize the nodal information using an exact interior distance field.

In addition to nodal information, tetrahedra that are intersected by crack surfaces store exact local representations of those crack surfaces. Each tetrahedral edge may be cut at most once, therefore, the storage requirements are limited to six plane equations.

4.2.2.2 Distance Updates upon Fracture

After each fracture event, we locally update the fragment distance field where needed, following a front propagation approach. The run-time computation of the exact distance field is computationally prohibitive, but we propose a fast approximation that fulfills desired properties. It is important to remark that the interior distance of a fragment is used in the computation of penetration depth and contact normal in the collision detection lgorithm 6. The amount of penetration depth is used by the drift correction algorithm during collision response, and the contact normal is used for the definition of non-penetration contact constraints. Distances need not be accurate, but they must grow monotonically in the interior of an object, locally approximate Euclidean distance, and converge to the true distance as we get close to the surface of the object. Normal directions, on the other hand, must point outward of the object, and should vary smoothly to avoid competing contact constraints for nearby points. It turns out that the algorithm for consistent penetration depth computation of Heidelberger *et al.* [Heidelberger 04] fulfills exactly these properties, hence we have adapted this algorithm for interior distance field approximation.

Next, we summarize the application of Heidelberger's algorithm to our problem. When an object A suffers a fracture, we first visit all tetrahedra intersected by the newly created crack surfaces, and initialize distance field information at their nodes. This implies assigning a fragment identifier f, and computing a distance d and a direction $\mathbf{n}$, based on the exact

surface information stored at the tetrahedra. For each fragment, we initialize a front with the visited nodes. Then, we iterate front propagation steps on the graph defined by tetrahedral edges, until no distances are reduced. Figure 4.1 illustrates the front propagation inside a fragment.

If the front propagation reaches a node at position $\mathbf{p}$ in step $i+1$, we compute a distance to the surface as an average propagation of distances from its neighbors reached in step i (denoted as $N_i(\mathbf{p})$), in the following manner:

$$d = \frac{\sum_{j \in N_i(\mathbf{p})} w_j \left(d(\mathbf{p}_j) + \mathbf{n}(\mathbf{p}_j)^T (\mathbf{p}_j - \mathbf{p}) \right)}{\sum_{j \in N_i(\mathbf{p})} w_j}. \tag{4.1}$$

Following Heidelberger et al., we use as neighbor weights $w_j = 1/\|\mathbf{p}_j - \mathbf{p}\|^2$. If the distance d is shorter than the current value stored at $\mathbf{p}$, we update the distance and add $\mathbf{p}$ to the front at step $i+1$. We also update the direction at $\mathbf{p}$ as the weighted average direction of neighbors reached in step i:

$$\mathbf{n} = \frac{\sum_{j \in N_i(\mathbf{p})} w_j \, \mathbf{n}(\mathbf{p}_j)}{\sum_{j \in N_i(\mathbf{p})} w_j}, \tag{4.2}$$

followed by a normalization step.

Figure 4.2 shows an accurate interior distance field for an object A, the accurate distance fields of its fragments after a fracture, and our approximate distance fields. The image illustrates the monotonic growth of distances inside the fragments. Our distance field approximation does not require high-quality tetrahedral meshes in practice. In our examples, we used TetGen for mesh generation, with a maximum radius-edge ratio between 1 and 2, and enforcing interior edges to be shorter than twice the length of the longest surface edge.

Even though we have used tetrahedral meshes, our data structure could be extended to other settings, such as hexahedral meshes or even meshless methods. The mesh is used in two ways: (i) Its edges define a graph for the propagation of distances; (ii) Distances can be interpolated inside mesh elements. On hexahedral meshes, the graph may be constructed using the edges of the mesh or adding other connections, and interpolation can also be defined inside mesh elements. On meshless methods, a graph may be constructed using neighboring particle information which is easily updated upon fracture events [Steinemann 06], and interpolation can be defined based on neighboring nodes [Müller 04a].

4.2.2.3 Inside-Outside Query: insideTest($\mathbf{p}, D(f)$)

As a preprocess, we build a k-d tree with the tetrahedra of the mesh. To decide whether a point $\mathbf{p}$ is inside a fragment f, we first use the k-d tree to retrieve the tetrahedron that encloses $\mathbf{p}$. If all nodes of the tetrahedron are in the same fragment, then the query is trivially answered. If only some nodes are in fragment f, then we use the exact local representation of the crack surfaces to answer the inside-outside query.

4.2.2.4 Penetration Depth Query: penetration($\mathbf{p}, D(f)$)

If the tetrahedron containing a point $\mathbf{p}$ is intersected by crack surfaces, we use the exact local surface information to compute the penetration depth and direction to the surface of fragment f. If the tetrahedron is completely inside the fragment, then we use the fragment distance field. In particular, we use as neighbors $N_i(\mathbf{p})$ the four nodes of the tetrahedron, and we apply equation 4.1 to compute the distance to the surface of f, and 4.2 to compute the penetration direction.

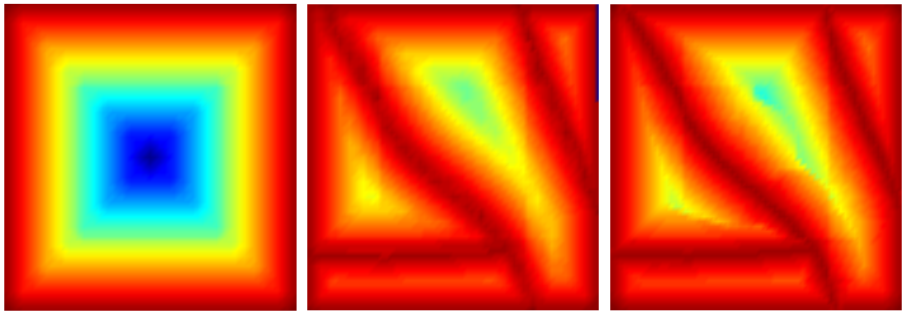


Figure 4.2 – Comparison between recomputed and updated distance field. From left to right: interior distance field of a 2D object, distance fields of its four fragments after fracture, and our approximate distance fields computed using a fast propagation method.

Close to original surfaces of the object, where fracture does not modify distances, it is possible to obtain more accurate penetration information in a simple manner. As a preprocess, we compute a distance field on a dense regular grid. This regular-grid distance field is used for the initialization of the fragment distance field at nodal positions, but we also query it at run-time. We simply use the minimum of the distances returned by the (precomputed) regular-grid distance field and the (dynamically updated) fragment distance field.

4.2.2.5 Sphere Intersection Query: sphereTest($\mathbf{p}, r, D(f)$)

The fragment distance field stores only interior distance information for the fragments. When the query point $\mathbf{p}$ is outside fragment f, the fragment distance field provides the distance d to the surface of some other fragment. This distance d is a lower estimate of the distance to f, and can be used for culling in lgorithm 6 if it is larger than the radius r of the sphere. To handle far fragments, we use the largest between d and the distance to a bounding box of fragment f.

The procedure described above performs well in most cases, but fails for large non-convex fragments surrounded by small fragments, returning largely underestimated distances that produce little culling. We provide a less conservative solution for such situations. As a preprocess, we construct a multi-level grid on every object, and register pointers from the tetrahedra to their occupied cells. Every grid cell stores a bit mask indicating whether it contains each and every fracture fragment. Upon a fracture event, we traverse the tetrahedra of new fragments, mark the bit masks of their occupied cells, and then we perform a bottom-up update of the multi-level grid by simple logical AND operations. To test a sphere for intersection, we query the grid level with cell size immediately larger than $2\,r$. The sphere can be culled if none of the eight cells joining at the grid point closest to $\mathbf{p}$ contains fragment f.

4.2.3 Fracturable Adaptive Sphere Tree

Our novel sphere tree data structure is motivated by two requirements. First, to reduce the cost of both collision detection and response, in particular at collision-intensive fracture events, we seek a sphere tree data structure suitable for adaptive collision detection. We construct a sphere tree by optimizing at each level the coverage of the fragment, in a way similar to the point-shell hierarchy of Barbič and James [Barbič 08]. In this way, we achieve good contact sampling through simple breadth-first traversal of the sphere tree. Second, the data structure should allow very fast updates upon fracture events. We construct the sphere tree by covering both the surface and the interior of an original object. Prior to fracture,

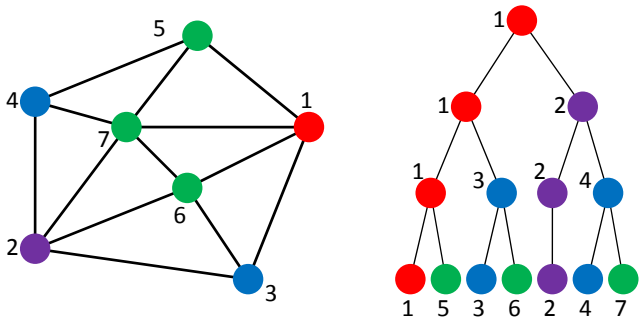


Figure 4.3 – Construction of the hierarchy. A 2D polygon with surface vertices and interior nodes (left) and its sphere tree (right). The points are numbered according to maximum distance ordering, and colored according to the level in which they are added to the tree.

interior parts are easily culled and produce almost no overhead. After they get exposed by fracture, on the other hand, they are quickly accessed during tree traversal.

Next, we describe the ordering of points and the construction of our fracturable adaptive sphere tree, the procedure for updating the sphere tree upon fracture, and the execution of self-adapting collision detection.

4.2.3.1 Ordering and Construction of the Sphere Tree

We build a sphere tree on a set of points $P = \{\mathbf{p}_i\}$ representing an object A. As discussed in Section 4.2.2.1, we assume that the fracture algorithm relies on a volumetric mesh associated with the object. To anticipate fracture events, the set of points consists of the union of the surface vertices and the nodes of the volumetric mesh. The full role of interior nodes during tree updates will be explained in Section 4.2.3.2.

For good adaptive collision detection during tree traversal, we seek a good sampling of the set P on every level of the tree. This can be achieved by selecting most distant points, which would produce a good coverage of the object. Barbič and James [Barbič 08] seed random points for each level of the tree, and achieve good coverage thanks to a relaxation algorithm. Although their approach might be adapted to our setting, we seek two additional features: our set of points includes interior points in addition to surface points, and the sampling retains the original surface vertices and interior nodes, to later accommodate fracture updates. In addition, we assume meshes sampled in a semi-regular way.

We achieve good coverage of the object at each level through *maximum distance ordering* of the points P. We initialize an ordered list L_2 with the two furthest points. Then, given the ordered list with m points, L_m, we append the point that is furthest from its closest point in L_m, i.e., $L_{m+1} = (L_m, \mathbf{p}^*)$, with $\mathbf{p}^* = \arg\max_{\mathbf{p}_i \notin P_m} \min_{\mathbf{p}_j \in P_m} \|\mathbf{p}_i - \mathbf{p}_j\|$.

Given the full ordered list, level l of a sphere tree, with 2^l nodes, is trivially constructed by selecting the first 2^l points in the list. Then, level $l+1$ is constructed using those same 2^l nodes and the following 2^l nodes in the list. We set as parent of a node in level $l+1$ its closest node in level l, just like Barbič and James [Barbič 08]. This heuristic groups nodes based on proximity and increases the chances for pruning during run-time queries. Figure 4.3 shows a 2D example with the maximum distance ordering and the tree construction. The sphere tree construction is a preprocess, and we have followed an unoptimized $O(n^2)$ implementation based on pair-wise distance computation, but accelerations are possible.

Each node of the tree must store the sphere center (i.e., the point position $\mathbf{p}$) and radius. For a node with center at $\mathbf{p}$, we precompute the radius as the distance to its furthest descendant. Each point may be present at multiple levels in the tree (but with different radii).

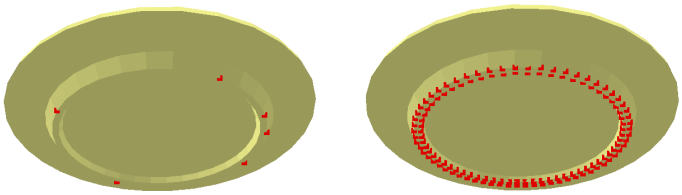


Figure 4.4 – Illustration of the contact selection mechanism. Rolling plate on a transparent ground, with contacts output by (left) our self-adapting collision detection (up to 6), and (right) full collision detection (up to 128). See the accompanying video.

We define a contact constraint only the first time the point is queried in lgorithm 6, and we cache its inside-outside status for subsequent queries down the tree. Choosing the point $\mathbf{p}$ as the center of the sphere does not yield optimally tight spheres. We tried approaches that produce tighter spheres with better culling, but the overall query times were worse as we could not exploit caching.

4.2.3.2 Tree Updates upon Fracture

We restructure the sphere trees through simple local modifications of parent-child relationships. Given a node belonging to a fragment f_i, if its parent belongs to a different fragment f_j, we make the node a child of its closest ancestor in f_i. If it has no ancestor in f_i, the node becomes a root for f_i. A direct implication of this decision is that sphere trees may become forests of several trees after fracture events.

When an edge of the volumetric mesh is cut by a crack surface, two new points need to be added to two different fragments. Recall that we assume fracture algorithms adopting the virtual node approach [Molino 04], and then each edge is cut at most once. Given a new point $\mathbf{p}_i$, we make it a child of its neighboring original point $\mathbf{p}_j$. Note that $\mathbf{p}_i$ is a surface point, and we follow an insertion approach that tries to place $\mathbf{p}_i$ high in the hierarchy.

The insertion of new points also requires the modification of sphere radii, as they may no longer be conservative. When $\mathbf{p}_i$ is added, we check if the radius of its parent $\mathbf{p}_j$ is shorter than $\|\mathbf{p}_i - \mathbf{p}_j\|$, and we update it accordingly. We also propagate updates up in the tree if needed.

Compared to full tree recomputation, our fast tree update offers a much more efficient solution at fracture events, and a good compromise for subsequent simulation frames. In simulations where fracture events are distributed over time, our approach could be extended with full tree recomputation as a parallel task, followed by data structure swapping.

4.2.3.3 Self-Adapting Collision Detection

The fragment distance field and the fracturable sphere tree enable fast queries and fast data structure updates upon fracture. However, in situations with many penetrating points or with parallel surfaces in close proximity, the cost of collision detection is inevitably high, and collision response is affected by the large number of contacts. We have designed a self-adapting collision detection algorithm that produces a reduced set of contact constraints, and at the same time guarantees the absence of penetration (at the final stable configuration). Our algorithm relies on a velocity-level constraint-based contact solver plus drift correction, the gold standard solution in rigid body engines in video games.

During breadth-first traversal of the sphere tree, we may output contact constraints high in the hierarchy as outlined in lgorithm 6. Thanks to the good sampling provided by the

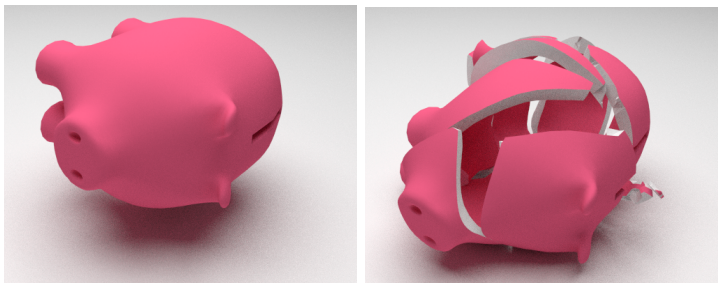


Figure 4.5 – Piggy bank used for comparisons and analysis.

maximal distance ordering, a few of the first encountered contacts are probably sufficient for the velocity-level constraint-based solver, while further contacts become redundant. We initialize a collision query between two fragments f_i and f_j by setting a maximum number of contacts m (8 in our experiments), and if this number is reached we simply interrupt the query. Drift correction quickly resolves the contacts that have been detected, but if this number is m, then other contacts may have been missed. In that case, we increment $m \leftarrow m + 1$, and continue the sphere tree traversal with a negative tolerance $-\epsilon$ (in our experiments $\epsilon = 0.2\%$ of the object radius), i.e., we search for contacts that penetrate further than ϵ. Effectively, with this approach collision detection self-adapts until the number of contacts guarantees non-penetration up to a tolerance ϵ on a stable configuration. Figure 4.4 compares the number of contacts for a $5,392$-triangle plate rolling on a transparent ground with our approach vs. full collision detection. Self-adapting collision detection requires at most 6 contacts, while full collision detection outputs up to 128 contacts. In self-adapting collision detection, adaptivity could also be guided by error metrics of collision response, but existing approaches do not address the complex interactions of constraint-based collision response.

We found that, to be effective at fracture events, self-adapting collision detection requires a small positive detection tolerance ϵ, i.e., we output contacts closer than a small distance ϵ. The reason is that the tree traversal stops only when m contacts are output, and parallel surfaces just about to touch would allow little culling but produce no contacts.

4.2.4 Experiments and Results

We evaluated our approach on 5 scenarios: (i) a piggy bank dropped on the ground (Figure 4.5), (ii) 27 bunnies dropped at different times (Figure 4.8), (iii) 32 bricks crashed against the ground (Figure 4.9), (iv) an interactive scenario where the user drops balls on 4 plates placed on a shelf (Figure 4.10), and (v) another interactive scenario where the user manipulates and fractures 5 bunnies (Figure 4.11). The sizes of the surface and volumetric meshes of the different objects are summarized in Table 4.1. Our collision detection algorithm has been integrated with the rigid body engine of Havok. The 'freezing' utility of Havok Physics was deactivated in all experiments, for better analysis. All experiments were executed on a 3.4GHz Intel i7-2600 processor with 8GB of RAM, using only one core.

Tables 4.2 and 4.3 report various simulation statistics and timings for the 5 benchmarks. The 'piggy bank', 'bricks', 'plates', and 'interactive bunnies' benchmarks are all real-time, including dynamics simulation, fracture simulation, collision detection, and collision response. The 'drop bunnies' scenario, on the other hand, was executed with a subframe time step (5ms) to illustrate robust contact handling with small fragments and high impact velocities.

Figure 4.12 shows plots of timings and simulation statistics for the 'drop bunnies' and 'bricks' scenarios. In both examples, the cost of collision detection grows steadily as more

Object	# Triangles	# Tetrahedra	# Points
Piggy bank	9,722	20,807	5,870
Bunny	7,940	18,767	5,089
Brick	468	594	224
Plate	5,392	8,617	2,711
Shelf	4652	10,200	2,989

Table 4.1 – Number of triangles, tetrahedra, and points (including surface vertices and interior points) for the different objects used in the experiments.

Scenario	# Triangles Original/Fractured	# Fracture Fragments	# Output points Average (Max)	# Colliding points Average (Max)
Piggy bank	9,734 / 17,622	14	93 (463)	290 (4,173)
Drop bunnies	137,403 / 435,068	166	1,235 (3,245)	4,993 (18,592)
Bricks	15,036 / 40,750	156	671 (1,379)	1,334 (2,700)
Plates	26,268 / 45,762	44	309 (524)	X
Interactive bunnies	39,724 / 51,752	22	326 (470)	X

Table 4.2 – Simulation statistics for the different scenarios: number of triangles of the scene before and after fracture; total number of fragments; number of contacts selected by collision detection for collision response; and total number of colliding points (not measured in the interactive scenarios).

Scenario	Time step	Total time Average (Max)	Collision detection Average (Max)	Physics Average (Max)	Update Max	Fracture Max
Piggy bank	15	1.94 (15.2)	1.74 (9.46)	0.14 (0.59)	1.2	12.5
Drop bunnies	5	24.5 (81.5)	23 (70)	1.3 (3.66)	4.4	22
Bricks	30	11.7 (29.5)	10.7 (27.3)	1 (2.46)	1.05	2
Plates	30	7.83 (17.4)	6.78 (12.2)	0.36 (0.7)	0.7	8
Interactive bunnies	15	2 (10.6)	1.64 (4)	0.26 (0.41)	1	9

Table 4.3 – Break-up of timings for the different scenarios, all given in milliseconds, and showing average and maximum cost per time step: time step size (with frames rendered every 30ms); total cost per time step; time for collision detection queries; time for physics computations, numerical integration, and collision response; time for data structure updates; and time for fracture computations. The last two times are measured only at fracture events.

objects are dropped. However, we can draw the important observation that collision detection does not suffer noticeable spikes at fracture events, despite the large number of colliding points, thanks to our self-adapting contact selection. Both scenarios show computational peaks at fracture events due to the cost of fracture computation. The cost of data structure updates was always smaller than the cost of fracture computation, and is not showed for clarity (but it is summarized in Table 4.3).

We have also analyzed the influence of resolution (both for the surface mesh and the interior sampling of the volumetric mesh) on data structure updates and collision detection queries (for the sphere in Figure 4.6). The timings for a reference sphere (2.5K triangles and 4K interior points) are: 1.54ms for updates upon fracture, and 1.16ms on average (3.27ms max) for queries. Changing the surface resolution (to 10K triangles), while keeping the interior sampling fixed, timings are: 1.8ms for the update, and 1.37ms on average (4.17ms max) for queries. Changing the interior sampling (to 435 points), while keeping the surface

fixed, timings are: 0.46ms for the update, and 0.68ms on average (2.26ms max) for queries.

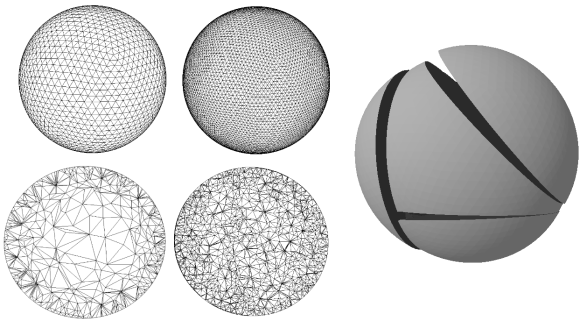


Figure 4.6 – Sphere used for the analysis of sampling resolution on update and query costs. The top left images show two different samplings of the surface, and the bottom left images show two different samplings of the interior.

Finally, we have also analyzed the overhead introduced in collision detection queries by our data structures, which trade fast updates upon fracture for not fully optimal culling. Figure 4.7 plots several comparisons for the 'piggy bank' scenario in Figure 4.5. Our approach updates the distance field (D) and the sphere tree (S) when the piggy bank crashes. We have compared collision detection query times and the number of visited points in the sphere trees, with other combinations where we recompute the exact distance field and/or we recompute the sphere tree for the new surface (with no interior points).

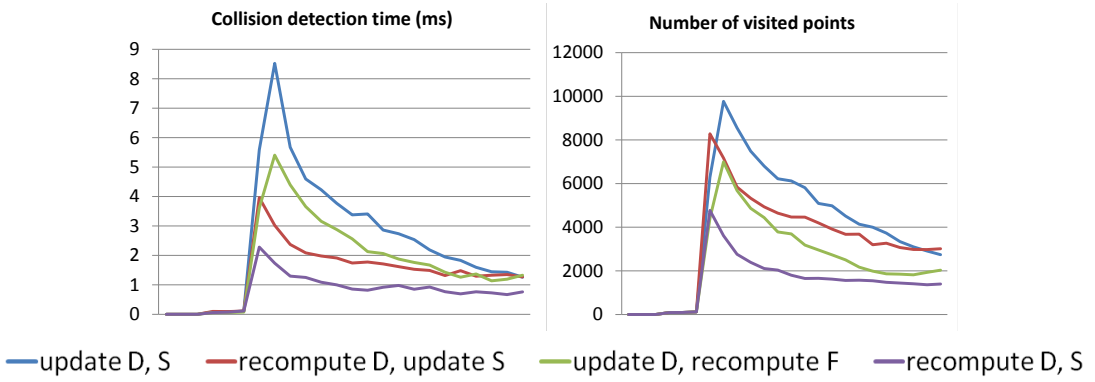


Figure 4.7 – Comparison of query times and statistics for the 'piggy bank' scene. We compare our fast update of the distance field D and sphere tree S data structures to other combinations that fully recompute an exact distance field and/or a sphere tree of the new surfaces. We achieve similar culling efficiency with much faster updates at fracture events.

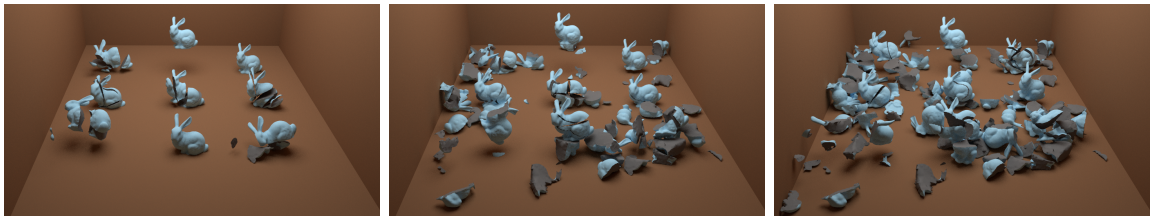


Figure 4.8 – Bunnies are dropped in three batches and fractured into 166 fragments and 435K triangles. The complete simulation runs at 24.5ms per time step on average, with a maximum of 81.5ms.

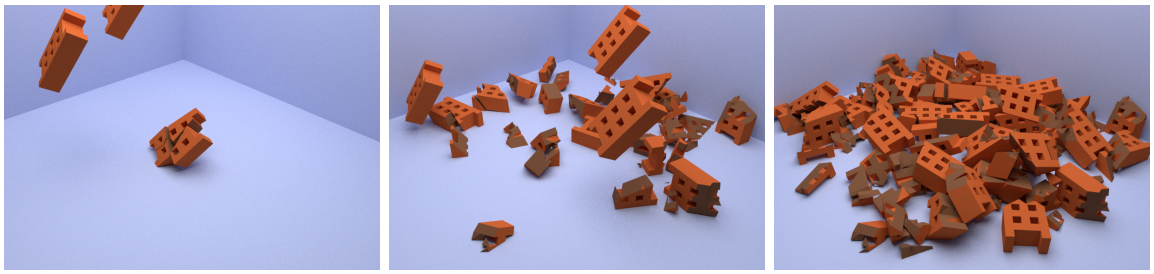


Figure 4.9 – Real-time demo of crashing bricks, totaling 156 fragments and 40K triangles. The complete simulation runs at 11.7ms per time step on average, with a maximum of 29.5ms.

Figure 4.10 – Interactive smashing of plates. The user drops balls in real-time to smash the plates, and at the end of the simulation the scene consists of more than 45K triangles. The complete simulation runs at 8ms per time step on average, with a maximum of 17ms.

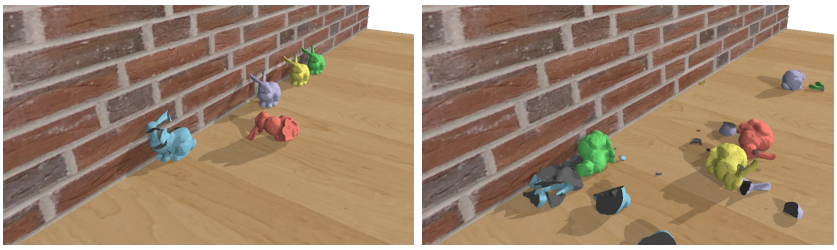


Figure 4.11 – The user manipulates bunnies interactively with the mouse, producing fractures and collisions. The complete simulation runs at 2ms per time step on average, with a maximum of 10ms.

4.2.5 Discussion and conclusion

We have proposed an efficient solution for collision detection in brittle fracture simulations. Our solution is composed of algorithms that address the two main challenges in such simulations: the update of acceleration data structures upon topology changes, and the efficient computation of contacts between newly created crack surfaces.

Our algorithms demonstrate high performance in challenging scenarios, including real-time user manipulation of fracturing objects, and scenes with hundreds of fragments and tens of thousands of triangles simulated at video game rates. Some limitations remain however, including the possibility to miss collisions with small features and robustness problems under large penetrations. Solving these limitations requires non-trivial extensions to incorporate continuous collision detection.

We envisage other interesting extensions as well. One is the design of parallel versions of our algorithms, to exploit the computing power of graphics processors. Another one is the application of our solutions to penalty-based collision response. The adaptive contact selection was designed for constraint-based response algorithms and may not be trivially adapted, but other components, such as the fragment distance field, may be easily integrated.

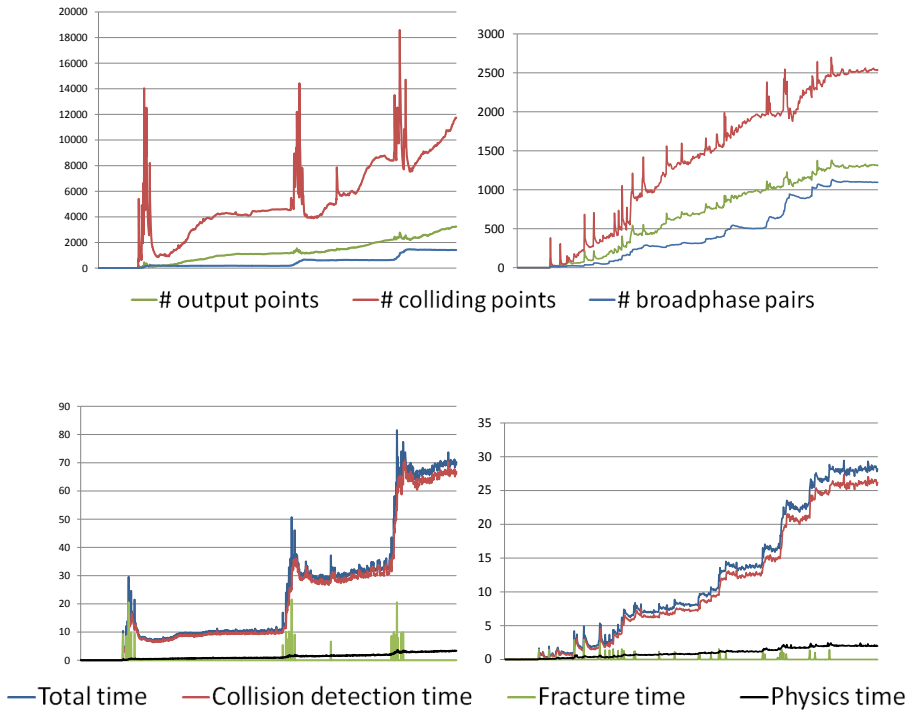


Figure 4.12 – Plots for the 'drop bunnies' scene (first and third plots) and the 'bricks' scene (second and fourth plots). The two left plots show collision detection statistics: number of points output by adaptive collision detection (green), number of actual colliding points (red), and pairs of bodies output by broad-phase collision detection (blue). The two right plots show timings per time step: total (blue), collision detection (red), fracture computation (green), and physics computations (black). Times to update collision detection data structures were always shorter than fracture computation, and are not included for clarity.

Yet another interesting extension is handling ductile and/or progressive fracture and elastic deformations. Since our approach already updates data structures at fracture events, it should also be possible to update those data structures as objects deform and fractures progress. Distance fields and sphere trees could also serve for self-collision detection algorithms.

The results of our experiments open promising perspectives for the use of our solutions in real-time applications such as video games and haptic interaction.

4.3 Haptic interaction with fracturing bodies

The haptic interaction, or interaction with the sense of touch (presented in section 2.6.3) is a rich way of interacting with the virtual world. As opposed to unidirectional interactions such as interaction with the keyboard or the mouse, haptic rendering allows feeling what is happening in the virtual scene thanks to our kinesthetic sensors. Haptic rendering not only is useful as an output device, but can be helpful to check perceptually the behavior of a simulation. When something not visually perceptible is going wrong in the simulation, a haptic output often reveals the artifacts.

In this section, we show how we made possible the coupling between the fracture simulation and the haptic rendering. First, we present the feasibility of the coupling between third party rigid body engines and haptic rendering. Second, we present how to maintain the stability of the interaction even when the number of body is growing.

4.3.1 Benchmarking the rigid body engines for haptic

We present in this section an evaluation of dynamic engines, based on relevant criteria chosen with respect to haptic rendering of contact between rigid bodies. The objective of this evaluation is to determine whether popular real-time dynamic engines are well-suited for haptic rendering of rigid bodies, and to highlight their limits. The list of dynamic engines evaluated below is not exhaustive, but we tried to choose the dynamic engines that seemed the most promising to us. Moreover, we defined an environment for our experiments that is modular enough to be able to integrate any physical simulation library.

4.3.1.1 Selected rigid body engines

We selected the following popular rigid dynamic engines for the evaluation:

- **Havok physics** (`http://www.havok.com`). Havok was created in 1998 in Dublin (Ireland), and is the world leader provider for rigid body dynamics solution. It also implements sophisticated collision detection algorithms, ragdoll animation, vehicles dynamics and character animation toolkit.

- **NVidia PhysX** (`http://www.nvidia.com/`, section PhysX in technology menu). PhysX (former NovodeX) has been created in 2002 by the semi-conductor company Ageia that was the first to propose on the market a hardware acceleration solution called PPU (Physic Process Unit) for physical simulation. NVidia PhysX implements collision detection algorithms, rigid, soft body and fluids simulation. PhysX has recently been acquired by NVidia (in February, 2008) to become NVidia PhysX (the hardware acceleration capabilities have been deported on the GPU).

- **Bullet physics** (`http://bulletphysics.org/`). Bullet physics is an open source physic engine founded in 2003 by Erwin Coumans, a former Havok employee and is supported by Sony Entertainment division. Like its competitors, Bullet physics provides collision detection and rigid body simulation. It also implements soft body simulation.

- **Open Tissue** (`http://www.opentissue.org/`). OpenTissue is born from a research project initiated by Kenny Erleben [Erleben 07]. He presents in his papers inviting algorithms for rigid body simulation. OpenTissue integrates collision detection system and simulation of rigid and deformable bodies.

All these dynamic engines use Gauss-Seidel like iterative solvers to manage non-penetration and user-defined constraints, that are expressed at the velocity level, *i.e.* the solver will try to find impulses that prevent or correct the constraints violations. (OpenTissue performs a final shock-propagation stabilization step as presented by its author). They are all implemented at least on Windows, Linux, MacOSX, and, except for OpenTissue, Playstation 3, Xbox 360 and Nintendo Wii.

Our evaluation is focused on a comparison between the results obtained with the different libraries. We do not present technical comparisons of the methods used by the dynamic engines. Moreover, since neither Havok physics nor NVidia PhysX is open source, it remains difficult to have information on the underlying algorithms used for collision detection and constraints resolution.

4.3.1.2 Performance Criteria

We designed our experiments in order to measure the quality of haptic rendering through the three following performance criteria:

- **Computation time**. A stable and realistic haptic rendering needs high refreshment updates (see *e.g.* [Colgate 95]). It is commonly admitted that a haptic display that renders contacts and impacts between rigid bodies should be performed at about 1kHz, *i.e.* the computation time spent to compute the state of the world from time t_i to time t_{i+1} must be performed in less than one physical millisecond. In order to measure this criterion, we start a timer before the call to the simulation method of the dynamic engine (this call includes collision detection and constraints solving) and stop it after the end of the call.

- **Stability**. The stability of the simulation indirectly measures how laws of physics such as energy conservation are respected. Passivity of a virtual world [Colgate 95] (*i.e.* the fact that the world only dissipates energy) is a sufficient condition for ensuring its stability. Therefore, we made measurements of the variations of the total world energy to conclude on its stability.

- **Accuracy** The accuracy indicates how well the physical phenomena (such as dry or sliding friction, bouncing, ...) are reproduced in the virtual world. To give a mark for spatial accuracy, we performed spatial measurements of penetrations distances (using Euclidean distance between bodies' centers as metric). We also visually appreciated the results based on reference simulations of the real world.

For each test case presented in the next section, we measure the average and maximum computation times, the sum of the total energy of all the bodies of the world, and we give a mark on the visually appreciated end state. If $\mathcal{B}$ is the set of bodies' indexes of the world, m_i, v_i, ω_i and J_i are respectively the mass, linear velocity, angular velocity and inertia matrix (related to the center of mass) of body i, the energy e of the world is computed as:

$$e = \sum_{i \in \mathcal{B}} \frac{1}{2} \left(m_i \, v_i^2 + \omega_i^T J_i \omega_i \right) \tag{4.3}$$

4.3.1.3 Tests Cases

We used four discriminant test cases for the measurements of our performance criteria (the words in italic facing the name of the tests indicate which main criterion is measured through the test):

1. **Pile of 50 cubes** – *stability.* The classical pile of cube test (Figure 4.13a) is a challenging structure because of its contact disposition: naive iterative solvers have a very slow convergence rate in order to propagate the non-penetration constraints [Milenkovic 01]. Also, this test measures the efficiency of the error correction due to interpenetration occurring.

2. **Seven-stages card house** – *friction accuracy.* The card house (composed of 89 cards, Figure 4.13b) is a structure that fully depends on an accurate simulation of friction phenomena [Kaufman 08]. If the friction is too much approximated or enforces penetrations, the card house is destabilized and collapses.

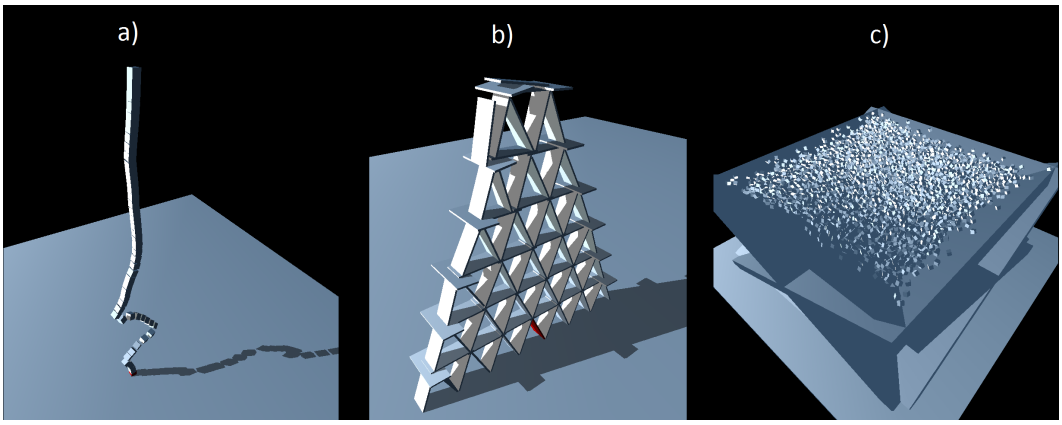


Figure 4.13 – Snapshots of three of the four test cases. **a**: A pile of 50 cubes. **b**: A 89-cards, 7 stages card house. **c**: 8000 cubes in a basin.

3. **8000 cubes in a basin** – *computation time.* In order to check the scalability of the libraries, the third test consists in dropping 8000 cubes in a basin (Figure 4.13c). A lot of objects are put in a high contact configuration (8 contacts per body in average), measuring the evolution of the timings of the solvers when the numbers of bodies and contact increase.

4. **Heavy block on a light block** – *efficiency of solvers.* This test puts Gauss-Seidel like solvers into slow convergence rate. If not enough iterations are used, or the correction methods are inappropriate, the upper heavy block penetrates the light one, and the system becomes unstable.

4.3.1.4 Test Parameters

Since our tests are related to rigid body simulation, we retained two physical parameters:

- • **The coefficient of friction**. According to the Coulomb model of friction, the coefficient of friction μ tells that the maximum tangential force that can be applied to oppose tangential motion at a contact point between two rigid bodies, is $\mu \times f$, f being the magnitude of the normal force acting at the contact point to prevent interpenetration. As an example, the coefficient of friction between two dry pieces of steel is approximately 0.15. Although friction is a very common phenomenon, it is far from simple to include accurately into rigid body dynamics [Baraff 91]. Thus, we vary the coefficient of friction from 0 (no friction) to 1 (high friction) for each test case in order to see its influence on the obtained results, and how well the dynamic engines handle it.

- • **The coefficient of restitution**. The coefficient of restitution indicates what percentage of energy is conserved and dissipated during an impact between two rigid bodies (0 means a total inelastic impact, while 1 means a perfect elastic and bouncy impact). In contact resolution between rigid bodies, resting contacts are often separated from collision events and are treated using different algorithms. To decide whenever a resting contact or a collision occurs, the relative velocity between colliding bodies at contact point is often considered. Since the restitution coefficient has an influence on how relative velocities are modified, we chose to study the influence of the restitution coefficient on the simulation results.

All our tests have also been performed by varying the time step. Depending on the integration scheme used, the time step has namely a great influence on the stability of the constraints solver. The different combinations of parameters are summed up on Table 4.4.

Friction coefficient	Restitution coefficient	Time step (s)
0.5	0.4	1/60
0.5	0.0	1/60
0.5	0.4	1/100
1.0	0.4	1/60
0.0	0.4	1/60

Table 4.4 – The different configurations of parameters used for each test case.

Configuration of the dynamic engines

Each dynamic engine has its own parameters that may have impacts on the results. For example, it is possible to set the number of iterations used for constraint solving. Since the default configuration of the dynamic engines is set to optimize a trade-off between computation time and accuracy performances, we chose to let the default values. However, other aspects such as aggressive freezing strategies can alter the results. Another aspect is the collision detection system that can be turn into continuous or discrete mode, having great impacts on computation time and results. To make the comparison possible, we disabled freezing and continuous collision detection for all the results presented in this paper.

4.3.1.5 Results

This section summarizes the results obtained for each of our performance criteria, for the four test cases previously defined. We believe that the collision detection system of OpenTissue slows down the simulation times drastically, and noticed that the stability of the simulation is lower than the other libraries. Therefore, we do not present the results obtained with OpenTissue since they are not comparable to the results obtained with the other dynamic engines.

Computation times comprise both the collision detection, constraint solving and integration time. We performed all our tests on an Intel Pentium D (3.40 GHz) with 2.0 GB RAM on Windows XP. Except the third test for which we measured the results after 10 seconds of simulation (an arbitrary chosen representative value), we stop the simulation and the measurements whenever the simulation has reached a stable state.

Pile of cubes

Havok physics brought the best results for the simulation of the pile of cubes. We obtained the best average computation times, and the best stability. It is possible to make the pile to stand up for time step going up to 1/60s. Figure 4.14a shows the dissipation of the total energy that led to a stable state for the pile of cube using Havok physics and a time step of 1/100s.

With NVidia PhysX, we notice a visible penetration between cubes at the beginning of the simulation, followed by a counter reaction that destabilizes the pile which breaks before 5 seconds of simulation if a time step over 1/100s is used. Using tiny time steps (less than 1/800s), it is possible to make the pile to hold, but it never reaches a stable state, and small oscillations can be observed.

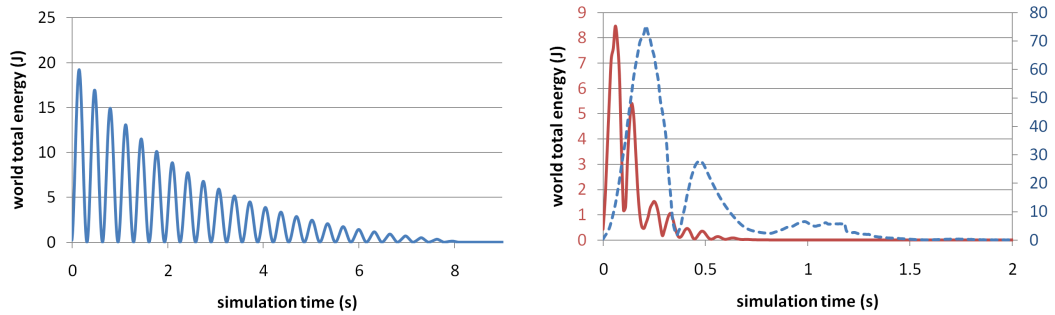


Figure 4.14 – Plot of the total energy over time of the pile of cube test. **a**: Pile of cube test with Havok physics. **b**: Card house test with Havok physics (red solid line) and NVidia PhysX (blue dashed line). Time step is 1/100s.

Bullet physics brought not as good computation time results as its competitors, and had similar results than NVidia PhysX on the stability plan (we did not manage to make the pile hold more than 4 seconds of simulation for time steps over 1/100s).

Using a null coefficient of friction, none of the engine enables to make the pile hold. However, in this case, we obtained the best results with Havok physics that maintains the pile for more than 4 seconds of simulation against 1.5 seconds for NVidia PhysX, and less than 1 second for Bullet physics.

Card house

Havok physics enables to simulate the card house in a visually realistic manner, and with the best computation times. With a time step of 1/100s, and a friction coefficient of 1, even the top most stage of the card house remains in place. NVidia PhysX shows the same phenomenon as for the pile of cubes: after a penetration between the cards, a counter reaction occurs (see Figure 4.14b at time 0.5s) and the card house is destabilized. However, although the card house is maintained for a long time, it never reaches a stable state before the card house is almost completely broken (after more than 50 seconds of simulation). Bullet physics shows a weakness on this test case: its default solver cannot handle properly dry friction. Indeed, whatever the coefficient of friction used, the cards slide and the whole house is broken at the beginning of the simulation.

8000 cubes in a basin

On this test case, we mainly measured the average and maximum computation times. For this simulation, we saw that Havok physics and NVidia PhysX brought very similar results. We noted however a slightly greater maximum computation time for Havok physics when the coefficient of friction is zero. Bullet physics is on the third place, with the highest computation times.

Figure 4.15 shows the computation time evolution with respect to the number of bodies, and the number of contacts in the virtual world. On this chart, we separated the call to the collision detection module and the constraints solving, enabling to show the time spent for collision detection (solid line) and the time used for constraint solving (dashed line).

Heavy block on a light block

None of the dynamic engines has been able to simulate visually plausible results for this test whenever the mass ratio between the two bodies is above 500 (see Figure 4.16, white body is

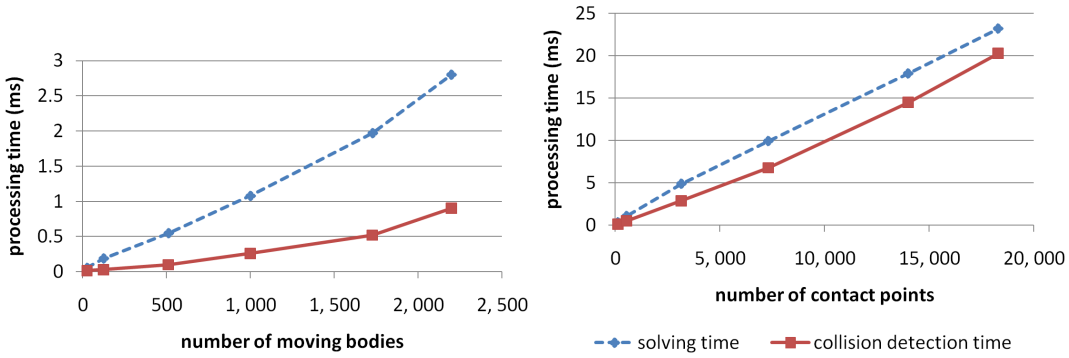


Figure 4.15 – Complexity of the scene versus computation time. **a**: number of bodies and computation time. **b**: number of contacts and computation time.

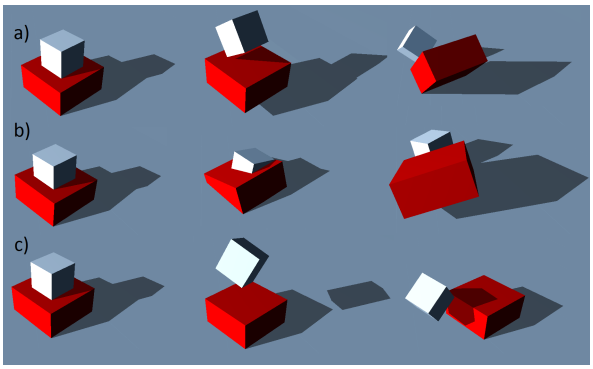


Figure 4.16 – Fourth test case. A heavy cube (white) is dropped on a light one (dark red) with a scale factor of 1000 on the masses. **a**: results with Havok physics. **b**: results with NVidia PhysX. **c**: results with Bullet physics. Time step = 1/800s, friction coefficient = 0.5, restitution coefficient = 0.4.

heavy while the dark one is light). Actually, using a time step of 1/1000s and masses equal to 1 and 500, it is possible to reach a stable state with Havok physics where the heavy cube is resting on the light one, after several unnatural bounces. We did not achieve to have the same state using NVidia PhysX or Bullet physics with this mass factor. NVidia PhysX has a different behavior, allowing the heavy cube to penetrate the light one (see Figure 4.16b, middle). This makes the light cube in red becoming unstable, being randomly shook until it is ejected away from the heavy cube that does not move. With Havok or Bullet physics, the interpenetration is corrected in such a manner that the heavy cube (and the light one for Havok) bounces abnormally over the light one, until both cubes are separated by tangential forces. Using a bigger time step (1/60s), the heavy cube penetrates the light one, and the latter is pushed away horizontally.

4.3.1.6 Summary of the Evaluation

We performed four discriminant tests, each of them was designed to measure one of the performance criterion. Figure 4.17 sums up the average timings (over all the configurations of parameters of Table 4.4) for each of the three first test cases (we do not show timings for the fourth test, since the scene composed of two bodies is not relevant for computation time comparisons).

We noticed that Havok physics has a small advance on computation time performance on NVidia PhysX, while Bullet seems to be less optimized. In average, we measured that it takes about 0.012 ms to solve for one cube in a contact configuration of 8 contacts per cube

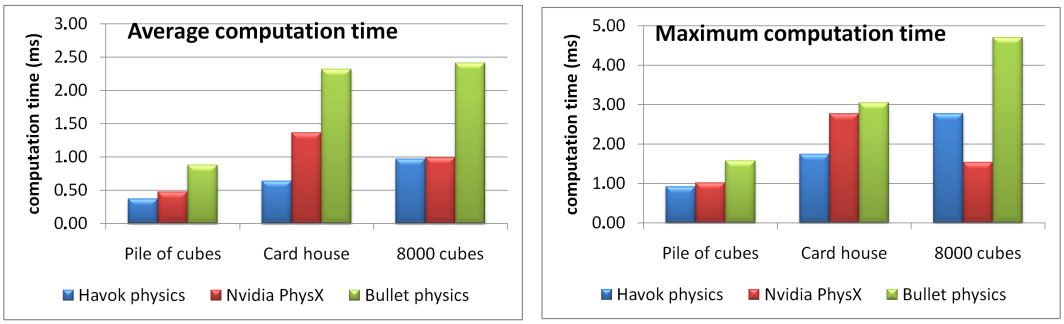


Figure 4.17 – Average and maximum processing time for the three first test cases.

for Havok physics or NVidia PhysX, with our hardware configuration.

Table 4.5 sums up the accuracy conclusion. We noticed that Havok leads to the more stable and accurate simulation, allowing simulating the pile of cubes and the card house successfully. NVidia PhysX shows some weaknesses on those challenging structures, allowing an initial interpenetration that destabilizes the simulation further. Bullet physics does not seem to handle properly dry friction, making structure such as the card house impossible to be correctly simulated.

	Pile of cubes	Card house	Heavy/light block
Havok physics	++	++	0
NVidia PhysX	+	+	0
Bullet physics	+	0	0

Table 4.5 – Summary of the stability and accuracy appreciations. Legend ++: good accuracy, stable simulation. +: visually realistic results, stability reduced. 0: simulation not successful.

4.3.2 Coupling rigid body engines and haptic rendering

In this section, we focus on how it is possible to couple the two main components of our haptic application, *i.e* how to couple the dynamic engine with the haptic rendering API, in order to obtain a modular and generic system. For that purpose, we introduce a scaling interface between the two components that in order to tune the haptic display without altering the overall stability of the system.

We define our coupling with the assumption of admittance controlled haptic device. Indeed, this control mode is well-suited for a simple coupling between a dynamic engine and a haptic rendering API, because the dynamic engines provide successive positions of the proxy (the body in the virtual world which is coupled for haptic interaction) over time, that can be used to pilot the haptic device. The forces returned by the haptic device can be applied on the proxy and integrated into the dynamic engine. Also, admittance control can be emulated from an impedance-controlled device using a virtual coupling, as presented in section 2.6.3.2. Figure 4.18 shows the flow of data between the simulation and the haptic control when using admittance mode.

It is however too restrictive to plug directly the simulation results to the haptic API as shown on Figure 4.18. Indeed, values returned by the dynamic engine may have a numerical representation different from the one used by the haptic API, the frame coordinates can be different, and the order of magnitude of the values used are likely to be incompatible. Moreover, haptic display must be tunable: changing the range of efforts allowed and calibrating

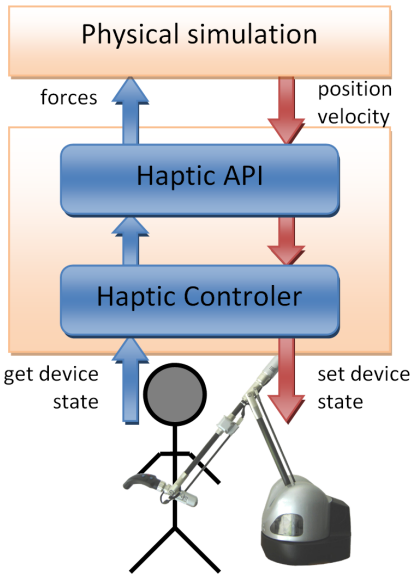


Figure 4.18 – Admittance control of a haptic device. Haptic rendering refers to the haptic API, the haptic controller and the haptic device.

the interaction so that most of the user's movements stay in the haptic device working space are two manipulations that should offer haptic applications. These observations motivated us to define a scaling interface between the two modules as presented in the next section.

4.3.2.1 Scaling Factors between Virtual and Real World

Each haptic device has its own mechanical constraints and working space. The haptic device used during our experiments (a 6 DOF Virtuose™6D35-45, from Haption, Soulge sur Ouette, France) supports less than 40 Newtons of effort, and has a workspace which is approximately a 30cm side cube, while a PHANTOM Omni haptic device from Sensable (Woburn, MA 01801, USA) has a $16 \times 12 \times 7$cm working space, supporting around 3.3 Newtons of effort. Of course, the two devices are not designed for the same applications, but their differences highlight the fact that the physical simulations values cannot be directly used for haptic rendering. Moreover, we want to be able to manipulate objects of a mass of several tons in huge environments as well as interacting with microscopic objects.

It is possible to amplify the movements of the user, and to give him more force so that the interactions fit into the working space of the haptic device and its range of allowed efforts. However, altering the coupling has consequences in both the simulation and the control of the admittance haptic device and the physical simulation. In this section, we propose an interface between the physical simulation and the haptic rendering that considers the two modules as black boxes, and that enables to freely give values for scaling factors, without altering the stability of the system to obtain a generic coupling.

In the following, the subscript x_{simu} denotes a value x coming from the physical simulation module, while the subscript x_{hapt} denotes a value x coming from the haptic rendering module.

The scaling factor

We define the scaling factor z_{fact} as the factor linking the virtual world unit of length (denoted v_l) to the real world unit of length, the meter. For example, a factor of 10 means that 1 virtual unit of length (*i.e.* the value 1 in the numerical representation of the length in the dynamic engine) is equivalent to 10 meters in the real world. This factor is useful for example to

choose a value for the gravity of the virtual world that has the same representation than on the earth (*i.e.* around $98 v_l.s^{-2}$ for an earth equivalent gravity if $z_{fact} = 10$). Also, this factor can be helpful when creating the virtual world, and computing the mass of the objects from their density in $kg.m^{-3}$ to the virtual world density in $kg.v_l^{-3}$. The scaling factor has only an influence on the physical simulation (and the conception of the world) and thus does not belong to the scaling interface between the physical simulation and the haptic rendering. However, it is really helpful to cleanly separate the scaling interface, and to avoid mixing independent and linked variables into the coupling.

Since the physical quantities used for haptic rendering are related to space, mass and time dimensions, we define additional length and mass factors as presented in the following.

The length factor

We define a length factor s_{fact} that modifies the unit of length of the virtual world v_l so that the simulated scene fits into the desired working space. To ensure the good separation of the systems of units of the physical simulation and the haptic rendering, we use this factor to modify the linear position x_{simu} in meter, the linear velocity v_{simu} in $m.s^{-1}$ and the linear forces f_{simu} in $kg.m.s^{-2}$ linearly, while the torque τ_{simu} in $kg.m^2.s^{-2}$ and the inertia matrix I_{simu} using the square of the factor:

$$x_{hapt} = \frac{x_{simu}}{s_{fact}} \quad ; \quad v_{hapt} = \frac{v_{simu}}{s_{fact}} \tag{4.4}$$

Concerning the orientation values q_{simu} and angular velocity ω_{simu}, it would be possible to similarly define a rotation factor, that could amplify or decrease the angular motion. We however kept the original values for orientation and angular velocities (the angular quantities have no unit, and do not scale with length or mass):

$$q_{hapt} = q_{simu} \quad ; \quad \omega_{hapt} = \omega_{simu} \tag{4.5}$$

The mass factors

We finally define the force factor f_{fact} and torque factor τ_{fact} that increase or decrease mass values. The force factor modifies linearly the force returned by the haptic device f_{hapt} and the mass m_{simu}. The torque factor modifies linearly the torque τ_{simu} and the inertia matrix I_{simu}:

$$f_{simu} = f_{hapt} \times f_{fact} \times s_{fact} \quad ; \quad m_{hapt} = \frac{m_{simu}}{f_{fact}} \tag{4.6}$$

$$\tau_{simu} = \tau_{hapt} \times \tau_{fact} \times s_{fact}^2 \quad ; \quad I_{simu} = \frac{I_{hapt}}{\tau_{fact} \times s_{fact}^2} \tag{4.7}$$

Figure 4.19 sums up the scaling interface between the physical simulation module and the admittance haptic rendering module.

4.3.2.2 Synchronization with Physical Time

Although length and mass factors can be easily interpreted, time factor is more confusing. We want the virtual unit of time to coincide with the real second. Therefore, we perform an active wait at the end of each simulation step using the system clock, enabling to synchronize the simulation time with physical time (see Figure 4.20). To be feasible, this solution implies that the computation time needed to move the world n seconds forward is smaller than n

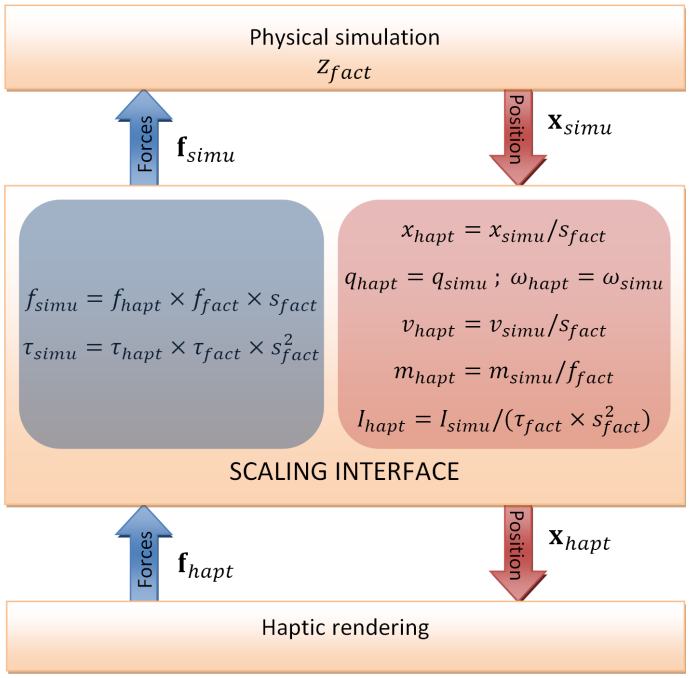


Figure 4.19 – Scaling interface between the physical simulation and the admittance haptic rendering. Bold values depict global state vector or global effort vector.

physical seconds. Otherwise, the simulation takes some latency. To avoid this latency, the number of bodies in the virtual world can be limited. Indeed, if we master the time needed to simulate one body in the worst case (see the conclusion of the evaluation section), then we can estimate the maximum number of bodies that it is possible to simulate within a given time step in order to avoid the simulation to take some latency.

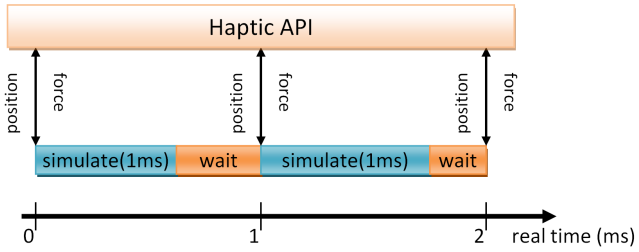


Figure 4.20 – Synchronization with physical time using active wait.

4.3.2.3 Results

We implemented the presented coupling using an admittance capable haptic device (a 6 DOF Virtuose™6D35-45, from Haption). In order to avoid the visual display to slow down the haptic frequency, we performed virtual world image synthesis into a separated thread that performs read-only operation on the virtual world (and is not synchronized with the simulation thread).

We experienced stable haptic interactions using Havok physics, NVidia PhysX and Bullet physics. The more pleasant and accurate interactions have been obtained with Havok physics, where we could feel dry and sliding frictions performing up to 1kHz haptic rendering for the pile of cube, and nearly 800Hz haptic rendering for the card house test.

Comparable interactions have been obtained with NVidia PhysX. In simple cases, fine interactions with dry and sliding frictions can be performed. However, interactions involving stacking structure such as the pile of cube and the card house, where visual oscillations appear, do not spare the haptic display that suffers from the same artifact.

Finally, we also performed satisfying interactions using Bullet physics. The main difference between the two previous haptic displays is the friction phenomena. Indeed, the tangential motion is felt viscous, and no clear dry friction is haptically reproduced.

4.3.2.4 Discussion and conclusion

The scaling interface enables us to easily tune the haptic display using few factors, but can also manage changes in frame coordinate or numerical representation interface. We presented simple linear or quadratic scaling functions for physical quantities, based on unit analysis. These functions have the advantage of isolating two systems of units, and thus conserve the original stability. The definition of other scaling functions must be carefully done in order to avoid loss of stability leading to unstable situations in haptic rendering.

For the purpose of our evaluation, we defined a multithreaded test environment and a physic abstraction layer in order to have a full control on the parameters and the measurements. However, the abstraction layer and test environment are steps that can be skipped for general haptic rendering. Using a multithreaded environment is common in haptic rendering, to avoid loss of time for external tasks such as visual rendering. Although we do not detail this aspect in this paper, it does not invalidate the scaling interface, which is proper to haptic rendering.

The list below sums up the steps that quickly and simply lead to a stable and modular haptic application:

1. Choose a dynamic engine, or an abstract layer that includes several dynamic engines such as PAL (`http://www.adrianboeing.com/pal/`) which embeds about ten dynamic engines in a unified interface.

2. Choose a haptic device that is well-suited for the application and that can be controlled in admittance mode (an abstraction layer for haptic device control would be even more elegant).

3. Define the scaling interface presented in the second part of the paper (and optionally the synchronization time procedure), and plug it to the dynamic engine and the haptic API as shown on Figure 4.19.

4.3.3 Dealing with a growing number of bodies and haptics

In order to haptically render stiff contacts between rigid bodies, a high refreshment frequency must be maintained in order to ensure a stable and accurate interaction [Colgate 95]. Therefore, it is often considered to create two processes in haptic rendering applications: one for the physical simulation, and another one for the haptic rendering. The simulation process extracts at its rate a so-called intermediate representation [Adachi 95] from the virtual world. This intermediate representation is a simplified and local model of the world that is exploited by the haptic process to generate orders at frequencies allowing good haptic rendering quality.

There are different models used for intermediate representation in the literature. Some authors proposed using the Jacobian of forces with respect to a displacement of the tip of the haptic device [Cavusoglu 00]. Another approach consists in extracting a simplified geometry

from the virtual world, around the interaction area. The nature of the geometry can be a single plane [Adachi 95], a set of planes [Mark 96], parametrized surfaces [Balaniuk 99], set of nearest triangles [Mendoza 00], or other data for collision detection [Davanne 02]. Finally, multiresolution approaches have been proposed for deformable bodies, where parts of the mesh that are close to the tip of the device are extracted, and simulated at different rates using different levels of details [Astley 97]. A linearized version of deformation simulation can also be used in the haptic process [Cavusoglu 00]. More recent work has been proposed in [Otaduy 05] to deal with haptic display of bodies having complex geometries. The geometry is dynamically simplified in a sensation preserving way around contact points.

Most of the methods have been developed in order to allow haptic interactions with deformable bodies. However, contacts between rigid bodies are simulated using different models, and need higher haptic frequency. Therefore, the methods proposed for deformable bodies cannot be applied to rigid bodies. In this section, we present a multiresolution approach for haptic rendering between rigid bodies. The main idea of our new coupling scheme is to dynamically extract a subset of the global world and to build a second physical world as an intermediate model. We call this second physical world the *haptic sub-world*. As presented in Section 4.3.3.1, the haptic sub-world is built from a limited number of carefully selected bodies, and can therefore be simulated at a higher frequency into the haptic process, as depicted on Fig. 4.21. The results presented show that our coupling scheme enables to increase the complexity of the global world without any perceptible alteration of the haptic display.

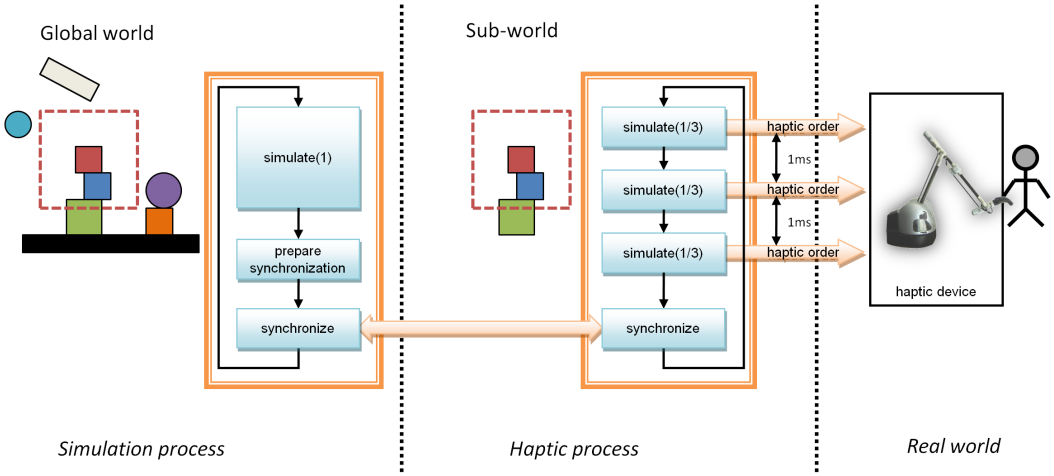


Figure 4.21 – Haptic-sub world main algorithm. The simulation process simulates the global world once with a time step of dt, while the haptic process simulates for example 3 times its haptic sub-world using a time step of $1/3 \times dt$ (we call the ratio between the time steps the *simulation ratio value*, noted r_{sim}). At the end of the haptic cycle, the two worlds are synchronized, exchanging bodies positions and efforts information.

4.3.3.1 The Graph-based Haptic Sub-world Coupling Scheme

We define the haptic sub-world as a subset of the bodies of the world that is simulated at a frequency allowing a good haptic rendering. We noticed that during the interactions, only the bodies that are directly or indirectly in contact with the proxy (the body linked to the haptic device) can influence it. Therefore, we define the haptic sub-world based on the graph of contacts between the bodies. Let us define β, the set of bodies of the global world and $p \in \beta$ represents the proxy. If we suppose that we have a function called $contact : \beta \times \beta \rightarrow \mathbb{R}$ that provides the number of contacts occurring between a pair of bodies, then we can define

the graph $\mathcal{G} = (\beta, \mathcal{V})$, where an edge $v = (b_1, b_2)$ from the edges set $\mathcal{V} \subset \beta \times \beta$ exists only if there is a direct contact between b_1 and b_2, i.e.:

$$\forall\, b_1, b_2 \in \beta,\ contact(b_1, b_2) \geq 1 \Rightarrow (b_1, b_2) \in \mathcal{V} \tag{4.8}$$

From the graph $\mathcal{G}$, we define $\mathcal{H} = (\beta_h, \mathcal{V}_h), \beta_h \subset \beta, \mathcal{V}_h \subset \mathcal{V}$ as the connected subgraph of G that contains p (the proxy). Fig. 4.22 shows an example of how the graph $\mathcal{H}$ is obtained.

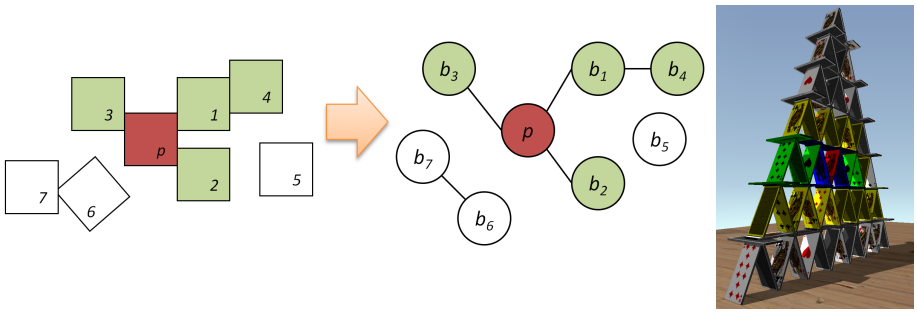


Figure 4.22 – Graph-based haptic sub-world. The graph $\mathcal{G}$ (middle, containing all nodes) defines the contact configuration of the bodies of the global world. The colored connected components of the graph containing the proxy (in red) form the graph $\mathcal{H}$ (containing only colored nodes) of the bodies that compose the haptic sub-world. The card house (right) illustrates the haptic sub-world composed of colored cards extracted from the proxy (red card).

In practice, we only need the graph $\mathcal{H}$ (the connectivity information of the other bodies in $\mathcal{G}$ is useless for our purpose). Therefore, we designed an algorithm that progressively builds the graph $\mathcal{H}$ over the simulations, starting from the proxy and the bodies that are directly in contact with it.

From this definition, a body can belong to the haptic sub-world only if it has a (direct or not) contact with the proxy. However, a haptic cycle is needed before a body is added to the sub-world. Therefore, we anticipate all potential contacts by bringing bodies close to the proxy even if there is no contact with it (Paragraph 4.3.3.1). Also, we must limit the complexity of the haptic sub-world to preserve the haptic frequency (see Fig. 4.22, card house). However, if the graph is limited, it may remain contacts between the two worlds, and we must manage the exchange of energies to avoid loss of physical plausibility (Paragraph 4.3.3.1).

Building the Haptic Sub-world

A body close to the proxy is a body that can be reached by the proxy in less than two haptic cycles. Indeed, the linear velocity of the proxy has a limit v_{max} defined by the mechanical constraints of the haptic device. For a given fixed haptic frequency rendering f, and a chosen simulation ratio value r_{sim}, the maximum displacement p_{max} of the proxy during two haptic cycles is $p_{max} = \frac{r_{sim}}{f} \times v_{max}$. Assuming that the size of the proxy bounding cube side is c_{size}, then we define a boundary cube with a side size of $c_{size} + 2 \times p_{max}$ centered around the proxy that represents the limit of close bodies. All the bodies intersecting the boundary box are added into the sub-world to avoid the loss of physical information. We use broadcast collision detection to detect all bodies bounding boxes which are intersecting the boundary cube. During simulation of the global world, events are triggered, warning that a new body has entered the boundary box, or a body that was inside left it. We keep all the events in a list. At synchronization time, we check the leaving/entering list of bodies to update the haptic sub world, and we use the position of the proxy to re-center the boundary cube.

Moving the boundary cube may generate new events that are added to the list and are taken into account at the next synchronization.

Building the Graph and the Interface

As previously mentioned, using a haptic sub-world with a small number of bodies enables to perform a parallel simulation at high rates for the haptic rendering concerns. However, it is possible that the haptic graph $\mathcal{H}$ has too many nodes (we may have $\mathcal{H} = \mathcal{G}$, the global graph), and in that case, the sub-world loses its sense. To ensure a fixed haptic frequency, we limit the propagation of the graph to a maximum number of bodies. We define an interface that manages the energy exchanges between the global world and the haptic sub-world when the graph is limited. Three different exchanges are controlled.

First, the efforts coming from the global world. We define border bodies as bodies of the haptic sub-world having a contact with a body of the global world (*e.g.* the yellow cards on Fig. 4.22). During the simulation of the global world, we store non-penetration impulses applied at these contacts and apply them on border bodies at each simulation step of the haptic cycle. On a pile of cube at rest, this constraint enables to have the same static behavior for the bodies of the haptic sub-world and their equivalent bodies in the global world, as shown in Figure 4.23.

Second, the friction on border bodies. To include friction at border, we create a "friction point" constraint at each border contact where an impulse is applied. This constraint imposes the body point to stay along the contact normal line. If a big enough tangential effort is applied, the constraint is freed. The threshold is determined using Coulomb friction model. Namely, knowing the normal impulse magnitude i applied at border contact and the coefficient of friction μ at that point, the intensity threshold used for the perpendicular effort is $\mu \times i$. Sliding friction is not applied, but approximated by the fact that the friction point constraint is updated at each synchronization.

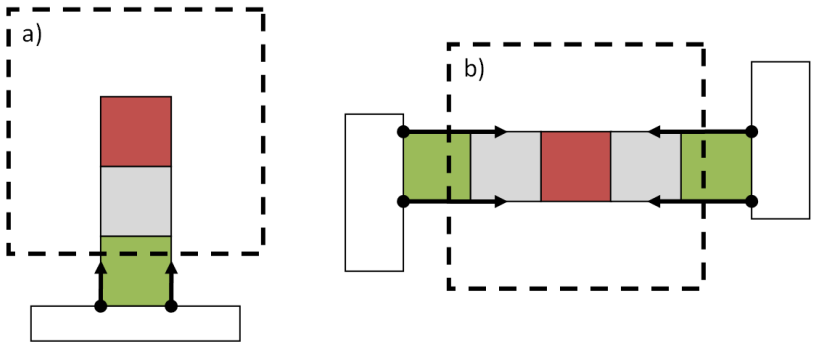


Figure 4.23 – Handling of the effort exchanges at the boundaries of the haptic sub-world. By applying appropriate retrieved impulses at border contacts, we can have helpful information on the surrounding global world **a**, or conserve compression information of the world, that allows simulating the same friction or resistance behavior as in the simulation world **b**. The green bodies (at which border contacts are present) are border bodies.

Third, the efforts coming from the haptic sub-world. In order to propagate the normal actions coming from the haptic sub-world to the global world, we simply impose the positions of the bodies of the haptic sub-world to their equivalent body in the global world. For tangential efforts, we store at each friction constraint the sum of impulses applied to maintain the constraint, and apply them on each body of the global world that has a contact with border bodies.

4.3.3.2 Results and Evaluation

Our tests and experiments have been performed on PC running on Windows XP, equipped with 2 processors Intel Pentium D (3.4 GHz) and 2 Go RAM. The haptic device used is a 6 DOFs Virtuose 6D35-45 (Haption company, France). The graphic card for display is an NVidia Quadro FX 3450. We used Havok physics software (`http://www.havok.com`) for collision detection and rigid body simulation (the solver complexity is $\mathcal{O}(n)$, n being the number of constraints). We used a third process to manage visual rendering in order to increase computation time performances.

Same Haptic Frequency for more Complex Scenes

The simulation process has r_{sim} haptic periods of time to simulate its world and prepare the synchronization (see Fig. 4.21). Namely, the number of bodies allowed into the global world is approximately equal to the number of bodies allowed into the haptic sub-world multiplied by r_{sim}. For example, on our configuration, using a value of 10 for r_{sim} enabled us to perform satisfying haptic rendering in virtual worlds containing more than 500 cubes at a frequency of 1kHz. Without our method, this frequency can be maintained only for scene containing less than 50 cubes.

Accuracy Measurements

We performed accuracy measurements based on the velocity of the bodies on scenes containing up to 250 bodies, and about 2000 contacts. We measured that the sub-world method generates a loss of energy of 6% in the worst case (compared to the simulation without sub-world), using a simulation ratio value of 8. This loss of energy is due to the approximations and filters applied during the haptic cycle in order to avoid energy gain, that could lead to instability. Even if these approximations increase with r_{sim}, we experienced that the haptic feedback remains the same for the user, even with high r_{sim} values (> 10). Figure 4.24 shows an analysis on the velocities of our coupling method.

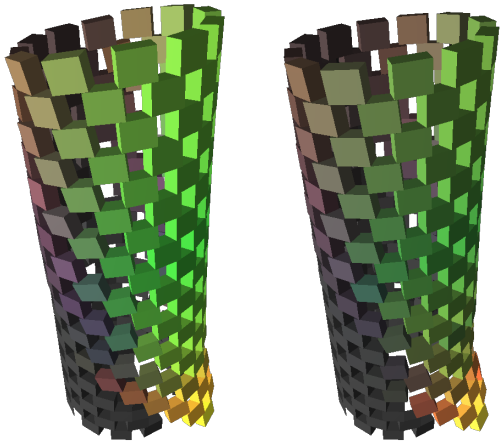


Figure 4.24 – Analysis of the accuracy on velocities. Measurement of velocity accuracy on a complex structure (about 2000 contacts) and a pile of 50 cubes being pushed at the bottom, after 1 second of simulation. The color intensity is linearly dependent to the body velocity. The left picture is the result without our coupling scheme, and the right picture with it.

Comparison with Other Coupling Methods

We implemented and compared the haptic feedback obtained with our method to other coupling schemes between the physical simulation and haptic rendering: the direct coupling, the interpolation of position and a static sub-world coupling. With a direct coupling, a stable and accurate haptic rendering is obtained, but the complexity of the virtual world is limited. With the interpolation method, the efforts returned by the haptic device are stored during the haptic cycle, and are then applied all at a time during synchronization. This behavior produces artifacts and decreases the stability of the haptic feedback, even for small values of r_{sim} (lower than 4). Using a static sub-world, we no longer simulate the haptic sub-world: it is frozen and all objects are fixed. The position of the proxy is imposed into the simulation process at each synchronization. This produces a stable simulation, but the late integration of the action of the proxy produces annoying vibrations artifacts. Using our sub-world method, we can increase the complexity of the virtual world while avoiding the artifacts produced using the interpolation or the static sub-world methods.

4.3.3.3 Discussion and conclusion

We have presented a new coupling scheme based on a haptic sub-world principle with the use of a graph to manage the contacts between rigid bodies. The scheme allows coupling the physical simulation of rigid bodies and haptic rendering, and enables to increase the complexity of the virtual world without having perceptible loss in haptic rendering quality.

We also presented an interface that manages the exchanges of energy between the sub-world and the global world when contacts remain between the two worlds. With our coupling scheme, we have been able to perform 1kHz and satisfying haptic rendering in scenes containing more than 700 cubes with many contacts.

Future work will be focused on the use of our method as a tool for haptic rendering applications. We also project to validate it on user-based scenarized scenes. Also, we think it is possible to use our coupling scheme to perform several haptic displays together from the same global world without loss of computation time performance.

4.4 Chapter conclusion

This chapter treated two aspects present in interactive simulations: the collision detection, and the interaction with haptic rendering. In the context of brittle fracture, these aspects present unique challenges. For the collision detection, new bodies with arbitrary shapes and created at run-time must be handled, and existing solutions based on precomputed data cannot be applied. We presented a new collision detection algorithm based on a mesh-based distance field and a reconfigurable sphere tree that are both updated efficiently at fracture events. We also introduced a contact selection mechanism that allows avoiding the computational peaks due to the challenging configuration of the fragments at the first frames after the fracture events.

The results of our collision detection system coupled with our impact-based fracturing method presented in the previous chapter have shown to be fast enough to pretend to be used in interactive applications such as haptic rendering, opening doors for new applications. We proposed coupling our fracture simulation system to a haptic device. To this purpose, we first analyzed the possibilities of coupling of common rigid body engines to create haptic displays. Our results showed that a stable haptic feedback can be obtained from these engines, through a benchmark. We also presented scaling mechanisms that allow the tuning of the

mass and length factors between the virtual world and the real world without loss of stability. Finally, we proposed a new haptic coupling based on the extraction of a sub-world simulated at higher frequencies, that allows maintaining the needed simulation frequencies for haptic rendering even when new bodies are added to the scene, as it is the case in brittle fracture scenarios.

We demonstrated the usability of our fracture simulation method for interactive applications that requires high computation performances such as haptic rendering. Our method is stable and can be integrated to any third party rigid body engine to enrich it seamlessly with brittle fracture simulation.

Evaluation and validation

5

Contents

The evaluation and validation of the fracture simulation methods are rarely addressed. Indeed, validating brittle fracture with experiments present unique challenges. First, it is difficult to say whether a fracture pattern is valid or not, since the patterns are not reproducible in real life. What makes the fracture correct or not is not well defined in the literature. Second, as presented in section 3.3, in the case of impacting bodies, the dynamic aspects of the simulation play a key role in the fracture outcome. Although static cases are widely studied in fracture mechanics, the dynamic cases are poorly addressed because it is difficult in practice to measure the deformations and the formation of cracks in stiff materials.

We decomposed the validation of the fracture into two sub-validations. First, the validation of the crack propagation, or how the cracks propagate. Second,the validation the fracture opening (or fracture initiation), or checking whether and where the cracks start. These two independent validations are justified by the different needs of the brittle fracture application. For example, on a virtual prototyping application, knowing whether an object would break is the main feature required, and knowing how it breaks may be secondary. On the other hand, on application where the visuals are more important such as a game, the perceived realism of fracture pattern obtained will be more important than the correctness of the simulation.

In this chapter, we first propose validating the crack propagation aspects based on statistical information extracted from the fracture geometries. We define a metric based on this statistical information to define the visual similarity between two fracture patterns. To validate this statistical approach, we present a perceptive experiment based on a user study. This work has been done in collaboration with the university of Yale (Holly Rushmeier), the university of Girona (Carles Bosch), and the INRIA Sophia-Antipolis (George Drettakis). We then propose a subjective validation of the haptic perception of the fracture, and check if the haptic feedback is consistent with the simulation parameters. Finally, we present the preliminary results on the validation of the impact-based fracture methods, based on experiments of real objects fracturing.

5.1 Perception of the fracture pattern and fracture statistics

5.1.1 Statistical features of a fractured body

The statistical properties of fracture patterns have been analyzed in fields such as mechanics [Griffith 21], paleontology [Chowdhury 09] and other fields [Cloos 55, McDanels 06]. Their main goal is to use statistics to help to understand the reconstruction of fractured objects, based on shapes and adjacency information. In computer graphics, [Shin 10] recently analyzed fracture patterns observed in wall paintings, to improve fresco fragments matching [Funkhouser 11].

Our hypothesis is that fracture patterns that appear similar can be identified by comparing statistical information on the geometry of the patterns produced. To confirm this hypothesis, and to determine which statistics are most important, we performed a user study where human subjects were asked to choose between patterns matching different types of statistics, as presented in the following.

5.1.2 User-study on the perception of the fracture pattern

We gathered a list of potential statistics from previous work [Shin 10] and eliminated redundant measures. We finally considered three types of statistics:

- **Fragment statistics** ($S1$): number of fragments and distribution of the area of the fragments.
- **Crack statistics** ($S2$): number of cracks and distribution of crack the lengths.
- **Junction statistics** ($S3$): the density of junctions.

We performed a 2-alternative forced choice experiment. The interface of the user study can be seen in Figure 5.1. At the top is an image of a reference pattern manually extracted from a photograph. Each lower image is generated to match the top image for a subset of the statistics (e.g., S1 or S2+S3). A table of the images generated for the user study is presented in Figure 5.2. We used brute force optimization to determine parameters of the simulation. The users then had to compare the lower two images and choose the one that looked most visually similar to the real example. We used a "line-drawing" rendering style to avoid the use of other cues such as texture.

Experimental Apparatus

The user was instructed to select their preferred fracture pattern in terms of similarity to the real image by clicking on the preferred image. The participants had 7s to give their answer. Timeouts or errors were very rare (less than 1% of the total number of answers) and were ignored in the analysis. The participants had the option to take breaks by pressing a "Pause" button.

Population and Collected Data

20 participants (14 males and 6 females) aged from 23 to 50 (mean=31.82, standard deviation=7.67) performed the experiment. For each pair of conditions, we recorded the choices of the participants and their time to answer.

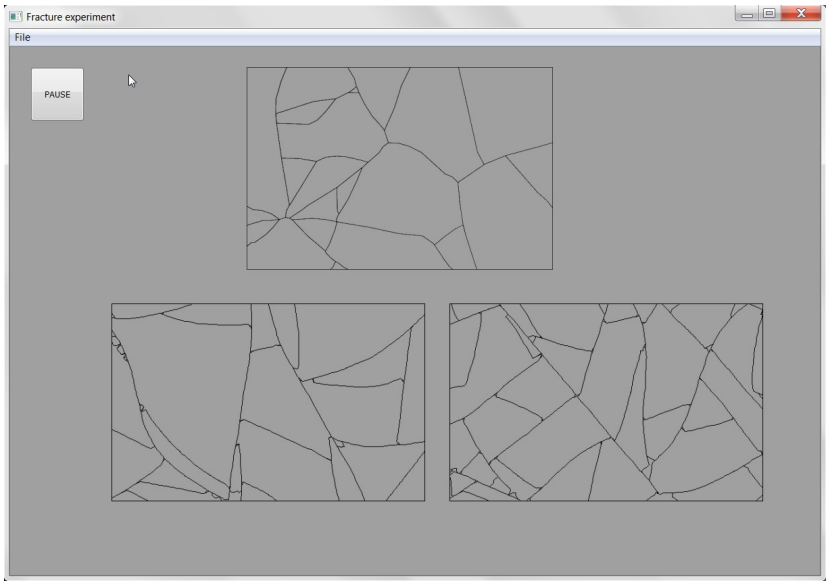


Figure 5.1 – The interface for the user study. Above we show the reference fracture pattern taken from an input photograph. Below, two patterns for which different statistics have been matched (Left: optimization matched the three statistics, Right: the optimization matched the crack length distribution). The user is asked to select the most visually similar pattern to the reference.

Experimental Conditions

We used a within-subjects design to evaluate seven different pattern conditions. The seven conditions corresponded to an image simulated by optimizing one of the three types of statistics or a combination of these statistics. The conditions were : (i) $S1$, (ii) $S2$, (iii) $S3$, (iv) $S1 + S2$, (v) $S2 + S3$, (vi) $S1 + S3$, (vii) $S1 + S2 + S3$. We had 5 reference images which are given in supplementary material. All the possible pairs of the different conditions were tested twice in both orders (i.e., left or right image). For each group of possible combinations, the order between the different pairs was randomized as well as the reference image. The experiment lasted approximately 20 minutes.

Results

We analyzed the answers of the participants for the different pattern conditions to determine which conditions gave the highest percentages of answers in terms of similarity between the fracture patterns of the real image and the simulation. Each individual performed 212 comparisons. Under the null hypothesis of equal preference between two conditions, the number of times an individual preferred the first condition follows a binomial distribution with parameters 10 and 1/2. After standardization, such a variable can be approximated by a standard normal random variable. Thus, for each pair of conditions, we tested the presence of a preferred condition using an exact binomial test. The p-values were adjusted with a Bonferroni correction. The percentages of time a condition is chosen are given in Table 5.1.

The analysis showed that $S1$ was significantly chosen more than all the other conditions, except the combination $S1 + S3$ ($p = 0.1$). $S2$ and $S3$ were significantly chosen less than all the other conditions (lines 2 and 3 of Table 5.1). No significant effect was found between condition $S1 + S2$ and $S2 + S3$ ($p = 0.15$), as well as between condition $S2 + S3$ and $S1 + S2 + S3$ ($p = 0.55$). We can conclude that $S2$ was always less significantly chosen than

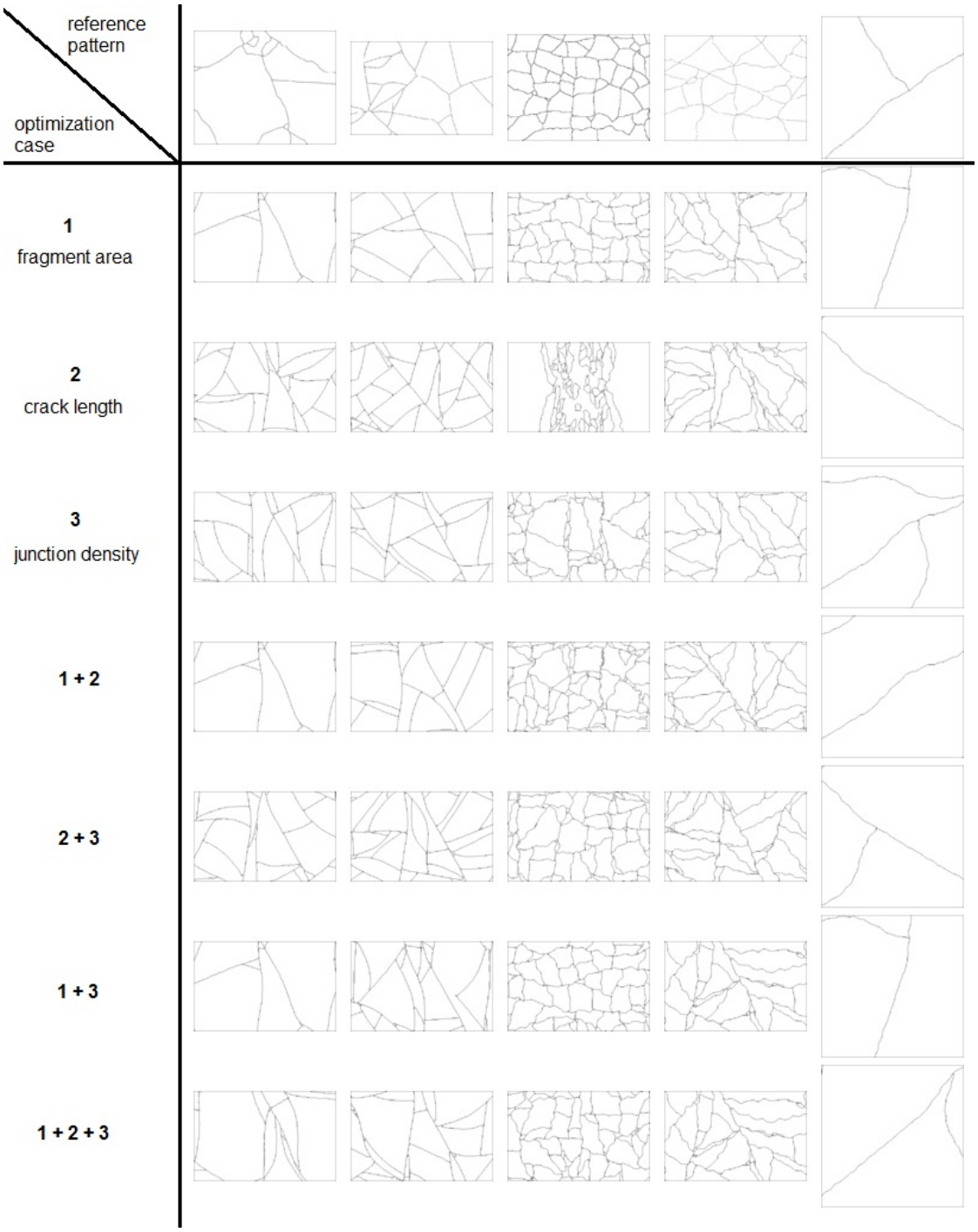


Figure 5.2 – Reference and simulated images generated for the user study.

all the other conditions, except for the conditions where $S2$ is included. Condition $S3$ gives the same results as $S1$ when combined with it.

These results indicate that matching fracture pattern statistics have a significant impact on the visual similarity of the resulting patterns. If pattern statistics were irrelevant, there would have been no significant effect when different pattern statistics were matched. Further, we conclude that fracture area (S1) and junction statistics (S3) appear to have a dominant effect compared to crack length and number (S2) in perceiving visual similarity of fracture

patterns.

The user study thus provides a strong indication of the validity of using statistics as a metric for evaluating visual similarity of fracture patterns. It also determines the relative importance of each statistic.

	$S1$	$S2$	$S3$	$S1+S2$	$S2+S3$	$S1+S3$	$S1+S2+S3$
$S1$		91	79.5	76	72.5	50	79
$S2$	9		27	24.5	32	22	24.5
$S3$	20.5	73		42	38.5	27.5	40.5
$S1+S2$	24	75.5	58		44.5	39.5	41
$S2+S3$	27.5	68	61.5	55.5		29	47.5
$S1+S3$	50	78	72.5	60.5	71		75.5
$S1+S2+S3$	21	75.5	59.5	59	52.5	24.5	

Table 5.1 – Percentage of instances that the statistic type for the row was chosen over the type for the column. For example, in the first line, patterns with statistic type S1 were selected in 91 percent of the instances where they were compared to patterns with statistic type S2 matched. The light gray values indicate that there was no significant effect found between the two conditions.

5.1.3 Optimizing the parameters of the simulation from images

We used the statistical measures to define a similarity metric between fracture patterns. This metric provides a distance between two input patterns, and can be used as a function to minimize into a minimization process. Namely, a set of statistics of a desired fracture pattern is be extracted from *e.g.* an image, and the optimization find a set of parameters for which that produces fracture statistics that are close to the input statistic pattern as illustrated in Figure 5.4.

Starting with a photograph of the desired pattern, the location of the cracks is first specified by the user through a standard graphics editor. The obtained pattern is then processed into edges, junctions and fragments, and the set of statistical properties is then deduced. The extraction of these statistics is illustrated in Figure 5.3, where the extraction process and the resulting statistics are presented for four different input images.

5.1.3.1 Parameters to be optimized

The input required for a crack simulation is a set of material properties (E, ν, ρ, R_{c_m}), an age parameter (i.e. total simulation time), junction density, the parameters that control the crack initiation and propagation ($d\sigma_e$, r_{relax}, v_{relax}, $R_{c_{var}}$, l) and the noise values A and f for the crack propagations.

The material properties are determined by specifying the material observed in the exemplar image. The junction density is computed directly from the exemplar. The age parameter is specified by the user, and the noise values are adjusted manually.

Range of Values for Optimized Parameters

We give a valid range for the optimization of each parameter (see Table 5.2) to guide the optimization. To scale with as many materials as possible, the range of possible values is expressed as a function of some known parameters.

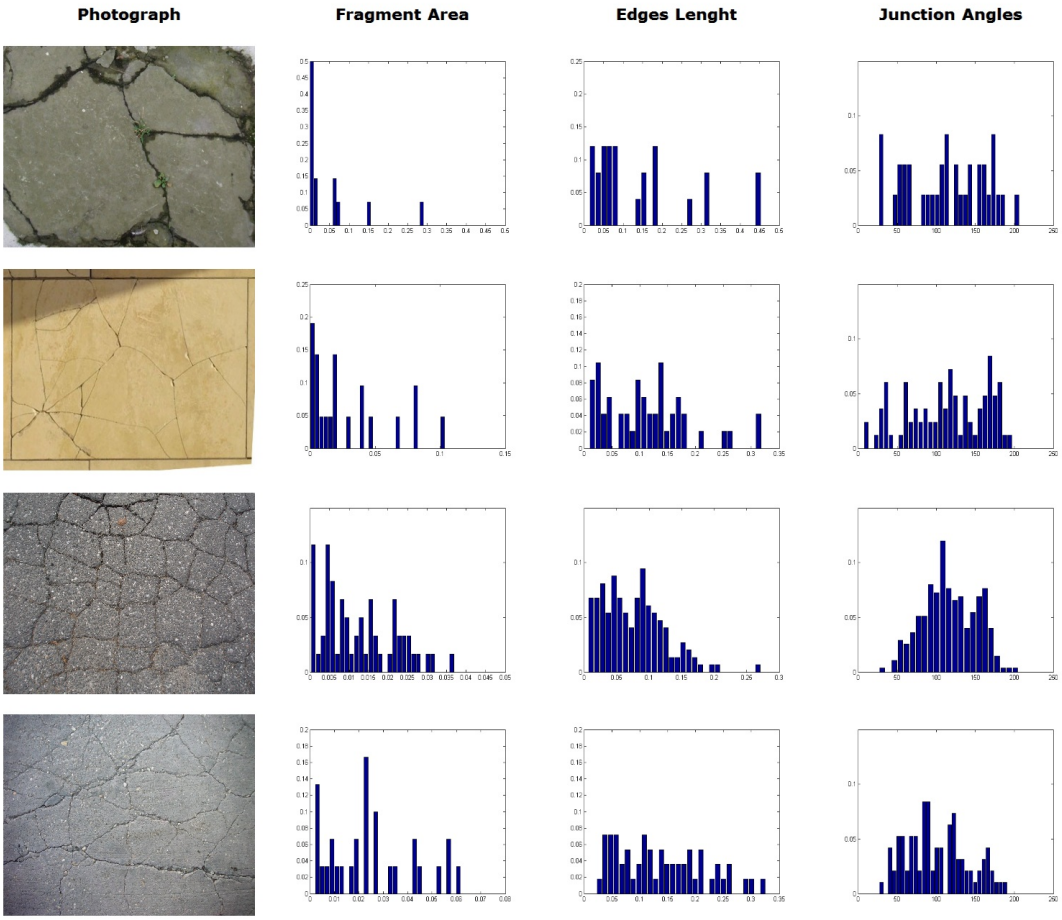


Figure 5.3 – Statistical informations extracted from images. Fragments, cracks and junctions statistics are computed from are vectorized version of input pictures.

Parameter	Min value	Max value	*Ref* parameter
$d\sigma_e$	0%	33%	R_{c_m}
r_{relax}	0%	100%	$\sqrt{\text{fragment area}}$
v_{relax}	0%	33%	R_{c_m}
$R_{c_{var}}$	0%	100%	R_{c_m}
l	0%	3000%	body mass

Table 5.2 – Range of valid values for the optimized parameters. Minimum and maximum values are expressed as a percentage of the reference parameter *Ref*. R_{c_m} is the average resistance to fracture of the material.

Normalization of the Statistics

The input statistics $\mathbf{s}_{obs}$ coming from the images are initially computed with pixel-unit length, while the statistics $\mathbf{s}_{sim}$ from the simulation have the unit length of the virtual world. In order the make the two values comparable, we normalize all our statistics: the areas of the fragment are divided by the total fractured area, the crack lengths by the square root of the total area.

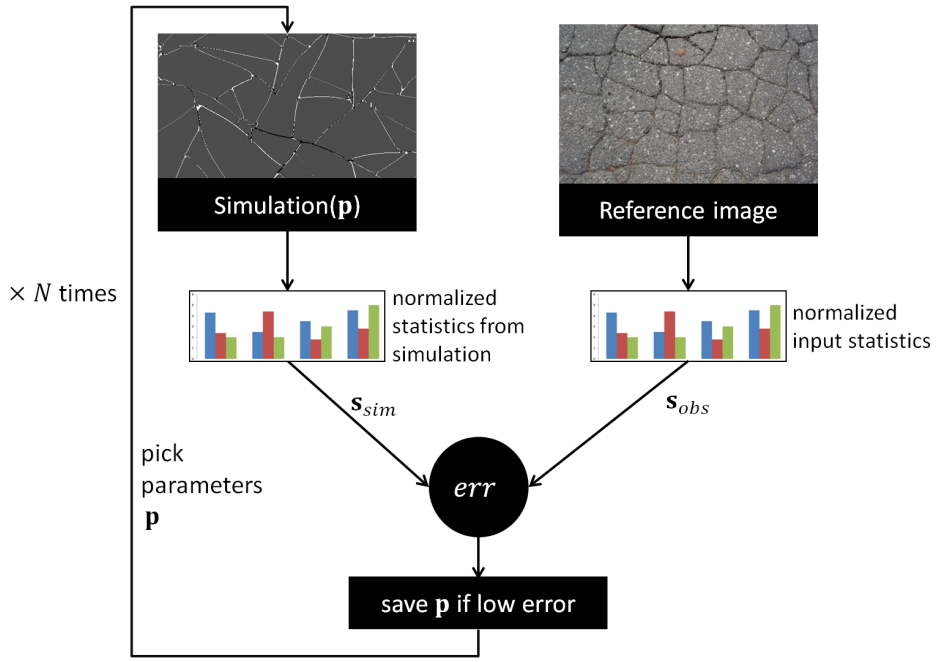


Figure 5.4 – Overview of the optimization process. Parameter vector $\mathbf{p}$ is optimized iteratively. After each simulation, the statistics of the simulation are compared to the statistics of the reference image. The *err* function is our metric (Eq. 5.1).

5.1.3.2 Optimization process

Since we have a wide range of possible values for the remaining parameters, we need to have a more efficient optimization approach than the brute force algorithm. As the distribution of resistance to fracture as well as the force position are random variables, we follow the same approach used in [Beaumont 02] to find our parameters, which is based on an Approximate Bayesian Computation.

We need to find appropriate 5-tuples for the values of the parameters of vector $\mathbf{p} = (d\sigma_e, r_{relax}, v_{relax}, R_{c_{var}}, l)$ which will result in patterns with statistics similar to those measured from the chosen fracture input image. The high level structure of the process is illustrated in Figure 5.4.

The main principle of the algorithm consists in sampling the five parameters of Table 5.2 in their respective valid ranges, following a uniform law. At each trial, the distribution of the resistance to fracture is sampled following a normal law. The magnitude l of the loading forces is optimized in our approach since we do not know its value from the reference images. We use an *a priori* probability distribution in the optimization process to take into account the randomness of the force position. We run N simulations, each one with a different set of parameters $\mathbf{p}$, and each one generating different output statistics $\mathbf{s}_{sim}$. After each simulation we compare the fracture input image and the simulated one using the following metric:

$$\begin{aligned} err(\mathbf{s}_{obs}, \mathbf{s}_{sim}) = \quad & w_{frag}.EMD(\mathbf{s}_{fragobs}, \mathbf{s}_{fragsim}) \\ + \quad & w_{crack}.EMD(\mathbf{s}_{crackobs}, \mathbf{s}_{cracksim}) \\ + \quad & w_{junc}.EMD(\mathbf{s}_{juncobs}, \mathbf{s}_{juncsim}) \end{aligned} \tag{5.1}$$

where 'frag' stands for fragment area statistics, 'crack' for crack length statistics, and 'junc' for junction statistics. Since all the statistics correspond to histograms of values measured in the image or simulation result, we chose to use the Earth Movers Distance [Rubner 98]

(noted EMD in Eq. (5.1)), which is known to give good results [Pele 08]. Finally, the w are weighting values that allow giving more importance to some statistics w.r.t. the others.

In our examples, we use $w_{frag} = 3$, $w_{crack} = 1$, $w_{junc} = 1$. These weights have been selected based on the user study, giving more weight to fragment area statistics than to the other two. For the user study, all weights were set to 1 to generate the synthetic patterns, since the weightings were not known. At the end of the N simulations, we take the set of parameters with the lowest error computed using our metric. This set of parameters can be then used to generate as many similar patterns as desired, by varying the resistance to fracture and the loading force position.

We found that $N = 20,000$ produced a good set of parameters for all our exemplars. A rectangular tile geometry, proportioned similar to the exemplar and discretized into 24,000 elements, is used for parameter estimation. Although a single force position is considered during estimation, multiple positions could be easily added to handle more complex patterns.

5.1.3.3 Results of the optimizations

Figure 5.5 – Application of our example-based fracturing method on different scenes. Photographs of input fracture patterns are shown in the insets. Left to right: (1) a bathroom tile has been broken, and each fragment can be manipulated separately. (2) Fracture pattern obtained from a ground tile photograph applied on a basin. (3) Sidewalk and curb from an urban scene fractured

This section shows results of our age-based with the optimization of the parameters method applied on different scenarios. Figure. 5.5 shows several fracture simulations using parameters optimized with our metric. Although the geometries of the simulated objects are very different, the visual features of the input fracture patterns are correctly reproduced. This can also be observed in Figure 5.6.

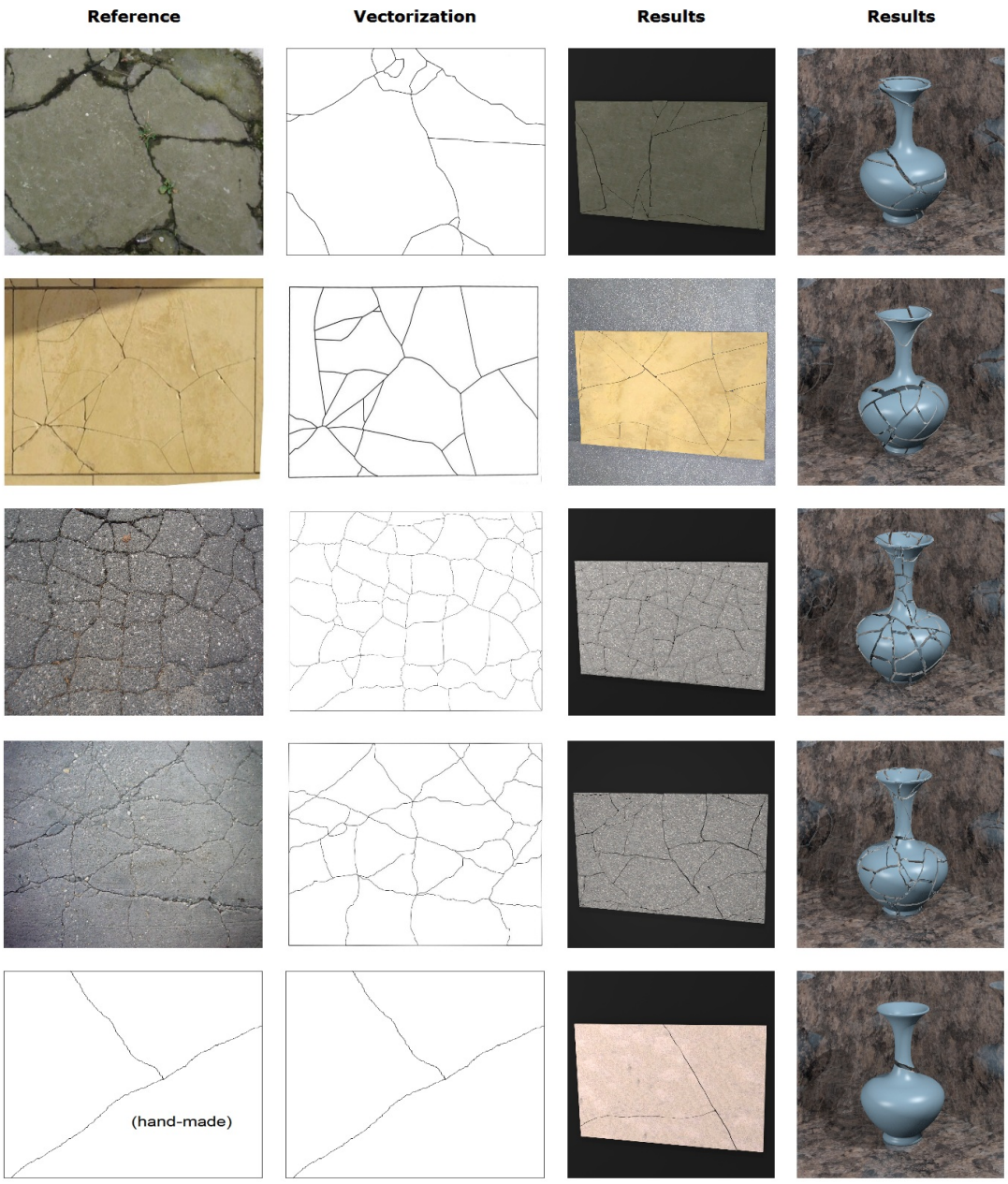


Figure 5.6 – Fracture patterns applied on different objects.

Figure 5.7 – Similar fracture patterns can be automatically generated and propagated from a single input (inset).

Since we are simulating the interior of the objects, a complete closed surface mesh is generated for each fragment.

Once the parameters have been optimized for a specific pattern, we can easily simulate the state of a fractured object at different ages.

Figure 5.7 shows an example of an editing session on the urban scene, using our interactive fracture modeler. The editor provides ways to add external input forces onto the object allowing richer and physically-based fracture patterns. We can also easily propagate a simulated pattern over similar objects, as shown in Figure 5.7 right.

The processing of each input exemplar takes between 2 and 5 minutes depending on the number of fragments. The optimization then takes 15 to 25 minutes for 20,000 trials using a body with about 40K tetrahedra. Computing each trial involves running the simulation with the current parameters, extracting the statistics of the obtained pattern and computing the difference using our metric based on EMD. Each such trial takes about 60 milliseconds.

The meshes used in our example applications vary from 7K tetrahedra for the ground tiles shown in Figure 5.7, to 54K tetrahedra for the road model of Figure 3.23. The simulation time varies from 16 ms to generate 4-5 cracks and 4-6 fragments on the tile mesh, to 264ms for 271 cracks and 71 fragments on the road mesh. The interactive editor runs between 30 and 100 fps, depending on the complexity of the scene. The latter framerate is obtained in the urban scene using 100 fracturable bodies along with the rest of the scene. Note however that most of this time is spent in visualizing the models. All timings were obtained in an Intel Core 2 Extreme with 2.3 GHz and 4GB of RAM, using a NVidia Quadro FX 3700M.

5.1.4 Discussion and conclusion

We validated through a user study the statistical approach as a way to measure the visual similarity of fracture patterns. From the statistical measurements, a similarity metric based on statistics extracted from the observed fractured pattern has been defined. Finally, we showed how the set of physical parameters can be optimized so that the resulting simulation mimic the fracture patterns observed on a photography.

Results shows that the same input pattern can be applied on different geometry, while producing the same key features of the pattern. Finally, the simulation times are fast enough to allow a user to apply pattern interactively on various scenarios.

5.2 Haptic virtual fracture: first experiments and results

The main purpose of the virtual fracturing study is to get perceptive feedbacks from naïve users of a fracturing simulation algorithm. In this section, we present a description of the system being tested, and the setup of the user study.

Figure 5.8 – Setup of the experiment. **Left**: photography of the experimental conditions. The haptic interface is linked to the ball in the simulation, and the user can break the plates with force feedback. **Right**: screenshots of the simulation during a trial at the initial condition (top) and after one plate has been broken (right).

5.2.1 Overview

The global system being tested is the fracture algorithm presented in section 3.3 coupled with a six input and output degrees of freedom haptic device using the coupling presented in section 4.3.2. The efficiency of the fracturing system allows setting up scenes with freely moving objects at rates of 600Hz including the fracture simulation. The motion of the rigid bodies is managed by a third party rigid body dynamic engines, leading to a robust and stable simulation, as well as a natural coupling with any admittance piloted haptic device.

In order to get a subjective validation of this fracturing algorithm, we set up a user study that has been designed with two main objectives: (1) Check the behavior of the simulation at a perceptive level by checking if the users are able with their own strategies to actually find out the weakest objects in a simple scene. (2) Get a general perceptive feedback of the fracturing simulation and of the haptic display at the time of impact.

5.2.2 User study

The user study setup is shown in Figure 5.8. The user manipulates a virtual ball through a Haption Virtuose 6D haptic device (www.haption.com, Haption S.A., Soulgé sur Ouette, France), and breaks two plates of different weaknesses. We performed a 2-alternative forced choice (2-AFC) experiment. During one trial, the user has to find out which plate (either the left one, or the right one) is the weakest one.

Experimental apparatus

Using the strategy of their choice, the users were asked to find the weakest plate (*i.e.* the plate the less resistant to fracture) among the two plates displayed, using the haptic interaction. When the user has decided, he had to press a button on the keyboard corresponding to the chosen weakest plate. There was no limit of time for each trial, and the user could take a break at any time by pressing the space key.

Population and collected data

12 participants (10 males, 2 females) aged from 23 to 40 (mean = 26.8, standard deviation = 20.1) performed the experiments. At each trial, we recorded the weakness value of the chosen plate, the weakness value of the other plate, and the time elapsed from the beginning of the trial to the time of the choice. At the end of each session, the participants fill a questionnaire asking for the number of different perceived weaknesses (between 0 and 5), the confidence of their choices (using a likest-scale between 0 and 7), a textual description of the strategy they adopted, and any additional remarks.

Experimental conditions

We used 3 different values for the weaknesses (w_1, the weakest one, w_2 and w_3, the stronger one) of the plates among all the trials and all the users (however, the users do not know how many different plates they are going to experience). The weakness of each plate is a parameter of the fracture simulation algorithm that represents the amount of deformation the body can undergo before it begins to fracture. We chose w_1, w_2 and w_3 experimentally with the help of an experimented and a naive user to set the min and max values. The differences between w_2 and w_3 and between w_1 and w_2 are identical. During each trial, the user has to compare either the weakness w_1 vs. w_2, w_2 vs. w_3, or w_1 vs. w_3, leading to 6 different possible trials if we take into account the symmetry of the scenario.

One experimental session is composed of 30 trials (5 instances for each possible case). The order of the trials for each session is chosen randomly. Each user performs 2 sessions:

- One session without force feedback: the user interacts with the scene using the haptic device, but with no force output. In that case, the user can use the acceleration and velocity to break the plate as haptic cues, but no force is generated when he hits the plates.
- One session with force feedback: in that session, the user has another haptic cue, which is the perceived force when he hits a body with a ball, and when a plate being hit by the ball fractures.

5.2.3 Results

We analyzed the answers of the participants for the different haptic conditions in order to determine which condition gave the highest percentages of answers in terms of weakness value recognition. Each individual performed 720 comparisons. A summary table of the answers of the participants is presented in Table 5.3. Over a total of 720 trials, 153 errors (20.1%) were made: 80 errors (11.1%) in the session with force feedback, and 73 errors (10%) in the session without force feedback. Under the null hypothesis of equal preference between two conditions, the number of times an individual preferred the first condition follows a binomial distribution with parameters 6 and 1/2. After standardization, such a variable can be approximated by a standard normal random variable. Thus, for the different pairs of weakness, we tested the presence of a preferred weakness for each condition using an exact binomial test. The p-values were adjusted with a Bonferroni correction. The percentages of good answer w.r.t. the conditions (weakness of the plates and with or without force feedback) are given in Table 5.4. For the condition without force feedback, the analysis showed that there was a significant effect for the comparison between w_1 and w_2 ($p = 0.006$) as well as between w_1 and w_3 ($p < 0.001$) but not for the comparison between w_2 and w_3 ($p = 0.39$). For the condition with force feedback, participants were able to compare the weaknesses w_1

and w_2 ($p = 0.007$), as well as the difference between w_1 and w_3 ($p < 0.001$) and between w_2 and w_3 ($p = 0.006$). We then compared the answers between the two conditions (with or without force feedback) with a signed rank test. We found no significant result for the comparison between w_1 and w_2 ($p = 0.15$) as well as between w_2 and w_3 ($p = 0.77$) . However, the analysis showed that there was a significant effect for the comparison between w_1 and w_3 ($p < 0.001$).

With force feedback

	w_1	w_2	w_3
w_1		85	106
w_2	35		89
w_3	14	31	

Without force feedback

	w_1	w_2	w_3
w_1		95	111
w_2	25		81
w_3	9	39	

Table 5.3 – Summary tables of the choices of the participants. For a cell of row w_i, column w_j, and value v, read the weakness value w_i has been v times considered to be lower than the weakness w_j. Therefore, the values on the lower left triangles of the matrices are the number of errors made by the participants.

	With force feedback	Without force feedback
w_1 vs. w_2	71.8 (sd = 13.8)	79.2 (sd = 13.1)
w_2 vs. w_3	74.2 (sd = 15.6)	67.5 (sd = 23.8)
w_1 vs. w_3	88.3 (sd = 11.4)	32.5 (sd = 7.5)

Table 5.4 – Percentage of good answers and standard deviation (sd) for each condition.

Concerning the subjective questionnaire, we performed a Friedman test on the confidence mark given by the participant. We found no significant effect on the confidence mark ($\chi^2 = 0.67$ and $p = 0.11$). However, the mean value for the condition with force feedback ($M = 4.39$, $SD = 0.72$) was slightly higher than for the condition without force feedback ($M = 5.17$, $SD = 1.47$).

5.2.4 Discussion and conclusion

We noticed through the experiments and the questionnaires that all the participants converged quickly to the same strategy. They first place the ball above the center of one plate, and hit it with a small velocity. Then, they increase the height of the ball and the velocity of impact (giving also stronger forces and acceleration to the ball) until the plate breaks. They do the same with the other plate, and try to remember the plate that broke with the less effort. This strategy has been used for both sessions (with force feedback and without force feedback). On the session with force feedback, the comparison between the two weakest plates is responsible for most of the errors. This can be explained by the way the fragments are simulated once the plate is broken. Boundary volumes are used to detect the collisions between the fragments, and this approximation generates overlapping between the fragments yielding to exaggerated reactions.

On the other hand, between the two stronger plates, it seems that haptic display helped the user to remember the strength of the impact (in addition to the information of velocity and force given by the participants). Indeed, bigger forces are involved and the extra forces generated by the collision detection approximation have less impact.

Regardless of the user study, the haptic feedback coupled with the fracture simulation allowed us to perceive artifacts that were not perceivable on the visual rendering. For examples, we discovered bugs into the simulation that produced inconsistent fracture reactions w.r.t. the input effort thanks to haptic rendering. From this experience, we encourage the use of haptic as a mean of subjective validation of simulation algorithms.

5.3 Preliminary validation of the impact-based fracture model based on real data

The main purposes of our experiments were to have a comparison basis with real-life data to evaluate and calibrate the impact-based fracture model presented in section 3.3. We designed experiment set ups to collect the data for this goal. In this section, we first present the experimental set ups and conditions. Then, we present the collection of image data obtained. Finally, we discuss the results obtained, with preliminary conclusions and comparison with our model. This work has been done in collaboration with the LARMAUR, University of Rennes 1, France.

5.3.1 Experiment types and setups

We chose two different objects for our experiments: 15x15x0.6cm ceramic tiles and 15x15x0.2cm glass slabs as shown in Figure 5.9.

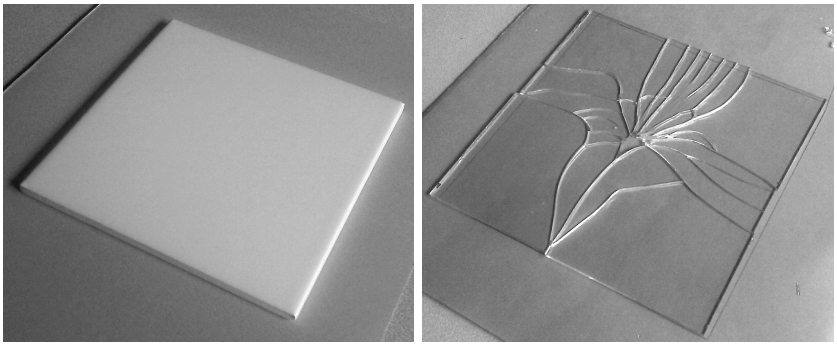


Figure 5.9 – Example of ceramic tile (left) and glass slab (right) used in our experiments.

We set up various types of experiments:

- **Identification of the material elastic properties**. We measured the exact dimension, density, Young's modulus, Poisson ratio of the ceramic tiles.
- **Identification of the Rankine threshold**. We measured the value of stress before fracture (Rankine threshold) of the ceramic tiles applying static loadings.
- **Impact breaking test - 4 punctual contacts**. We made experimental breaking test by dropping a metallic ball from the same height onto ceramic tiles and glass slabs resting on 4 contact points.
- **Impact breaking test - fitted**. We made experimental breaking test by dropping a metallic ball from the same height onto ceramic tiles fitted in an aluminum support.
- **Impact breaking test - varying ball height**. Varying the height of the ball to observe fracture can help to understand the dynamic aspects of the deformation on the fracture results.

- **Other exploratory experiments** We designed exploratory experiments to help to understand the behavior of glass slabs.

Each experiment is described below with more details.

5.3.1.1 Identification of the material elastic properties

From accurate measurements of the volume of a ceramic tile and its weight, we found the ceramic to have a density $\rho = 1738kg.m^{-1}$.

In order to identify the Young's modulus and the Poisson coefficient of the ceramic, the velocity of propagation of a transversal and longitudinal wave is measured with a piezoelectric transducer. We found a velocity of $v_{long} = 2820m.s^{-1}$ for the longitudinal wave, and a velocity of $v_{trans} = 1849m.s^{-1}$ for the transversal wave. From these values, we deduce the Young's modulus E and the shear modulus G as:

$$E = \rho \left(\frac{3v_{long}^2 - 4v_{trans}^2}{\frac{v_{long}^2}{v_{trans}^2} - 1} \right) = 13.35GPa \tag{5.2}$$

$$G = \rho v_{trans}^2 = 5.94GPa \tag{5.3}$$

From the Young's modulus and shear modulus, the Poisson ratio can be obtained with:

$$\nu = \frac{E}{2G} - 1 = 0.123 \tag{5.4}$$

5.3.1.2 Identification of the Rankine threshold

In order to determine the Rankine threshold (the value of maximum tensile stress before fracture), we performed a ring on ring test as shown in Figure 5.10. The ceramic tile is squeezed between a pair of concentric rings with different diameters. The ring above the tile is smaller than the ring below it. In our experiment, the upper ring had a diameter of 60.5mm, while the lower ring had a diameter of 125.1mm. This test allows obtaining a uniform stress on the lower part of the tile, and to compute the Rankine threshold R_c of the material with the following relation:

$$R_c = \frac{3f\left(2\ln\left(\frac{a}{b}\right)r^2(\nu+1) - (a^2-b^2)(\nu-1)\right)}{4e^2\pi r^2} \tag{5.5}$$

where:

- f: force applied at the instant of fracture.
- e: thickness of the tile.
- a: diameter of the upper ring.
- b: diameter of the lower ring.
- r: tile width.
- ν: Poisson coefficient of the tile.

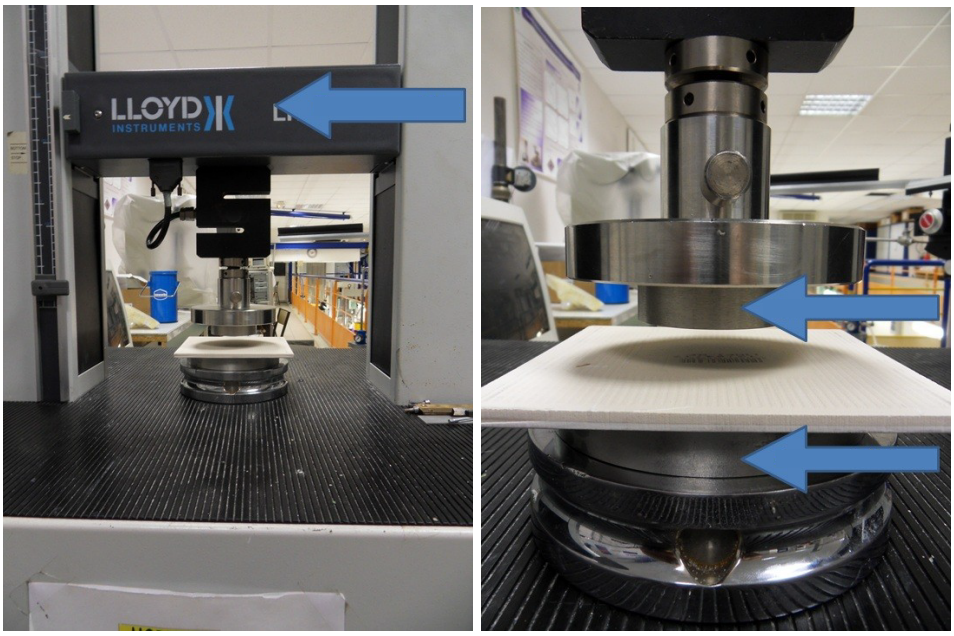


Figure 5.10 – Set up of the ring on ring test. **Left**: the upper axis (blue arrow) is translating down. **Right**: the ceramic tile is resting on a large ring, while an upper ring with a lower diameter is loading the tile.

We did the experiment on 30 tiles, with 3 different velocities of approach of the upper ring, namely 10 tiles at $1mm.min^{-1}$, 10 tiles at $10mm.min^{-1}$ and 10 tiles at $100mm.min^{-1}$. These three velocities can be considered extremely small compared to the velocity of propagation of the wave into the material, and should produce configurations of the tile that can be studied in a static context. For each trial, the test machine outputs the maximum force f that the upper ring that applied on the tile before it breaks. From these forces, we deduced the Rankine threshold R_c for each tile using equation (5.5). The plots of the values obtained for the Rankine threshold are shown in Figure 5.11.

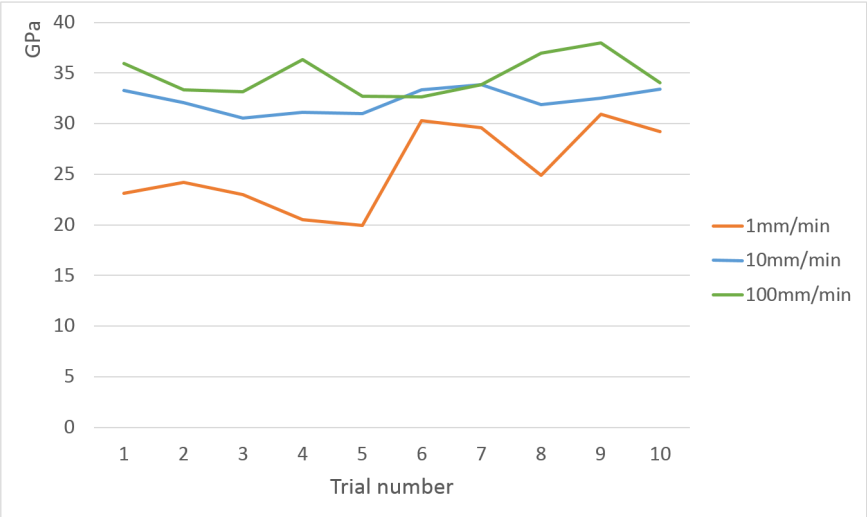


Figure 5.11 – Results of the ring on ring tests.

A surprising fact can be noticed on the trial made at a very low velocity $1mm.min^{-1}$. Indeed, we observed values of Rankine thresholds lower than the ones obtain at 10 and

$100mm.min^{-1}$. The explanation of the observation is not clear. We believe it is a consequence of stress corrosion cracking due to the humidity of the air and the possible start of uncompleted cracks that allow that corrosion. At higher velocities, the results on the Rankine threshold are consistent (while we think to have experienced stress corrosion cracking also at $10mm.min^{-1}$), with an average of $35MPa$. In Figure 5.11, we show photos of the broken tiles. We observed that the tiles that broke in fewer pieces led to a lower Rankine threshold. At the velocity of $10mm.min^{-1}$, some of the tiles broke apparently the same way as the slowest speed, and some of them broke in more pieces. Figure 5.12, 5.13 and 5.14 show the pictures of the broken tiles on the ring on ring test. Irregularities on the results on the Rankine threshold of the first condition are also observed on the pattern of the fracture.

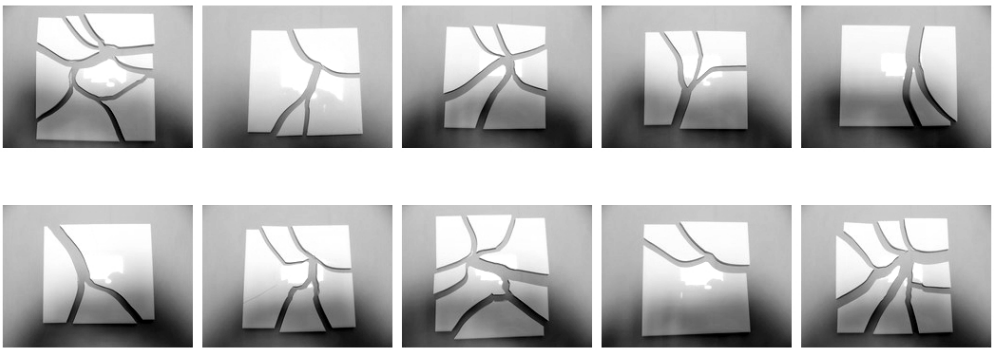


Figure 5.12 – Pictures of the 10 broken tiles ($1mm.min^{-1}$) on the ring on ring test.

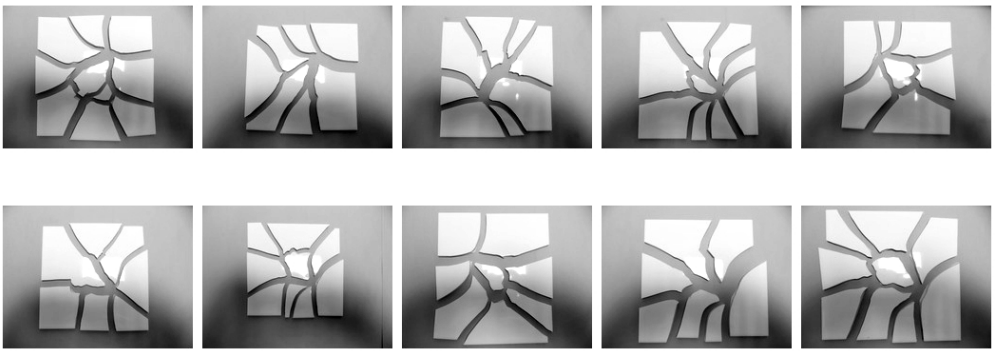


Figure 5.13 – Pictures of the 10 broken tiles ($10mm.min^{-1}$) on the ring on ring test.

5.3.1.3 Impact breaking test - 4 punctual contacts

To measure the similarity and the variance that is be observed on the fracture the same type of object under the same conditions, we designed the 4 punctual contacts test shown in Figure 5.15. The tile is resting on 4 accurately positioned metallic balls. A 110g metallic is dropped at the top of a plastic cube, and hits the tile on its support.

We performed 30 trials that are presented on Figure 5.16.

The results show that even if the conditions are the same, the number of fragment obtained is different on the trials. The fracture statistics on the fragments are provided in Table 5.5.

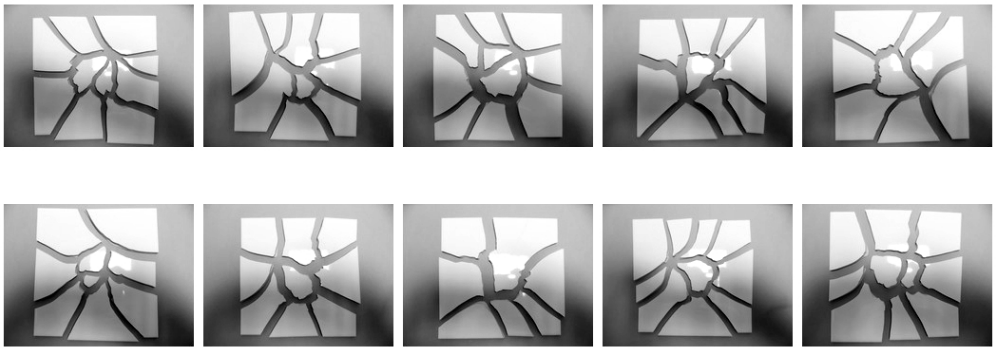


Figure 5.14 – Pictures of the 10 broken tiles ($100mm.min^{-1}$) on the ring on ring test.

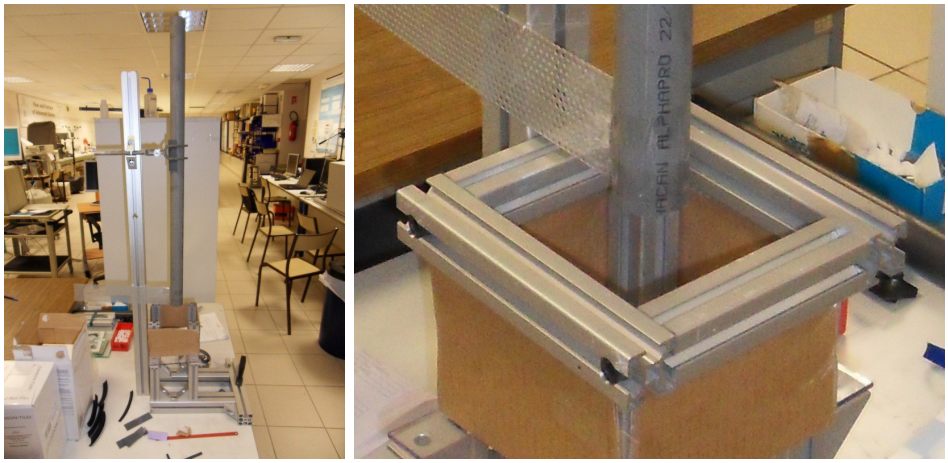


Figure 5.15 – Test bench designed for the 4 punctual contacts breaking and the fitted tiles tests. **Left**: overview of the bench, a ball is dropped in the tube to fall on a tile set up on its support. **Rigth**: zoom on the aluminum support of the tile. For the 4 punctual contacts test cases, 4 metallic balls were fixed on the top of the aluminum support.

	Frag. number	Frag. area (mm^2)	Crack length (mm)	junction angle
Mean value	4.7	5746	83	76
Std deviation	0.74	1565	10	19

Table 5.5 – Mean value and standard deviation of the fragment number, area, crack length and junction angles extracted from the 30 trials of the 4 punctual contacts case (Figure 5.16).

5.3.1.4 Impact breaking test - fitted

In order to check if the variance obtained in the 4 punctual contacts test came from the small variation of the condition or from the variation of the material, we designed the fitted breaking test. In this condition, the tile is squeezed into an aluminum support shown in Figure 5.15, right. The main difference with the punctual contact is that the positioning error of the contact as well as the bouncing effects that could modify the results vanishes. However, it is less clear in this situation if our contact model still holds. We present preliminary results of this test in Figure 5.17.

In all cases, we observe that five cracks are produced, with no variance. More trials are however needed to be able to conclude on the variance.

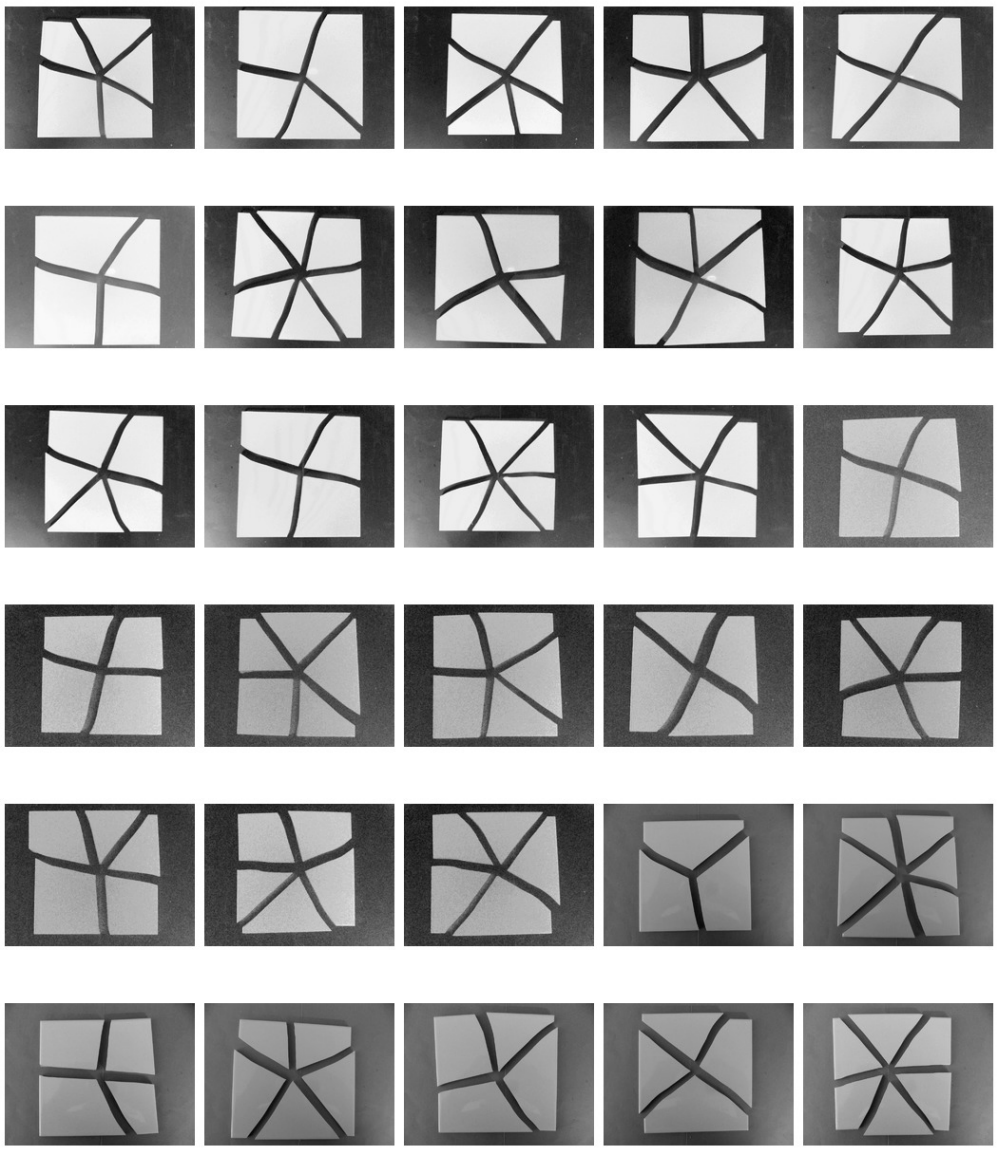


Figure 5.16 – Pictures of the tiles broken on the 4 punctual contacts test. Even if similarities are observed between the trials, a non-zero variance is observed on the number of fragments obtained.

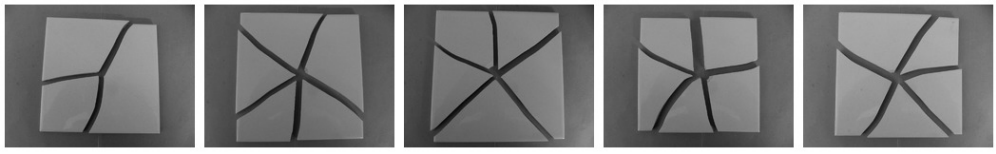


Figure 5.17 – Pictures of the tiles broken on the fitted test. In this condition, the variance on the number of cracks seems to be lower than the previous condition. On the first picture, 5 cracks propagated, but only 3 of them crossed the tile.

5.3.1.5 Impact breaking test - varying ball height

To main objective of the varying ball height test was to check the height at which the tile break, and to be able to compare it with our simulation method. The experiment consists in setting the tile on 4 punctual contacts, and dropping the same metallic ball from a small height, and increase the height until the tile breaks. The value measured is the height of the ball for which the tile break. We did 10 trials. The height before fracture varied between 48cm and 52cm, with an average of about 50cm for a metallic ball of mass 0.11kg, and a diameter of 25mm. This means that dropping a ball from this height on this ceramic tile theoretically generated a maximum stress of about 35MPa at the center of the tile. These values should be observed in numerical simulations of fracture, and are a basis for comparison of simulation and real fractures.

5.3.1.6 Test on glass slabs

We present here preliminary test we performed on 150x150x2mm glass slabs. Indeed, the first results obtained were surprisingly sparse, and we decided to concentrate on the understanding of the difference of the behavior of the glass w.r.t. the ceramic rather than trying to obtain statistical information of the glass fractures.

Figure 5.18 – Experiment on glass slabs - 4 punctual contacts. The results obtained are various and hard to interpret. Some noise may bias the experiments.

The first experiment was identical to the 4 punctual contacts, but using glass slabs instead of ceramic tiles. As shown in Figure 5.18, the results of this experiment are hard to interpret, with either multitude of cracks and fragments, or just a few of them that not always propagate from the center. This observation made us believe that small noise in the experimental condition introduced bias on the experiments. For example, the four contacts on which is resting the slab may not be perfectly on the same plane, and the secondary vibrations of the glass slab plus the bounces of the metallic ball may generate more fragments than initially planned. This noise would be responsible for the random observed results. The ceramic tiles were thicker and though less sensible to this noise.

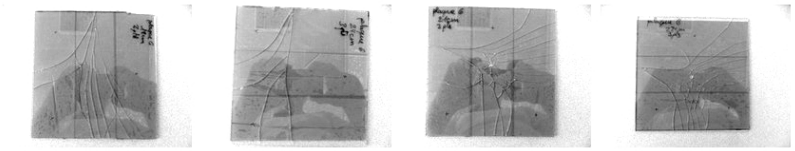


Figure 5.19 – Experiment on glass slabs - 3 punctual contacts. Noise is still observable is this condition.

We also noticed that some cracks propagate from the exterior of the slab as highlighted in Figure 5.9, right. This means that weaknesses are present on the borders of the slabs, that allow cracks to propagate easily from the exterior. In many cases, we observed a mixture of cracks propagating from the center and cracks propagating from the borders. This is

observable on experiments we performed with 3 punctual contacts to ensure the contact points to be coplanar. The results of this condition are presented in Figure 5.19. In order to limit the propagation of the crack from the border weaknesses, we tried to put the slab on a ring as support instead of punctual contacts. However, as highlighted in Figure 5.20, some noise remains in the conditions, and it is difficult to reproduce the same patterns.

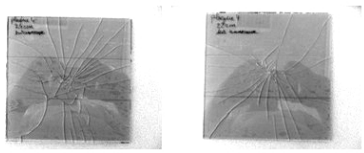


Figure 5.20 – Experiment on glass slabs - ring support. Noise is still observable is this condition.

To further reduce the noise in the experimental condition, we tried to put the slab on a stiff moss to drop the ball on it. In this condition, a curious case appears as shown in Figure 5.21. Either the glass shatter with a clear star pattern propagating from the center, or propagation from the borders occurs and the pattern obtained is radically different.

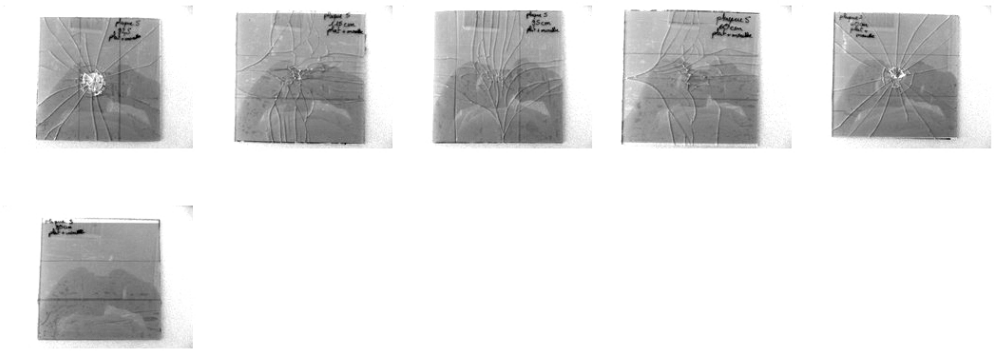


Figure 5.21 – Experiment on glass slabs - moss support. Two kind of pattern appears. First (trial 1 and 5) clear star pattern. Second (other trials) noisy patterns.

Finally, in order to prevent a maximum the crack to propagate from the borders, we did some trials with the glass slab resting on another glass slab, guaranteeing a clean surface and a few deformations at the extremity of the slab (the support slab is resting on the ground). Results obtained in this condition are shown in Figure 5.22. This condition demonstrated more stable results, with crack propagating only from the center. It is however not clear how the slab will deform in this case, and how tensile stress will open the fracture.

The results on glass slabs are preliminary, but open perspectives for more detailed experiments on which statistical information could be obtained, as for the ceramic tiles. Figure 5.23 shows the results obtained by the simulation when the $110g$ metallic ball is dropped from 50cm high. We can see that some features observed in real experiments such as the star-like pattern starting from the impact point is reproduced correctly in the simulation. However, some features produced by the simulation are not observed in the real experiments. The close and almost parallel fractures generated by the simulation are not present in reality.

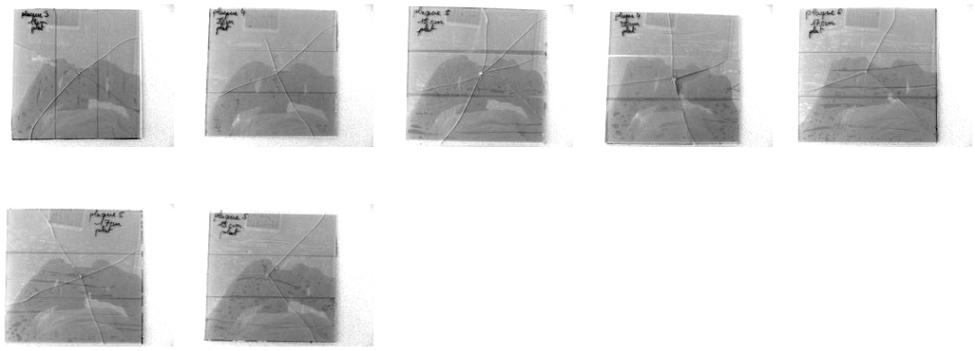


Figure 5.22 – Experiment on glass slabs - planar glass support. In this condition, no crack propagates from the borders, and the fracture patterns seem reproducible.

5.3.2 Discussion and conclusion

We performed various experiment on the static and dynamic fracture of ceramic tiles and glass slab. Although still preliminary, these experiments are a good starting point for the evaluation and the comparison of our fracture simulation approach. We designed in our fracture simulation framework the virtual version of the 4 punctual contacts experiment. The dimension, density, elastic properties and fracture properties have been set according to the measurements.

To simulate the tiles of Figure 5.23 we applied a factor of 1/5 on the contact duration estimation. Indeed, in the first simulations (without this scaling) the tiles did not break at $50cm$, the height at which they break in real experiments. Increasing the height of the ball to $55cm$ made the tiles break, but not with the star pattern, due to a large propagation of the deformation into the tiles, as shown in Figure 5.24, top. With this factor on the contact duration, we managed to generate more local deformation (Figure 5.24, bottom), and the cracks propagate from the center.

Figure 5.23 – Results of the simulation of the ceramic tiles. Star patterns that are observed in real experiments appear on the simulated tiles.

Also, we found that the simulated tiles broke with about the same threshold on the height of the ball ($48cm$). Under this threshold, the simulated tiles did not break, and above the threshold, they did. However, because stress relaxation is not simulated, we found that the energy dissipated during the fracture that is not dissipated using our model led to generate more fragments than observed in real life. Indeed, when a crack starts to propagate, the stress around the crack path should be reduced due the energy of creation of new surfaces, lowering the probability for another crack to start in almost the same direction. Although we simulate this phenomenon for the age-based cracking, we do not simulate it for the impact-based approach because of it computational cost. Simulating the stress relaxation would become the bottleneck of the simulation and compromise its interactivity.

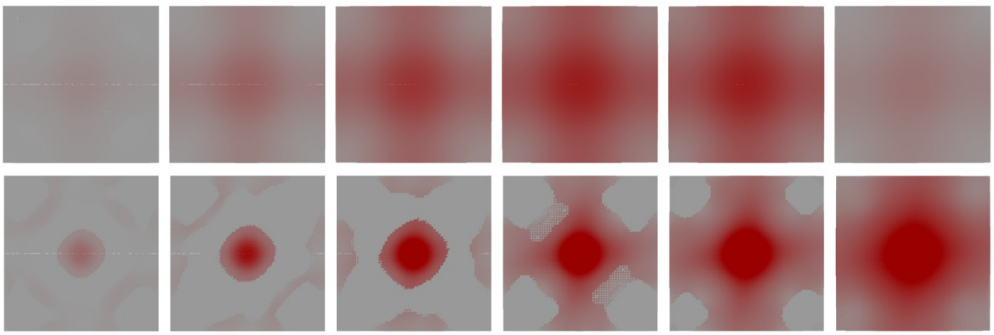


Figure 5.24 – Scaling of the duration of the contact. Bottom view of the deformation state of the tile during the contact force application. Only the maximum tensile stress is represented with a red scale (pure red means the Rankine threshold is reached). **Top**: the duration of the contact is not scaled, and the deformation generated by the impact propagates without generating local deformations. **Bottom**: with a scaling on the contact duration, local deformations are created, allowing reproducing the star pattern observed in real experiments.

All these experiments are preliminary and deserve more study. However, we observed important results. First, the fracture pattern seems to be statistically reproducible. Namely, interpreting the pattern with a statistical point of view allow observing the similarity between the results. Second, our real-time simulation approach seems suitable for prototyping. After some calibration on the contact duration, we were able to reproduce the fracture test condition of the ceramic tiles. Setting all the measured materials parameters in the simulation allowed us to retrieve the fracture threshold observed in the experiments. Some additions to the current models such as the simulation of stress relaxation are however necessary to obtain more correct fracture patterns w.r.t. our pattern similarity metric.

5.4 Chapter conclusion

In this chapter, we tackled the challenge of the evaluation and validation of the brittle fracture simulation. We divided the validation in two parts: the crack propagation, and fracture initiation part, which we consider to be different goals of a fracture simulation.

We proposed to define a similarity metric between fracture patterns based on statistical information extracted from the fracture geometries. We showed through a user study that this approach is valid for the perception of the similarity between fracture patterns. We also presented an optimization process that fits a set of physical parameters to make the simulation match an input pattern extracted from an image. Results show that realistic images are automatically obtained from this process. In a second part, we proposed a subjective validation of our brittle fracture simulation and haptic rendering coupling methods based on haptic feedback. We conducted an experiment and showed that subjects are able to perceive the variation of fracture parameters through the haptic interaction.

In a third part, we presented experiments of real ceramic tiles and real glass slabs that have been broken in different conditions to validate our impact based algorithm. We showed that statistical information used for the perception studies could be relevant to compare objects broken under the same conditions. We also showed through the ceramic tiles experiments that our simulation is able to reproduce the main features observed on the real broken tiles. The experiments also allowed us to highlight artifacts in the simulations (such as the crack energy that is not dissipated) that would be a good start for future studies.

Conclusion

6

In this thesis, we tackled the topic of real-time and physically-based simulation of brittle fracture. We divided the project around three complementary axes. First, we addressed the **modeling of the fracture phenomena**, proposing more efficient and more physically-based models and methods for the simulations of brittle fracture in real-time. To allow interactive simulations with brittle fracture simulations, the collision detection and the haptic feedback have been addressed as the second axis of study of this manuscript, with new algorithms that tackle the unique challenges involved by the brittle fracture simulations for interactive applications. In a third part called **evaluation and validation**, we studied the limitations and the physical correctness of the models proposed through perceptive studies and real data from experiments.

6.1 Modeling of the fracture phenomena

Concerning the modeling of the fracture phenomena, we proposed new models and methods that allow simulating impact-based and age-based brittle fracture more efficiently and with an improved physical correctness compared to previous real-time methods.

Modeling the fracture state and crack propagations

We first presented a model that stores the fracture state of a body based on three components: a damage state stored in the elements of a volumetric mesh, a fracture surface sampling stored in the edges of the same volumetric mesh, and a fragment identifier stored at each node of the mesh. The design of this model allowed having an accurate representation of the fracture state of a body, as well as defining the modifications of the fracture state due to fracture events. A propagation algorithm that updates the fracture state model based on an implicit surface definition of the fracture paths has been presented. The complexity of this algorithm is linear in the number of cut elements. An energy stop criterion based on the strain energy has also been proposed to stop the crack propagation and generates partial fractures. We also described an efficient meshing algorithm that converts the fracture state into a set of surface meshes (one per fragments) ready to be displayed with classical 3D hardware. This results in an efficient representation of the fracture without constraining the fracture surfaces to follow predefined boundaries.

An impact-based fracture simulation method based on modal analysis

For the impact-based fracture, we presented a new method based on modal analysis capable of estimating the contact durations, choosing adaptive time steps, and simulating efficiently the deformations of impacted bodies, taking into account dynamic effects such as inertia and damping. The contact durations are estimated using the vibrational properties of the highest excited deformation modes of the body, while the time step is chosen using the Shannon-Nyquist sampling theorem applied on the highest frequency modes. Results showed that

our method has computational requirements compatible with interactive applications, and is able to reproduce dynamic effects of fracture that were not possible to simulate in real-time previously.

An efficient age-based cracking algorithm

For the age-based fracture, we presented an extension of previous aging methods to build a more efficient but still physically-based aging simulation of brittle fracture. Our method is based on a stress map that evolves over time, and a new stress relaxation algorithm designed for efficiency. Results demonstrate that our method produces visually realistic patterns that are applied at interactive rates on large scenes.

A database approach for fragment physics

Finally, we presented an approach based on a database of precomputed physical data as a way to handle the physics of the fragments. Modal analysis is precomputed for a range of parametrized shapes, and stored in a database, labeled with mesh descriptors. We studied three types of mesh descriptor: a moment-based descriptor, a voxel-based descriptor, and an improved version of the voxel-based descriptor. When new fragments are created at run-time, their mesh descriptors are computed and a quick search is performed into the database to find the entry of the database that matches the geometry of the fragments. We showed that the database approach can be applied in real-time, and produces visually credible visual feedback.

6.2 Interaction and brittle fracture

We addressed the problem of the collision detection between the fragments by proposing new methods suited for brittle fracture scenarios. Also, to demonstrate the applicability of our method for interactive applications, we proposed a haptic feedback of the brittle fracture simulation, which necessitate high efficiency and stability to perform.

Collision detection for brittle fracture

We presented a new algorithm for collision detection between fragments based on three mechanisms: a mesh-based distance field storing at each node of the mesh a distance to the closest surface, a reconfigurable sphere tree, and a contact selection mechanism that avoids peak computation time at fracture events. We proposed a way to update the mesh-based distance field using a front propagation algorithm, and to build the sphere tree considering the volume of the body instead of its surface, allowing efficient updates at fracture events. Our results showed that good computation time performances are achieved with this method, and that scenes composed of non-convex objects and thousands of triangles can be handled in real-time.

Haptic rendering for brittle fracture

On the haptic interaction side, we first demonstrated through a benchmark and haptic tests that common rigid body engines can be coupled robustly with a haptic device controlled or emulated in admittance mode. We also proposed coupling mechanisms that allow modifying the ratios between the mass and length quantities of the real world w.r.t. the virtual world while guaranteeing a stable simulation. Then, we proposed a new coupling scheme for haptic

rendering based on the extraction of a sub-world around the area of interaction that is simulated at a different rate. From the contacts between the rigid bodies, we build a graph starting from the manipulated object, and linking recursively the bodies that are in contact to create a sub-world that is simulated at higher frequencies for haptic rendering purposes. If the sub-world becomes too big to be simulated at haptic rate, the depth of the graph is limited, and mechanisms that manage the interface between the sub-world and the whole world are proposed. This coupling scheme allows maintaining the high frequency required for realistic haptic feedback even if the number of bodies to simulate is increasing, as it is the case in fracture scenarios.

6.3 Evaluation of the models

In order to evaluate and validate the methods proposed, we presented a new similarity metric for fracture, perceptive studies, and experiments based on real-life breaking scenarios.

A similarity metric for fracture patterns based on statistics

We first tackled the problem of the formal definition of the similarity of fracture patterns. We proposed to define a similarity metric based on statistical information extracted from the fracture geometry, namely the fragments number and areas, the cracks number and lengths, and the junction properties. We evaluated the importance of each category of statistic in the human perception of the fracture patterns by performing a user study. We found that the distribution of the fragment areas is the information that influences the most our perception of the fracture pattern. Then, we defined a similarity metric and proposed a method that optimizes the simulation parameters to make the fracture pattern match with the one of an input exemplar photography. The optimization process tries to minimize the error between the normalized input statistics extracted from the exemplar image, and the normalized statistics generated from the fracture simulation with a Bayesian process. Our results demonstrated that believable and realistic fracture patterns are automatically reproduced from exemplar images using this approach.

Subjective perception with haptic feedback

We also proposed a perceptive study of the believability of our impact-based simulation method through the haptic feedback. Users were asked to break plates of different resistance to fracture through a haptic interaction. We found that the subjects were significantly able to discern the plate properties correctly. We also found haptic feedback to be a helpful tool to detect the artifacts in the simulations that were not possible to perceive with the visual feedback.

Validation based on real-life breaking objects

Finally, we proposed to validate the impact-based model for the simulation of brittle fracture with real experiments. We designed a benchmark where ceramic tiles and glass slabs were broken under different conditions. We measured the elastic properties and the resistance to fracture of ceramic tiles, and performed various breaking tests designed to validate individual aspects of the simulation. On the ceramic tiles, we observed statistically similar results between the tiles broken under the same conditions. The experimentations on the glass slabs were subject to noise introduced by weaknesses on the border of the slabs that made them

break differently. We explored five different impact conditions to reduce this noise, and found that a flat support seemed to cancel the noise introduced by the borders, opening perspectives for further exploration and experiments. Finally, we compared the results obtained in these experiments with its virtual version simulated our impact-based model. We found that the star patterns observed in the experiments are also present in the simulation with a factor on the contact duration estimation. We also found artifacts present in the simulation that were not present in the real-life tests. First results of the validation are encouraging and helped us to understand better the strengths and weaknesses of our model, and provided ideas for future work.

6.4 Discussion and perspectives

All the models and studies presented in this manuscript are complementary, and allowed us to build a real-time and physically-based simulation of brittle fracture, with free crack propagation, simulation of the deformations during impacts, and efficient collision detection between fragments. Of course, the proposed methods have some limitations that draw interesting topics of research for future work, as discussed in the following.

An adaptive fracture state model

The resolution of the fragments from our fracture state model cannot be higher than the resolution of the volumetric mesh contained in the fracture state. This can be a desired feature in applications where the memory consumption has to be bounded and controlled, but in off-line simulation, this can be a limitation. A possible way to circumvent this problem would be to allow a dynamic sampling of nodes into the mesh of the fracture state model. However, geometrical and numerical problems may appear, and the global efficiency of the method would be compromised.

Other representations of the fracture surfaces

Our propagation algorithm may miss some features expected in some types of materials. For example, the branching phenomena during crack propagation is not handled and could not be defined with implicit surfaces. To extend the propagation algorithm, another representation of the fracture surface should be adopted, and the propagation order should be modified.

Energy released during fracture

Our impact-based approach that relies on modal analysis could also be extended in some ways. First, it could handle larger deformations using *e.g.* modal derivatives, or other extension of the modal analysis linear basis such as the ones presented in section 2.4.3.2. This would allow simulating the brittle fracture of less stiff materials. Introducing plasticity into the integration of the modal analysis steps could also allow extending our model to ductile fracture of stiff materials. Also, we saw that the close parallel fracture surfaces created in the simulation are not observed in the real experiments. We think this is due to the stress relaxation phenomenon that is not handled in our impact-based fracture approach. The stress relaxation method presented in the age-based fracturing section could be applied to the impact-based method, but at a price of a drastic drop in efficiency.

Handling the physics of the fragments

Our impact-based fracturing method relies on a precomputation of the vibration mode to compute efficiently the deformations at run-time. However, once the initial bodies for which the precomputations had been done break, the deformations can no longer be computed on the fragments. We saw that using a database of precomputed physical data could be a way to generate approximate deformations on the fragments. However, the possible number of fragment topologies is too big to have a representative database. An interesting idea for future work would be to re-use the initial deformation modes for the fragments whose volume is close to the volume of the initial bodies (small fragments can generally be well approximated with ellipsoids). It would be also interesting to find out whether it is possible to relate the changes in the fundamental vibration frequencies of a body w.r.t. its topology changes.

A continuous collision detection using triangles instead of vertices

Our collision detection method presents several limitations. First, the collision test is performed using point shell, preventing edge/edge collision to be detected. Even if we did not found that problematic since the density of vertices of the meshes used is high enough to avoid visible penetrations, this could be a problem for sparser sampled meshes. In this case, the elementary primitive tested should not be a point, but a surface primitive, such as a triangle. Our method could be extended by building the hierarchy from the set of triangles of the mesh instead of set of points. However, two issues should be studied. First, the pruning level of the hierarchy build with triangles will be less good than using vertices. Second, the collision queries should be rewritten to check for a triangle against a distance field instead of a point against a distance field, which would compromise the performances. The second main limitation of our collision detection algorithm is that we perform a discrete collision detection, raising problems with thins objects or big time steps. The extension of the method to continuous collision detection is not straightforward, and would be worth to study.

Future work for the evaluations of brittle fracture methods

The evaluations presented allowed us to assess the validity of several aspects of our simulation methods. However, the results presented on the real ceramic tiles and the glass slabs are still preliminary, and more study should be performed in that direction. It seems that the breaking tests are reproducible considering the statistical information of the geometry of the produced patterns. More tests should be performed to obtain significant values. Also, it is still not clear how the deformation energy is dissipated during the crack propagation. Breaking a tile by dropping a ball at twice more height will neither generate twice more fragments, nor twice more cracks. These aspects can still not be observed in the simulations. Also, a thin objects such as the glass slabs (2mm thick), materials weaknesses seem to have a great importance in the fracture outcome, as well as secondary waves effects. On the glass slab examples, the variance obtained on the height at which the slab broke is huge, and some aspects of the fracture patterns obtained are difficult to explain. We believe there is still plenty of room to understand fully the dynamic aspects of brittle fractures, and to be able to model them through macroscopic rules compatible with interactive simulations.

6.5 Long term perspectives

The contributions presented in this manuscript represent interesting advances in fracture simulation, and we hope it paved the way for future research, such as the long term projects

presented in the following.

Towards fully destructible virtual environments

In virtual environments, the freedom that the operator has to interact and to explore the environment is crucial for his sensation of immersion. Virtual environments are becoming more and more dynamic, and the possibilities of interactions have been constantly improving. For example, in recent video games, virtual worlds are populated with more and more animated characters, rigid bodies, clothes, vehicles and particles. All are simulated with physically-based approaches, providing great experiences for the players. However, it is still not possible to simulate fully destructible environments for several important technical issues.

The first issue is the **precomputed data**. A lot of precomputations are done on static data in virtual worlds. Structures such as navigation meshes, triangle meshes for rendering, collision detection structures as well as data for rendering such as baked textures, light maps, ... are all precomputed. In the case of a fully destructible environment, no heavy precomputation is possible, or it needs to be updated efficiently at each fracture event, which would imply great modifications of current simulation and rendering engines.

The second issue is the **graphic hardware**. Historically, graphics cards were designed to accelerate the rasterization of simple primitives on a 2-D screen, with a few input parameters to influence the position and aspects of the primitives rendered. However, the input triangle meshes to be rendered were supposed to be static. Nowadays, graphics cards evolved a lot, and the rendering pipelines is mainly composed of fully programmable stages such as vertex shaders, geometry shaders, and pixel shaders. However, the pipeline is still fed with vertices streams that are often stored on the graphic card memory, and static. The costly CPU/GPU memory transfers are discouraging for the generation of the triangle meshes on CPU, and a current optimal solution would be to handle all the physics only on the GPU to avoid these transfers. Therefore, current graphic hardware are still not designed to handle the generation of new meshes at run-time.

The third issue is the **memory consumption**. Allowing an unlimited number of new bodies to be created at run-time generate memory issues. In a game engine, each component has a memory and CPU budget that should be mastered. We believe that considering the material around points could be a solution to limit the memory consumption as presented in our fracture state model. However, it also bound the size of the smallest fragment that can be created. An adaptive subdivision with a maximum memory budget could be defined to circumvent this issue, while guaranteeing a boundary on the memory consumption. Nevertheless, this is a complex issue that has been rarely addressed so far.

Towards a real-time and verified multi-physics engines for virtual prototyping

More and more physics engines are becoming "multi-physics", namely they handle *e.g.* rigid body, soft body, and fluid simulations. Although very helpful for special effects in movies and games, their use in virtual prototyping applications is moderated because a few information on the **physical correctness** of the simulations that are provided. We think that a standardization of the physical correctness should be defined for each physical phenomenon, in order to provide an "accuracy mark" for each engine. The standard tests could be defined based on real-life examples accepted by the community, and easily accessible. Nowadays, relevant criteria are not well defined for each phenomenon to simulate. Also, validating independently each physical phenomenon does not guarantee that their coupling is validated as well. Conditions on the interfacing of the different phenomena such as solid and liquid coupling should be defined as well for verified multi-physics engines.

Real-time constraints generally imply simplifications and approximations on the simulations, but if the errors are quantified on each aspect of the simulation, the usability of each simulation method for a particular application could be defined formally. For these reasons, a great research on the validations and the conditions of coupling of multi-physics engines could be performed to ensure the quality of chosen criteria. We think that statistical approaches such as the one presented in this thesis for brittle fracture is a promising way to go for these validations.

Author's publications

The work presented in this manuscript led to the following publications:

International conferences:

- Loeiz Glondu and Sara C. Schvartzman Maud Marchal and Georges Dumont and Miguel A. Otaduy. *Efficient Collision Detection for Brittle Fracture.* In Proceedings of the ACM/Eurographics Symposium on Computer Animation. 2012. *Honorable mention award.*
- Loeiz Glondu and Lien Muguercia and Maud Marchal and Georges Dumont and Carles Bosch and Holly Rushmeier and George Drettakis. *Example-Based Fractured Appearance.* In Computer Graphics Forum. 2012.
- Loeiz Glondu and Maud Marchal and Georges Dumont. *Real-Time Simulation of Brittle Fracture Using Modal Analysis.* IEEE Transactions on Visualization and Computer Graphics. 2012.
- Loeiz Glondu and Benoit Legouis and Maud Marchal and Georges Dumont. *Precomputed Shape Database for Real-Time Physically-Based Simulation.* In Proceedings of VRIPHYS. 2011.
- Loeiz Glondu and Maud Marchal and Georges Dumont. *A New Coupling Scheme for Haptic Rendering of Rigid Bodies Interactions based on a Haptic Sub-World using a Contact Graph.* In Proceedings of EuroHaptics. 2010.
- Loeiz Glondu and Maud Marchal and Georges Dumont. *Evaluation of Physical Simulation Libraries for Haptic Rendering of Contacts Between Rigid Bodies.* In Proceedings of the ASME World Conference on Innovative Virtual Reality. 2010.
- Loeiz Glondu and Maud Marchal and Georges Dumont. *A Sub-world Coupling Scheme for Haptic Rendering of Physically-based Rigid Bodies Simulation.* In Proceedings of VRIPHYS. 2009.
- Laurent George and Maud Marchal and Loeiz Glondu and Anatole Lécuyer. *Combining Brain-Computer Interfaces and Haptics: Detecting Mental Workload to Adapt Haptic Assistance.* In Proceedings of EuroHaptics. 2012. *Best paper award.*

National conferences:

- Loeiz Glondu and Maud Marchal and Georges Dumont. *Accurate 6-DOFs Haptic Rendering of Frictional Contacts between many Rigid Bodies.* In Proceedings of journés de l'AFRV. 2010.
- Loeiz Glondu and Maud Marchal and Georges Dumont. *Simulation physique de fracture d'objets fragiles en temps réel.* In Proceedings of journés de l'AFIG. 2011. *Second best paper award.*

Bibliography

[Abdelaziz 08] Yazid Abdelaziz & Abdelmadjid Hamouine. *A survey of the extended finite element*. Computers and Structures, vol. 86, pages 1141–1151, 2008. 46

[Adachi 95] Yoshitaka Adachi, Takahiro Kumano & Kouichi Ogino. *Intermediate Representation for Stiff Virtual Objects*. In Proceedings of the Virtual Reality Annual International Symposium, 1995. 60, 118, 119

[Allard 10] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez & Paul G. Kry. *Volume Contact Constraints at Arbitrary Resolution*. In Proceedings of ACM SIGGRAPH, volume 29, August 2010. 50

[Andriot 07] Claude Andriot & Florian Gosselin. Le traité de la réalité virtuelle - tome 2, chapitre Commande d'une interface à retour d'effort, pages 203–216. Fush F. Moreau G., 2007. 56

[ARPACK] ARPACK. http://www.caam.rice.edu/software/ARPACK/. 86

[Astley 97] Olivier R. Astley & Vincent Hayward. *Real-time Finite-elements Simulation of General Visco-elastic Materials for Haptic Presentation*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1997. 62, 119

[Avril 09] Quentin Avril, Valérie Gouranton & Bruno Arnaldi. *New Trends in Collision Detection Performance*. In Simon Richir & Akihiko Shirai, editeurs, Proceedings of Laval Virtual VRIC, pages 53–62, 2009. 48

[Balaniuk 99] Remis Balaniuk. *Using Fast Local Modelling to Buffer Haptic Data*. In Proceedings of Fourth PHANTOM Users Group Workshop, 1999. 61, 119

[Bandi 95] S. Bandi & D. Thalmann. *An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies*. In Computer Graphics Forum, volume 14, pages 259–270, 1995. 50

[Bao 07] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran & Ronald Fedkiw. *Fracturing Rigid Materials*. IEEE Transactions on Visualization and Computer Graphics, vol. 13, pages 370–378, 2007. 8, 45

[Baraff 89] David Baraff. *Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies*. ACM Transactions on Graphics, vol. 23, no. 3, pages 223–232, 1989. 53

[Baraff 90] David Baraff. *Curved surfaces and coherence for non-penetrating rigid body simulation*. In Proceedings of ACM SIGGRAPH, pages 19–28, 1990. 50

[Baraff 91] David Baraff. *Coping with Friction for Non-Penetrating Rigid Body Simulation.* In Proceedings of ACM SIGGRAPH, pages 31–41, 1991. 53, 110

[Baraff 94] David Baraff. *Fast Contact Force Computation for Nonpenetrating rigid Bodies.* In Proceedings of ACM SIGGRAPH, pages 23–34. ACM Press New York, NY, USA, 1994. 53

[Baraff 98] David Baraff & Andrew Witkin. *Large Steps in Cloth Simulation.* In Proceedings of ACM SIGGRAPH, pages 43–54, 1998. 24, 38

[Baraff 01] David Baraff. *Implicit Methods for Differential Equations.* In SIGGRAPH Course Notes, 2001. 24

[Barbič 08] J. Barbič & D. L. James. *Six-DoF Haptic Rendering of Contact between Geometrically Complex Reduced Deformable Models.* IEEE Transactions on Haptics, vol. 1, no. 1, 2008. 100, 101

[Barbič 07a] Jernej Barbič. *Real-time reduced large-deformation models and distributed contact for computer graphics and haptics.* PhD thesis, 2007. 39, 41, 49, 60, 80

[Barbič 07b] Jernej Barbič & Doug L. James. *Time-critical Distributed Contact for 6-DoF Haptic Rendering of Adaptively Sampled Reduced Deformable Models.* In Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation, volume 2, pages 171–180, 2007. 40, 49

[Barbič 11] Jernej Barbič & Yili Zhao. *Real-time large-deformation substructuring.* In Proceedings of ACM SIGGRAPH, pages 91:1–91:8, 2011. 40, 80

[Battlefield 12] Battlefield. http://www.battlefield.com/fr/battlefield3, 2012. 15

[Beaumont 02] M.A. Beaumont, W. Zhang & D.J. Balding. *Approximate Bayesian computation in population genetics.* Genetics, vol. 162, no. 4, pages 2025–2035, 2002. 131

[Belytschko 99] Ted Belytschko & T. Black. *Elastic crack growth in finite elements with minimal remeshing.* International Journal for Numerical Methods in Engineering, vol. 45, no. 5, pages 601–620, 1999. 46

[Bielser 03] Daniel Bielser, Pascal Glardon, Matthias Teschner & Markus Gross. *A State Machine for Real-Time Cutting of Tetrahedral Meshes.* In Proceedings of Pacific Conference on Computer Graphics and Applications, pages 385–415, 2003. 42, 45

[Bonet 97] J. Bonet & R.D. Wood. Nonlinear continuum mechanics for finite element analysis. Cambridge Univ Pr, 1997. 29

[Bridson 02] Robert Bridson, Ronald P. Fedkiw & John Anderson. *Robust Treatment of Collisions, Contact, and Friction for Cloth Animation.* ACM Transactions on Graphics, vol. 21, pages 594–603, 2002. 45

[Bustos 05] Benjamin Bustos, Daniel A. Keim, Dietmar Saupe, Tobias Schreck & Dejan V. Vranić. *Feature-based similarity search in 3D object databases.* ACM Computer Survey, vol. 37, pages 345–387, 2005. 84

[Cavusoglu 00] Murat Cenk Cavusoglu & Frank Tendick. *Multirate Simulation for High Fidelity Haptic Interaction with Deformable Objects in Virtual Environments*. Proceedings of IEEE International Conference on Robotics and Automation, vol. 3, pages 2458–2465, 2000. 62, 118, 119

[Chang 97] Beeling Chang & J. Edward Colgate. *Real-time Impulse-Based Simulation of Rigid Body Systems for Haptic Display*. In Proceedings of the ASME Interational Mechanical Engineering Congress and Exhibition, pages 1–8, 1997. 60

[Che 06] Yinghui Che, Jing Wang & Xiaohui Liang. *Real-time Deformation using Modal Analysis on Graphics Hardware*. In Proceedings of GRAPHITE, pages 173–176, 2006. 77

[Chowdhury 09] Ananda S. Chowdhury, Suchendra M. Bhandarkar, Robert W. Robinson & Jack C. Yu. *Virtual multi-fracture craniofacial reconstruction using computer vision and graph matching*. Computerized Medical Imaging and Graphics, vol. 33, no. 5, pages 333–342, 2009. 126

[Chung 96] Kelvin Chung & Wenping Wang. *Quick elimination of non-interference polytopes in virtual environments*. In Proceedings of the Eurographics workshop on Virtual environments and scientific visualization, pages 64–73, 1996. 50

[Cloos 55] E Cloos. *Experimental analysis of fracture patterns*. Geological Soc. Of America Bulletin, vol. 66, no. 3, pages 241–256, 1955. 126

[Colgate 95] J. E. Colgate, M. C. Stanley & J. M. Brown. *Issues in the Haptic Display of Tool Use*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 3, pages 140–144, 1995. 16, 59, 109, 118

[Constantinescu 05] D. Constantinescu, S. Salcudean & E. Croft. *Haptic Rendering of Rigid Contacts using Impulsive and Penalty Forces*. IEEE Transactions on Robotics, vol. 21, no. 3, pages 309–323, 2005. 60

[Cotterell 65] B. Cotterell. *On brittle fracture paths*. International Journal of Fracture, vol. 1, no. 2, pages 96–103, 1965. 70

[Criswell 11] Brice Criswell, Jeff Smith & David Deuber. *Deformable rigid bodies and fragment clustering for film*. In SIGGRAPH Course Notes, 2011. 15

[Davanne 02] Jérôme Davanne, Philippe Meseure & Christophe Chaillou. *Stable Haptic Interaction in a Dynamic Virtual Environment*. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 3, pages 2881–2886 vol.3, 2002. 119

[Desbenoit 05] Brett Desbenoit, Eric Galin & Samir Akkouche. *Modeling Cracks and Fractures*. The Visual Computer, vol. 21, no. 8-10, pages 717–726, 2005. 8, 42

[Dick 11] Christian Dick, Joachim Georgii & Ruediger Westermann. *A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects*. IEEE Transactions on Visualization and Computer Graphics, vol. 17, pages 1663–1675, 2011. 42

[Dobkin 90] David P. Dobkin & David G. Kirkpatrick. *Determining the Separation of Preprocessed Polyhedra - A Unified Approach.* In Proceedings of the International Colloquium on Automata, Languages and Programming, pages 400–413, 1990. 50

[Elices 02] M. Elices, G.V. Guinea, J. Gomez & J. Planas. *The cohesive zone model: advantages, limitations and challenges.* Engineering Fracture Mechanics, vol. 69, no. 2, pages 137–163, 2002. 42

[Erleben 07] Kenny Erleben. *Velocity-Based Shock propagation for Multibody Dynamics Animation.* ACM Transactions on Graphics, vol. 26, no. 2, pages 1–20, 2007. 54, 108

[Etzmuβ 03] Olaf Etzmuβ, Michael Keckeisen & Wolfgang Straβer. *A Fast Finite Element Solution for Cloth Modelling.* In Proceedings of the Pacific Conference on Computer Graphics and Applications, page 244, Washington, DC, USA, 2003. IEEE Computer Society. 33

[Faure 08] François Faure, Jérémie Allard, Florent Falipou & Sébastien Barbier. *Image-based Collision Detection and Response between Arbitrary Volumetric Objects.* In ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 155–162, 2008. 50

[Faure 11] François Faure, Benjamin Gilles, Guillaume Bousquet & Dinesh K. Pai. *Sparse Meshless Models of Complex Deformable Solids.* ACM Transactions on Graphics, 2011. 36

[Fisher 01] S. Fisher & M. C. Lin. *Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Fields.* Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001. 49

[Frisken 00] S. Frisken, R. Perry, A. Rockwood & R. Jones. *Adaptively Sampled Distance Fields: A General Representation of Shapes for Computer Graphics.* In Proceedings of ACM SIGGRAPH, pages 249–254, 2000. 49

[Funkhouser 11] Thomas Funkhouser, Hijung Shin, Corey Toler-Franklin, Antonio García Castañeda, Benedict Brown, David Dobkin, Szymon Rusinkiewicz & Tim Weyrich. *Learning how to match fresco fragments.* Journal on Computing and Cultural Heritage, vol. 4, pages 7:1–7:13, 2011. 126

[Gibson 97] Sarah F. F. Gibson & Brian Mirtich. *A Survey of Deformable Modeling in Computer Graphics.* MERL Technical Report Cambridge, vol. 19, 1997. 30, 38

[Gilbert 88] Elmer G. Gilbert, Daniel W. Johnson & Sathiy S. Keerthi. *A fast procedure for computing the distance between complex objects in three-dimensional space.* IEEE Journal of Robotics and Automation, vol. 4, pages 193–203, 1988. 50

[Glondu 10] Loeiz Glondu, Maud Marchal & Georges Dumont. *Evaluation of Physical Simulation Libraries for Haptic Rendering of Contacts Between Rigid Bodies.* In Proceedings of ASME World Conference on Innovative Virtual Reality, 2010. 77

[Gregory 99] Arthur Gregory, Ming C. Lin, Stefan Gottschalk & Russell Taylor. *A Framework for Fast and Accurate Collision Detection for Haptic Interaction.* In Proceedings of the IEEE Virtual Reality Conference, 1999. 60

[Griffith 21] Alan A. Griffith. *The Phenomena of Rupture and Flow in Solids.* Royal Society of London Philosophical Trans. Series A, vol. 221, pages 163–198, 1921. 14, 126

[Gross 06] D. Gross & T. Seelig. Fracture mechanics: with an introduction to micromechanics. Springer Verlag, 2006. 76

[Guendelman 03] Eran Guendelman, Robert Bridson & Ronald Fedkiw. *Nonconvex Rigid Bodies with Stacking.* ACM Transactions on Graphics, vol. 22, no. 3, pages 871–878, 2003. 49, 54

[Gumerov 04] Nail Gumerov & Ramani Duraiswami. Fast multipole methods for the helmholtz equation in three dimensions. Elsevier, 2004. 86

[Hahn 88] James K. Hahn. *Realistic Animation of Rigid Bodies.* In Proceedings of ACM SIGGRAPH, pages 299–308. ACM New York, NY, USA, 1988. 54, 76

[Hauser 03] Kris K. Hauser, Chen Shen & James F. O'Brien. *Interactive Deformation Using Modal Analysis with Constraints.* In Graphics Interface, pages 247–256, 2003. 40

[Havok] Havok. www.havok.com. 77

[Heczko 02] Martin Heczko, Daniel A. Keim, Dietmar Saupe & Dejan V. Vranic. *A method for similarity search of 3D objects.* Datenbank-spektrum, vol. 2, pages 54–63, 2002. 84

[Heidelberger 04] B. Heidelberger, M. Teschner, R. Keiser, M. Müeller & M. Gross. *Consistent Penetration Depth Estimation for Deformable Collision Response.* Proceedings of Vision, Modeling and Visualization, 2004. 98

[Heo 10] J.-P. Heo, J.-K. Seong, D. Kim, M. A. Otaduy, J.-M. Hong, M. Tang & S.-E. Yoon. *FASTCD: Fracturing-Aware Stable Collision Detection.* Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2010. 51

[Hilaga 01] Masaki Hilaga, Yoshihisa Shinagawa, Taku Kohmura & Tosiyasu L. Kunii. *Topology matching for fully automatic similarity estimation of 3D shapes.* In Proceedings of ACM SIGGRAPH, pages 203–212, 2001. 84

[Hirota 98] Koichi Hirota, Yasuyuki Tanoue & Toyohisa Kaneko. *Generation of Crack Patterns With a Physical Model.* The Visual Computer, vol. 14, pages 126–137, 1998. 38, 42

[Hoffmann 89] Christoph M. Hoffmann. Geometric and solid modeling: an introduction. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. 48

[Huang 10] Jin Huang, Yiying Tong, Kun Zhou, Hujun Bao & Mathieu Desbrun. *Interactive Shape Interpolation through Controllable Dynamic Deformation.* IEEE Transactions on Visualization and Computer Graphics, vol. 99, pages 1–8, 2010. 40, 80

[Iben 06] Hayley N. Iben & James F. O'Brien. *Generating Surface Crack Patterns.* In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 177–185, Sept 2006. 8, 44, 45, 81, 82

[Irving 04] Geoffrey Irving, Joseph Teran & Ronald Fedkiw. *Invertible Finite Elements for Robust Simulation of Large Deformation.* In Proceedings of the SIGGRAPH/Eurographics symposium on Computer animation, pages 131–140, Aire-la-Ville, Switzerland, Switzerland, 2004. 33

[Irwin 57] George R. Irwin. *Analysis of stresses and strains near the end of a crack traversing a plate.* Journal of Applied Mechanics, pages 361–364, 1957. 15

[James 02] Doug L. James & Dinesh K. Pai. *DyRT: dynamic response textures for real time deformation simulation with graphics hardware.* In Proceedings of ACM SIGGRAPH, pages 582–585, 2002. 40

[James 06] Doug L. James, Jernej Barbič & Dinesh K. Pai. *Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources.* In Proceedings of ACM SIGGRAPH, pages 987–995, 2006. 86

[Jeřábková 09] Lenka Jeřábková & Torsten Kuhlen. *Stable Cutting of Deformable Objects in Virtual Environments Using XFEM.* IEEE Computer Graphics and Applications, vol. 29, pages 61–71, 2009. 46

[Johnson 87] K.L. Johnson. Contact mechanics. Cambridge Univ Pr, 1987. 74

[Kaufman 05] Danny M. Kaufman, Timoty Edmunds & Dinesh K. Pai. *Fast Frictional Dynamics for Rigid Bodies.* In International Conference on Computer Graphics and Interactive Techniques, volume 24, pages 946–956, 2005. 54

[Kaufman 08] Danny M. Kaufman, Shinjiro Sueda, Doug L. James & Dinesh K. Pai. *Staggered Projections for Frictional Contact in Multibody Systems.* ACM Transactions on Graphics, vol. 27, no. 5, pages 164:1–164:11, 2008. 55, 109

[Kaufmann 09] Peter Kaufmann, Sebastian Martin, Mario Botsch, Eitan Grinspun & Markus Gross. *Enrichment textures for detailed cutting of shells.* In Proceedings of ACM SIGGRAPH, pages 501–510, 2009. 46

[Keyser 97] J. Keyser, S. Krishnan & D. Manocha. *Efficient and accurate B-rep generation of low degree sculptured solids using exact arithmetic.* In Proceedings of ACM symposium on Solid modeling and applications, pages 42–55. ACM, 1997. 49

[Klosowski 98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral & K. Zikan. *Efficient collision detection using bounding volume hierarchies of k-DOPs.* IEEE Transactions on Visualization and Computer Graphics, vol. 4, pages 21–36, 1998. 50

[LAPACK] LAPACK. http://www.netlib.org/lapack/. 86

[Larsson 01] T. Larsson & T. Akenine-Möller. *Collision Detection for Continuously Deforming Bodies.* Eurographics, 2001. 51

[Lin 91] Ming C. Lin & John F. Canny. *A fast algorithm for incremental distance calculation.* In Proceedings of IEEE International Conference on Robotics and Automation, pages 1008–1014. IEEE, 1991. 50

[Lin 95] Ming C. Lin & D. Manocha. *Fast interference detection between geometric models.* The Visual Computer, vol. 11, no. 10, pages 542–561, 1995. 48

[Lin 98] Ming C. Lin & Stefan Gottschalk. *Collision Detection Between Geometric Models: A Survey.* In Proceedings of IMA Conference on Mathematics of Surfaces, pages 37–56, 1998. 48

[Luciano 05] Cristian Luciano, Pat Banerjee, Lucian Florea & Greg Dawe. *Design of the* ImmersiveTouch*: a High-Performance Haptic Augmented Virtual Reality System.* In Proceedings of Salvendy G: Human Computer International, 2005. 62

[Mark 96] William R. Mark, Scott C. Randolph, Mark Finch, James M.Van Verth & Russell M. Taylor II. *Adding Force Feedback to Graphics Systems: Issues and Solutions.* In Proceedings of ACM SIGGRAPH, pages 447–452, 1996. 60, 119

[McDanels 06] S. J. McDanels, B. M. Mayeaux, T. E. Collins, G. A. Jerman, R. S. Piascik, R. W. Russell & S. R. Shah. *An Overview of the Space Shuttle Columbia Accident from Recovery Through Reconstruction.* Journal of Failure Analysis Prevention, vol. 6, no. 1, pages 82–91, 2006. 126

[McNeely 99] William A. McNeely, Kevin D. Puterbaugh & James J. Troy. *Six degree-of-freedom haptic rendering using voxel sampling.* In Proceedings of ACM SIGGRAPH, pages 401–408, 1999. 49

[Mendoza 00] Cesar Mendoza & Christian Laugier. *A Solution for the Difference Rate Sampling between Haptic Devices and Deformable Virtual Objects.* In International Symposium on Robotics and Automation, 2000. 61, 119

[Meseure 07] Philippe Meseure & Abderrahamne Kheddar. Le traité de la réalité virtuelle - tome 3, chapitre Modèles pour le rendu haptique, pages 141–154. Fush F. Moreau G., 2007. 60

[Milenkovic 01] Victor J. Milenkovic & Harald Schmidl. *Optimization-Based Animation.* In Proceedings of ACM SIGGRAPH, pages 37–46, 2001. 54, 109

[Mirtich 96] B.V. Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD thesis, University of California, 1996. 54

[Moës 99] Nicolas Moës, John Dolbow & Ted Belytschko. *A finite element method for crack growth without remeshing.* International journal for numerical methods in engineering, vol. 46, pages 131–150, 1999. 46

[Molino 04] Neil Molino, Zhaosheng Bao & Ron Fedkiw. *A Virtual Node Algorithm for Changing Mesh Topology during Simulation.* In Proceedings of ACM SIGGRAPH, page 4, 2004. 45, 102

[Moore 88] Matthew Moore & Jane Wilhelms. *Collision Detection and Response for Computer Animation.* Proceedings of ACM SIGGRAPH, vol. 22, no. 4, pages 289–298, 1988. 54

[movie2012 09] movie2012. http://www.sonypictures.com/homevideo/2012/, 2009. 15

[Müller 01] Matthias Müller, Leonard McMillan, Julie Dorsey & Robert Jagnow. *Real-time Simulation of Deformation and Fracture of Stiff Materials*. In Proceedings of the Eurographic workshop on Computer animation and simulation, pages 113–124, 2001. 44, 80, 98

[Müller 02] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow & Barbara Cutler. *Stable Real-time Deformations*. In Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 49–54, 2002. 33, 34

[Müller 04a] Matthias Müller & Markus Gross. *Interactive Virtual Materials*. In Proceedings of Graphics Interface, pages 239–246, 2004. 34, 35, 44, 45, 82, 99

[Müller 04b] Matthias Müller, Richard Keiser, Andrew Nealen, Mark Pauly, Markus Gross & Marc Alexa. *Point Based Animation of Elastic, Plastic and Melting Objects*. In Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 141–151, 2004. 36, 41

[Müller 04c] Matthias Müller, Matthias Teschner & Markus Gross. *Physically-Based Simulation of Objects Represented by Surface Meshes*. In Proceedings of the Computer Graphics International, pages 26–33, Washington, DC, USA, 2004. IEEE Computer Society. 44

[Müller 05] Matthias Müller, Bruno Heidelberger, Matthias Teschner & Markus Gross. *Meshless Deformations Based on Shape Matching*. ACM Transactions on Graphics, vol. 24, no. 3, pages 471–478, 2005. 36

[Müller 07] Matthias Müller, Bruno Heidelberger, Marcus Hennix & John Ratcliff. *Position Based Dynamics*. Journal of Visual Communication and Image Representation, vol. 18, no. 2, pages 109–118, 2007. 52, 55

[Müller 08a] Matthias Müller. *Hierarchical Position Based Dynamics*. In Proceedings of Virtual Reality Interactions and Physical Simulations, 2008. 52, 55

[Müller 08b] Matthias Müller, Joe Stam, Doug James & Nils Thürey. *Real time physics: class notes*. In ACM SIGGRAPH Course notes, page 88, 2008. 23

[Nealen 06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman & Mark Carlson. *Physically Based Deformable Models in Computer Graphics*. Computer Graphics Forum, vol. 25, no. 4, pages 1–24, 2006. 30

[Neff 99] Michael Neff & Eugene Fiume. *A Visual Model for Blast Waves and Fracture*. In Proceedings of the conference on Graphics interface, pages 193–202, 1999. 42

[Nesme 05] Matthieu Nesme, Yohan Payan & François Faure. *Efficient, Physically Plausible Finite Elements*. In Proceedings of Eurographics, 2005. 33

[Norton 91] Alan Norton, Greg Turk, Bob Bacon, John Gerth & Paula Sweeney. *Animation of Fracture by Physical Modeling*. Visual Computer, vol. 7, no. 4, pages 21–219, 1991. 42

[O'Brien 99] James F. O'Brien & Jessica K. Hodgins. *Graphical Modeling and Animation of Brittle Fracture*. In Proceedings of ACM SIGGRAPH, pages 137–146, 1999. 8, 15, 43, 44, 98

[O'Brien 02a] James F. O'Brien, Adam W. Bargteil & Jessica K. Hodgins. *Graphical Modeling and Animation of Ductile Fracture*. ACM Transactions on Graphics, vol. 21, no. 3, pages 291–294, 2002. 44

[O'Brien 02b] James F. O'Brien, Chen Shen & Christine M. Gatchalian. *Synthesizing Sounds from Rigid-body Simulations*. In Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 175–181, 2002. 40, 86

[Oda 05] Ohan Oda & Stephen Chenney. *Fast Dynamic Fracture of Brittle Objects*. In ACM SIGGRAPH Posters, page 113, New York, NY, USA, 2005. ACM. 45

[Osada 02] Robert Osada, Thomas Funkhouser, Bernard Chazelle & David Dobkin. *Shape distributions*. ACM Transaction on Graphics, vol. 21, pages 807–832, 2002. 84

[Otaduy 05] Miguel A. Otaduy & Ming C. Lin. *Sensation Preserving Simplification for Haptic Rendering*. In International Conference on Computer Graphics and Interactive Techniques, pages 72–83, 2005. 61, 119

[Otaduy 06] Miguel A. Otaduy & Ming C. Lin. *A Modular Haptic Rendering Algorithm for Stable and Transparent 6-dof Manipulation*. IEEE Transactions on Robotics, vol. 22, no. 4, pages 751–762, 2006. 60, 61

[Otaduy 07] Miguel A. Otaduy, O. Chassot, D. Steinemann & Markus Gross. *Balanced Hierarchies for Collision Detection between Fracturing Objects*. In Proceedings of IEEE Virtual Reality Conference, 2007. 51

[Palmer 94] I. J. Palmer & R. L. Grimsdale. *Collision Detection for Animation using Sphere-Trees*. Computer Graphics Forum, vol. 14, no. 2, pages 105–116, 1994. 50

[Paquet 00] E. Paquet, A. Murching, T. Naveen, A. Tabatabai & M. Rioux. *Description of shape information for 2-D and 3-D objects*. Signal Processing: Image Communication, vol. 16, pages 103–122, 2000. 84

[Parker 09] Eric G. Parker & James F. O'Brien. *Real-Time Deformation and Fracture in a Game Environment*. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 156–166, 2009. 33, 45

[Pauly 05] Mark Pauly, Richard Keiserand, Bart Adams, Philip Dutré, Markus Gross & Leonidas J. Guibas. *Meshless Animation of Fracturing Solids*. ACM Trans. Graph., vol. 24, no. 3, pages 957–964, 2005. 9, 36, 46, 47

[Pele 08] Ofir Pele & Michael Werman. *A Linear Time Histogram Metric for Improved SIFT Matching*. In ECCV, pages 495–508, 2008. 132

[Pentland 89] A. Pentland & J. Williams. *Good vibrations: modal dynamics for graphics and animation*. ACM Transactions on Graphics, vol. 23, no. 3, pages 207–214, 1989. 40, 80

[Perlin 02] K. Perlin. *Improving noise.* ACM Transactions on Graphics, vol. 21, pages 681–682, July 2002. 70

[PhysX] PhysX. www.nvidia.com/object/physx_new.html. 77

[Picinbono 99] Guillaume Picinbono & Jean-Christophe Lombardo. *Extrapolation: a Solution for Force Feedback ?* volume 1, pages 117–125, 1999. 61

[Pietroni 09] Nico Pietroni, Fabio Ganovelli, Paolo Cignoni & Roberto Scopigno. *Splitting cubes: a fast and robust technique for virtual cutting.* Visual Computer, vol. 25, no. 3, pages 227–239, 2009. 42

[Pocheville 04] Aurélien Pocheville & Abderrahmane Kheddar. *I-Touch: a Framework for Computer Haptics.* In Workshop on Touch and Haptics. IEEE/RSJ IROS, 2004. 62

[Ratatouille 07] Ratatouille. http://www.disney.fr/ratatouille/, 2007. 15

[Renz 01] M. Renz, C. Preusche, M. Pötke, H. Kriegel & G. Hirzinger. *Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm.* In Proceedings of EuroHaptics. Citeseer, 2001. 49

[Rubner 98] Yossi Rubner, Carlo Tomasi & Leonidas J. Guibas. *A metric for distributions with applications to image databases.* In ICCV, pages 59–66, 1998. 131

[Ruffaldi 08] Emanuele Ruffaldi, Dan Morris, Federico Barbagli, Ken Salisbury & Massimo Bergamasco. *Voxel-Based Haptic Rendering Using Implicit Sphere Trees.* In Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, pages 319–325, 2008. 60

[Ruspini 00] D. Ruspini & O. Khatib. *A Framework for Multi-Contact Multi-Body Dynamic Simulation and Haptic Display.* In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, 2000. 62

[Samet 95] Hanan Samet. *Spatial data structures.* Modern Database Systems, The Object Model, Interoperability and Beyond, pages 361–385, 1995. 50

[Shewchuk 94] Jonathan Richard Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Rapport technique, CMU-CS-94-125, 1994. 25

[Shin 10] Hijung Shin, Christos Doumas, Thomas Funkhouser, Szymon Rusinkiewicz, Kenneth Steiglitz, Andreas Vlachopoulos & Tim Weyrich. *Analyzing Fracture Patterns in Theran Wall Paintings.* In VAST, 2010. 126

[Sifakis 07] Eftychios Sifakis, Kevin G. Der & Ronald Fedkiw. *Arbitrary cutting of deformable tetrahedralized objects.* In Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 73–80, 2007. 42

[Smith 01] Jeffrey Smith, Andrew Witkin & David Baraff. *Fast and Controllable Simulation of the Shattering of Brittle Objects.* Computer Graphics Forum, vol. 20, pages 81–91, 2001. 43

[Spillmann 07] Jonas Spillmann, Markus Becker & Matthias Teschner. *Non-iterative Computation of Contact Forces for Deformable Objects.* Journal of WSCG, vol. 15, no. 1-3, pages 33–40, 2007. 53

[Steinemann 06] Denis Steinemann, Miguel A. Otaduy & Markus Gross. *Fast Arbitrary Splitting of Deforming Objects.* In Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 63–72, 2006. 49, 99

[Stewart 00] D. Stewart & J.C. Trinkle. *An Implicit Time-stepping Scheme for Rigid Body Dynamics with Coulomb Friction.* In IEEE International Conference on Robotics and Automation, volume 1, pages 162–169, 2000. 52, 54

[Sud 06] A. Sud, N. K. Govindaraju, R. Gayle & D. Manocha. *Interactive 3D Distance Field Computation Using Linear Factorization.* Proceedings of ACM Symposium on Interactive 3D Graphics and Games, 2006. 49

[Sukumar 00] N. Sukumar, N. Moës, B. Moran & T. Belytschko. *Extended finite element method for three-dimensional crack modelling.* International Journal for Numerical Methods in Engineering, vol. 48, pages 1549–1570, 2000. 46

[Teran 03] J. Teran, S. Blemker, V. Hing & R. Fedkiw. *Finite volume methods for the simulation of skeletal muscle.* In Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 68–74. Eurographics Association, 2003. 31

[Terzopoulos 87] Demetri Terzopoulos, John Platt, Alan Barr & Kurt Fleischer. *Elastically Deformable Models.* ACM Transactions on Graphics, vol. 21, no. 4, pages 205–214, 1987. 30

[Terzopoulos 88] Demetri Terzopoulos & Kurt Fleischer. *Modeling Inelastic Deformation: Viscolelasticity, Plasticity, Fracture.* In Proceedings of ACM SIGGRAPH, volume 22, pages 287–296, 1988. 15, 30, 43

[Teschner 05] M. Teschner, Kimmerle S., B. Heidelberger, Zachmann G., Raghupathi L., Futhrmann A., Cani M.-P., Faure F., Faure N., Fagnetat-Thalmann N.and Strasser W. & Volino P. *Collision Detection for Deformable Objects.* In Computer Graphics Forum, volume 24, pages 61–81, 2005. 48, 50, 51

[TetGen] TetGen. http://tetgen.berlios.de/. 85

[Viet 06] Huynh Quang Huy Viet, Takahiro Kamada & Hiromi T.Tanaka. *An Algorithm for Cutting 3D Surface Meshes.* Proceedings of the International Conference on Pattern Recognition, vol. 4, pages 762–765, 2006. 42

[Vranic 01a] D. V. Vranic & D. Saupe. *3D Shape Descriptor Based on 3D Fourier Transform.* In Proceedings of EURASIP, pages 271–274, 2001. 84

[Vranic 01b] D. V. Vranic, D. Saupe & J. Richter. *Tools for 3D-object retrieval: Karhunen-Loeve Transform and spherical harmonics.* In IEEE International Workshop on Multimedia and Signal Processing, pages 293–298, 2001. 84

[Wan 11] *International Journal on Interactive Design and Manufacturing.* International Journal on Interactive Design and Manufacturing, 2011. 40

[Weller 09] R. Weller & G. Zachmann. *Inner Sphere Trees for Proximity and Penetration Queries*. Robotics: Science and Systems, 2009. 50

[Witkin 90] Andrew Witkin, Michael Gleicher & William Welch. *Interactive Dynamics*. ACM Transactions on Graphics, vol. 24, no. 2, pages 11–21, 1990. 53

[Witkin 01a] Andrew Witkin. *Constrained Dynamics*. In Physically Based Modeling, 2001. 53

[Witkin 01b] Andrew Witkin & David Baraff. *Differential Equation Basics*. In Physically Based Modeling, 2001. 23

[Zafar 11] Nafees B. Zafar & Mark Carlson. *Destruction and dynamics artist tools for film*. In SIGGRAPH Course Notes, 2011. 43

[Zheng 10] Changxi Zheng & Doug L. James. *Rigid-Body Fracture Sound with Precomputed Soundbanks*. In Proceedings of ACM SIGGRAPH, volume 29, July 2010. 13, 45, 80, 83, 89

[Zilles 95] C. B. Zilles & J. K. Salisbury. *A constraint-based god-object method for haptic display*. In Proceedings of the International Conference on Intelligent Robots and Systems, volume 3, pages 31–46, 1995. 58