Davide Zerbato

Frictional Contact in Interactive Deformable Environments

Ph.D. Thesis

June 30, 2010

Università degli Studi di Verona
Dipartimento di Informatica

Advisor:
prof. Paolo Fiorini

Series N°: **TD-11-10**

Università di Verona
Dipartimento di Informatica
Strada le Grazie 15, 37134 Verona
Italy

No bird soars too high, if he soars with his own wings
W. Blake

Summary

The use of simulations provides great advantages in term of economy, realism, and adaptability to user requirements in many research and technological fields. For this reason simulations are currently exploited, for example, in prototyping of machinery parts, in assembly-disassembly test or training and, recently, simulations have also allowed the development of many useful and promising tools for the assistance and learning of surgical procedures. This is particularly true for laparoscopic intervention.

Laparoscopy, in fact, represents the gold standard for many surgical procedures. The principal difference from standard surgery is the reduction of the surgeon ability to perceive the surgical scenario, both from visual and tactile point of view. This represents a great limitation for surgeons who undergo long training before being able to perform laparoscopic intervention with proficiency. This, on the other hand, makes laparoscopy an excellent candidate for the use of simulations for training.

Some commercial training softwares are already available on the market, but they are usually based on rigid body models that completely lack the physical realism. The introduction of deformable models may leads to a great increment in terms of realism and accuracy. And, in the case of laparoscopy trainer it may allow the user to learn not only basic motor skills, but also higher level capabilities and knowledge. Rigid bodies, in fact, represents a good approximation of reality only in some situations and in very restricted ranges of solicitations. In particular, when non engineering materials are involved, as happens in surgical simulations, deformations cannot be neglected without completely loosing the realism of the environment. The use of deformable models, however, is limited for the high computational costs involved in the computation of the physics undergoing the deformations and because of the reduction in pre computable data in particular for collision detection between bodies. This represents a very limiting factor in interactive environments where, to allow the user to interactively control the virtual bodies, the simulation should be performed in real time.

In this thesis we address the simulation of interactive environment populated with deformable models that interact with frictional contacts. This includes the analysis and the development of different techniques which implement the various parts of the simulation: mainly the methods for the simulation of deformable mod-

els, the collision detection and collision solution techniques but also the modeling and the integration of suitable friction models in the simulation.

In particular we evaluated the principal methods that represent the state of the art in soft tissue modeling. Our analysis is based on the physical background of each method and thus on its realism in terms of deformations that the method can mimic and on the ease of use (i.e. method understanding, calibration and ability to adapt to different scenarios) but we also compared the computational complexity of different models, as it represents an extremely important factor in the choice and in the use of models in simulations.

The comparison of different features in analyzed methods motivated us to the development of an innovative method to wrap in a common representation framework different methodologies of soft tissue simulation. This framework has the advantage of providing a unified interface for all the deformable models and thus it provides the ability to switch between deformable model keeping unchanged all other data structures and methods of the simulation.

The use of this unique interface allows us to use one single method to perform the collision detection phase for all the analyzed deformable models, this greatly helped during the identification of requirements and features of such software module. Collision detection phase, when applied to rigid bodies, usually takes advantage of pre computation to subdivide body shapes in convex elements or to construct partitions of the space in which the body is defined to speed up the computation. When handling deformable models this is not possible because of the continuous changes in bodies shape. The collision detection method used in this work takes into account this problem and regularly adapt the data structures to the body configuration.

After collisions have been detected and contact points have been identified on colliding bodies, it is necessary to solve the collision in a physics based way. To this extent we have to ensure that objects never compenetrate during the simulation and that, when solving collisions, all the physical phenomena involved in the contact of real bodies are taken into account: this include the elastic response of bodies during the contact and the frictional force exerted between each pair of colliding bodies. The innovative method for solving collision that we describe in this thesis ensures the realism of the simulation and the seamless interaction with the common framework used to integrate deformable models.

One important feature of biologic tissues is their anisotropic behavior that usually comes from the fibrous structure of these tissues. In this thesis we propose a new method to introduce anisotropy in mass spring model. The method has the advantages of preserving the speed and ease of implementation of the model and it effectively introduces differentiation of the model behavior along the chosen directions.

The described techniques have been integrated in two applications that allows the physical simulation of environments populated with deformable models. The first application implements all the described methods to simulate deformable models, it performs precise collision detection and solution with the possibility to chose the most suitable friction model for the simulation. It demonstrates the effectiveness of the proposed framework. The main limitation of this simulator, i.e. its high computation time, is tackled and solved in a second application that

exploits the intrinsic parallelism of physical simulations to optimize the implementation and to exploit parallel architecture computational power. To obtain the performances required for an interactive environment the simulation is based on a simplified collision detection algorithm, but it features all the other techniques described in this thesis. The parallel implementation exploits graphic cards processor, a highly parallel architecture that update the scene every milliseconds. This allows the rendering of smooth haptic feedback to the user and ensures the realism of the physics simulation.

The implemented applications prove the feasibility of the simulation of complex interactions between deformable models with physics realism. In addition, the parallel implementation of the simulator represents a promising starting point for the development of interactive simulations that can be used in different fields of research, such as surgeon training or fast prototyping.

Acknowledgments

Paolo thanks for the assistance and the guidance during the whole thesis. I really like your approach to research and how the laboratory is organized. Discussions with you have always been source of inspiration (and additional work...).
Debora and **Gianni**, you convinced me to start this PhD, thanks for having put your trust in me and for the brilliant idea.
Vincent Hayward and **Cristian Secchi**, thanks for the useful reviews of my thesis and thanks for having been part of my thesis committee together with **Herman Bruyninckx** and **Luigi Palopoli**: your comments have been very useful for the final version of this thesis.
Past and present Altair laboratory people, thanks for making the lab such a nice place to work and thanks for all the useful (and funny) discussions and projects - I am still waiting for the Mojito machine...
Stefano: it's been a privilege working with you.
Paola, **Cesco** and **Vero** probably you do not even know each other, but thanks for the unexpected and significant help you gave me in this last period.
Ivan and **Max** thanks for the continuous support you gave me in all the situations I faced during this Phd.
Anna thanks for your help and patience in the first years of this adventure.
My family gave me all the means to achieve this important result, thank you for the trust you put in me even in bad patches and for having been my ultimate support in every situation.

Contents

1 Introduction 1
1.1 Simulation Components 2
1.1.1 User Input 3
1.1.2 Physical Modeling 3
1.1.3 Temporal Integration 4
1.1.4 Interference Detection 5
1.1.5 Collision Solution 5
1.1.6 Scene Rendering 6
1.2 Contributions 6
1.3 Structure Of The Thesis 8

2 Soft Tissue Simulation Techniques 11
2.1 Finite Elements 13
2.1.1 Condensation 16
2.1.2 Boundary Element Method 17
2.1.3 Modal Analysis 18
2.2 Mass Spring Models 19
2.2.1 Damping in Mass Spring Models 20
2.2.2 Volume Preservation 21
2.3 Meshless Models 22
2.3.1 Physically Based Meshless 23
2.3.2 Shape Matching Based Meshless 26
2.4 Conclusions 26

3 Common Representation Framework 29
3.1 Adaptive Models 29
3.1.1 Multi Resolution Models 30
3.1.2 Hybrid Models 34
3.2 Common representation 36
3.2.1 Point Based Approach 36
3.2.2 Geometric Analysis 37
3.2.3 Dynamic Analysis 39
3.3 Conclusions 42

4 Collision Handling ... 43
4.1 Collision Detection ... 43
4.1.1 General Approach ... 44
4.1.2 Collision Detection for Deformable Models ... 45
4.1.3 Collision Detection Library ... 48
4.2 Collision Solution ... 50
4.2.1 Problem Statement ... 51
4.2.2 Method ... 53
4.3 Results ... 55
4.3.1 Structure Update ... 56
4.3.2 Collision Detection ... 57
4.3.3 Collision Solution ... 58
4.4 Conclusions ... 59

5 Friction Models ... 63
5.1 Dynamic Components of Friction ... 64
5.2 Friction Models ... 65
5.2.1 Static Models ... 66
5.2.2 Dynamic Models ... 68
5.3 Model Comparison ... 71
5.3.1 Coulomb Model ... 72
5.3.2 Karnopp Model ... 73
5.3.3 Dahl Model ... 73
5.3.4 LuGre Model ... 73
5.3.5 Elasto Plastic Model ... 74
5.4 Integration ... 75
5.4.1 Velocity Approximation ... 76
5.4.2 Force Approximation ... 77
5.4.3 Force Distribution ... 78
5.4.4 Examples ... 78
5.5 Conclusions ... 81

6 Anisotropic Mass Spring Models ... 83
6.1 Related work ... 84
6.2 Method Description ... 87
6.2.1 Analytical Description ... 87
6.2.2 Geometrical Interpretation ... 89
6.3 Results ... 91
6.4 Conclusions ... 95

7 Implementation ... 97
7.1 *GPU* Implementation ... 98
7.2 Physics Simulation ... 101
7.2.1 Physical Model Representation ... 101
7.2.2 Elastic Force Computation ... 103
7.2.3 Volume Preservation ... 104
7.2.4 Temporal Integration ... 105
7.3 Deformable Model Interaction ... 106

7.3.1 Collision Detection With Fixed Structures 106
7.3.2 Probing 107
7.3.3 Grabbing 108
7.3.4 Cutting 108
7.3.5 Interaction Forces Computation 109
7.4 Results 110
7.5 Optimized Graphical Rendering 114
7.5.1 Remote Rendering Overview 114
7.6 Integration in the Simulation 117
7.6.1 Deformable Models Rendering 117
7.6.2 Architecture Scalability Test 119
7.6.3 Network Performance Test 121
7.6.4 Model Complexity Test 121
7.7 Conclusions 122

8 Conclusions and Future Work 125

References 129

1 Introduction

The simulation of deformable bodies is an active research topic since the last years of the '80s [102]. Several methods have been developed to simulate the deformations of soft materials undergoing external forces and many research areas take advantage from the progress made in this field. Simulations based on deformable models suffer for the high computational time required by the update of soft bodies state and thus physically based simulations are often limited to non interactive applications.

For interactive applications many simplifications are introduced into the physics of the environment, in particular in the modelization of soft tissues and in the simulation of contacts between bodies. For example, it is common to discard physically based deformable models to use, instead, simpler and faster models even if they cannot guarantee the realism of the results. These kind of simulation are usually referred to as physically plausible and proved to be very useful in computer animations or computer games.

Some applications, on the other hand, cannot tolerate the errors introduced by physically plausible simulations. This is the case, in particular, of surgical simulations. The use of computer assistance during the different phases of the diagnosis, intervention and patient follow up, spreads in the last years thanks to the possibilities provided by automatic image analysis tools. One important goal of research in computer assisted intervention is the development of surgical simulators that are capable to provide realistic environment, both for the training of surgeons and for the planning and the analysis of actual interventions [6, 12].

In this work we define as "physically based" the simulation of physical phenomena that does not differ *too much* from the reality. This definition is clearly subjective and it greatly depends on the aspects of the simulation that the user considers important. In surgical training, for example, it is important to have deformable models whose behavior resembles the real tissue behavior whereas in surgical planning the realism of the deformation should be ensured also quantitatively but, usually, they involve smaller ranges of deformations and forces.

Physically based simulations of deformable environments thus represent a major achievement in the development of computer assistance tool in surgery. These simulations should provide realistic behavior for the soft anatomical tissues undergoing deformations and should accurately model the interactions of contact-

ing tissues or organs and between surgical tools and organs. Some research gave promising results in this directions. In particular, effort at *INRIA* (Institut national de recherche en informatique et en automatique, the French national institute for research in computer science and control) resulted in the development of many methods and algorithms for the simulation of deformable models that converged into SOFA: a framework targeted at real-time simulation, with an emphasis on medical simulation [52].

The framework implements and integrates different techniques to obtain real time simulations of environments composed of rigid or deformable bodies. In particular it provides different deformable bodies for the simulation of solid tissues: principally mass spring models but also linear finite element models and rotationally invariant finite element models (see Chapter 2). In addition it provides point based models to mimic the behavior of fluids (see Chapter 2). Collision between rigid bodies are solved exactly whereas soft bodies are handled as collection of rigid bodies connected through soft constraints or are approximated by a set of spheres. One big limitation of this framework is its inability in simulating frictional contacts.

Another approach have been followed in the development of ODE an open source, high performance library for simulating rigid body dynamics [94]. This library can simulate deformable models by treating them as sets of rigid bodies linked by soft constraints. It provides collision detection algorithms and simulates friction in the contacts. The big drawback in using this library is the difficulty in modeling soft bodies with articulated rigid bodies. Moreover it is not targeted to real time simulations, thus it may be not suitable for interactive environments.

Along with the benefits introduced by a realistic simulation in the graphic rendering of the environment, the physical model of the scene can be exploited to increase the sense of immersion of the user by adding force feedback to his/her actions. This is of particular importance in the medical field, where surgeons learn to discriminate tissues by palpating them and where the application of excessive forces can seriously damage living tissues.

The goal of the work described in this thesis is the analysis and implementation of different components of an interactive simulator that provides force feedback to the user with advanced physical simulation based on deformable models and realistic friction computation.

1.1 Simulation Components

Interactive, physically based simulation of deformable environments with force feedback represents a great challenge for computer science today. Due to the complexity of the problem it involves different areas of research: i.e. computer science, physics but also human machine interaction. A simulator, in fact, can be decomposed in many different parts that cooperate to provide the needed realism. The building blocks of a simulator can be roughly summarized as follows:

- user input acquisition;
- physical modeling of deformable models;
- temporal integration;

- collision detection;
- collision solution;
- rendering of the scene to the user.

These phases are repeated during the simulation to provide a constant update of the scene physics and the scene rendering. The different parts can easily be identified in the structure of simulators. The final use of a simulator defines the characteristics of each part, as they should adapt to the requirements of the application. A brief description of each part is presented in the following.

1.1.1 User Input

This part represents essentially the interface that allows the user to interact with the scene. Interaction is usually obtained by moving objects or applying constraint or forces to some parts or points of simulated bodies. The actual input can came from standard peripherals, such as keyboard or mouse, but it can also involve more complex devices that allow a more direct manipulation of the environment. The use of proper devices that allow the user to freely move in a tridimensional space usually helps the interactions. In fact it is generally difficult for a human to interact with a 3D environment by using a two dimensional device such as a mouse or with a keyboard.

Different approaches have been developed to increase the dexterity of the user. Basically they monitor the movements of the user by using optical tracking instruments, or by using hardware devices. Some hardware devices provide the possibility to render forces (haptic rendering) to the user, this increases the virtual environment perception. This class of device is called haptic devices. The introduction of haptic rendering greatly improves the immersivity sense, but, as we will discuss in Section 1.1.6 it also introduces some tight constraints to the simulation. An example of an haptic device, the one used during the development and test of this work, can be seen in Figure 1.1. This devices provide six actuated degrees of freedom, i.e. it allows the user to define point and orientation in a tridimensional space and to experience forces and torques with his/her hand.

1.1.2 Physical Modeling

The modeling of the physics of the bodies in the scene is a key aspect of the simulation. The objective of this phase is the computation of soft bodies internal forces that are due to deformations or to external constraints (see Figure 1.2). To this extent the body is usually discretized in parts that contribute locally to the overall model behavior. Many different models have been proposed in the literature to mimic the deformation of soft tissues. They can be split in two categories on the base of their theoretical background. In fact it is possible to distinguish between physically based models and non physically based models (see Chapter 2 for more details).

We restrict our attention to physically based models because the parameters that control their behavior can be related to physical measures of soft tissue characteristics, such as stiffness or compressibility. This also guarantees the minimum level of realism that is needed in surgical applications. The main drawback in the

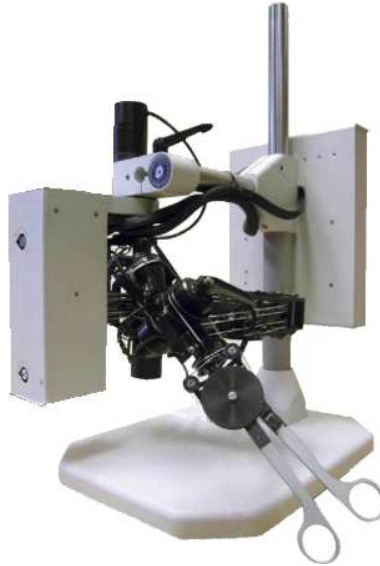


Fig. 1.1. The Freedom 7s haptic device.

use of physically base models is their computational complexity. Thus we analyzed some of the most common methods and identified their advantages and limitations.

1.1.3 Temporal Integration

Temporal integration evolves the scene in accordance with the user input and with the internal forces computed by the physical model. Forces acting on each body of the scene are summed up, and the whole scene is updated by a numerical integration technique. This moves the configuration of the bodies toward a status of equilibrium.

The introduction of physically based model into the simulation imposes very tight constraint on the temporal step used in the integration. As it will be detailed in Chapter 2, in fact, the maximum temporal step used in the simulation is related to the characteristics of the simulated tissues. Furthermore the interactivity of the application requires that the computation of each time step completes in an amount of time that is less than the simulated time. In addition, as we explain later in this chapter, the introduction of haptic rendering imposes that the scene is updated every millisecond, at least. These two aspects limit the choice of the

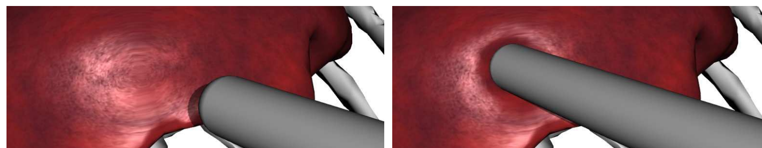


Fig. 1.2. A soft model undergoing deformations.

numerical integration technique: explicit techniques are a very common choice for interactive applications as they provide a good trade off between computation complexity and stability.

The characteristics of numerical integration techniques are well known, and an analysis of their application in simulation is beyond the scope of this thesis. For this reason we will only present the technique used, i.e. Verlet integration technique in Section 7.2.4.

1.1.4 Interference Detection

The goal of this phase is the identification of intersections between objects surfaces. A naive approach to collision detection will test all the triangles of each model with all the triangles of other bodies in the scene. The computational complexity of this approach makes it not suitable for complex scenes or for interactive simulations. For this reason many techniques have been developed to speed up the computation of collisions. They basically work by exploiting spatial partitioning or on mesh decomposition.

When applied to deformable model simulations these techniques suffer for the difficulty of precomputing data and/or to update data structures to maintain the consistency with object configurations. Some of the principal libraries targeted to collision detection between deformable models have been reviewed and compared to the requirements of our scenario. This led to the identification of the most suitable one and to the integration of this library into the implemented simulator.

1.1.5 Collision Solution

Collision solution aims at restoring the scene in a consistent state after one or more collisions have been detected. In our scenario the collision solution should not only respect the basic requirements of avoiding inter penetrations between solid objects, but it should also provide realistic behaviors for the colliding surfaces. To this extent, when solving a collision between contacting bodies, the physics of the contact, composed of interaction and friction force, should be simulated (Figure 1.3).

The introduction of realistic behavior into the computation of interaction requires an extension to standard collision solution techniques. In fact, most of the methods proposed in literature to handle collisions in interactive simulations introduce many simplifications to the problem. They usually compute the penetration depth of the colliding bodies and use it to obtain a penalty force (an approximation of the contact force) that moves the bodies apart. Although this method works well in video games or rigid bodies simulations, it does not model the physics of the contact nor it considers the friction during the computation of reaction forces.

To overcome this limitations we introduce a framework that provides a common representation of different class of deformable models. This framework allows the correct solution of collisions and, in addition, it allows the introduction of realistic friction models into the simulation that seamlessly integrates with the physics of the bodies.

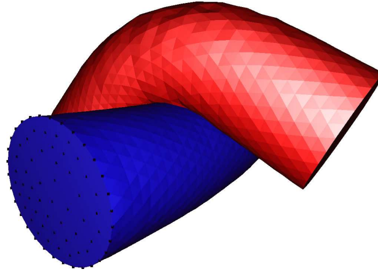


Fig. 1.3. Two deformable models interacting with collision detection and solution computation.

1.1.6 Scene Rendering

The last step of the simulation is the rendering of the scene to the user. Along with the classic graphical rendering we consider also force feedback. The force feedback (or more generically haptic feedback) allows the user to actually feel the forces that are generated during the interaction with the environment. This is very useful in simulations where the dynamic component of gestures is important, as in the case of surgical simulations, where fast movements can cause damages to tissues.

Tests show that to provide realistic force sensations the update of interaction forces should be faster than $1KHz$. If the force rendering frequency drops under this threshold the perceived realism of the environment decreases. In addition, delay or jitter in force rendering can cause instabilities to the simulation that degrade the realism and the correctness of the simulation.

For this reason it is very important to optimize the implementation and to guarantee a refresh of the physics of the scene at least every millisecond. To obtain this we implemented a simulator that exploits the graphics card processing unit and that is capable of ensuring the frequency requirement in the simulation of complex scenes. In addition we propose a method to perform graphical rendering remotely. This increases physics simulation performance and allows the introduction of advanced graphical rendering techniques without affecting the speed of the simulation (see Figure 1.4).

1.2 Contributions

This work introduces many novel methods to enhance the physics in interactive simulations of deformable models with force feedback. In particular, we describe a framework that unifies the handling of different techniques for soft tissue modeling. This framework wraps the physics of deformable models with a surface that is used to graphically represent the body and to exchange forces and displacements with the surrounding environment.

This approach provides great advantages to the definition of environments with multiple deformable models. In fact each deformable body can be modeled with the most appropriate physical model: the wrapping surface provides a unified interface to handle interactions. This means that the collision detection library

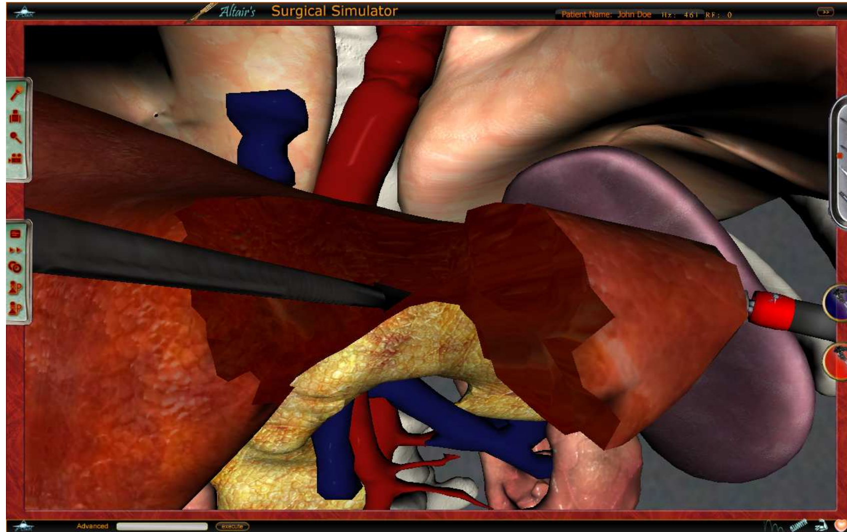


Fig. 1.4. A screenshot of the simulation of deformable environment with haptic feedback and advanced graphical rendering.

or the frictional model does not need to be tailored to the different features of physical models, instead one single algorithm will be suitable for all the different modelization techniques, since it will always interact with the wrapping surface.

The development of this approach requires some attention to the propagation of interactions between the two parts: surface and physics of the model. We address this problem by developing methods to guarantee the coherence between the two parts of the model and that extend methods used for point based model rendering (see Chapter 3). These methods define how forces applied to the model surface are propagated to the internal element of the model representing its physics and how forces computed on internal elements are rendered on the surface of the model. Similarly, the displacement of surface nodes is translated into displacement of physical elements that, in turn, determines the configuration of the surface after the temporal integration.

This approach is based on some precomputation that defines the interdependence between the physics and the surface, and thus reduces the overhead introduced to the computation of the scene.

The introduction of frictional contacts into the physics of the scene takes advantage of the mentioned framework. In fact, the unification of different deformable simulation techniques allows us to simplify the introduction of friction models into the simulator. We thus developed a single method that seamlessly integrates friction models into the physics of the environment.

This approach provides the ability to choose the most suitable friction model depending on the requirements of the simulation, thus it gives the user the ability to adapt the simulation to his/her needs. The method computes the different variables that are needed by friction models: tangential and normal forces and tangential velocity. It makes them available to the chosen friction model and then

receives back the computed friction force that is distributed to the points involved in the contact as explained in Chapter 5.

This work also propose an innovative method to introduce anisotropic behavior in mass spring model simulations. Anisotropy is a key feature of many real materials. Its contribution to the behavior of biological tissues is very important, as many biological materials are composed of fibers and thus their response to solicitations is strictly dependent on the direction of fibers and stimuli [37].

The basic idea of the approach described in Chapter 6 is to associate a value to each spring of the body. This value is used during the computation of the physics to determine the anisotropic contribution of the element. The major achievement of our method is the simulation of anisotropic behavior with a very small computational overhead. In fact, it increases the dimensions of the space in which the model is defined and evolves the model in this augmented space. The model is then projected back to its original space to obtain the result of the simulation.

Another important contribution of this thesis is the development of a parallel implementation of a physical simulator that handles deformable models and allows the user to freely interact with them and to perceive the environment through graphic and haptic rendering. The parallel implementation makes the simulator suitable to exploit the graphics card processing unit, which is composed of many basic processors that work in parallel.

The obtained simulator provides the user with the ability to grab and probe deformable models, but also to change model topology, principally by cutting it with appropriate virtual tools. Our approach minimizes the data exchange between CPU and graphic card that represents the main bottleneck of graphic cards programming. The whole computation is thus carried out at a frequency that is above $1KHz$ also when the environment is populated with complex models as shown in Figure 1.4.

To further improve the performance of this simulator, we extended it with an innovative remote rendering technique. This relieves the graphic card used for the physics simulation from the graphic rendering job and entrusts a remote machine with it. The data encoding and transmission is optimized to reduce the delay in the process and to proceed in parallel with the physics update. The resulting architecture provides excellent results in physical simulation of complex scenes and in rendering them to the user.

1.3 Structure Of The Thesis

In the following we present the different parts that composes this work. The structure of this thesis follows the structures of simulation components as described in Section 1.1.

In particular, Chapter 2 introduces the problems related to the simulation of deformable models and reviews the most diffused techniques that have been developed to mimic the behavior of soft tissues in numerical simulations. Three main classes of deformable models are recognized, finite element, mass spring and point based models. Methods used in this thesis are described in details, moreover advantages and limitations of each class of models are highlighted keeping into account their computational complexity.

In Chapter 3 we describe the approach we developed to uniform the handling of different simulation methods. We present the theoretical background, drawn on techniques coming from graphical rendering. Then we detail the extensions we added to adapt the underlying concepts to the needs of our scenario. The definition of a wrapping surface based on point based rendering techniques requires the introduction of methods to adapt the surface of the model to the underlying physical representation and to distribute forces and displacements between the two entities.

Chapter 4 identifies the requirements of interactive, physically based simulations for the collision detection phase. Then it provides a comparison of the principal methods and libraries developed to perform collision detection. This comparison allows us to determine the most suitable library for our needs. In particular we identified V-collide as the one that offers the best trade off between computational time and performances when applied to deformable models.

Then, in Chapter 5, we analyze the principal aspects of frictional contacts to identify the features that are required to provide realistic results in simulations. Some of the friction models provided in the literature are analyzed and their behaviors compared. An important aspect of this analysis is model computational complexity, as deformable model simulations require the solution of many contacts at each time step. In this chapter we also detail the integration of deformable models into the physical simulation, in particular we describe the method we developed to compute required variables from the framework presented in Chapter 3 and to distribute forces computed by friction models to body surface points.

Chapter 6 proposes the enhancement to mass spring models that allows to simulate anisotropic materials with a simple extension to the basic model. We detail the theoretical background that supports our method and then we provide an intuitive, graphical representation of the method. The method is then compared with the state of the art for modeling anisotropic models. Graphical simulations are provided to show the realism of the obtained models.

The implementation of the simulator demonstrating all the methods developed is described in Chapter 7. The chapter describes in depth the data structures we developed to store the physical models in the graphic card memory. Then it details the algorithms that simulate the physics of the model and that handle the interaction with the user. The combination of "ad hoc" data structures and algorithms allows the simulation to run completely on the graphic card and to reach the frame rate needed to provide realistic haptic feedback. Along with the details of the physics implementation we provide also the description of a method we developed to perform the graphical rendering remotely. This method asynchronously download data required for rendering from the graphics card carrying out the physical simulation and then send them to the remote machine. The remote machine decodes the data, reconstructs the scene and perform the actual graphical rendering.

Finally, in Chapter 8 we summarize the results of the work presented in the previous chapters and we outline some possible extensions to this work.

2

Soft Tissue Simulation Techniques

In this chapter we introduce the most important techniques that have been developed to model and simulate soft tissue deformations. We will briefly describe the linear approximation that is usually adopted in interactive simulations and then the three main classes of deformable models will be introduced with their variants. The comparison of different models properties leads to the choice of the right modeling technique for each specific application.

Deformable models are abstractions that allow to approximate the behavior of soft tissues undergoing deformations. They provide accurate results in load simulations and stress/strain analysis for engineering structures as in the rendering of deformable materials in computer graphics. Recently the use of deformable models in haptic rendering has been investigated, showing the possibility to obtain the required, high frame rate and plausible results also in interactive simulation with force reflection.

A deformable object is usually defined by its rest, or undeformed, shape and by a set of material parameters that define how the material deforms under external forces. Several methods for computing the response of a deformable object have been developed, starting from different considerations: the main feature that distinguishes the various methods is whether the material is considered as a continuum medium or as made of a discrete set of elements. The physical background that guides the development of models and the identification of model parameters is based on the laws used for the computation of the internal stress tensor (usually labeled $\sigma \in \mathbb{R}^{3\times3}$). For an in depth description of deformable tissue modeling the reader can refer to [37]. Most works in simulation use Hookean linear material law that relates the stress tensor and the strain tensor ($\varepsilon \in \mathbb{R}^{3\times3}$).

In fact, the stress of a linear elastic material can be derived from the stored energy potential function of the strain (also called strain energy density function). Therefore it is possible to define an elastic material to be one that satisfies:

$$\sigma = \frac{\partial w(\varepsilon)}{\partial \varepsilon} \qquad \text{or} \qquad \sigma_{ij} = \frac{\partial w}{\partial \varepsilon_{ij}} \tag{2.1}$$

where e is the strain energy density function. If the material in addition to being elastic has a linear stress-strain relation it is possible to write:

$$\sigma = \mathbf{E}\varepsilon \qquad \text{or} \qquad \sigma_{ij} = E_{ijkl}\varepsilon_{kl} \tag{2.2}$$

The quantity **E** is called the stiffness tensor, or the elasticity tensor.

Deformable models usually require a discretization of the object volume. This discretization is obtained by defining a cloud of point internal to the object's volume and by defining on them a structure such as a tetrahedral mesh or a lattice. The structure is then used to compute interactions between different points of the model. As we will explain in the following sections, meshless models do not require this structure and use support functions to define and weight the interactions among points.

A key aspect of deformable models simulation that comes from the temporal integration performed, is the dependency of results on simulated temporal step: big temporal integration steps applied to a high resolution model causes errors in the results that can lead to instability of the simulated environment. Typically the time step in simulation is defined empirically, by starting from a small interval and increasing it until the simulation becomes instable. This method cannot guarantee the stability of the simulation under all circumstances, as the errors in the temporal integration depend on the point velocities and thus on the forces applied by the user. Some works, e.g. [85] discussed in Chapter 3, provide an adaptive time step whose choice is based on theoretical considerations. In particular, Courant-Friedrich-Lewy condition [78] relates the time step t to the velocity of a sound wave in the simulated tissue v_{sound} and to the spatial discretization step (the minimum distance between two points) d_{min}:

$$t < \frac{d_{min}}{v_{sound}} \tag{2.3}$$

One interpretation of Equation 2.3 is related to Shannon theorem: in fact the maximum allowed temporal step guarantees that the sampling of a sound wave traveling in the tissue at the nodes of the model does not introduce aliasing. Since the velocity of sound waves in tissues depends on tissue stiffness and density, Equation 2.3 imposes tight constraints on the spatial resolution of the model given the desired temporal step and the stiffness of the simulated tissue.

Some methods for the simulation of deformable models are based on the off line computation of a set of interactions [5, 53]. Thee interactions are used as a sort of base of deformation space: at run time the actual response of the body to user interaction are computed as a weighted combination of pre computed contributions. These methods are particularly fast but they are less generic than on line methods, in addition, it usually difficult to allow topology changes (such as cuts or tears) during the simulation or they can be modeled only along predefined surfaces as in [70].

In the following sections we will present the principal on line methods that represent the state of the art in deformable model simulations for interactive applications. The described methods are based on both continuous and discrete representations, most of them rely on the linear approximation of the stress-strain equation 2.2 for its simplicity and ease of implementation.

2.1 Finite Elements

Finite element models can give the most accurate results in simulations of deformable tissues, their strong theoretical background make them realistic and easy to calibrate, thus their use is widespread in engineering and in those fields where accuracy is the most important goal [11] [61] and where they are used for static and dynamic simulations. We focused our work on linear isotropic *FEM*, so in the following only this class of method will be described, additional information regarding *FEM* analysis can be found in [7] and [20]. The finite element method for a homogeneous elastic material is based on the generalized Hooke's law in Equation 2.2 where **E** is the elasticity tensor. Since **E** is a rank four tensor it has 81 coefficients, but due to the symmetry of the stress tensor, strain tensor, and stiffness tensor, only 21 elastic coefficients are independent in a generic material. For a linear isotropic tissue the elasticity matrix can be expressed as:

$$\mathbf{E} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{2.4}$$

and Equation 2.2 can be rewritten as:

$$\sigma = \lambda tr(\varepsilon)\mathbf{I} + 2\mu\varepsilon \tag{2.5}$$

showing that in this case only two variables are independent. λ and μ are Lamé coefficients: the first parameter λ has no physical interpretation, but it serves to simplify the stiffness tensor in Hooke's law. The two parameters together constitute a parametrization of the elastic moduli for homogeneous isotropic media, and are thus related to the other elastic moduli. In fact, there are different ways

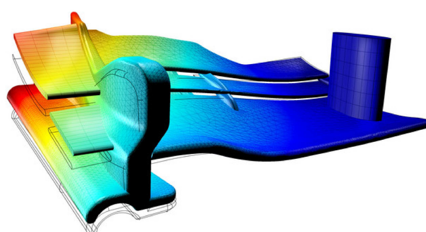


Fig. 2.1. Analysis of deformation for a sport car wing structure undergoing airflow pressure, from Comsol Multiphysics

to express this elasticity matrix. The most common and direct method is using Young's modulus (usually labeled as E) and Poisson's ratio (usually ν). Young's modulus represents a measure of stiffness of the tissue, and can be determined experimentally from the slope of a stress/strain curve. Poisson's ratio describes the

compressibility of the material and ranges from 0, for a completely compressible material, to 0.5 which represents incompressible materials. From Equation 2.2 the strain energy of a linear elastic body Ω can be obtained as:

$$E_{strain} = \frac{1}{2} \int_{\Omega} \varepsilon^T \sigma dx \tag{2.6}$$

Static equilibrium between deformation energy and external force is achieved when the first variation of $E(u)_{strain}$ vanishes, introducing the Cauchy strain tensor B (or linear strain tensor, since it considers only small deformations and strains so that higher terms can be neglected) the equilibrium condition can be written as:

$$\delta E(u)_{strain} = 0 = \int_{\Omega} B^T DBu \, dx - f \tag{2.7}$$

and since everything inside the integral is constant it can be reduced to a constant matrix K_E called the stiffness matrix and the whole system can be expressed as:

$$f = K_e u \tag{2.8}$$

For complex models, composed by many tetrahedra, the contribution of each single element can be summed up into a unique matrix K. In three dimensions the resulting matrix is a square matrix of size $3n \times 3n$ (with n the number of nodes of the model). The resulting matrix is symmetric, as a consequence of the symmetry of the K_e matrices, moreover the block of elements in position $[3i, 3i{+}2], [3j, 3j{+}2]$ describes how the i-th and j-th elements interacts and is obtained as the sum of the contributions of all tetrahedra that shares nodes i and j. Equation 2.8 provides the force acting on the model nodes, given the node displacements. In some case deformable bodies are controlled "in force" i.e. applying forces to their nodes and obtaining a displacement as a result (this is the case, for example, of structural analysis). In these cases the matrix has to be inverted, but it is necessary to impose some constraints to the model to obtain a non singular matrix . In particular, in a 3D space 6 degrees of freedom of the model should be constrained. This is equivalent to fix at least three points of the model. To fix the points of a *FEM* one needs to put zeros on the three columns and rows related to that point and to put ones corresponding to the diagonal elements of the matrix relative to those points. In the case of a 1D system, composed by two truss elements as shown in Figure 2.2, the system stiffness matrix K is:

$$K = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{2.9}$$

where k_i is the stiffness or the i-th truss element. K is singular because column two is a linear combination of columns one and three. As result, there is no unique solution to the system unless one of the nodes is constrained. The first node of the system can be fixed by modifying the equation describing the system in:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} 0 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} 0 \\ f_2 \\ f_3 \end{bmatrix} \tag{2.10}$$

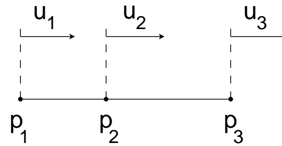


Fig. 2.2. A simple 1D truss system

where f_i represents the force acting on the i-th node of the system.

One of the main limitations of linear finite element method is their rotational variance: when a linear finite element model undergoes large rotations its volume increases in an unrealistic manner. To overcome this problem Green's nonlinear strain tensor can be used [26], loosing the linearity property for small deformations, or forcing the rotational invariance for each single element of the model, as done in corotational finite elements [80]. In this paper it is showed how to obtain rotationally invariant finite element using the Green's strain tensor instead of Cauchy's. The Green tensor is non linear and rotationally invariant but it leads to heavier computations and it is not able to linearly relate deformations to displacements except asymptotically for small displacements. An alternative approach has been proposed in [80] by Müller and improved in [46] and [83]. The common idea is to identify the local rotation applied to the elements of the model and to remove the rotational component from the estimated force (see Figure 2.3). Müller proposed to extract a rotation matrix for each node of the model as a mean of the rotations of its incident edges. As stated in [46] to be consistent with finite elements is better to identify a rotation matrix per tetrahedron thus keeping the Jacobian of the system symmetric, whereas using a rotation per node breaks symmetry. To

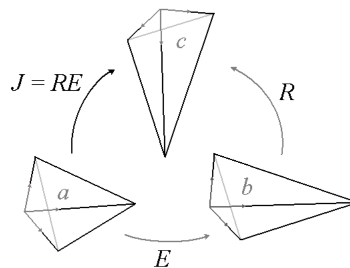


Fig. 2.3. An initial tetrahedron a is deformed into c by the transformation J composed of a rigid motion R and a local deformation contained in E. In b the deformed element is rotated back into the original coordinate frame.

obtain the requested rotation matrix two steps should be performed: the first is the construction of a matrix J that encodes the transformation between the deformed frame of reference and the original frame, the second is the decomposition of J into a rotation matrix R and a strain component matrix E. A suitable approxima-

tion of the matrix J can be constructed choosing deterministically three edges of each tetrahedron in the initial frame and three in the deformed frame of reference (respectively u_n and v_n where $n \in 1, 2, 3$) and define J as:

$$J = [\mathbf{u}_1\mathbf{u}_2\mathbf{u}_3][\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3]^{-1} \tag{2.11}$$

The rotation matrix R_e to be extracted from J is usually not unique since the tetrahedra undergo deformation during simulation. To obtain R_e it is possible to use the SVD decomposition:

$$J = USV^T \qquad R_e = VU^T. \tag{2.12}$$

or to use a Gram-Schmidt QR factorization, that allows computing the needed matrix directly:

$$J = QR, \qquad R_e = Q. \tag{2.13}$$

Once the R_e matrix is known the B matrix used in Equation 2.7 should be recomputed, as changes in the topology of the tetrahedra lead to changes in the interpolation matrix. Rotational invariant *FEM*s or corotational *FEM*s solve some of the problems of linear *FEM* at the cost of a higher computational complexity and a reduced stability.

Many techniques have been developed to speed up the computation of *FEM*, based on the assumption that the user cannot interact with model internal points or by observing that some deformations (modes) of the system can not be "seen" by the user. In the rest of this Section we will present the two most important methods: condensation and modal analysis.

2.1.1 Condensation

If some points of the model can not be manipulated during the simulation (usually because they are internal) then the system degrees of freedom can be reduced through condensation: calling u_s and u_i the displacement of external and internal points respectively and f_s and f_i the force acting on external and internal points respectively, we rewrite the system equations by reordering the row and columns of the matrix K:

$$\begin{bmatrix} f_s \\ f_i \end{bmatrix} = \begin{bmatrix} K_{ss} & K_{si} \\ K_{is} & K_{ii} \end{bmatrix} \begin{bmatrix} u_s \\ u_i \end{bmatrix} \tag{2.14}$$

where K_{ss} collects the entries of the matrix K related only to surface points. From Equation 2.14 a condensed system, containing only the surface nodes of the original system can be obtained:

$$K'_{ss} = K_{ss} - K_{si}K_{ii}^{-1}K_{is} \tag{2.15}$$

$$f'_s = f_s - K_{si}Kii^{-1}f_i \tag{2.16}$$

$$K'_{ss}u_s = f'_s \tag{2.17}$$

The matrix K'_{ss} is, usually, not sparse as the original matrix K is and, therefore, it can not take advantage of optimizations related to sparse systems. Since its size depends on the nodes that can be manipulated, this optimization gives good

results if the number of "hidden" nodes is large compared to the number of surface node. The drawback of condensation is the limitation of the interaction that can be simulated: in fact internal points do not explicitly appear in the computation and their position can not be computed. This is not a limitation when the topology of the model remains constant during the whole simulation, but when topological changes (e.g. cuts) may happen the interior part of the model should be considered. This implies the computation of a new K'_{ss} matrix with a consequent increment of the computational time. Thus condensation can not be used (or hybrid models should be considered [112]). Moreover using the matrix K to linearly relate forces to displacements allows obtaining the static equilibrium solution of the system, i.e. the configuration of the deformed body after an infinite time and without any damping. For interactive simulations, when the user need to perceive the deformations of the model, the dynamic behavior of the soft tissue is required. To obtain it Equation 2.8 has to be extended as:

$$M\frac{d^2u}{dt^2} + D\frac{du}{dt} + Ku = f \tag{2.18}$$

where M and D are, respectively, the mass and damping matrices of the system. The computation of the two matrices is not straightforward, and for simplicity they are usually computed as lumped (diagonal) matrices. The solution of the system is approximated using numerical methods.

2.1.2 Boundary Element Method

Boundary element method (*BEM*) are similar, in the approach, to condensed *FEM*. *BEM*, in fact, consider only the boundary of the modeled body and compute the equilibrium of forces and displacement by discretizing body surface. This method is only suitable for linear homogeneous tissues, as it does not take into account inclusions or variations in the composition of bulk materials of the body.

The domain of the deformable model is denoted by $\Omega \in \mathbb{R}^3$ and its boundary is Γ. *BEM* shares with *FEM* the law that governs the deformation (Equation 2.8). In the remainder of this section we consider $f = 0$ (i.e. no external forces are present) to simplify the notation.

The boundary is separated in two parts: the first one, denoted Γ_u, allows to define displacement boundary conditions, i.e. areas where the body is in contact with external structures. The second one, denoted Γ_p, defines areas where it is possible to specify the acting force, or the traction p defined as the force over the unit area. Γ_p is useful to define which parts of the body surface are free to move ($p = 0$).

It is possible to rewrite Equation 2.8 in boundary integral formulation:

$$cu + \int_\Gamma \mathbf{p}^*\mathbf{u}\, d\Gamma = \int_\Gamma \mathbf{u}^*\mathbf{p}\, f\Gamma \tag{2.19}$$

Where c is a known function that depends only on the geometry of the boundary and $\mathbf{u}^*$ and $\mathbf{p}^*$ are fundamental solutions which depend only on known elasticity properties.

To solve Equation 2.19 numerically it is discretized by approximating $\mathbf{u}$, $\mathbf{u}*$, etc. in finite dimensional function spaces. This requires the discretization of the boundary Γ into a set of N noon-overlapping elements which represent the displacement and traction by functions which are piecewise interpolated between the element's nodal points. Then the integral equation is applied to each of the n boundary nodes, and the resulting integrals are computed over each boundary element to generate an undetermined system of $3n$ equations involving the $3n$ nodal displacements and the $3n$ nodal tractions. The boundary conditions are then applied, fixing n nodal values (either displacement or traction) per direction. The resulting linear system of $3n$ equations is determined and may be solved to obtain the unknown nodal boundary values.

The resulting system, before the application of boundary conditions, has the form:

$$Hu = Gp, \tag{2.20}$$

after the boundary conditions have been applied, it is possible to bring the unknowns to the left-hand side and the knowns to the right side and obtain a linear system:

$$Av = z \tag{2.21}$$

which may be solved for the unknown nodal quantities v. A key advantage of *BEM* is that all the unknowns are on the boundary Γ of the body whereas *FEM* also includes unknowns of the interior. This reduces the computational time required to solve *BEM* but, on the other hand, limits their field of application to homogeneous bodies.

2.1.3 Modal Analysis

An approach that share some concepts with *FEM* condensation is modal analysis. It works by decomposing the single modes of vibration of a model and synthesizing model deformation as a linear composition of those modes. An in depth discussion of modal analysis and its use with finite element method can be found in [20]. A more detailed presentation of modal analysis mathematical background can be found in [71]. The technique was firstly proposed by Pentland and Williams in [86]. They used linear and quadratic deformation fields defined over a rectilinear volume instead of the object's actual modes and they deformed the model embedding it the region and using a method similar to free form deformation. The method allows to efficiently simulate deformations only for compact objects that can be approximated by a rectilinear solid. More recently modal analysis have been extended to handle geometrically complex, real-time deformation models, [45] also exploiting the computational power of hardware architectures [54] and [117]. As modal analysis is based on the linear Cauchy tensor introduced in Equation 2.7, it shares a drawback with linear *FEM*, i.e. the lack of rotational invariance. In [18] modal analysis is extended to accommodate rotations. Although the approach is not guaranteed to perform well for large deformations, these results are visually better than standard modal analysis.

The modal decomposition of a physical system starts from Equation 2.18 and diagonalizes it. One approximation introduced to simplify the diagonalization concerns the D matrix: it is supposed to be a linear combination of matrix K and

M. This restriction is known as Rayleigh damping and it gives better results of the simple mass damping commonly used (mass damping assumes $D = \alpha M$ where *alpha* is a positive scalar). Using the proposed approximation Equation 2.18 can be expressed as:

$$K\left(d + \alpha_1 \frac{du}{dt}\right) + M\left(\alpha_2 \frac{du}{dt} + \frac{d^2u}{dt^2}\right) = f \tag{2.22}$$

where α_1 and α_2 are the Rayleigh coefficients. If we define W as the matrix whose columns are the solutions of the generalized symmetric eigenproblem $Kx + \lambda Mx = 0$ and call Λ the diagonal matrix of eigenvalues, then the system can be rewritten as

$$\Lambda\left(z + \alpha_1 \frac{dz}{dt}\right) + M\left(\alpha_2 \frac{du}{dt} + \frac{d^2z}{dt^2}\right) = g \tag{2.23}$$

where $z = W^{-1}d$ is the vector of modal coordinates and $g = W^T f$ is the external force vector in the modal coordinate system. Each row of this system corresponds to a decoupled second order differential equation and represents one mode of the original system. Each of the modes can be obtained analytically solving the differential equation, moreover a consideration about the mode can be made: the square of the eigenvalue associated with a mode is the mode natural frequency (in radians per second).

The decoupled system is not an approximation of the original system, it represent the same behavior of the original one, but it can be solved analytically without using a numerical integration method. The great advantage of modal analysis is that it allows to discard unwanted or unimportant modes. In fact if ω_i is the solution of the i-th second order differential equation in Equation 2.23, its imaginary part determines the frequency that a model will vibrate at. Modes that vibrate at more than half the simulation frame rate will cause temporal aliasing and can be neglected. Moreover if the eigenvalue λ_i associated to a mode is large, then the force required to cause a noticeable displacement of that mode will be large. As it is usually possible to make assumptions on the forces that will be handled during a simulation, modes that require too high of a force can be discarded.

2.2 Mass Spring Models

Mass spring models are the simplest of all deformable models. They are not based on continuum equations, as finite element method and, as we will describe later, meshless models are, instead they discretize the body in a set of point masses that are connected to each other by a network of ideal springs [41] [40] [2].

The state of the system at a given time t is defined by the positions $\mathbf{x}_i$ and the velocities $\mathbf{v}_i$ of the masses $i = 1...N$. Each mass is subject to a force $\mathbf{f}_i$ computed as the sum of the forces due to all springs that the mass is connected to, and to external forces such as gravity, friction, etc. The motion of each particle is governed by Newton's second law $\mathbf{f}_i = m_i\ddot{\mathbf{x}}_i$, where m_i is the mass associated to the i-th point. The entire system can then be expressed as:

$$\mathbf{M}\ddot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{v}) \tag{2.24}$$

where $\mathbf{M}$ is a $3n \times 3n$ diagonal mass matrix.

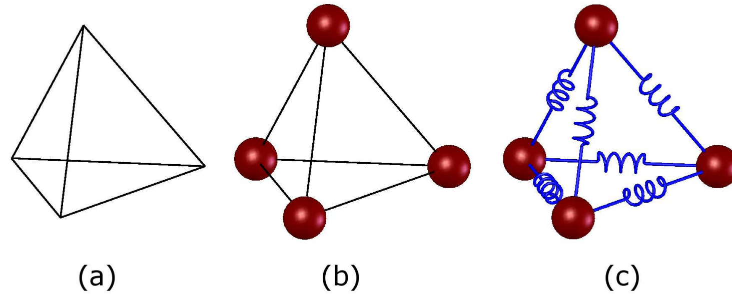


Fig. 2.4. The construction of a simple mass spring model: starting from a mesh (a) masses are placed at mesh nodes (b) and springs are added along mesh edges (c).

The main characteristic that differentiate mass spring models is the topology of the spring net. There are many possibilities to arrange springs: points can be regularly spaced in the body volume and springs can connect them to form a regular lattice, or points can be unevenly positioned to satisfy some criteria depending on the model surface curvature or on the local density. Moreover springs can be arranged to compose cubes, tetrahedra or other regular or irregular elements. As stated in the introduction, the model assume linear spring behavior: the force exerted by a spring connecting mass i to mass j depends only on its elongation:

$$\mathbf{f}_i = k_s(\|\mathbf{x}_j - \mathbf{x}_i\| - l_{ij})\frac{(\mathbf{x}_j - \mathbf{x}_i)}{\|\mathbf{x}_j - \mathbf{x}_i\|} \tag{2.25}$$

where k_s is the spring stiffness and l_{ij} is its rest length. Mass spring models are simple to understand and implement, moreover they are computationally efficient and can easily handle large deformations. However, since they are not built upon elastic theory, mass spring systems are not always accurate. They are, generally, not convergent: i.e. refining the mesh does not produce more realistic results. Another difficulty related to the use of those models is the identification of model parameters: mass values and spring stiffness [119]. Several extensions to *MSM* have been proposed in the literature to enrich their behavior and make them more realistic. In particular two necessary properties to ensure the realism are internal damping and volume preservation, as described in the next section.

2.2.1 Damping in Mass Spring Models

To enrich the behavior of the model and make it more realistic, energy dissipation is introduced. To account for this, while preserving the simplicity of the model, viscoelastic springs are introduced. Thus, in addition to the force computed with Equation 2.25 each spring exerts a viscous force that can be computed as the difference of velocities at its ends weighted by a damping factor k_d. Different

and more complex configurations can be taken into account, as showed in Figure 2.5, and each model leads to a different behavior, described in Figure 2.6. In the following we will describe mass spring models based on the Voigt model (Figure 2.5.b). It is the most commonly used in literature as it provides better results in interactive simulations.

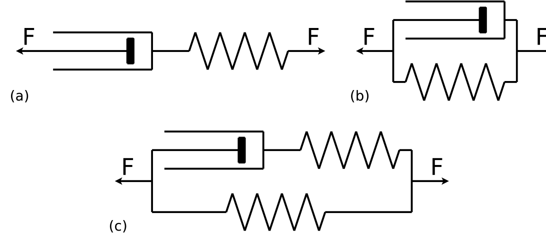


Fig. 2.5. Different viscoelastic models: in (a) a Maxwell body, in (b) a Voigt model and in (c) a Kelvin, or standard linear, model.

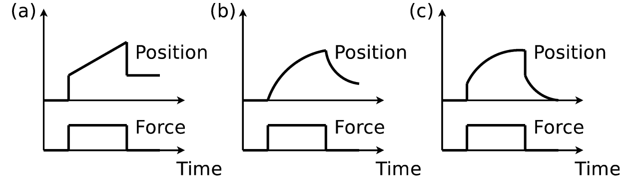


Fig. 2.6. Viscoelastic model responses for an applied force: in (a) for a Maxwell body, in (b) for a Voigt model and in (c) for a Kelvin model.

This simple approach leads to unrealistic simulation since rigid rotations or motions of the model result damped. A more realistic behavior can be obtained by projecting the velocities along each spring direction:

$$\mathbf{f}_i = k_d \left(\frac{(\mathbf{v}_j - \mathbf{v}_i)^T (\mathbf{x}_j - \mathbf{x}_i)}{(\mathbf{x}_j - \mathbf{x}_i)^T (\mathbf{x}_j - \mathbf{x}_i)} \right) \tag{2.26}$$

With this approach any rigid movement of the model in the space is not damped as the relative velocity of any couple of its nodes $(\mathbf{v}_j - \mathbf{v}_i)$ is zero. The identification of the damping coefficient to associate to each spring of the model requires suitable calibration methods.

2.2.2 Volume Preservation

Volume preservation is a property that can not be neglected in tissue modeling. Mass spring models do not guarantee that model volume remains constant during the simulation since the acting forces counterbalance the compression of linear and

not volumetric elements. It is then useful to introduce radial forces (that can not be modeled with springs positioned along the edges of tetrahedra) that oppose the variations of the tetrahedron volume. To obtain the desired radial forces, they can be computed as pointing from the vertices of the tetrahedron to its center of mass [76]. The modulus of the force can be computed as suggested in [9]

$$F_{it} = \left(\sum_{j=1}^{4} \|\mathbf{r}_j - \mathbf{r}_b\| - \sum_{j=1}^{4} \|\mathbf{r}_j^0 - \mathbf{r}_b^0\| \right) \frac{\mathbf{r}_i - \mathbf{r}_b}{\|\mathbf{r}_i - \mathbf{r}_b\|} \tag{2.27}$$

where the sums range over the 4 vertices of the tetrahedron. Each vertex has current and rest coordinates $\mathbf{r}_i$ and $\mathbf{r}_i^0$, where $\mathbf{r}_b$ and $\mathbf{r}_b^0$ are the current and original coordinates of the center of mass of the tetrahedron. The volume conservation forces that act on vertex i should be summed up over tetrahedra that share the vertex to obtain the whole volume force. The obtained volume force is usually weighted by a constant k_V (that can be seen as the stiffness coefficient of a volumetric spring) that weights the contribution of the volume preservation over the contribution of other forces (elastic and damping).

2.3 Meshless Models

Meshless (or mesh-free) methods have been widely used for solving partial differential equations (PDEs) numerically [99] based on a set of scattered nodes without connectivity properties thus without the mesh structure that comes from model volume discretization and that must be used for finite element methods. Their use have been recently extended to interactive simulations [92] [69] [68]. The advantages of meshless methods for computer animations are manifold: there is non need to generate a mesh of nodes for simulation, the nodes only need to be scattered within the solid object, which is much easier to handle in principle. Properties such as spatial adaptivity (node addition or elimination) and shape function polynomial order adaptivity (approximation or interpolation types) are naturally provided. Data management overhead can be minimized during the simulation. There are many variants of the meshless method, the most common use the Moving Least Square (or *MLS*) [1] shape functions which has been first employed in the Element Free Galerkin (EFG) method [100] [115] because of its high rate of convergence and high efficiency.

Mesh free method are an approximation technique that only requires a set of nodes (sampling nodes) distributed across the entire analysis domain. The value of the investigated or field function is approximated at those nodes by using shape or weight functions. One shape function is associated to each node and it controls the contribution of that node to the surrounding ones. For simplicity in model definition and computation, shape functions differs, in a single model, only for some parameters that are tuned considering local node density and model topology. It is possible to express a finite element model as a meshless model where the shape functions are constructed utilizing the mesh of the elements. In sharp contrast with *FEM* and *MSM*, the shape functions in mesh free methods are constructed using only the sampling nodes without any connectivity.

2.3.1 Physically Based Meshless

One of the most common methods used to obtain the weights needed for meshless approximation is *Moving Least Squares* or *MLS* [115]

We associate each node I of the model with a positive shape function w_I of compact support. The support of the shape function defines the domain of influence of the node Ω_I:

$$\Omega_I = \left\{ \mathbf{x} \in \mathbb{R}^3 w_I(\mathbf{x}) = w(\mathbf{x}, \mathbf{x}_I) > 0 \right\} \tag{2.28}$$

where $w(\mathbf{x}, \mathbf{x}_I)$ is the shape function associated with node I evaluated at position $\mathbf{x}$ and is used to weight the contribution of the node I to the field function.

The approximation of the field function f at a position $\hat{\mathbf{x}}$ is only affected by those nodes whose weights are non-zero at $\hat{\mathbf{x}}$ (the active set $\mathcal{A}(\mathbf{x})$).

If $f(\mathbf{x})$ is the field function defined on the analysis domain Ω its approximation at position $\mathbf{x}$, indicated with $f^h(\mathbf{x})$ can be computed using *MLS*. With *MLS* it is possible to obtain the proper shape function for each node of the model. In fact we can define $f^h(\mathbf{x})$ as the sum of some polynomial basis functions $P_i(\mathbf{x})$ weighted for proper coefficients $a_i(\mathbf{x})$ and express in in vector form:

$$f^h(\mathbf{x}) = \sum_{i=i}^{m} p_i(\mathbf{x}) a_i(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\mathbf{a}(\mathbf{x}) \tag{2.29}$$

where m is the number of polynomial basis functions in the column vector $\mathbf{p}(\mathbf{x})$ and $a_i(\mathbf{x})$ and their coefficients, which are functions of the spatial coordinates $\mathbf{x}$. Even though higher order function are possible, linear basis functions are usually chosen (for $3D$ case: $\mathbf{p}^T_{(m=4)} = \{1, x, y, z\}$.

From the previous equation it is possible to derive $\mathbf{a}(\mathbf{x})$ minimizing a weighted L_2 norm:

$$J = \sum_{I \in \mathcal{A}(\mathbf{x})} w(\mathbf{x} - \mathbf{x}_I) \left[\mathbf{p}(\mathbf{x}_I)\mathbf{a}(\mathbf{x}) - f_I \right]^2 \tag{2.30}$$

where f_I is the nodal value associated with node I. Equation 2.30 can be rewritten in matrix form:

$$J = (\mathbf{Pa} - \mathbf{f})^T \mathbf{W}(\mathbf{x})(\mathbf{Pa} - \mathbf{f}) \tag{2.31}$$

where:

$$\mathbf{f}^T = (f_1, f_2, ..., f_n)$$

$$\mathbf{P} = \begin{bmatrix} p_1(\mathbf{x}_1) & p_2(\mathbf{x}_1) & ... & p_m(\mathbf{x}_1) \\ p_1(\mathbf{x}_2) & p_2(\mathbf{x}_2) & ... & p_m(\mathbf{x}_2) \\ ... & ... & ... & ... \\ p_1(\mathbf{x}_n) & p_2(\mathbf{x}_n) & ... & p_m(\mathbf{x}_n) \end{bmatrix}$$

$$\mathbf{W}(\mathbf{x}) = \begin{bmatrix} w(\mathbf{x} - \mathbf{x}_1) & 0 & ... & 0 \\ 0 & w(\mathbf{x} - \mathbf{x}_2) & ... & 0 \\ ... & .. & ... & ... \\ 0 & 0 & 0 & w(\mathbf{x} - \mathbf{x}_n) \end{bmatrix}$$

To find the coefficients $\mathbf{a}(\mathbf{x})$, we obtain the minimum of J by setting:

$$\frac{\partial J}{\partial \mathbf{a}} = \mathbf{P}^T \mathbf{W}(\mathbf{x})\mathbf{Pa}(\mathbf{x}) - \mathbf{P}^T \mathbf{W}(\mathbf{x})\mathbf{f} = \mathbf{A}(\mathbf{x})\mathbf{a}(\mathbf{x}) - \mathbf{B}(\mathbf{x})\mathbf{f} = 0 \tag{2.32}$$

So we can obtain:

$$\mathbf{a}(\mathbf{x}) = \mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x})\mathbf{f} \tag{2.33}$$

And the shape functions are given by:

$$\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), ..., \phi_n(\mathbf{x})] = \mathbf{p}^T(\mathbf{x})\mathbf{A}^{-1}(\mathbf{x})\mathbf{B}(\mathbf{x}) \tag{2.34}$$

To obtain a certain consistency of any desirable order of approximation, it is necessary to have a complete basis. The basis functions $\mathbf{p}(\mathbf{x})$ may include some special terms such as singularity functions, i.e. special functions that ensure the consistency of the approximation and improve the accuracy of results in case of discontinuities in the modeled body (like cracks or difference or cuts).

The weight functions $w(\mathbf{x}, \mathbf{x}_i)$ play an important role in constructing the shape functions: they should be positive to guarantee a unique solution for $\mathbf{a}(\mathbf{x})$, they should decrease in magnitude as the distance to the node increases to enforce local neighbor influence and they should have compact support which ensure the sparsity of the global matrices (for computational efficiency) [84].

One key attractive property of *MLS* approximations is that their continuity is directly related to the continuity of the weighting functions. Thus a lower-order polynomial basis $\mathbf{p}(\mathbf{x})$ such as the linear basis can still be used to generate highly continuous approximations by choosing appropriate weight functions with certain smoothness requirements.

When applied to deformable model simulation *MLS* allows to reconstruct the displacement field $\mathbf{u}$ in the body volume and to compute its spatial derivative $\nabla\mathbf{u}$. Those computations are then used to obtain the strains and stresses of the material and to update the model configuration.

In particular, given the displacement vector field $\mathbf{u} = (u, v, w)^T$ represented by the scalar displacements $u = u(x, y, z)$, $v = v(x, y, z)$, $w = w(x, y, z)$ the deformed position of a point originally located at $\mathbf{x}$ is $\mathbf{x} + \mathbf{u}$. The Jacobian of the mapping is given by:

$$\mathbf{J} = \mathbf{I} + \nabla\mathbf{u}^T = \begin{bmatrix} u_{,x}+1 & u_{,y} & u_{,z} \\ v_{,x} & v_{,y}+1 & v_{,z} \\ v_{,x} & w_{,y} & w_{,z}+1 \end{bmatrix} \tag{2.35}$$

Then the strain can be expressed, using the quadratic Green - Saint-Venant strain tensor:

$$\varepsilon = \mathbf{J}^T\mathbf{J} - \mathbf{I} = \nabla\mathbf{u} + \nabla\mathbf{u}^T + \nabla\mathbf{u}\nabla\mathbf{u}^T \tag{2.36}$$

Assuming a Hookean material, i.e. $\sigma = \mathbf{E}\varepsilon$, the elastic body force U is obtained with:

$$U = \frac{1}{2}(\varepsilon\sigma) = \frac{1}{2}\left(\sum_{i=1}^{3}\sum_{j=1}^{3}\varepsilon_{ij}\sigma ij\right) \tag{2.37}$$

and the elastic force per unit of volume at a point $\mathbf{x}_i$ is the negative gradient of the strain energy density with respect to this point displacement $\mathbf{u}_i$ (that is, the directional derivative $\nabla_{\mathbf{u}_i}$ and can be expressed as:

$$\mathbf{f}_i = -\nabla_{\mathbf{u}_i}U = -\frac{1}{2}\nabla_{\mathbf{u}_i}(\varepsilon \cdot \mathbf{E}\varepsilon) = -\sigma\nabla_{\mathbf{u}_i}\varepsilon \tag{2.38}$$

The described *MLS* method is then used to obtain the approximation of $\nabla \mathbf{u}$ from the model point displacements. We will describe the method for the x-component u of the displacement field, a similar approach can be applied to obtain the y and z-components. In fact the continuous scalar field $u(\mathbf{x})$ can be approximated, in a neighborhood of $\mathbf{x}_i$ using the Taylor expansion:

$$u(\mathbf{x}_i + \Delta\mathbf{x}) = u_i + \nabla u|_{\mathbf{x}_i} \cdot \Delta\mathbf{x} + O(\|\Delta\mathbf{x}\|^2) \tag{2.39}$$

where $\nabla u|_{\mathbf{x}_i}$ is $(u_{,x}, u_{,y}, u_{,z})$ computed at point i. Given u_i and its spatial derivative at point i we can approximate it at point j with:

$$\tilde{u}_j = u_i + \nabla u|_{\mathbf{x}_i} \cdot (\mathbf{x}_j - \mathbf{x}_i) = u_i + (\mathbf{x}_j - \mathbf{x}_i)^T \nabla u|_{\mathbf{x}_i} \tag{2.40}$$

where $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$. The error e of this approximation is defined as the square difference between the approximated values $\tilde{u}_j$ and the known values u_j, weighted by the weight function:

$$e = \sum_j (\tilde{u}_j - u_j)^2 - w(\mathbf{x}_j, \mathbf{x}_i) \tag{2.41}$$

Composing equations 2.40 and 2.41 yields to:

$$e = \sum_j (u_i + u_{,x} x_{ij} + u_{,y} y_{ij} + u_{,z} z_{ij} - u_j)^2 \tag{2.42}$$

where x_{ij}, y_{ij} and z_{ij} are the x, y and z components of $\mathbf{x}_{ij}$ respectively. Given the positions of the sampling points $\mathbf{x}_i$ and the sampled values u_i we want to find the values for $u_{,x}$, $u_{,y}$ and $u_{,z}$ that minimize the error e. Setting the derivatives of e with respect to $u_{,x}$, $u_{,y}$ and $u_{,z}$ to zero yields three equations in the three unknown:

$$\left(\sum_j \mathbf{x}_{ij} \mathbf{x}_{ij}^T w(\mathbf{x}_j, \mathbf{x}_i) \right) \nabla u|_{\mathbf{x}_i} = \sum_j (u_j - u_i) \mathbf{x}_{ij} w(\mathbf{x}_j, \mathbf{x}_i) \tag{2.43}$$

The 3×3 moment matrix $\mathbf{A} = \sum_j \mathbf{x}_{ij} \mathbf{x}_{ij}^T w(\mathbf{x}_j, \mathbf{x}_i)$ can be precomputed and inverted, and then used during the computation of v and w. If the matrix $\mathbf{A}$ is non-singular the derivatives can be computed as:

$$\nabla|_{\mathbf{x}_i} = \mathbf{A}^{-1} \left(\sum_j (u_j - u_i) \mathbf{x}_{ij} w(\mathbf{x}_j, \mathbf{x}_i) \right) \tag{2.44}$$

Some issue arises when the number of point in the neighborhood of sampling point i is less than 4 (including point i) or if points are co-planar or collinear the matrix $\mathbf{A}$ is singular and cannot be inverted. This can be avoided using a proper sampling step in the body volume. The matrix is inverted using SVD to reduce problems related to matrix ill-conditioning [79].

2.3.2 Shape Matching Based Meshless

A completely different approach that is worth considering for its simplicity, performance and results, is shape based meshless modeling, that has been presented in [81] and extended in [91]. The method is not based on a physical background, instead it matches the current configuration of the model with the original one, it constructs a transformation matrix from which a rigid motion and a strain component can be extracted. Discarding the effect due to the rigid motion it is possible to simulate a deformation on the model that is stable and efficient. Given a cloud of points that represents the model at the rest position x_i^0 and at the deformed position x_i the problem is to find the rotation matrix R and translation vectors t_0 and t that minimize

$$\sum_i w_i(R(x_i^0 - t_0) + t - x_i)^2 \tag{2.45}$$

where the w_i are the weights of individual points. The natural choice of these weights is the mass associated to the point in the simulation. Moreover the optimal translation vectors are the center of mass of the model in the initial end in the deformed state respectively ($t_0 = x_{cm}^0$ and $t = x_{cm}$). To find the optimal rotation it is useful to define $q_i = x_i^0 - x_{cm}^0$ and $p_i = x_i - x_{cm}$ so that the term to be minimized became $\sum_i m_i(Aq_i - p_i)$. The matrix A can be computed as:

$$A = \left(\sum_i w_i p_i {q_i}^T\right)\left(\sum_i m_i q_i {q_i}^T\right)^{-1} = A_{pq}A_{qq} \tag{2.46}$$

The rotation matrix can be found considering that A_{qq} contains only scaling and not rotation, and rewriting $A_{pq} = RS$ where $S = \sqrt{A_{pq}^T A_{pq}}$. Finally the goal position for the current configuration can be obtained as:

$$g_i = R(x_i^0 - x_{cm}^0) + x_{cm} \tag{2.47}$$

At each step of the simulation, points are attracted towards their goal position, the more they are moved toward the goal position the stiffer the body will appear. As stated in [81] the method is stable i.e. it does not require smaller temporal steps as the body became stiffer. Moreover it can handle quadratic deformations and plasticity.

2.4 Conclusions

The choice of the proper method to simulate soft body deformations highly depends on the requirements of the simulation. In fields were the computational time can be sacrificed to precision, i.e. mechanical engineering or brain surgery planning, nonlinear dynamic finite elements are used, whereas in interactive applications, such as laparoscopic surgery trainers, linear approximation usually offers a good trade off between realism and speed. When complex models are needed even linear models result computationally too heavy for standard PC architectures. In these cases parallel implementations help in reducing the computational time. One

of the most common approaches exploits graphics card hardware (Graphics Processing Unit or *GPU*) stream processors as a highly parallel architecture. This implementation method is well suited for interactive simulations, where temporal requirements are more important than realism. In fact the high number of processors embedded in a modern graphics card allows to obtain very high frame rate in the simulation of complex models. Moreover they proved to be suitable also for haptic rendering. In Section 7 we will present and discuss a *GPU*-based method that improves and extends *MSM* for interactive simulations with haptic feedback.

3

Common Representation Framework

As can be observed from Chapter 2 many methods have been developed to simulate deformations in virtual environments. They present many differences in both their implementation and in the "interface" they provide to other components of the simulation (such as, for example, collision detection algorithm, constraints imposition, ...). In this section we will address the problem of providing one single approach to handle different classes of deformable models. This approach is based on, and extends, previous works developed to handle meshless models in graphical rendering.

Very few works address the problem of representing the same tissue with different models but this ability can provide great advantages in simulations from the point of view of realism and computational speed up. In fact, it is not easy to define a priori the most appropriate model for a simulation. The model choice depends on many factors such as the required realism, the actions that the user perform on the model and even the rendering provided to the user. Thus a linear *FEM* can be appropriate for small deformations, when a good level of realism is required, but can fail in the simulation of cuts as the time required to update the data structures can be too high to ensure smooth rendering. Similarly *MSM*s are better in handling big deformations and changes in topology, but they are not suitable in handling volume variations. Meshless models can provide a good realism and the ability to simulate changes in topology, but their computational complexity makes them unsuitable for high resolution models. By developing a standard interface between the external world, represented by the force applied to the surface and visual rendering as well, we ensure that the best modeling can be used to simulate specific tasks and deformable objects. In the rest of this chapter we provide a brief survey of multi resolution and hybrid methods and we detail the proposed approach.

3.1 Adaptive Models

Methods that adapt the model to the simulation needs can be categorized in two main classes: multi resolution methods, that refine the model in the area where a higher realism is required, and hybrid methods, that combine different mod-

elization techniques in a single model and use their different features to balance the realism and the computational requirements of the simulation. Both methods introduce additional computations in the simulation, to adapt the number of elements (tetrahedra, points, ...) to the simulation needs or to create and handle the proper data structures.

3.1.1 Multi Resolution Models

One approach that can be followed to enhance the realism of the simulation is the use of multi resolution models. The ability to adapt the level of detail of the model in accordance to simulation needs can both speed up the computation and improve the realism of the simulation. Basically there are two approaches to multi resolution modeling. The first one is based on the off line definition of different meshes with increasing resolutions and on the pre computation of their physical parameters. The second approach consists in refining the model during the simulation. This approach requires on line remeshing methods and a proper algorithm to update the physical parameters of the structures involved in the simulation.

The inherent higher complexity of this second approach is justified by its versatility. In fact the use of precomputed data does not always allow to correctly adapt the model to the user needs. For example, in the case of a cutting action, the remeshed elements should follow the direction of the cut, but it is difficult to know a priori this direction and thus to pre compute the proper data. One limitation of the use of adaptive remeshing is the update of physical parameters. When *FEM*s are used it is usually straightforward to update finite element physical parameters due to the continuous nature of the approximation. In the case of meshless models the update of the shape functions introduced in Section 2.3.1 introduces some delays in the computation and, in the case of *MSM*s, the update of springs coefficients requires non trivial methods.

In particular, in [105] the author proposes a method to ensure a homogeneous behavior for *MSM*s at different resolutions. The methods is based on the observation that assigning a unique stiffness to all springs in a model does not lead to a homogeneous behavior. The author noticed that, under the previous assumption, higher point density leads to higher stiffness and developed a method to compute spring stiffness that is function of both the spring length and the volume associated to tetrahedra incident on the spring.

This approach has been used in some works where multi resolution is applied to *MSM*. In [19] the authors propose a multi resolution surface deformable model that can be dynamically adapted to user input. The described method is based on mass spring model whose topology is simplified and or detailed to match simulation needs. Three criteria of subdivision are used in mesh refinement: vertices (and springs) are added where points are subject to force higher than a threshold, where their velocity is too high or if the surface curvature exceeds a threshold. The model is refined using a modified butterfly method [30], extended to handle irregular cases. Butterfly method is defined on meshes where all vertices has valence six, i.e. each vertex has six neighbors. The proposed scheme defines the new vertex in the case it lies on an edge that connects two vertices whose valence is not six.

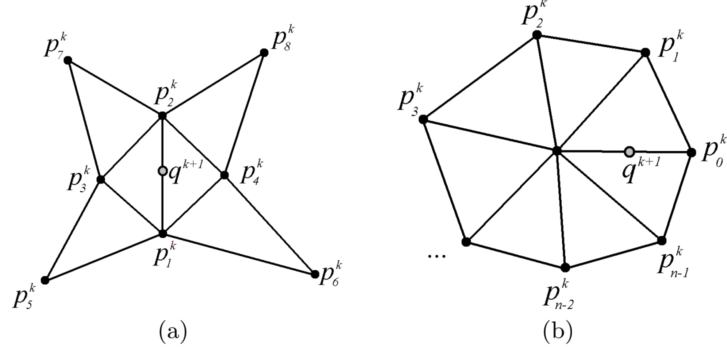


Fig. 3.1. Modified butterfly subdivision scheme: (a) regular cases, (b) irregular cases

Edges to be subdivided can be classified depending on the valence (i.e. number of neighbors) of connected vertices. The added point coordinates (q^{k+1}) are obtained as a function of the surrounding points (p^k).

- The edge connects two vertices of valence six. In this regular case, the position of the new vertex is determined from the following equation (refer to Figure 3.1(a):

$$q^{k+1} = \frac{1}{2}(p_1^k + p_2^k) + \frac{1}{8}(p_3^k + p_4^k) - \frac{1}{16}(p_5^k + p_6^k + p_7^k + p_8^k) \tag{3.1}$$

- The edge connects a K-vertex (its valence is not six) and a 6-vertex. The neighbors of the K-vertex are used as indicated in Figure 3.1(b). In this case the following formula is used:

$$q^{k+1} = \sum_{i=0}^{N-a} S_i p_i^k \tag{3.2}$$

where

$$S_i = \frac{\frac{1}{4} + cos\left(\frac{2\pi i}{N}\right) + \frac{1}{2} cos\left(\frac{4\pi i}{N}\right)}{N}, \qquad 0 \leq i \leq N-1 \tag{3.3}$$

.

- The edge connects two K-vertices. If both of the vertices are irregular, the above formula is applied to both vertices and the results are averaged.

The obtained model solves the problem of adapting spring stiffness to the changes in topology with an approach based on the method proposed by Van Gelder in [105] and thus ensures the integrity of dynamical behavior at different resolutions. Results are shown in Figure 3.2

Another approach that adapts at run time the topology of the model is described in [85]. The work applies to both *FEM*s and *MSM*s, in fact it describes a method to refine and simplify tetrahedral meshes ensuring quality of the modified mesh. The quality of the mesh is preserved by applying a flip operation to

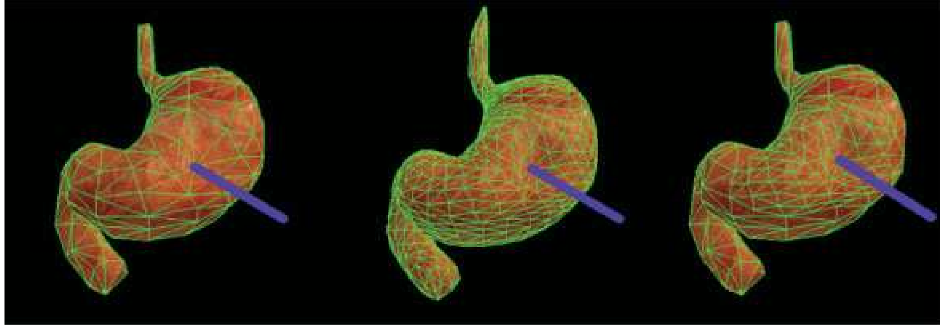


Fig. 3.2. Comparison of low resolution mesh on the left, high resolution model in the center and adaptive mesh obtained with method proposed in [19].

tetrahedra: by flipping the edges of adjacent tetrahedron to maximize the minimum corner measure the algorithm always produce the optimal mesh (in terms of simulation stability). An important aspect that is considered in this work is the adaptation of the temporal step used in the simulation. In fact, as described in Chapter 2 temporal integration step is closely related to mesh resolution. The authors propose a higher bound to temporal step that ensures the stability of the simulation and the bounding of simulation errors. The complete method is applied to both *MSM* and nonlinear *FEM*, dynamic parameters are adapted at run time to maintain the dynamic behavior (see Figure 3.3). One great advantage of the obtained models is their ability of simulate changes in topology.

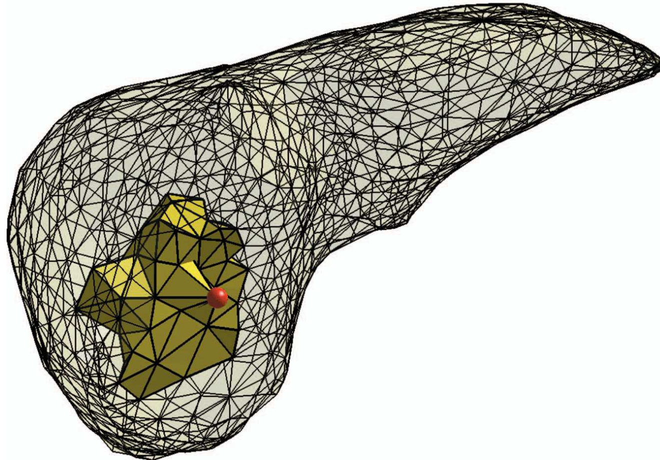


Fig. 3.3. Liver model used in [85] undergoing deformation modeled with *FEM*.

A different approach is detailed in [74] where a regular lattice is used to sample the body volume. This regular grid is adapted by two operators of reduction and

reconstruction. The model includes an elastic force approximation that is similar to shape matching based meshless models discussed in Section 2.3. Furthermore it does not require two adjacent elements to share nodes on the common face. The force acting on each element node is computed by equating to zero the resultant internal forces and the moment of internal forces:

$$\sum_i \mathbf{r}_i \times k(\mathbf{R}_i - r_i) = k \sum_i \mathbf{r}_i \times \mathbf{R}_i = \mathbf{0}. \tag{3.4}$$

Where $\mathbf{R}_i$ and $\mathbf{r}_i$ are the original and deformed positions of vertex i with respect to element center of gravity and k represent the element stiffness. The model handles changes in topology, e.g. a cut, with a two steps approach. First, elements on cut surface are split and internal elements are recursively split to adapt to neighbors size, then elements are recombined to locally preserve the element density. Results of the work are presented for bi-dimensional models only.

In [114] another adaptive *FEM*-based method is described. The underlying deformable model is a non linear *FEM* computed through the mass lumping technique, that allows obtaining diagonal matrices and thus, real time computation. The mass lumping technique adopted by the authors is column summation. Given the model mass symmetric matrix (introduced in Equation 2.18) M the correspondent lumped matrix $\overline{M}$:

$$\overline{M}(i,j) = \delta(i,j) \sum_{k=1}^{N} M(i,k) \tag{3.5}$$

where N is the number of rows of the matrix N and $\delta(i,j)$ is the Kronecker delta ($\delta(i,j) = 1$ if $i = j$ and equals 0 otherwise). The appropriate level of detail is ensured by an adaptive meshing techniques that exploits off line computed data to speed up the computation during the simulation. In fact the diagonalization of the matrix cannot be performed at run time, thus a hierarchy of meshes and the relative matrices are precomputed and used to obtain element parameters during the simulation. The main limitation of the method is its inability to fast adapt the mesh to the situation. In fact, since the refinements are precomputed, they can be not suitable in some situations: points cannot be placed at wish, instead new points must lie in one of the precomputed positions. This causes many steps of refinement to be performed to obtain the optimal mesh resolution.

Another method, proposed in [26], combines spatial and temporal multi resolution. The underlying model is a finite element model based on the Green elasticity tensor [102]. The body is partitioned in a non-nested multi resolution hierarchy of tetrahedral meshes. A quality criterion handles the changes in resolution of the model. The criterion is based on the error due to the linearization of the displacement field, that can be expressed as $\Delta \mathbf{d} h^2$ where h is the minimum distance of the considered node to its neighbors and $\Delta \mathbf{d}$ is the Laplacian of the displacement $\mathbf{d}$. The force $\mathbf{f}$ per volume V is directly linked to the Laplacian of the displacement field for almost incompressible objects [25]: $\mathbf{f} \approx V\mu\Delta\mathbf{d}$, where μ indicates the first Lamé coefficient. The quality criterion γ measures the adequacy of a node in the simulation using the simple approximation:

$$\gamma = \Delta \mathbf{d} h^2 \approx \frac{\mathbf{f} h^2}{\mu V} = \frac{\mathbf{f}}{\mu h} \tag{3.6}$$

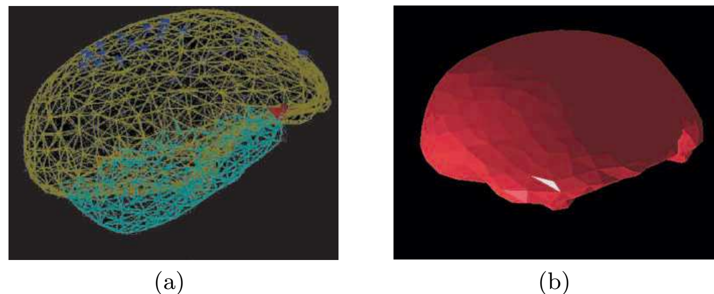
If γ exceeds a threshold γ_{max} or if it drops under γ_{min} the node is merged or split, respectively. Unlike the method discussed in [85] the temporal adaptation is defined on each particle of the model and it is not a unique simulation parameter. This greatly reduces the computational cost of the simulation but it must be correctly handled to interpolate computed values.

3.1.2 Hybrid Models

Research has also been made on the use of hybrid models. The most common approach to hybrid modeling consists in combining pre computation and on line computation to improve results or simulation time, but some work has been done in integrating different modelization techniques. The use of different models ensure the possibility to use the proper method in each part of the simulation and to adapt the realism to the simulation needs.

In [36] the realism of the contact with a deformable model is increased by the use of a hybrid model. The proposed method uses a standard *MSM* model to compute the overall deformation of the body and uses a *FEM* local precomputed model to simulate the area of the contact. The contribution coming from the *FEM* is weighted by the distance from the contact point and summed to the deformation provided by the coarse *MSM*.

Conversely, in [27] a quasi-static *FEM* (a *FEM* that neglects the dynamics of the model) is used to simulate the global deformation, whereas a local tensor mass model (an extension of *MSM*s) is used to simulate the region where the interaction takes place, allowing the simulation of topological changes (see Figure 3.4). In this approach the forces coming from the two models are not combined but each tetrahedron is assigned to one model. In this way no ghost forces appear when the topology changes. The main limitation of this method resides in the tensor mass model, in fact it is not invariant to rigid transformations: i.e. a translation or a rotation induce forces into the model. This makes the model valid only for small displacements.



(a) (b)

Fig. 3.4. A model obtained with the method described in [27]. In (a) the hybrid liver model seen in wire frame. The upper mesh corresponds to the precomputed, quasistatic, elastic model, whereas the bottom mesh corresponds to the tensor mass model; in (b) the hybrid liver model seen in flat shading.

Another hybrid approach to deformable model simulation is proposed in [72] and [73]. The method described in these works couples a *FEM* and a meshless model to model human heart. The *FEM* is based on Green's strain tensor and the material is considered isotropic. Internal damping is introduced in the *FEM* to increase the realism of the simulation and is computed from the strain rate tensor μ that measures the rate at which the strain is changing and it is the time derivative of ε:

$$v_{ij} = \left(\frac{\partial \mathbf{x}}{\partial u_i} \cdot \frac{\partial \dot{\mathbf{x}}}{\partial u_j}\right) + \left(\frac{\partial \dot{\mathbf{x}}}{\partial u_i} \cdot \frac{\partial \mathbf{x}}{\partial u_j}\right), \tag{3.7}$$

where $\mathbf{x}$ indicates the *FEM* point velocity and u_i with $i \in [1, 2, 3]$ is one of the directions along the frame of reference. The viscous stress σ^v is then:

$$\sigma^v_{ij} = \sum_{k=1}^{3} \phi v_{kk} + 2\psi v_{ij} \tag{3.8}$$

where ϕ and ψ control how fast the material looses or dissipates kinetic energy. Along with the *FEM* in areas where external forces will be applied during the simulation, sampling points will be added and used to create the meshless model. The complete model will handle global deformations with the *FEM* and will improve local accuracy by using the meshless model. The method allows the two models to overlap in some regions. Preliminary results of the method have been obtained in the modelization of left ventricle of human heart.

Heart modeling is also addressed in [75] but the proposed approach exploits *MSM*s and a continuum mechanical model. One important feature of cardiac tissue considered in the work is anisotropy. To correctly simulate anisotropic behavior *MSM* is defined over a regular cubic grid, and springs are placed on edges, faces and interior part of the cube. The continuum mechanical model is based on a strain energy function that, similarly to *FEM* based methods, allows to compute the material stress starting from its strain. The overall model is discretized in a regular volumetric lattice composed by voxels. Each voxel is associated to only one model and thus the two models only shares faces and edges of elements. The integration of the two models is then straightforward as it is simply performed by the summation of the contributions coming from the two different models on shared points.

In the work described in [112] a hybrid model is proposed and implemented exploiting graphic card processing unit. The approach splits the model in two areas, the first one, the non operational area is the part of the model where interactions happen only on the surface of the model and is modeled with condensed *FEM*. The second one, the operational area, models the zones where the user can apply topological changes to the model and exploits standard *FEM*. This approach requires a priori knowledge of the area where the user will interact during the simulation and thus greatly reduce the field of application of the method.

3.2 Common representation

In this section we describe the approach we propose to overcome the limitations of current adaptive methods. Multi resolution and hybrid approaches present some difficulties in ensuring a constant computational time when the complexity of the model, in terms of details or realism, changes. In fact when models are refined and new elements added, the computational time increases accordingly and it is usually not possible to know a priori how many new elements are needed to provide the required realism. We propose to construct different models of the same object, using different resolution and modeling techniques. The use of different models can better fit the requirements of the simulation. To ensure that the user does not perceive the difference between different models used, we separate the physical modeling from the graphical rendering. One unique graphical representation is used for all different models. This guarantee the ability to switch models without affecting user visual perception but requires a method to link the surface of the model to its physics element.

The approach we propose is based on the introduction of a new entity in the simulation, i.e. a "skin" wrapping the deformable object structure that hides model features and that is simulated separately from the object itself. This surface is encoded as a triangular mesh, defined on a set of points, called *surfels* (for surface elements). Surfels are the nodes of this representation, those nodes collect forces form the extern of deformable model (e.g. friction or constraints) and distribute them to the physics element of the model. They also control how internal forces are propagated to the external environment. At the same time they handle deformations of the model, in fact the displacement defined on the physics element is interpolated on surfels to control surface deformation and displacement due to collisions are imposed, by surfels, to internal elements. As we will discuss in Chapter 5, frictional contact depends on contact normal forces, and contacting surface velocities and properties. Thus the described structure is exploited to separate the deformation simulation from the handling of frictional contact. This requires that the surface embodies some physical properties such as friction coefficient. The separation of the deformable model from its surface representation allows to adapt the resolution of the physics to simulation needs without affecting the resolution of its surface, keeping unaltered the realism of the interactions. The main problem that rises from this separation is the need for keeping the surface model and the deformable model "synchronized" by propagating displacements and forces.

3.2.1 Point Based Approach

A similar problem can be found in meshless modeling and in point based rendering: where the physics and the graphics of models are handled separately. The problem can be formalized as the reconstruction of surfaces from unstructured data samples. The representation of two dimensional surfaces in three dimensional space can be classified in two groups: implicit surfaces and parametric surfaces.

Implicit surfaces are defined as the zero set of a scalar function over the whole three dimensional domain. The algebraic structure of this representation is simple and largely independent of the topological complexity of the surface. Surface deformations and topological changes can be easily applied while preserving the global

consistency of the surface. The scalar field can be specified through radial basis functions [15] or with level sets [82]. Point set surfaces are based on moving least squares approximation and a projection operator to implicitly define the surface.

Parametric surfaces are defined by mapping a two dimensional domain into three dimensional space. In computer graphics the use of Bézier curves, B-splines, NURBS and subdivision surface is quite common. A survey of this topic can be found, for example, in [32]. These representations are based on a mesh of control points with known connectivity. B-splines and NURBS surfaces require that the underlying mesh has a regular connectivity, whereas subdivision surfaces work also with semi regular meshes. Parametric surfaces provide some advantages in handling texture mapping, but they are more complex to adapt to topological changes.

In this work we focus on implicit surfaces because of their simplicity and ability to handle changes in topology. The method we have developed is based on approaches developed to render meshless based deformable models. As described in Section 2.3, meshless models are based on a set of physical points that are used to approximate the displacement inside the modeled body volume. A meshless model provides no information about point connectivity, thus it is not possible to extract a surface mesh representation from its structure. To overcome this limitation and to provide an effective method to visually render them, a set of surface points or *surfels* (in opposition to *phyxels*, the physics points) are scattered along the model surface before the simulation and are used in the graphical rendering.

3.2.2 Geometric Analysis

To compute the surfels, the surface of the undeformed body should be known. This appears as a reasonable assumption for solid bodies for which the surface can usually be extracted in form of an isosurface, signed distance function or simply by a polygonal mesh. During the simulation, data computed for the phyxels are used to approximate the displacement of surfels with a procedure similar to the one described in Section 2.3.1. The displacement vector $\mathbf{u}_{sfl}$ at the surfel position $\mathbf{x}_{sfl}$ is computed from the displacements $\mathbf{u}_i$ of the neighboring phyxels. To obtain a good approximation of surface displacement a *MLS* method is used. The value $\mathbf{u}_{sfl}$ can be expressed as:

$$\mathbf{u}_{sfl} = \frac{1}{\sum_i w(\mathbf{x}_i, \mathbf{x}_{sfl})} \sum_i \omega(\mathbf{x}_i, \mathbf{x}_{sfl}) \left(\mathbf{u}_i + \nabla \mathbf{u}_i^T (\mathbf{x}_{sfl} - \mathbf{x}_i)\right) \tag{3.9}$$

where $\mathbf{u}_i$ is the displacement computed for the phyxel in position $\mathbf{x}_i$ and ω is a proper weighting function. The choice of the weighting function ω is of great importance for the quality of results. In point based rendering the number of surfels is much bigger than the number of phyxels, and the time spent on the computation of the ω function can considerably slow down the simulation. For this reason simple weight functions are commonly used in the computation of surface deformation: a very common choice for ω is a truncated Gaussian function:

$$\omega'(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} e^{(-\|\mathbf{x}_j - \mathbf{x}_i\|^2/h^2)} & \text{if } \|\mathbf{x}_j - \mathbf{x}_i\| < h \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

The value h defines the region that affects the surfel behavior. Big value of h ensures a smooth surface at higher computational cost, whereas small values of h increase the computation speed and provide sharper surfaces, at the risk of surfels that are affected by only one phyxel or completely separate from the underlying physics in case of big deformations. Thus the choice of h requires particular attention. h value is usually not unique for all elements of the model. We use an approach similar to the one proposed in [79]: we allow irregular sampling of model volume and surface. For each element i we compute the average distance $\bar{r}_1$ to its 10 nearest neighbors then we define $h_i = 3\bar{r}_i$. This ensure good results of realism and stability in the simulation.

One limitation of the truncated exponential function is its discontinuity at $\|\mathbf{x}_j - \mathbf{x}_i\| = h$, where the function jump to 0 with a discontinuity. This is in contrast with the *MLS* approach that ensures the continuity in the physics simulation. In contrast with point based rendering techniques, our method works on a triangular mesh defined on the points on the model surface. This guarantees a good realism in the rendering even if a, relatively, small number of surfels is used. For this reason we can afford more complex ω' functions that preserve the continuity property also for the model surface. For these reasons we employ a polynomial weighting function [79]:

$$\omega(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \frac{315}{64\pi h^9}\left(h^2 - \|\mathbf{x}_j - \mathbf{x}_i\|^2\right)^3 & \text{if } \|\mathbf{x}_j - \mathbf{x}_i\| < h \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

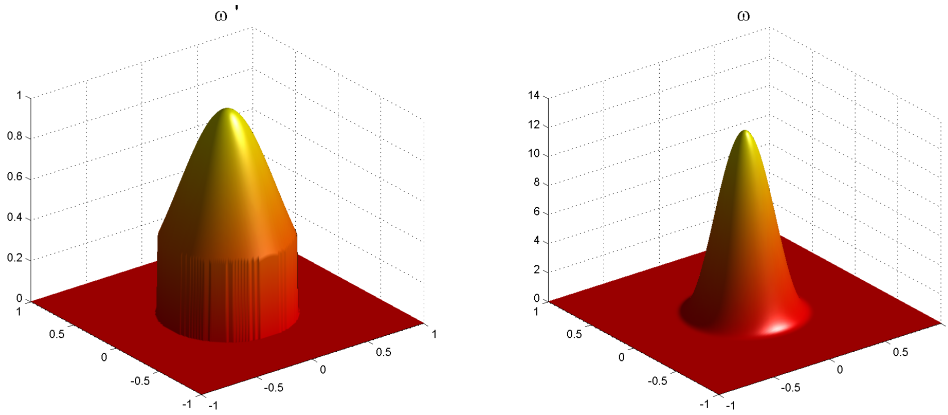


Fig. 3.5. The two proposed weighting functions ω' and ω on the left and the right respectively. The chosen value for h is 0.5.

The differences between the two weighting functions are depicted in Figure 3.5. The differences in the amplitude are due to the scale factor used in the computation of ω that normalizes the area beneath the curve. The scaling factor can be neglected if the displacement of the surfel is computed as in Equation 3.9 since the normalization is carried out by the division by the sum of the weights $\sum_i w(\mathbf{x}_i, \mathbf{x}_{sfl})$. An

example of interpolation using Equation 3.9 and weighting function ω defined in Equation 3.11 is shown in Figure 3.6.

This method is particularly suited to handle meshless models, where relatively few physical points are considered, and when the spatial derivatives $\nabla \mathbf{u}$ of the displacement field are known. For *FEM*s or *MSM*s, where the number of physical points needs to be higher to ensure a good realism, the overhead introduced to compute $\nabla \mathbf{u}$ is not affordable. Moreover, due to the high physical point density, the contribution of $\nabla \mathbf{u}$ to the displacement of surface points (Equation 3.9) can be neglected and visual realism is still good. Simulations of the same body, performed with different deformable models are shown in Figure 3.7. The three models behavior is clearly different, and this is due to the different approach in deformation modeling. The external mesh is the same for the three models and it is defined on surfels that are located at the same position with respect to the undeformed body. From the figure it is possible to see that the proposed method can handle sharp edges with no computational overhead and that it is suitable for meshless models too.

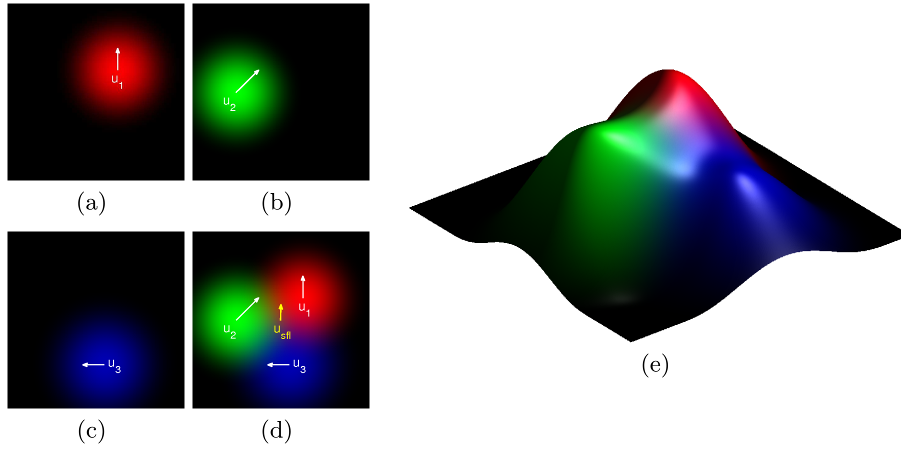
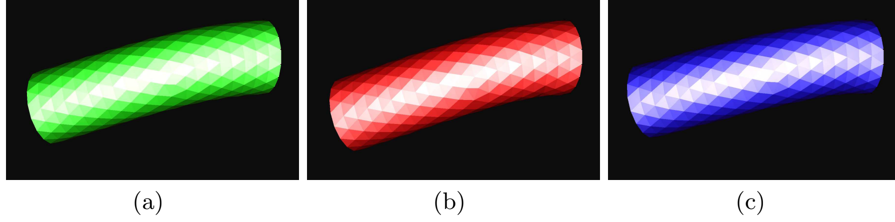


Fig. 3.6. Interpolation using Equation 3.9 and weighting function ω. In (a) the displacement value for phyxel 1, and the value of its weighting function. In (b) and in (c) the data for phyxel 2 and 3 respectively. In (d) the three contributions are plotted together and the displacement for a surfel placed in the center of the figure is computed. In (e) the distribution of weighting functions in 2D space is plotted.

3.2.3 Dynamic Analysis

Another important step in this approach to deformable bodies modeling is the propagation of forces between physics and surface. The propagation of forces can



(a) (b) (c)

Fig. 3.7. Deformation of the same body undergoing gravity force simulated with different methods. From left to right: finite element model, mass spring model and meshless model. In the first two simulations the contribution of $\nabla \mathbf{u}$ is neglected during the rendering phase.

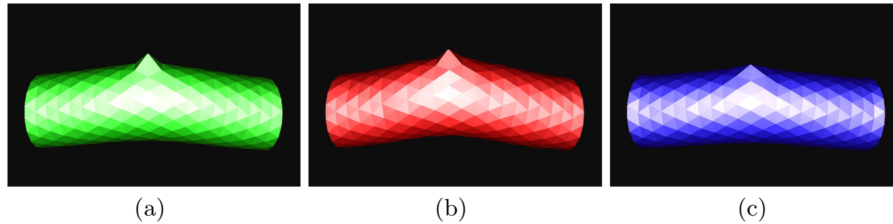
be handled similarly to the propagation of displacements but paying attention to avoid changing the overall forces in the system. This requires a different function for distributing forces from surfels to phyxels and vice versa. To ensure that no forces are created or destroyed during the propagation we used the following function to compute the force acting on phyxel j:

$$\mathbf{f}_j = \sum_{i \in S} \frac{\omega'(\mathbf{x}_i, \mathbf{x}_j)\mathbf{f}_i}{\sum_{k \in P} \omega'(\mathbf{x}_i, \mathbf{x}_k)} \tag{3.12}$$

where $\mathbf{f}_i$ indicates the force acting on the i-th surfel, S is the set of model surfels and P is the set of model phyxels. Equation 3.12 ensures that all and only the force acting on each surfel is distributed to neighboring phyxels. In fact the term in the denominator of Equation 3.12 represents the sum of the weights assigned to phyxels influences by surfels i. In this way the single contributions due to surfel i are normalized and the resulting weights sum to one. This ensure that all and only the forces that act on the single surfel i are propagated to model phyxels.

By swapping the role of surfels and phyxels the same method can be used to distribute the forces from the physical representation of the model to its surface and used during the computation of contact forces. The use of the truncated exponential function described in Equation 3.10, even if not continuous on its border, is justified by the nature of the approximated value: in fact forces applied by the surface to surfels are transferred to internal physical elements (and thus smoothed) by the physical simulation. The result of external forces applied to different deformable models can be evaluated in as Figure 3.8, where the three models presented in Figure 3.7 are deformed by an external force. The force and the application point is the same in all three cases.

The wrapping surface not only provides an interface to constraint forces and displacements between the deformable model and the external environment, but it is also used to handle the friction force that generates during the contact between deformable bodies and other entities. During a contact, in fact, two kind of forces act: reaction forces and frictional forces. Our framework uses the physics of deformable models to compute the reaction forces keeping into account both dynamic properties of the colliding bodies and their stiffnesses. Other values needed

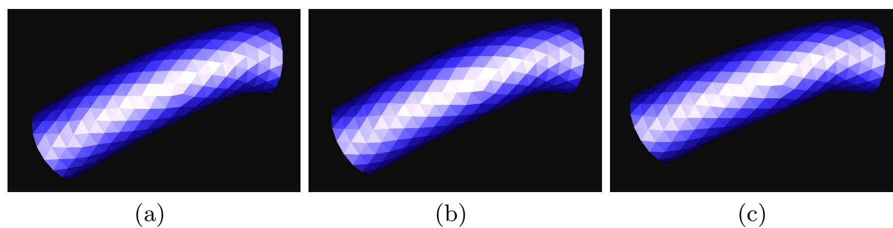


(a) (b) (c)

Fig. 3.8. Three different models deformed by the same external force. From left to right: finite element model, mass spring model and meshless model. In the first two simulations the contribution of $\nabla \mathbf{u}$ is neglected during the rendering phase.

to solve the frictional contact such as relative velocities between contacting surfaces and friction coefficient are obtained directly from the surface structures. This simplifies the computation of the frictional contacts allowing to abstract it from the underlying physical deformable model.

When propagating forces and displacements between the two parts of the model (surface and internal) the h value that governs Equation 3.10 plays a very important role. In fact it controls the area influenced by the contribution of the single element. Small values of h reduce the computational time, as less neighbors need to be processed for each element, but also lead to model where constraints (forces and displacements) effect is more local. On the other hand, higher values propagates constraints on wider areas but with a higher computational cost. To compute the h value for the ω' weighting functions we use the same approach defined in Section 3.2.2. For each element i we compute the average distance $\bar{r}_1$ to its 10 nearest neighbors then we define $h_i = 3\bar{r}_i$. Figure 3.9 compares the results obtained for a meshless physical model wrapped by surfaces with different values of h. In (a) we used a value $h_i = 1.5\bar{r}_i$, in (b) the used values is $h_i = 3\bar{r}_i$ whereas in (c) we used $h_i = 4.5\bar{r}_i$. As can be seen higher values of h propagates the constraint from the fixed end of the cylinder deeper along the model structure: this leads to an increment to the model rigidity.



(a) (b) (c)

Fig. 3.9. The same meshless model wrapped by different surfaces and deformed by gravity. Values of h are computed by $h_i = 1.5\bar{r}_i$, $h_i = 3\bar{r}_i$ and $h_i = 4.5\bar{r}_i$ in (a), (b) and (c) respectively.

3.3 Conclusions

The choice of the right model to simulate deformable environments is not straightforward. Different models generate very different results in terms of computational cost, fidelity of the simulation, and realism of the simulated task. To cope with this problem various approaches have been developed. Two main categories can be identified: multi resolution techniques and hybrid models. The first ones increases, at run time, the number of elements used in the simulation to increase the realism of the model. Hybrid methods, instead, use different modelization techniques to adapt the area of the model to the simulation requirements prior to the simulation. The main limitation of those methods is the increase in computational time in the former case and the strong constraints imposed on the simulation in the latter case. To ensure a bounded computational time and to allow the user to freely interact with simulated models, we developed a method that allows to handle in a common way different deformable models. The most appropriate simulation method is chosen off line based on simulation requirements. This method only affects the physics computation, ensuring a fixed computational time and the required realism of the simulation.

The proposed approach is based on a separation of the internal (or physical) part of the model from the visual representation, i.e. the surface, that acts as an interface between the internal part and the external world. The interactions between the world and the physical model are obtained through the exchange of forces and displacement from and to the model surface. Forces and displacements are handled similarly, with a method inspired by the point based rendering techniques. The difference in the nature of the exchanged values justifies the different weighting functions involved in the method. Preliminary results show that the distribution of the displacement on the surface of the model is realistic and adapt to different modelization techniques. At the same time the application of external forces to the model can be successfully achieved with the proposed method. This method may allow to switch, during the computation, from one physical model to another, this switch may not preserve the overall stabillity of the simulation, as it could introduce energy into the system. It is thus necessary that the simulation remains stable during and after the switch, with proper methods based on model physical properties (such as [105]) or on the actual switching (as it happens in [36] or in [73]).

4

Collision Handling

In this chapter we will address the problem of collision detection between bodies in virtual scenes. When simulating dynamic environments, bodies in the scene can come in contact and also penetrate each other. To ensure the realism of the simulation it is required that these situations are detected and properly solved. Collision are usually checked after the update of the scene and the results of the detection are used to restore the consistency. The whole process consists of two steps. The first part of the process, called collision detection, aims at obtaining an estimation of the instant of the contact and at identifying the points involved in the contact. The second part modifies the body configuration to avoid overlap between bodies and is usually called collision solution.

Collision detection represents an issue in many fields of research, it has been addressed in assembly and disassembly [47], in computer-aided design and machining [98], in tolerance verification [59] and computer simulated environments [64]. Its goal is the identification of the geometrical contact before it occurs or when it has actually occurred. Solving collision requires a proper method to handle the contact: contacting rigid bodies are usually displaced along the minimum penetration depth [120], whereas local methods are used to separate contacting deformable models [28, 38].

In this chapter we will introduce the principal techniques and libraries developed to perform collision detection, evaluating their use in the specific scenario of interactive, physically based simulation of deformable models. The introduction of complex contact mechanics in the simulation requires a high level of accuracy in the detection of contact points and in the solution of collisions, thus an innovative approach, specific for deformable triangular mesh, will be presented. The implementation of the described method relies on the use of root finding methods. Some algorithms for root approximations will be compared from the theoretical point of view, and their performance on real cases will be evaluated.

4.1 Collision Detection

Collision detection is a fundamental step in physically based simulations. The choice of the right collision detection algorithm is not always straightforward,

as it depends on the nature of the modeled phenomena, the time step involved in the simulation and the precision needed. Collision detection procedures have been developed to solve many classes of problems. Many criteria can be used to differentiate them but the main differences stay in the model properties that methods handle, the kind of queries that they can answer and the simulation environments they can handle.

4.1.1 General Approach

Different model representations have been developed to suit the needs of different tasks. Interacting bodies can be represented by non polygonal models, e.g. constructive solid geometry (*CSG*), where objects are obtained by primitives shapes (spheres, cubes, cones, ...) combined with union, difference or intersection operators. *CSG* is quite difficult to apply to deformable models, moreover an accurate boundary or surface representation, useful for rendering or interference detection can be hard to compute from *CSG* representations [58].

Another method to describe objects uses implicit surfaces. Implicit surfaces exploit a map from the 3D space to real numbers $f : \mathbb{R}^3 \mapsto \mathbb{R}$. The surface is the locus of points where $f(x, y, z) = 0$. Implicit surfaces are generally closed manifolds and have been used to model deformable bodies [13]. When looking for the interference between two implicit representations the surfaces are sampled and the collisions are checked using the approximated models obtained from sampled points.

Rigid bodies can also be represented by using parametric surfaces: a parametric surface is a mapping between a plane to the 3D space $f : \mathbb{R}^2 \mapsto \mathbb{R}^3$. Parametric surfaces do not always represent closed manifolds, thus they do not describe a complete solid model, but rather its surface boundary. Thanks to its simple domain, the function f is usually easy to polygonalize and to render graphically. A special case of parametric surface commonly used in CAD are Non-Uniform Rational B-spline (*NURBS*) [122].

The most common representation used in deformable body simulation is polygonal models that describes a model through a collection of triangles that are not required to be geometrically connected, nor to have a topological structure. When the polygons form a closed manifold the solid has well defined interior and external parts. Some collision detection algorithms exploit this structure, many apply only to convex, closed manifold, models.

Another difference between collision detection algorithms is the information that they can provide. Some applications, such as assembly-disassembly simulations, requires to know only if objects interpenetrate or touch. In our scenario, where the contact between bodies is not restricted to point contact but it can involve multiple areas of the objects, we still need to know which points are in contact but we also require the identification of overlapping object parts. In other scenarios, such as robot motion planning, the knowledge of the separation distance between the bodies is required. The separation distance is also useful to estimate the collision time and it is used in physical simulations to adapt the simulation step to the environment [65].

Collision detection algorithms differentiate also for how they handle time in the simulation. Static collision detection is computed by retrieving the position of

the bodies in the scene and by considering them fixed. The computation is simpler, but those methods can miss many collisions: i.e. completely compenetrated bodies or small bodies that in one simulation step move from one side of an object to the other. Dynamic collision detection keeps into account the motion of the bodies during the simulation step. They usually approximate the motion of bodies with linear trajectories, associate a swept volume to each body or element (triangles, lines, ...) in the scene, and check for intersection between swept volumes. An intermediate approach increases the accuracy of static methods by sampling the trajectory of bodies during the simulation step and checking, statically, the collisions between bodies in the scene.

Collision detection in deformable environment requires particular attention, in fact the majority of the methods developed for rigid bodies collision detection cannot be applied to soft bodies. Collision detection algorithms for rigid structures usually rely on precomputed data to speed up the on line test phase. These structures include axis aligned bounding boxes (*AABB*), bounding spheres, spatial partitioning (such as octrees, binary space partitioning trees (*BSP tree*). The update of those structures when addressing soft bodies deformations is computationally too heavy to provide benefits. Thus, ad hoc methods have been developed to identify collisions between deformable models.

4.1.2 Collision Detection for Deformable Models

When applying collision detection algorithms to deformable models some of the assumptions exploited in collision detection of rigid bodies do not hold. In addition to point displacement, deformation also changes the size of the model elements. Thus the update of the collision detection structures requires a lot of computation and is, in general, not convenient.

Few methods have been developed to speed up collision identification for deformable models. They can be roughly subdivided in methods that handle implicit surfaces and methods that work on polygonal meshes. Implicit surfaces simulate deformations by updating control points of model surface. The control point update requires the re-computation of the surface and, usually, its re-tesselation to guarantee the quality of the result.

In [107] the authors present a method to detect geometric collisions between time dependent parametric surfaces. The described algorithm works on surfaces that are continuous and have bounded derivative. Surfaces are checked pairwise and the values corresponding to coincident points are obtained numerically. The big theoretical limitation of the method is that it is restricted to functions with computable Lipschitz number[1], from the practical point of view the main limitation is its high computational cost.

A method that handles parametric surfaces is detailed in [51]. The algorithm uses *AABB* and convex hulls of the objects to speed up the computation of pairs in close vicinity. Pseudo-normal patches and Gauss maps are used to detect self

[1] Lipschitz continuity is a smoothness condition for functions which is stronger than regular continuity. A Lipschitz continuous function is limited in how fast it can change; a line joining any two points on the graph of this function will never have a slope steeper than a certain number called the Lipschitz number of the function.

collisions and sweep and prune method is used to compute other collisions. Temporal coherence is utilized to achieve incremental computations. The algorithm has been implemented and provide contact computation for scenes undergoing second-order polynomial deformations at graphics frame rate (30 Hz). The computation of the pseudo-normal patch and the update of the *AABB* structure is very time consuming, and the method can hardly be adapted to haptic simulations.

In general the use of parametric surfaces to describe deformations represents a disadvantage for collision detection computation. In fact deformations of the reference shape requires the update of sampling points that can be computationally very expensive. Some work focused on the use of *NURBS* due to their reduced computational cost. In particular, [62] describes a method that relies on *NURBS* to perform deformation simulation, rendering and collision detection. The method relies on precomputation of *AABB* bounding hierarchy for each deformable object of the scene. Some additional data structure is used to allow the method to detect self collisions. All the structures are updated at run time, but objects are tested for collision only in the deformed area. This saves some computation but the method is not suitable to handle haptic interactions. Moreover it noticeably slows down in the presence of colliding large object patches.

Methods to compute collisions between deformable models represented with polygonal meshes have been developed too. They are usually based on decomposition of the scene objects in hierarchies of bounding volumes with convex shapes.

In [12] sets of spheres are used to represent the geometry of deformable models and are used to perform fast collision detection. The method extends the Quinlan algorithm [90] and works really fast in both the updating of data structures and in the actual collision detection. The principal limitation of this approach is the low resolution that can be obtained, as relatively big spheres should be used to keep their number low and the accuracy of detected collision depends on the radius of the spheres. Moreover the algorithm only detects one contact and does not handle changes in the topology of models.

Fig. 4.1. Collision detection between forceps and severed vessel, collision detection is performed with the method described in [12].

This approach is extended in [16] by identifying all the contacts between the structures and overcomes the limitations on the triangle size and, thus, in sphere

radius. However the resulting algorithm does not handle complex models at suitable frame rates for haptic interaction.

A different approach is described in [103] where the overlap test between $AABB$'s is improved to speed up the interference detection. Optimized $AABB$'s provide better performance than OBB's for rigid objects. Moreover the author presents an innovative method to update data structures when the model is deformed or its topology changes.

Some other methods do not use hierarchical representation to speed up the collision detection. [35] presents a linked volumes based approach to the simulation of deformable models that allows straightforward collision detection. In the described approach objects are mapped into an occupancy map. Collisions are detected by simply checking if two objects try to occupy the same voxel. The complexity of the method is quite high, as the representation of the objects is not hierarchical, moreover, the simulation of deformations lacks a clear physical interpretation.

[67] and [108] describe methods that are based on graphical rendering techniques to detect collisions. The approaches use the rasterization process that maps the scene object coordinates into the camera coordinate system, clip the tetrahedral faces outside the viewing volume, and project the remaining visible polygons into rasterized pixels. In this way, objects that are not rasterized at the same pixel do not collide. The resulting algorithm is simple and fast, but it has some important limitations that prevent it to be applicable to general collision detection. They cannot handle collisions between deformable models and, during the rasterizing process, they lose important tri dimensional information useful to compute the penetration depth such as the distance between colliding surfaces.

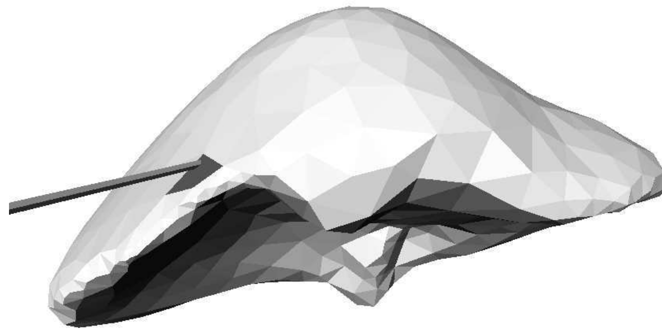


Fig. 4.2. Collision detection between a triangular mesh modeling a human liver and a static position of a tool (from [67]).

Some recent work focused on the use of GPU to improve collision detection performance. [44] and [43] describe a method that is based on chromatic decomposition and exploits GPU computational power to speed up collision detection. The main goal of the approach is to reduce the number of tests needed to detect the first time of contact between two bodies. The chromatic decomposition, computed at the beginning of the simulation, partitions the set of triangles that compose one model surface in k disjoint independent sets such that no primitive in the same set

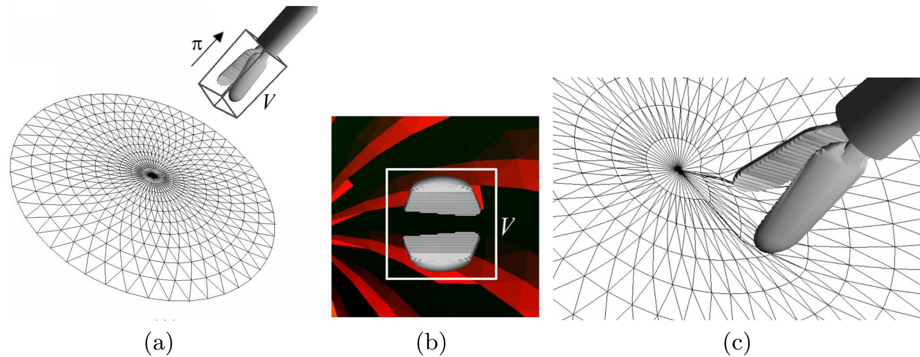


Fig. 4.3. Overview of the method described in [108]: (a) the object volume V is rendered with a parallel projection in direction π. (b) Each triangle of the membrane has a unique color. Triangles with hidden parts are colliding. (c) Application of deformation vectors.

is adjacent and that each primitive of a set has at most one adjacent primitive in each other set. The properties of this decomposition is exploited to discard tests during the actual collision detection phase. The main limitation of this method is its requirement that the model topology does not change during the simulation (otherwise the chromatic decomposition must be updated).

Another approach is described in [111]. The method reduces the number of tests by introducing four phases in the computation: a pre processing step marks the triangles of the model to filter colliding primitive pairs (triangles, edges and points are considered as primitives). Then a bit masking process assigns primitives to each triangle. At run time triangle markings are used to extract potentially interacting primitive pairs from each pair of potentially colliding triangles. The last step check for the actual interference between primitives, limiting the computation to one test for each pair of primitives. This method introduces less overhead in the pre processing phase with respect to the previous one, but it cannot handle topological changes too.

In [121] the collision detection is improved by the use of streams of $AABB$s that bound deformable models in the scene. The stream of bounding boxes is updated at run time when objects in the scene deform. GPU is exploited to parallelize the tests on the $AABB$s. Results are encoded to reduce the amount of memory that needs to be exchanged between GPU and CPU. The algorithm imposes no limitations on the shape of the model nor it requires that the topology remains constant during the simulation. The method shows to improve the performance with respect to standard collision detectors, but it cannot guarantee performance suitable for interactive haptic simulations, as it takes more than $8msec$ even for relatively simple models.

4.1.3 Collision Detection Library

We identified the requirements that our scenario impose on collision detection algorithm: our goal is to solve collisions between deformable models, represented as triangular meshes, with the possibility to change model topology and at a frame

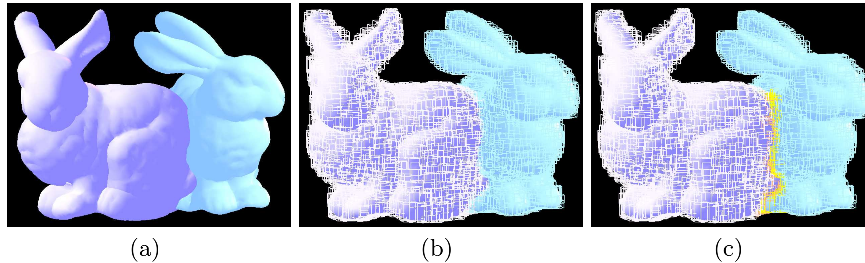


Fig. 4.4. Collision Detection using Streaming AABBs [121]. (a) shows intersecting two bunny models; (b) two bounding AABB streams are superimposed on the bunny models that they bound respectively; (c) highlights intersecting AABBs (shown as orange and yellow boxes).

rate that is suitable for haptic interaction. In addition, the detection of collisions should not stop after the first contact has been detected, but should provide all the pairs of triangles that collide between each pair of bodies. Thanks to the reduced time step used in the physical simulation we can avoid continuous collision detection and focus on static detection. Since we address deformable models, we cannot guarantee that objects are convex, and we discarded methods that rely on the decomposition of objects in convex parts as the complexity of the decomposition for our specific deformations is too high.

We investigated the features of some collision detection libraries, to identify the most suitable to our needs. In particular we analyzed four libraries: *RAPID* [42], *V-Collide* [50], *DeformCD* [101] and *SOLID* [104].

RAPID stands for Robust and Accurate Polygon Interference Detection, and is targeted to detect interferences between pairs of unstructured polygonal meshes by using *OBB*'s. It works with polygonal soups (i.e. models represented by triangles without connectivity information) but it does not explicitly handle changes in topology. It is most suitable for close proximity configurations between highly tessellated smooth surfaces. When deformations in the body occur the *OBB* tree associated to the body needs to be completely recomputed, slowing down the simulation.

V-Collide is a collision detection library for large dynamic environments that couples an n-nody algorithm : an algorithm that handle multiple bodies in contact and is not limited to pairs of objects, to the fast processing algorithms developed in *RAPID*. It is designed to operate on large number of static or moving polygonal objects and to allow dynamic addition or deletion of objects between time steps. The method suffers from the same limitations of *RAPID* since it uses the same algorithms for collision detection.

DeformCD is a fast collision detection library designed to accelerate calculation for deforming objects. For deforming objects, whose elements shape changes, a AABB refitting solution is used for collision detection. It exploits *GPU* to improve the performance of the detection, thus leading to a bottleneck when object topology changes and data needs to be uploaded to *GPU*. Very little documentation is provided on the use of the library.

SOLID is a library for interference detection of multiple three-dimensional polygonal objects undergoing rigid motion. It works with polygonal soups, it does not provide methods to handle changes in topology nor deformations. Its performances and features are similar to the ones of *V-Collide*.

We choose *V-Collide*, for its generality and for its ability to handle multiple objects in the scene. This requires that, at each time step, we rebuild the data structures involved in the collision detection. This greatly slows down the simulation, but we are not aware of any other algorithm that can compute collisions between deformable object undergoing general deformations that runs at frame rate that is suitable for interactive, haptic simulations.

4.2 Collision Solution

Once the collisions between bodies have been identified some action should be taken to restore the consistent configuration of the bodies in the scene. This phase of collision handling is often referred to as "collision solution". In many applications it is sufficient to warn the user about the compenetrating bodies or to recover the previous state of the scene. This is the case of assembly/disassembly simulations or tolerance verification. Computer aided design or machining, instead, require a more complex collision solution, in fact object shape should change in response to user input. Thus, it is necessary to update the configuration of the bodies to simulate the action of the virtual tool.

When applied to physics based simulations of rigid bodies, collision solution requires more information to correctly update the scene: usually it needs a good approximation of the collision time, a contact normal and a contact point for each colliding pair of objects. The contact point is the point where the objects first touch and the contact normal is the normal to a plane that passes through the contact point and is oriented such that it separates the object near the contact point. Objects are displaced along the normal until they do not interpenetrate. In addition contact forces can be computed and introduced into the simulation.

Solving collisions for physically based simulation of deformable bodies is even more complex. When a posteriori collision detection is employed, in fact, the definition of the contact point is not clear. The exact position of the contact point depends on the dynamical properties of material composing the contacting bodies, on their topology and on the body boundary conditions.

Some techniques have been developed to approximate the contact point using the knowledge of the body stiffness. In [28], for example, a virtual spring is introduced for each couple of bodies in contact. This spring is used to model the energy stored during the collision. The deformation of the bodies is not modeled, instead, a contact plane is associated to the point of initial contact of each body. To compute the separation plane used to solve the collision the algorithm simply evaluate a weighted mean of the contact planes. The exact position of the plane depends on the weights that are computed as functions of the relative stiffness of the two objects. This approach leads to good results in obtaining analytical description of contact between deformable bodies, but its application to general shapes and complex interactions is difficult, and cannot take into account properties such as object topology.

To address a wider range of interactions and topologies, we developed another method, that is not based on the analytical representation of the interacting shapes. Instead it exploits the characteristics that are typical of interactive deformable models simulation: the reduced time step employed and the explicit computation of model deformations. The simulation of physics based deformable environments requires the use of very small time steps in the temporal integration. This represents a stringent constraint for the computation of deformations but it allows some simplification in the process of collision solution. In fact, when handling deformable models, the contact forces that act during a single time step can be neglected due to the small module of the force itself and to the small amount of time considered.

To handle the contact between deformable model we identify the time of contact between the bodies, we recover a configuration in which the two bodies do not overlap and we rely on the physical simulation of deformations to compute the forces exerted during the interaction. The main issue in this procedures is the identification of the collision time.

4.2.1 Problem Statement

Given a pair of intersecting triangles we need to know with good accuracy when, during the last simulated time step, they first touched. The problem is quite common in rigid bodies collision solution and can be solved by computing the intersection of the segments covered by one triangle vertexes with the other triangles, then swap the triangles, repeat, and keep the minimum time obtained for triangle segment intersections.

In [77] a method to solve a similar problem for rigid triangles is described. The method is suitable to identify the collision between a static triangle and a segment. The method changes the frame of reference to place the triangle on the yz plane and to align its edges along the coordinate axes. Moreover it orients the segment along the x axis. The intersection point is simply obtained by solving an associated linear system.

In our scenario the triangle is not static nor it is rigid. Instead its three vertices are free to move during the whole time step. Thus we need to extend the previous method to handle deformations in the triangle.

The proposed method starts by assigning to each point of each model a time $\hat{t}$ equals to one, that means that the point position coincides with its position at the end of the simulated time step. Then it scans each pair of colliding bodies, for each body it scrolls the pairs of colliding triangles.

Given a pair of triangles, r and s. We indicate with $\vec{a}_r$, $\vec{b}_r$ and $\vec{c}_r$ the positions of the vertices of triangle r at the beginning of the last simulated time step and with $\vec{a}'_r$, $\vec{b}'_r$ and $\vec{c}'_r$ their positions at the end of the time step. Similarly we indicate with $\vec{a}_s$, $\vec{b}_s$ and $\vec{c}_s$ the positions of the vertices of triangle s at the beginning of the last simulated time step and with $\vec{a}'_s$, $\vec{b}'_s$ and $\vec{c}'_s$ their positions at the end of the time step.

During a time step, each point covers a linear trajectory that can be expressed as a linear combination of the starting and ending position: i.e. the trajectory associated with vertex a of triangle r is:

$$\vec{a}_r(t) = \vec{a}_r + t(\vec{a}'_r - \vec{a}_r) \text{ for } t \in [0,1] \tag{4.1}$$

the same procedures can be applied to obtain the segment associated to the other vertices.

The method checks each triangle for collisions with the segments associated to the other triangle vertices. The result of each test is a value of t for which a point on the segment lies on the triangle surface. The minimum between the obtained values of t represents the first time of contact between the two triangles.

We can analyze one single test without loss of generality: other tests are performed in a similar way. To identify the time of collision between triangle r and vertex a of triangle s we express a generic point $\vec{p}$ on the plane containing triangle r during the time step as a linear combination of triangle vertices. The position of $\vec{p}$ is a function of four parameters:

$$\vec{p}(t, \alpha, \beta, \gamma) = \alpha\vec{a}_r(t) + \beta\vec{b}_r(t) + \gamma\vec{c}_r(t) \tag{4.2}$$

where $\alpha, \beta, \gamma \geq 0$. The time of contact can be found by equating this parametric description of the point with a point moving along the segment defined by $\vec{a}_s(t)$:

$$\alpha\vec{a}_r(t) + \beta\vec{b}_r(t) + \gamma\vec{c}_r(t) = \vec{a}_s(t) \tag{4.3}$$

By substituting Equation 4.2 (and the corresponding equations for the other vertices) in the above equation we obtain:

$$\alpha(\vec{a}_r + t(\vec{a}'_r - \vec{a}_r)) + \beta(\vec{b}_r + t(\vec{b}'_r - \vec{b}_r)) + \gamma(\vec{c}_r + t(\vec{c}'_r - \vec{c}_r)) - \vec{a}_s - t(\vec{a}'_s - \vec{a}_s) = 0 \tag{4.4}$$

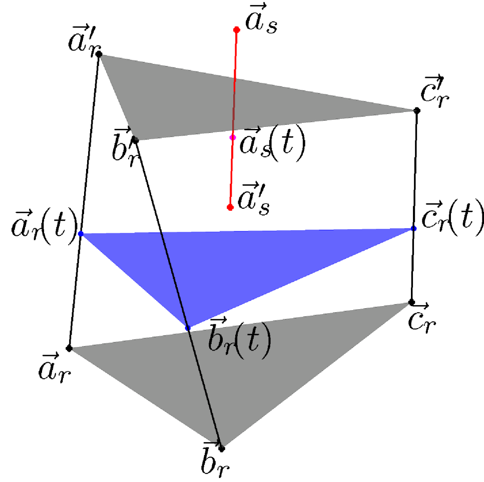


Fig. 4.5. Graphical representation of the values involved in Equation 4.4: in gray the initial and final triangle configuration, in blue the triangle on the plane defined by $\vec{p}(t, \alpha, \beta, \gamma)$

The non linear system that results from Equation 4.4 leads to the identification of the time instant in which the point on the segment lies on the plane containing

the triangle. After identifying the contact time of the six possible combinations (some contact may not happen) the minimum value $\hat{t}$ is used to update the time label associated to each vertex.

When all the colliding pairs have been checked and each colliding body tested the method updates each vertex position of the scene by moving it backwards along the trajectory covered during the last simulated time step. In particular the updated position for a point a of the model that moved from position $\vec{a}$ to a position $\vec{a}'$ is simply computed as:

$$\vec{a}_{new} = \vec{a} + t(\vec{a}' - \vec{a}). \tag{4.5}$$

This ensures the the two bodies do not overlap and that the two surfaces only touch on one or more points.

4.2.2 Method

As we noticed in the previous section, Equation 4.4 leads to a non linear system, whose analytical solution is quite complex. To rapidly compute the results of collision detection we adopted a numerical root finding method. The function we consider is the signed distance between the point $\vec{a}_s$ moving along the segment and the closest point on the plane containing the triangle r, as a function of time t.

To compute the signed distance we need the normal to the plane containing the triangle, since the triangle orientation changes during the time step the plane normal is a function of the time that can be obtained as the normalized cross product between two edges of the triangle:

$$\vec{n}_r(t) = \frac{(\vec{b}_r(t) - \vec{a}_r(t)) \times (\vec{c}_r(t) - \vec{a}_r(t))}{\left\|(\vec{b}_r(t) - \vec{a}_r(t)) \times (\vec{c}_r(t) - \vec{a}_r(t))\right\|}. \tag{4.6}$$

The signed distance is then computed as:

$$dist(t) = -\left\|\vec{a}_s(t) - \vec{a}_r(t)\right\| \frac{(\vec{a}_s(t) - \vec{a}_r(t)) \cdot \vec{n}_r(t)}{\|(\vec{a}_s(t) - \vec{a}_r(t)) \cdot \vec{n}_r(t)\|} \tag{4.7}$$

The first factor of the right side of Equation 4.7 actually computes the distance between the moving a_r point and the triangle plane, whereas the second factor computes controls the sign of the result. This ensures that when the point lies in the half space pointed by the normal (i.e. the point is outside the body) its distance is negative. On the other hand, when the point is inside the body its distance from the plane is positive.

We assume that, at the beginning of a time step ($t = 0$), all the objects are disjoint, thus each vertex of each body has a non positive distance with respect to all the triangles of other bodies. When after the temporal integration $t = 1$, two triangles intersect, at least one point of one triangle lies on the internal side of the other triangle. We identify these points and we compute the time in which they cross the triangle surface, i.e. the root of the distance function.

The previous consideration allows us to bracket the root: in fact function $dist(t)$ is continuous and its sign changes in the interval $[0, 1]$. A root finding method is thus suitable to approximate the time $\bar{t}$ in which $dist(\bar{t}) \approx 0$. Since the derivative of the distance function is hard to compute we cannot adopt methods that assume this knowledge, instead we focus on simpler methods. We analyzed the bisection method [57], the secant method [57] the regula falsi method [89] and the Brent method [89]. In the following we briefly present the four methods and discuss their principal features. All these methods start from the assumption that the value of the function changes in the interval $[a, b]$, that the function is continuous and that it presents only one root in the analyzed interval.

Bisection Method

Bisection method iteratively evaluates the function in the interval midpoint and examine its sign. Then it uses the midpoint to replace whichever limit has the same sign. After each iteration the bounds containing the root decrease by a factor of two. The method is guaranteed to converge, and its rate of convergence is linear.

Secant Method

Secant method approximates the function with a line inside the considered interval. At each iteration the computed solution is updated with the intersection of the approximating line and the t axis, using the following relation:

$$\bar{t}_{k+1} = \bar{t}_k - \frac{\bar{t}_k - \bar{t}_{k-1}}{dist(\bar{t}_k) - dist(\bar{t}_{k-1})} dist(\bar{t}_k) \tag{4.8}$$

If the analyzed function is twice continuously differentiable and the root is simple, then the rate of convergence of the method is super linear. One drawback of the method is that the estimated root can temporarily move outside the initial interval during the computation.

Regula Falsi Method

Regula falsi, or false position method, is similar in principle to the secant method, but it does not produce a succession of roots approximating the real root, instead it reduces the interval until its size is smaller than a desired threshold. At each iteration k it updates the interval $[a_k, b_k]$ by computing the value:

$$c_k = \frac{dist(b_k)a_k - dist(a_k)b_k}{dist(b_k) - dist(a_k)}. \tag{4.9}$$

The value c_k replace the interval limit with the same sign. The rate of convergence of the method is super linear in many cases, but an estimation of the exact order is not easy.

Brent Method

Brent method couples the advantages of root bracketing and bisection with an inverse quadratic interpolation. It exploits three prior points to fit an inverse quadratic function (t as a quadratic function of $dist(t)$) whose value at $dist(t) = 0$ is taken as the next estimate of the root $\bar{t}$. If the estimated root falls out of the current root brackets or if the bound does not collapse rapidly enough then the algorithm uses a bisection method to compute the new approximation.

Given three points, $\bar{t}_0$, $\bar{t}_1$ and $\bar{t}_2$, computed at step k the use of the inverse quadratic function leads to the following rule to update the approximation:

$$\bar{t}_{k+1} = \bar{t}_1 + \frac{S(T(R-T)(\bar{t}_2 - \bar{t}_1) - (1-R)(\bar{t}_1 - \bar{t}_0))}{(T-1)(R-1)(S-1)} \tag{4.10}$$

where

$$R \equiv \frac{dist(\bar{t}_1)}{dist(\bar{t}_2)} \qquad S \equiv \frac{dist(\bar{t}_1)}{dist(\bar{t}_0)} \qquad T \equiv \frac{dist(\bar{t}_0)}{dist(\bar{t}_2)}. \tag{4.11}$$

This method has, in many cases, super linear rate of convergence, in other cases it ensures at least linear convergence. The drawback is the increased complexity in computation that can penalize its performance when the number of required iterations is small.

Due to the difficulties in performing an accurate analysis of convergence for the presented methods we decided to implement the methods and to compare their mean performance to identify the most suitable for our scenario. Since we require that the root remains bracketed during the whole root finding process, to avoid errors in the solution, we discarded the secant method from our analysis.

4.3 Results

We integrate the collision detection library and the collision solution method with the deformable model simulation to test the behavior of the collision handling scheme in a realistic scenario. The resulting application simulates the interactions between two deformable models in contact. We analyzed the collision detection and solution step of the simulation that is composed by three basic sub steps:

- Collision detection structures generation: at each time step, after the models have been deformed by the physics computation, the collision detection structures have to be updated to the new model configurations.
- Collision detection test: once the structures have been updated they are used to detect the collision between the models. The result of the test consists in a list of pairs of interacting objects. For each interacting object pair a list of pairs of colliding triangles is generated.
- Collision solution: each pair of colliding triangles is analyzed with the method described in Section 4.2. Three different root finding methods have been evaluated to find the one that provides the best results in realistic cases.

To obtain meaningful results during our analysis all the tests we performed were based on two different models. The firs model, *coarse model*, is a *MSM* composed

by 554 points and 2729 tetrahedra, whose surface is defined by 906 triangles. The second one, *detailed model*, is a *MSM* composed by 2579 points, 15092 tetrahedra and with 2386 surface triangles. Both models are shown in Figure 4.6.

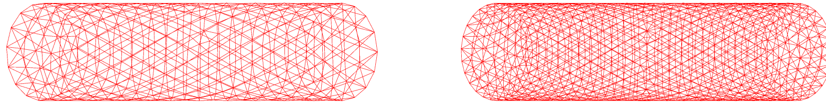


Fig. 4.6. The models used in the collision detection tests: *coarse model* on the left, composed by 554 points and 906 triangles and *fine model* on the right, composed by 2579 points and 2386 triangles.

4.3.1 Structure Update

The phase of structures update is handled directly by the collision detection library, it basically recomputes the *OBB* tree structures that wrap the analyzed objects. Two scenarios are analyzed during this test: in both scenarios the environment is composed by two cylinders. One cylinder is fixed at its basis. The other cylinder is free and falls under the effect of gravity. The second cylinder touches and compenetrates the first one during its fall. The scenarios differ only for the employed models: in one two *coarse model*s were used, whereas in the other we use two instances of *detailed model*. A graphical representation of the scene for the case of coarse models can be seen in Figure 4.9. Timings obtained in the two scenarios are showed in Figure 4.7. Computational time required for the update of the structure is analyzed during the evolution of the scene to identify potential dependencies of the algorithm on the model configuration (mainly the presence of long triangles).

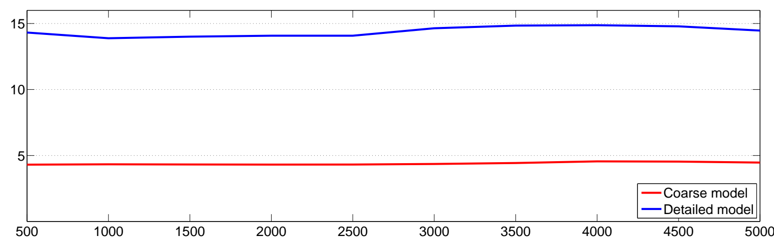


Fig. 4.7. Analysis of collision detection structures update. The evolution of the mean time required to update the structures used by the collision detector for the two models. Times are expressed in *msec*.

Results show that the time required to update the data structures is almost constant during the evolution of the scene. Instead it depends on the complexity of

the model: for the coarse model it takes about $4.4msec$ at each iteration, whereas for the *detailed model* it requires approximately $14.3msec$ to update the *OBB* trees.

4.3.2 Collision Detection

The scenarios used to evaluate the actual collision detection phase performance is the same used to test the structure update step. The required computational time was tracked during the first 5000 time steps to analyze the dependency of the procedure on the number of contacts. In fact, as the simulation proceeds the number of contacts increases until the two objects maximally compenetrates (after about 5000 time steps). Then the number of colliding triangles decrease and goes to zero when the red cylinder exits from the blue one. Plots of time required to compute the list of colliding triangles are shown in Figure 4.8.

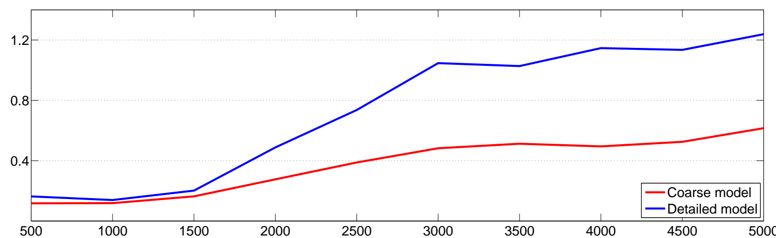


Fig. 4.8. Computational time required to perform actual collision detection for the two models during the simulation. Times are expressed in *msec*.

A graphical representation of the results of collision detection is showed in Figure 4.9. The figure shows the evolution of the scene as the red cylinder falls over the blue one. Interfering triangles of both models are highlighted in yellow: the collision detection algorithm correctly detects the colliding triangles. It can be noticed that the area of the models interested by the contact increases while the red cylinder penetrates the blue one. When the red cylinder exits the colliding areas decrease in size and the number of colliding triangles decreases too.

It is clear from Figure 4.8 that the complexity of the collision detection phase depends on the scene. When the number of colliding triangles increases the time required to detect all the interferences increases as well. This represents an important drawback for the integration of the collision detection phase in an interactive simulator, where each simulation step computation should complete in a fixed amount of time. Collision detection performance clearly depend on the complexity of the models that populate the scene, but thanks to the use of *OBB*s tree to speed up the computation an increment of a factor of 2.6 in the complexity of the models translates into an increment of a factor smaller than two in the time spent in the detection.

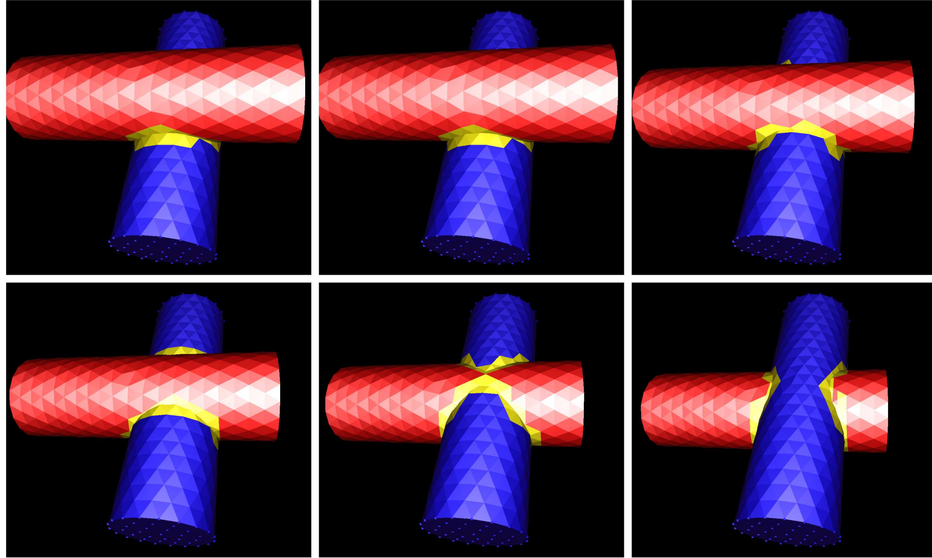


Fig. 4.9. Results of the collision detection phase. A sequence of two instances of the coarse model interpenetrating. Interfering triangles are highlighted in yellow. The blue cylinder is fixed at the two basis, the red cylinder is free, both are under the effect of gravity.

4.3.3 Collision Solution

During this test we evaluate the correctness of the approach described in Section 4.2, moreover we compare and analyze the performance of different numerical methods that can be used to approximate the time of collision between each pair of triangle. We integrated three root finding methods in the simulation: the bisection method, the regula falsi method and the Brent method. We use these three methods to solve collisions in the two scenarios used in previous tests. The analysis is carried out during the first 5000 steps of simulation to better identify method features. Timing of the three methods applied to the coarse and to the detailed models can be found in Figure 4.10 where plotted graphs represent the mean time for each invocation of the respective root finding method i.e. the total time required to identify one root (or the time of collision).

The analysis of the results shows that the three methods have very similar performance when applied to the coarse model. When applied to detailed model, regula falsi method shows worse performance with respect to other methods. This can be related to numerical instability of the method, as the detailed model has shorter edges that generates shorter vectors during the normal computations. The other two methods show a slight increment in the performance when applied to the pair of *detailed models* with respect to the case of the *coarse models*. The three methods correctly solve the collisions, results have been compared and the positions of model points after the collision solution step result equal. A graphical representation of the simulation after the phase of collision solution is shown in Figure 4.11.

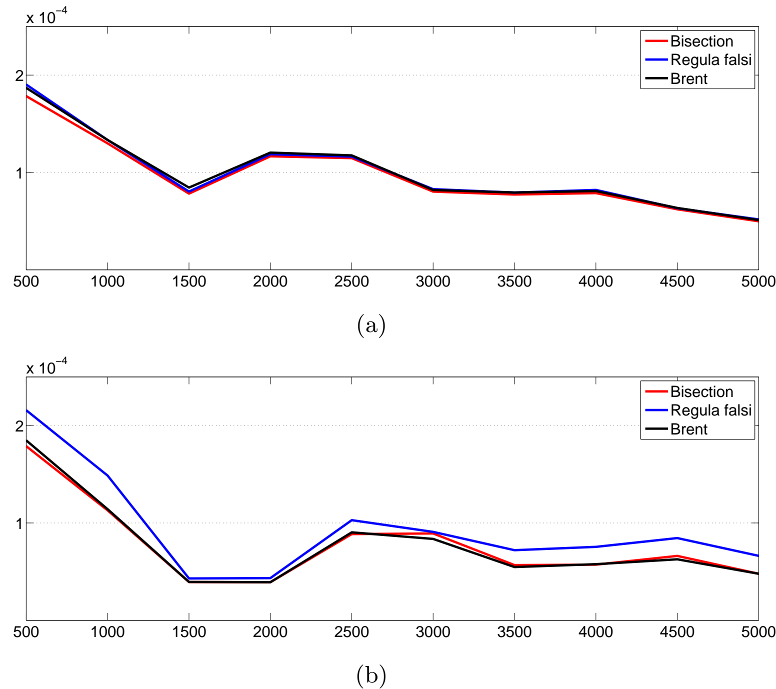


Fig. 4.10. Comparison of computational times (in *msec*) for the three root finding methods. On the x-axis the iteration number, on the y axis the mean time for one invocation of root finding method. (a) provides results for the coarse model (b) shows results for the fine model.

Time required to solve collisions between deformable models of average complexity suggest that some simplifications should be introduced to use them in simulations with haptic feedback. The overall time required to update data structures and to perform the actual collision detection is more than $1msec$. On the other hand, once that colliding pairs of triangles have been identified, the time required to compute the exact time of collision is quite small: all the three methods are suitable for interactive simulations as they can perform more than 10.000 tests in a millisecond (that is more than 1500 pairs of triangle). In addition, the simplicity and the fixed rate of convergence of the bisection method suggests that its implementation on parallel hardware may lead to important performance increments and to further increase its applicability in haptic simulations.

4.4 Conclusions

Collision detection and the solution of interferences between deformable models represent a bottleneck in the computation of interactive environments. Many approaches have been proposed in the literature to solve the problem ensuring suitable performance. We revised the state of the art in collision detection to identify

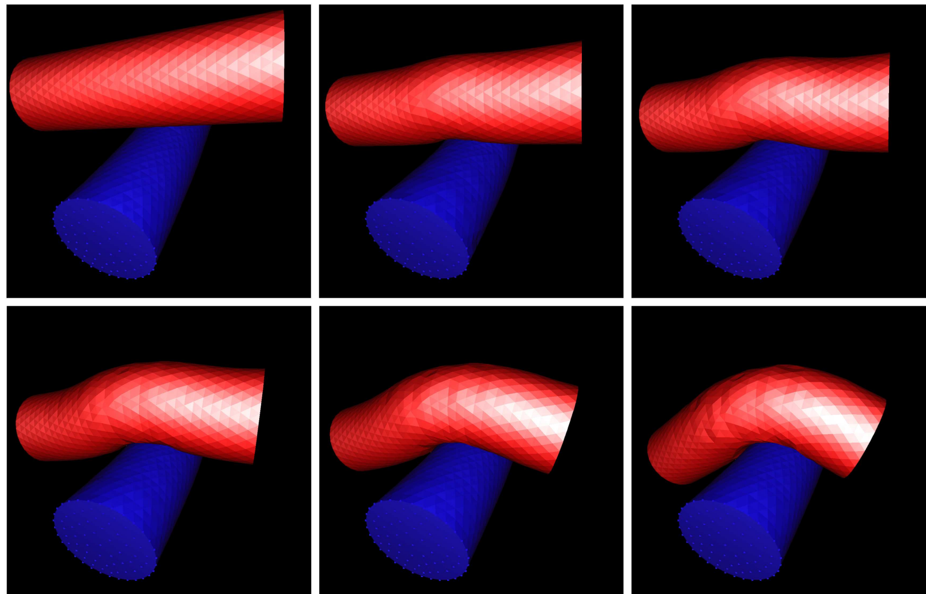


Fig. 4.11. Results of the collision solution phase. A sequence of two instances of *detailed model* interacting. The stiffness of the red model is lower than the stiffness of the blue one. The blue cylinder is fixed at the two basis, the red cylinder is free, both move under the effect of gravity.

the requirements of a collision detection algorithm suitable to handle the interactions between deformable models. We analyzed the principal collision detection libraries and compared their features.

We identify in V-Collide the library that best fits our needs and extended its capabilities with an innovative collision solution method. Collision detection is performed at the end of each time step and results are used to restore a physically correct configuration of the environment. Collision detection phase can return wrong results in some pathological situations: such as when a model, moving fast, crosses another body surface with its whole volume. In this case the collision detection will not identify the impact and thus the whole algorithm will not solve the compenetration. Other erroneous configurations may arise when two bodies in contact have very different sizes. In this case, in fact, numerical errors in the exact intersection between triangles and edges may lead to inaccurate results. This also happens when two colliding triangles are coplanar or two colliding segments collinear.

The complete method allows to approximate dynamic or continuous collision detection with reduced impact on the overall computation. The method is based on the computation of the time of contact between interfering triangles. During the simulation approximated values are used, due to the difficulties of solving the non linear system associated.

Different root finding methods have been analyzed and evaluated to identify the one that provides the best results in realistic conditions. The three methods provide

similar results; bisection method, despite its simplicity and the linear convergence rate, behave as well as the more complex Brent method. The use of the bisection method appears thus more suitable, also in anticipation of the implementation of a simulator on different architectures such as parallel processors or graphic cards that usually impose constraints on the number of available resources or the handling of branches and loops.

The simulation of virtual environments, comprising of physical simulation, collision detection with V-collide library and collision solution with the described approach runs at $55Hz$ when two *detailed models* are introduced in the scene and at $165Hz$ when two *coarse models* are simulated. The time required for the computation of the exact collision times is $0.94msec$ and $0.24msec$ respectively.

5

Friction Models

Due to the very evident effects it has in daily life, friction has been studied since Renaissance. Empirical studies identified three main parameters that rule the frictional contact between bodies: the normal force between the two contacting surfaces, the tangential force and the friction coefficient that is a characteristic of the two materials in contact. Two principal states can be identified by the observation of contacting bodies: sticking and sliding contact. In the sticking phase the external tangential force is completely balanced by the frictional force, whereas in the sliding phase friction only partially counterbalances the tangential force. The first observations of friction led to the development of three empirical laws that state that:

- friction force is directly proportional to the normal load;
- friction force does not depend on the apparent area of contact;
- kinetic friction is independent of the sliding velocity.

Even if modern studies proved that these three laws are not correct (in particular, the second one does not hold when applied to deformable models), they summarize the key aspects of friction as it can be perceived in common experiences. The development of correct friction models is nonetheless a key requirement for many research fields such as control, geology or automotive. It is essential in tribology but it is also an important aspect of the physical simulation of environments when simulated bodies interact.

In the following section we will provide a brief overview of some aspect of the friction that underline the need for realistic (and dynamic) friction models. Then we will review some of the principal models that allow to mimic the dry contact between solid bodies and their extension to lubricated contact. We will provide a comparison of the computational complexity of the different models and a qualitative analysis of their responses. Finally we will discuss how the different models can be integrated into the physical environment described in previous chapters.

5.1 Dynamic Components of Friction

During the study of friction, the identification of phenomena that cannot be explained with the cited laws lead to the development of more complex laws and models. These phenomena are principally related to the dynamic components of the friction effect. One of these phenomena is the dependency of the friction on the relative velocity of the contacting surfaces that is in sharp contrast with the third empirical law. It can be represented by plotting the frictional force against the relative velocity of the two surfaces, and the result is presented in Figure 5.1. The figure shows that the frictional force at constant normal load and constant velocity is a function of the velocity. The dip in the force at lower velocities is called Stribeck effect from the name of the researcher that first identified it [97]. The actual shape of the curve depends on the contacting surfaces, the temperature and the lubrication. Stribeck effect is only observed when two stiff materials (such as metals) are in rolling contact with lubricant. For this reason it is not a crucial component in the simulation of soft bodies in contact but we include it in our analysis for the sake of completeness.

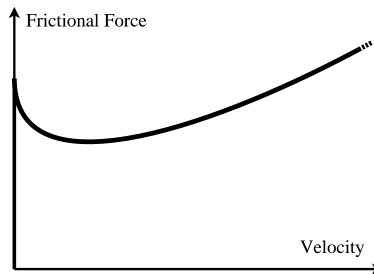


Fig. 5.1. The Stribeck effect: at constant normal load and velocity the modulus of the friction force is a function of the velocity.

Another effect that characterizes the contact between surfaces is the dependency of the break away force on the the rate of increment of the tangential force [55]. The break away force is the force that is needed to overcome the static friction and to initiate the motion. If the break away force is plotted against the rate of variation of the tangential force the obtained graph shows a curve similar to the one shown in Figure 5.2. The figure illustrates that for higher rates of variation of the external force, the force needed to switch from sticking to sliding phase decreases.

A further aspect of the sticking regime, that usually is not clearly noticeable, is the pre-sliding motion. It can be verified with two bodies that are in frictional contact with a tangential force that stays below a certain threshold: if the force is held constant, the displacement will likewise remain constant (except perhaps for creeping motion). When the force is decreased to zero, not all displacement will be recovered, i.e. there will be a residual displacement that is called pre-sliding motion.

The last important component of friction that we cover has been presented in [49], where authors describe experiments conducted by superimposing a periodic

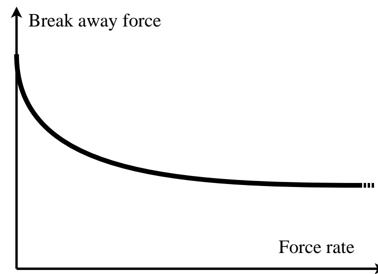


Fig. 5.2. The graph illustrates the dependency of the break away force on the ratio of increment of the force: smaller forces are required to switch from the sticking phase when higher force ratio are applied.

time-varying velocity on a bias velocity, so that unidirectional motion is obtained. In these conditions the relation between friction and velocity is characterized by hysteresis. The plot of the friction force against the velocity can be found in Figure 5.3. The size of the hysteresis loop increases with normal force, viscosity and frequency of the velocity variation.

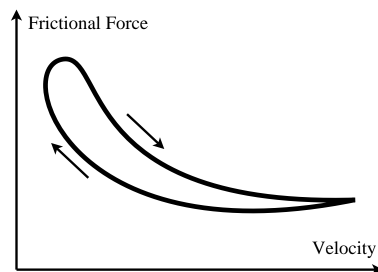


Fig. 5.3. Relation between friction force and velocity observed in [49]. The friction loop is lower for decreasing velocities than for increasing velocities. The hysteresis loop augments with the increment of the velocity variation frequency.

All these phenomena suggest that the laws previously presented cannot fully describe friction. Models should then be based on more complex laws and should take into account the dynamic components of the effect.

5.2 Friction Models

Friction models can be classified in two main categories: static models and dynamic models. In static friction models the friction force is given by a static function, except possibly for zero velocity. Dynamic friction models, on the contrary, take into account the evolution of the system in time and are usually defined through the application of the derivative operator to one or more variables of the system. They are thus more complex and computationally heavier. During our analysis we evaluated static and dynamic models, as, in some circumstances, the simplicity

and the computational speed of static models can balance the benefits of a more realistic but more complex dynamic model.

5.2.1 Static Models

Classical models identify a class of static models that can mimic many features of frictional contacts. They extend the law of Coulomb adding effects such as viscous friction, Stribeck effect and stiction. The basic model is called Coulomb friction model, it comes directly from the three laws cited at the beginning of this chapter and it models the friction as a force that opposes the motion with magnitude that is independent of velocity and contact area. The equation that rules this model is simply:

$$F = F_C sign(v), \tag{5.1}$$

where F_C stands for Coulomb force and is expressed as:

$$F_C = \mu F_\perp \tag{5.2}$$

where μ represents the friction coefficient between the contacting surfaces and $F_\perp$ is the force normal to the contact surfaces. This model does not define the force at zero velocity: its value depends on the definition of the sign function and can assume any value in the interval $[-F_C, F_C]$. A graphical representation of the model response can be found in Figure 5.4(a).

An extension to the Coulomb friction model can be obtained by taking into account the viscous friction, that depends on the relative velocity v of the contacting surfaces and can be expressed as:

$$F = (F_C + F_v|v|^{\delta_v})sign(v) \tag{5.3}$$

where δ_v allows a non linear dependence on the velocity and F_v is the viscous coefficient. The resulting relation between frictional force and velocity is shown in Figure 5.4(b) for the case $\delta_v = 1$.

To better approximate the real behavior of friction forces, stiction should also be taken into account. Stiction describes the friction force at rest as a force that completely balance the tangential force if it is under a threshold F_s and partially balances it otherwise:

$$F = \begin{cases} F_\parallel & \text{if } v = 0 \text{ and } |F_\parallel| < F_s \\ F_s sign(F_e) & \text{if } v = 0 \text{ and } |F_\parallel| \geq F_s \end{cases} \tag{5.4}$$

Using this approach friction force cannot be described just as a function of the velocity, instead when the velocity is zero, it becomes a function of the tangential force. In simulations, stiction can be integrated into the friction model by switching from static friction to dynamic friction when the tangential force exceed the threshold F_s. The whole model is graphically represented in Figure 5.4(c).

This model can be further extended to mimic the Stribeck effect that provides a continuous dependency of the force on the velocity, as plotted in Figure 5.4(d). The complete description of the Coulomb model with stiction, stribeck effect and viscous friction is summarized by:

$$F = \begin{cases} F_{\|} & \text{if } v = 0 \text{ and } |F_{\|}| < F_s \\ F(v) & \text{if } v \neq 0 \\ F_s sign(F_e) & \text{otherwise.} \end{cases} \tag{5.5}$$

Different functions can be used as $F(v)$ as detailed in [3], one of the most common

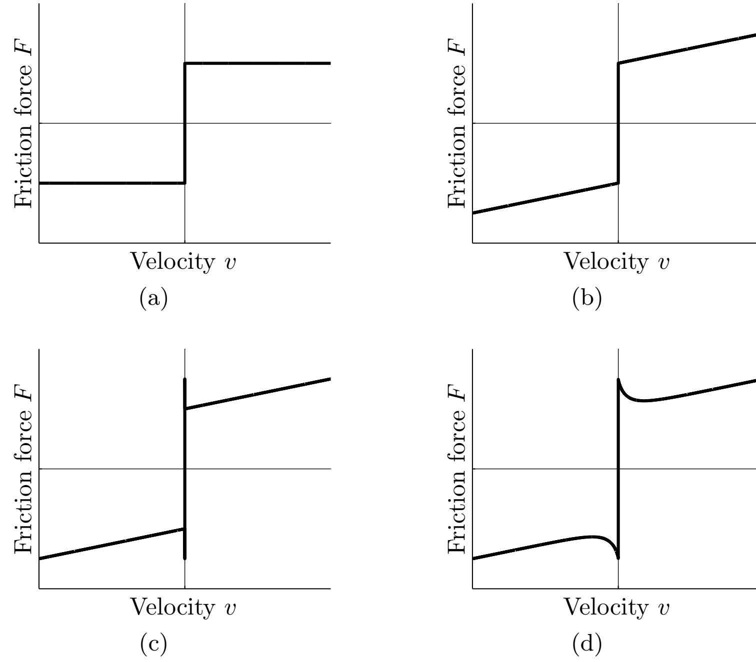


Fig. 5.4. Classical static friction models. (a) shows Coulomb friction, (b) plots Coulomb friction with viscous contribution, (c) add the contribution of stiction to Coulomb model and viscous model. (d) shows an example of Stribeck friction model.

relates the velocity to the force non linearly and can be expressed by:

$$F(v) = F_C + (F_s - F_C)e^{-|v/v_s|^{\delta_s}} + F_v v. \tag{5.6}$$

This choice computes the overall force as a weighted sum of the three discussed terms: the Coulomb force F_c, the stiction force F_s and the viscous force F_v.

Some problems arise when the models described are used in simulation. It is no easy, in fact, to correctly detect when the relative velocity v is zero and thus switch from one state to the others. A method developed to overcome this limitation is described in [56]. The proposed solution simply defines a dead zone of width $2\Delta v$ in which the model input velocity may vary and the model output is kept zero. The Karnopp model (by the name of the author) can be described as:

$$F = \begin{cases} F_{\|} & \text{if } v \leq \Delta v \\ F(v) & \text{otherwise,} \end{cases} \tag{5.7}$$

where the function $F(v)$ can be any arbitrary static function of the velocity. An example of the behavior of this model coupled with the function expressed in Equation 5.6 is plotted in Figure 5.5. The main limitation of this model is its requirement of the tangential force as input. Moreover the zero velocity interval, although of practical use, does not reflect the real behavior of friction.

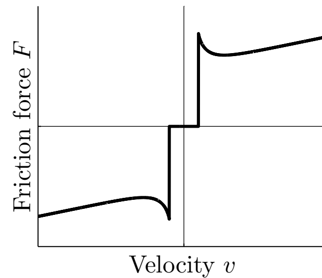


Fig. 5.5. Karnopp model: the graph of the friction force plotted against the velocity for a Karnopp model with $F(v)$ defined by Equation 5.6.

5.2.2 Dynamic Models

The use of static models can result in a lack of accuracy in some region of interest: such as pre sliding displacement in the stiction regime or hysteresis when changing the velocities of the contacting surfaces. In some simulations or models these aspect can be of great importance: slip stick phenomena, for example, are of great importance in the modeling of earthquakes [10] whereas hysteresis is required in the modelization of machine tools or servomechanisms [109]. To capture these behaviors it is necessary to extend the static models to keep track of the evolution of the system. This evolution is usually encoded in some state variables, thus dynamic models are also referred to as state variable models.

Dahl models is a dynamic friction model that is based on classic solid mechanics [23]. In solids subjected to stress, the friction force increases until rupture occurs, the stress strain curve can be expressed as a differential equation and used to model the friction. Dahl model is based on this approach and it defines the friction force as:

$$\frac{dF}{dx} = \sigma \left(1 - \frac{F}{F_C} sign(v)\right)^{\alpha} \tag{5.8}$$

where x is the relative displacement of the contacting surfaces, σ is the stiffness coefficient and α is a parameter that controls the shape of the stress strain curve. The Coulomb force F_C controls the slope of the derivative of the resulting force. In this model the friction force depends only on the displacement. Velocity is ignored in the computation of friction force, except for its sign. Thus this model cannot mimic Stribeck effect, as it is strictly related to velocity and to its variation rate.. The Dahl model can also be expressed in the time domain, and, for $\alpha = 1$ (a very common choice) it becomes:

$$\frac{dF}{dt} = \sigma v - \frac{F}{F_C}|v| \tag{5.9}$$

and then, by introducing $F = \sigma z$

$$\frac{dz}{dt} = v - \frac{\sigma|v|}{F_C}z \tag{5.10}$$

$$F = \sigma z \tag{5.11}$$

In the previous equation z describes the evolution of the system, and is used in implementations to keep track of the state of the system. The response of a Dahl model to a sinusoidal input velocity is show in Figure 5.6 in which hysteresis is clearly visible. Since the model does not take into account the velocity ratio, doubling the amplitude of the input (thus doubling its derivative) leads to bigger displacements, but the maximum exerted force remains constant.

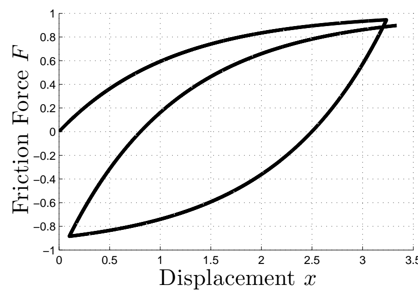


Fig. 5.6. The frictional force computed by a Dahl friction model to a sinusoidal input velocity. The force is plotted against the displacement of the body.

An extension of the Dahl model is the LuGre model [14]. It combines the Dahl model with arbitrary steady state characteristics such as the Stribeck effect. The internal state of the model can be interpreted as in the bristle model [22]: when two bodies are in contact each point of contact can be considered as a bound between flexible bristles. A relative movement of the contacting surfaces causes a change in the strain of the bristles which act as springs, giving rise to frictional force:

$$F = \sum_{i=1}^{N} \sigma_0(x_i - b_i) \tag{5.12}$$

where N is the total number of bristles, σ_0 is the stiffness of the bristles, x_i is the relative position of the bristles and b_i is the location where the bound was formed. When the deflection is large enough the bristles start to slip. The average bristle deflection for a steady motion is determined by its velocity. This model is able to mimic the Stribeck effect and rate dependent phenomena, such as the break away phenomena described in Figure 5.2 or hysteresis. The complete model can be expressed by:

$$\frac{dz}{dt} = v - \sigma_0 \frac{|v|}{g(v)} z \tag{5.13}$$

$$F = \sigma_0 z + \sigma_1(v) \frac{dz}{dt} + f(v), \tag{5.14}$$

where z encodes the average bristle deflection and $\sigma_1(v)$ is the damping of the bristles. Functions $g(v)$ and $f(v)$ model the Stribeck effect and the viscous friction respectively. A common choice for $g(v)$ is:

$$g(v) = \alpha_0 + \alpha_1 e^{-(v/v_0)^2} \tag{5.15}$$

in which α_0 represents the Coulomb force (F_C in previous equations), $\alpha_0 + \alpha_1$ corresponds to stiction force (previously F_s) and the parameter v_0 controls how the Stribeck effect varies within the interval $[\alpha_0, \alpha_0 + \alpha_1]$. The plot of the frictional force generated by a LuGre model in response to a sinusoidal velocity is plotted in Figure 5.7. In this case, by doubling the input velocity the maximum exerted force greatly increases.

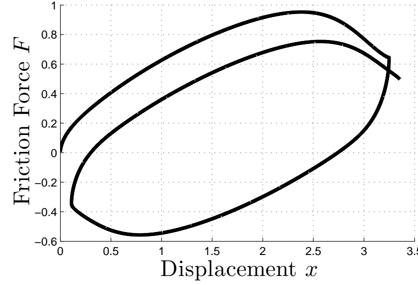


Fig. 5.7. An example of the response of a LuGre model to a sinusoidal input velocity.

This model is suitable to simulate lubricated contact. By using a decreasing function $\sigma_1(v)$, for example, it models the change in damping due to more lubricant being forced into the contact interface by the increased relative velocity.

By modifying the equation that rules the evolution of the state in Equation 5.14 it is possible to derive the Elasto Plastic model [29]. This model extend the LuGre model and introduces the possibility to simulate pre sliding displacement and stiction. The state variable z is defined as follow:

$$\frac{dz}{dt} = v \left(1 - \alpha(z,v) \frac{\sigma_0}{F_s(v)} sign(v) z \right)^i, \; i \in \mathbb{Z} \tag{5.16}$$

where the function $\alpha(z,v)$ is used to introduce stiction in the behavior of the model. An example of this function is:

$$\alpha(z,v) = \begin{cases} 0 & \text{if } |z| < z_{ba} \\ \frac{1}{2} \sin\left(\pi \frac{z - \left(\frac{z_{max} + z_{ba}}{2}\right)}{z_{max} - z_{ba}} \right) + \frac{1}{2} & \text{if } z_{ba} \leq |z| < z_{max} \\ 1 & \text{otherwise} \end{cases} \tag{5.17}$$

The response of an elasto plastic model computed for a sinusoidal input velocity is qualitatively very similar to the response of a LuGre model and can be found in Figure 5.8.

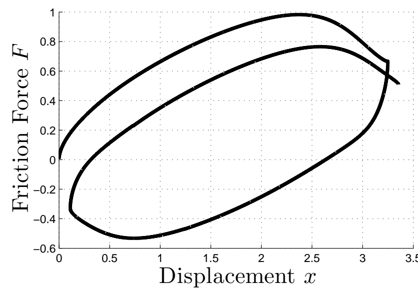


Fig. 5.8. The response of an elasto plastic friction model to a sinusoidal input. The behavior of the model is similar to the LuGre model behavior, but the output force is shifted of a positive amount (about 0.15 N).

The great number of friction models and their modifications suggest that the choice of the right model depends on the application for both the realism requirements and the computational complexity. In the next section we evaluate these requirements for the scenario of interactive simulations of deformable models.

5.3 Model Comparison

As we discussed in the previous section, there are big differences between the various friction models. As we are interested in interactive simulations the computational time represents a discriminant factor in the choice of the model. Thus we analyze a simple system composed by a mass that slides, with friction, over a plane. The weight of the system used for the simulations is $1kg$ subject to a gravity acceleration of $9.8m/s^2$. The parameters of the different models have been tuned to obtain similar physical characteristics: for example the same F_C has been used for Coulomb, Karnopp and Dahl models, whereas for LuGre and Elasto Plastic models their parameter α_0 has been initialized to F_C. A similar approach has been used in the definition of other constants.

We apply a force to the mass composed by a sinusoidal imposed over a continuous force, as shown in Figure 5.9. The small positive bias in the applied force allows us to highlight the absence of stiction effect in some models. We report and comment the evolution of the system with the different friction models and provide an evaluation of the time spent in the update of the friction force and of the model state at each simulation time step. We intentionally neglect to discuss and evaluate the parameters of the models and their effect on the output as the main goal of our analysis is the comparison of model features and complexity.

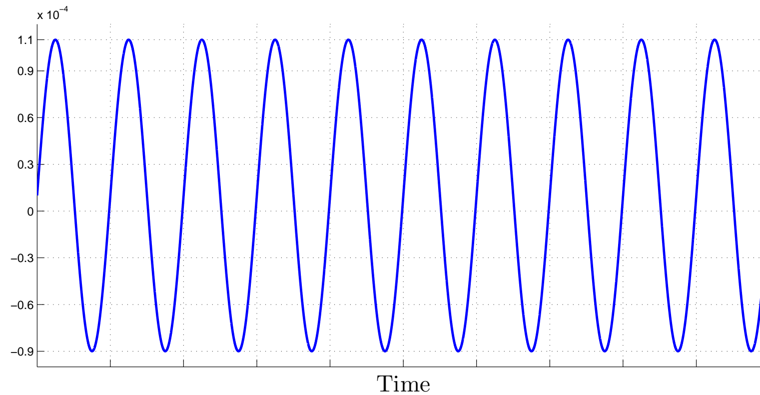


Fig. 5.9. The force used as input to the system during the evaluation of different friction models. The signal is composed by a constant value and a sinusoidal component.

5.3.1 Coulomb Model

The output obtained by coupling the physical system with a Coulomb friction model extended with stribeck effect is provided in Figure 5.10. The graph shows the position, the velocity and the frictional force during the evolution of the system. As can be noticed, the position of the mass increases in time as a sinusoidal signal superimposed to a constant slope. The underlying constant displacement demonstrates the absence of stiction in the evaluated model. The friction force shows a discontinuous behavior and its sign changes accordingly to the velocity.

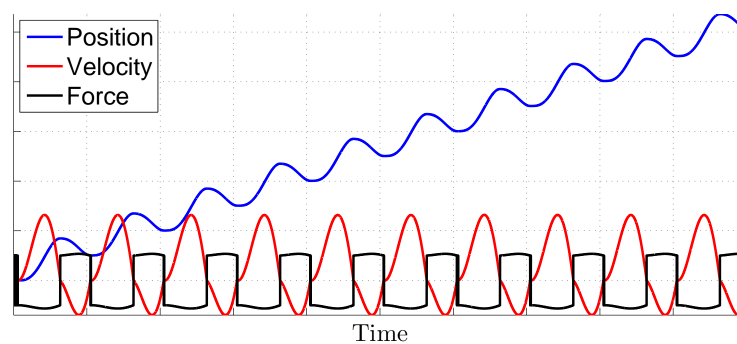


Fig. 5.10. The evolution of the physical system with friction modeled with a Coulomb model. Force graph has been scaled down to increase readability.

Some undesired jumps can be seen at the beginning of the simulation, when the velocity is zero, and right after the middle of the plot and are identified by the thicker lines in the graph. These jumps are due to the difficulty of the model

to correctly identify when the mass velocity is zero. The mean time required to compute the friction force at each time step is $1.58^{-2} msec$.

5.3.2 Karnopp Model

The Karnopp model gives results that are very similar to the Coulomb one. The graph of its output is shown in Figure 5.11. The evolution of velocity and position is indistinguishable from the ones obtained with Coulomb model, whereas the computed friction force does not show the spikes that are present the output of Coulomb model. Karnopp model, in fact, does not require to recognize zero velocity, thus provide a more stable behavior at low velocities. Its implementation is very similar to Coulomb model's one, but since it requires one branch less, the mean time required for the computation of one single update in the friction force is thus $5.73^{-4} msec$.

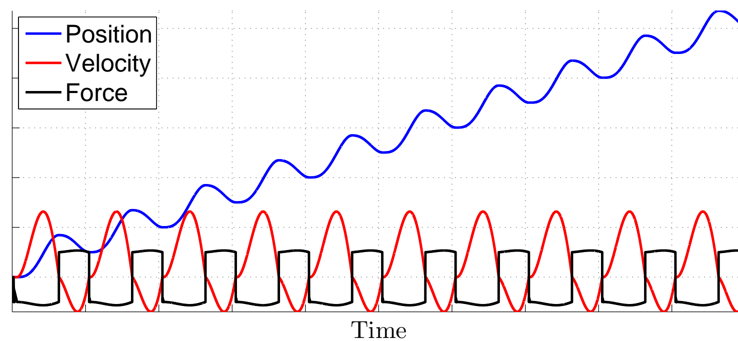


Fig. 5.11. The evolution of the physical system coupled with a Karnopp friction modeled. Force graph has been scaled down to increase readability.

5.3.3 Dahl Model

Dahl method is the first dynamic model we analyze, its response greatly differs from static method output as can be noticed from Figure 5.12. The graph clearly shows that the computed friction force reaches a stationary state after the initial phase, the force values has been amplified in the plot. Since the frictional force is smaller with respect to previous cases, the contribution to the velocity due to the continuous component of the input force increases. This leads to a drift of velocity toward the positive axis and to bigger displacement.

The mean computational time required to update the model state and to obtain the friction force estimation is $4^{-3} msec$.

5.3.4 LuGre Model

The LuGre model response is summarized in Figure 5.13. By isolating the periodic component in the force, it is possible to recognize a dip in the force that corresponds

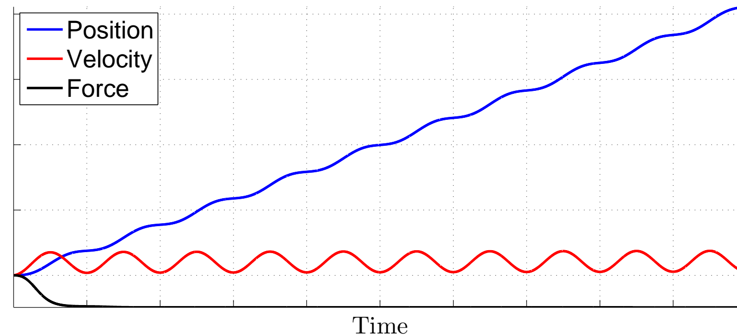


Fig. 5.12. The output of Dahl model applied to the analyzed system and the corresponding velocity and position profiles. Force values have been magnified in the graph to increase readability.

to the Stribeck effect. This model cannot reproduce stiction, in fact the body continues to increase its position during the whole simulation. The complexity of the model translates into a higher computational time, in fact, the mean time for the update of the model state and the computation of the friction force is $1.15^{-1} msec$.

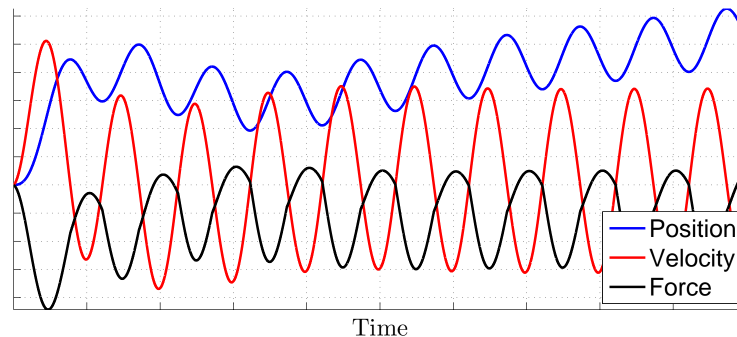


Fig. 5.13. Response of the LuGre model integrated into the simulated environment. Stribeck effect can be isolated in the force profile. Force has been scaled down to increase readability.

5.3.5 Elasto Plastic Model

The simulation of Elasto Plastic model provides interesting results. As expected, it is the only model (among the models evaluated in this work) that is capable of modeling stiction. The behavior of the model can be seen in Figure 5.14. Stiction is indicated by the absence of net motion when the system stabilizes. This is due

to the friction model completely balancing the small constant input force and this is due to its ability to model stiction. The model requires the computation of Equation 5.17 and thus is computationally heavier than the LuGre model. The computational time required to update the model and the force however is very similar and it is about $1.17^{-1} msec$ for a single step.

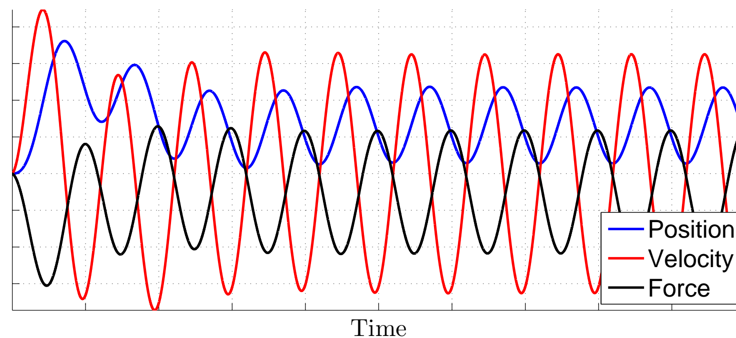


Fig. 5.14. Behavior of the Elasto Plastic model applied to the simulated environment. Stiction can be recognized by the absence of net motion after the initial phase. Force plot has been scaled down to increase readability.

The evaluation of the different models shows that their behavior is very different. Thus, to allow a wider range of simulations, we do not restrict our approach to one model, instead we developed a framework to seamlessly integrate different friction model into the physics of the environment.

5.4 Integration

The integration of friction models into the physical simulation requires the computation of three vectors for each contact point: the vector describing the relative velocity of the contact point, the vector containing the force normal to the contacting surfaces and the vector with the tangential force. These variables can be approximated from the results of the physical simulation. In fact, during the simulation we compute the direction of each surface point of the model, we estimate the force acting on that point and we obtain, through collision detection and solution, the exact point where the contacts happen.

To obtain the required values we consider a point that, during a simulation step, moved from a position $p(0)$ to a position $p(1)$. During this time step the point crosses the surface of a triangle tri at a time t (these information are provided by the collision detection and solution phase) as showed in Figure 5.15. Moreover, from the physical model, we obtain the force $f_i(0)$ that acted on each surface point i of the contacting bodies at the beginning of the time step.

With these data we can compute an approximation of the variables needed to feed the friction model. The basic idea is to detect the velocity and the force

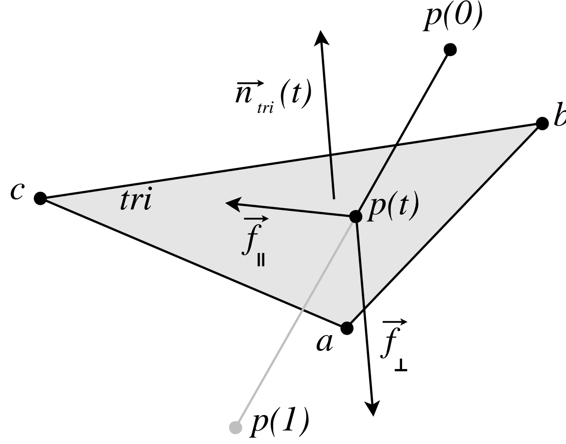


Fig. 5.15. A point colliding with a triangle: the picture shows some of the variables involved in the computation of the contact forces.

at the time of contact, and to decompose them into their tangential and normal components.

5.4.1 Velocity Approximation

During a single time step each point moves along a straight line with a constant velocity. Thus the velocity of the colliding point $\vec{v}_p$ does not change during the time step and at the instant of contact it is equal to the velocity $\vec{v}_p(0)$ it has at the beginning of that step. During the collision solution phase we freeze the colliding point and triangle at the instant of collision. Thus we exactly know the normal $\vec{n}_{tri}(t)$ to the plane containing the triangle at the time of collision. As the triangle also moves during the simulation we compute the force of the contact point on triangle surface $\vec{v}_t(t)$ as the linear combination of the velocities at the beginning of the time step of triangle's vertices:

$$\vec{v}_{tri} = \vec{v}_{tri}(0) = \alpha\vec{v}_a(0) + \beta\vec{v}_b(0) + \gamma\vec{v}_c(0), \tag{5.18}$$

where $\vec{v}_i(t)$ indicates the velocity at time t of the point i and α, β, γ are the convex combination of the triangle vertices that define the point on the triangle surface. These coordinates are defined as follows

$$\alpha a(t) + \beta b(t) + \gamma c(t) = p(t) \text{ with } \alpha + \beta + \gamma = 1 \text{ and } \alpha, \beta, \gamma \geq 0, \tag{5.19}$$

and allow to express each point inside the triangle as a weighted combination of triangle vertices. If the contact happens outside the triangle, we use the barycentric coordinates obtained by projecting the contact point on the closest triangle edge or vertex.

The relative velocity of the two points is thus simply the difference of the two velocities:

$$\vec{v} = \vec{v}_p - \vec{v}_{tri}. \tag{5.20}$$

We split this velocity in two components: a contribution along the triangle surface $\vec{v}_{\parallel}$ and a contribution normal to the triangle surface $\vec{v}_{\perp}$. Normal velocity is discarded whereas parallel component is used by the friction model if it needs it. The two components are simply computed by:

$$\vec{v}_{\perp} = \vec{v} \times \vec{n} \tag{5.21}$$

$$\vec{v}_{\parallel} = \vec{v} - \vec{v}_{\perp}. \tag{5.22}$$

5.4.2 Force Approximation

Forces are computed with a similar method. We consider that forces acting on each surface point remain constant during a single temporal step. This is in accordance with the numerical integration technique used to update model configuration. Another consideration supports this assumption: model surface in fact does not provide any inertial property during the contact, as no masses are associated to surface points. Thus, the assumption of a non null force at the end of the time step may provide a sort of inertia to the body surface.

In analogy with the velocity approximation, we compute the overall forces exerted on the contact point by the colliding point and by the triangle. The force on the triangle surface is parameterized with the barycentric coordinates of the point:

$$\vec{f}_{tri} = \vec{f}_{tri}(0) = \alpha \vec{f}_a(0) + \beta \vec{f}_b(0) + \gamma \vec{f}_c(0). \tag{5.23}$$

In the above equation $f_i(t)$ is the force acting on element i at time t and α, β and γ are the parameters identified during the velocity approximation.

The computation of the total force requires more attention, as a simple difference can produce wrong forces. We have to handle the cases where the two forces pull the point away from the triangle even if they have the same direction. For example if the two forces are both directed along the normal and the force acting on the point is bigger, in modulus, than the force acting on the triangle the point moves away from the triangle and the normal force should be zero. To obtain the correct estimation of the force we use:

$$\vec{f} = \begin{cases} 0 & \text{if } |\vec{f}_p| > |\vec{f}_{tri}| \text{ and } (\vec{f}_p - \vec{f}_{tri}) \cdot \vec{f}_p > 0 \text{ or} \\ & |\vec{f}_p| < |\vec{f}_{tri}| \text{ and } (\vec{f}_p - \vec{f}_{tri}) \cdot \vec{f}_{tri} < 0, \\ \vec{f}_p - \vec{f}_{tri} & \text{otherwise} \end{cases} \tag{5.24}$$

The case described in the previous paragraph is captured by the first part of the clause, the second part, instead, describes the situation when the resultant force is directed along $\vec{f}_{tri}$ and $\vec{f}_{tri}$ is the bigger force. The force is then split into two components, one perpendicular to the triangle surface and represented by $\vec{f}_{\perp}$ and the other directed long the triangle surface and indicated by $\vec{f}_{\parallel}$. The two components are obtained by:

$$\vec{f}_{\perp} = \vec{f} \times \vec{n} \tag{5.25}$$

$$\vec{f}_{\parallel} = \vec{f} - \vec{f}_{\perp}. \tag{5.26}$$

The computed relative velocity $\vec{v}_{\parallel}$ and the normal and tangential forces $\vec{f}_{\perp}$ and $\vec{f}_{\parallel}$ are fed to the friction model to obtain the frictional force exerted during the

contact. It should be noticed that all the variables are expressed from the point of view of the colliding point. If the friction model requires a description of its state, the value is stored in the colliding point data structure. This allows to keep a distinct model for each contact.

5.4.3 Force Distribution

Once that the frictional force $\vec{f}^f$ has been computed it can be applied to the surface point and used in the next simulation step to update the physics of the model. Handling the distribution of the force to the triangle is more complex, since forces can only be applied to the nodes of the surface mesh. To solve this problem without introducing ghost forces we distribute the force acting on the triangle using the barycentric coordinates introduced in Section 5.4.1. Moreover the friction force is oriented along the versor defined by the relative velocity of the colliding point and the triangle surface. The forces due to friction are thus:

$$\vec{f}^f_p = -\vec{f}^f \frac{\vec{v}}{|\vec{v}|} \tag{5.27}$$

$$\vec{f}^f_a = \alpha \vec{f}^f \frac{\vec{v}}{|\vec{v}|} \tag{5.28}$$

$$\vec{f}^f_b = \beta \vec{f}^f \frac{\vec{v}}{|\vec{v}|} \tag{5.29}$$

$$\vec{f}^f_c = \gamma \vec{f}^f \frac{\vec{v}}{|\vec{v}|} \tag{5.30}$$

To take into account the orientation of the friction force in the two cases, point and triangle, we change the sign of the force when applied to the colliding point, as the presented friction models return a force that is oriented as the velocity (or the tangential force).

5.4.4 Examples

We integrated the friction models described in Section 5.2 and the force/velocity computation method proposed in Section 5.4 with the framework described in Chapter 3. This integration lead to a virtual environment where deformable models interact with frictional contact. We use the obtained simulator to compute the evolution of the interaction between two deformable models. The considered models are the *coarse model* and the *detailed model* described in Section 4.3. When applied to a pair of coarse models the simulation enriched with Dahl model runs at $157Hz$ whereas the use of a pair of detailed models leads to an update frequency of $43Hz$.

In Figure 5.16 we compare the result of the simulation of two deformable models in contact with and without friction. The scene is composed by the two models introduced in Chapter 4 where the red model is free to fall under gravity force and the blue model deforms under gravity force but is fixed at his ends. The friction model used in the simulation is a Karnopp model. The different images are taken at corresponding simulation time.

The difference between the evolution of the two scenes is evident: in the non frictional contact the falling body slides along the surface of the fixed one and falls, whereas in presence of friction (with a proper value for the friction coefficient μ) it stops on the second model. Intermediate behaviors can be obtained by changing the parameters of the friction model. The use of different models does not lead to significant differences into this specific simulation.

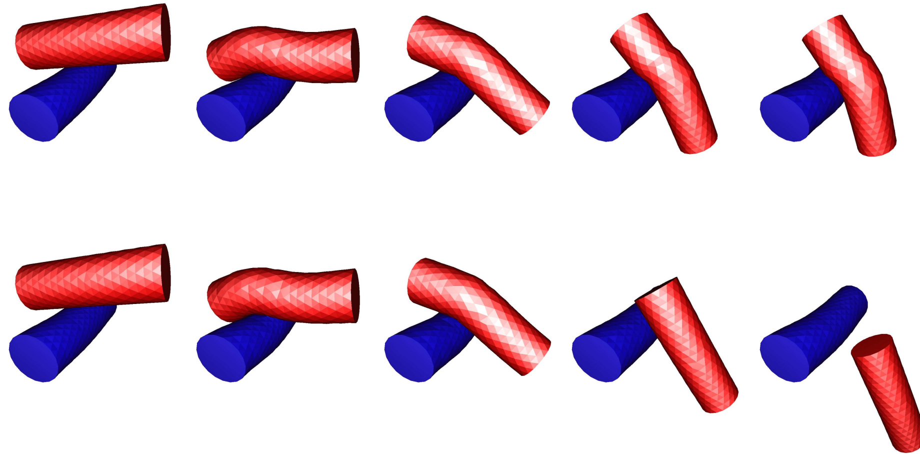


Fig. 5.16. The simulation of the same scene with and without the friction effect. In the upper row the friction computed by a Karnopp friction model slows down and stops the red model. In the lower row the absence of friction causes the red model to fall after sliding without friction on the blue model.

The simulations of a deformable sphere that falls over an inclined plane for different values of μ (and thus of F_C) are provided in Figure 5.17. Each column of the figure represents one simulation: in the leftmost column the contact between the plane and the ball has no friction, thus the ball simply slide along the plane without rolling. In the middle and in the rightmost column a Karnopp model is used to mimic the friction between the plane surface and the ball. The value of the parameter μ used in the right column is twice the value used in the middle column. The images cover ten seconds of simulation, and are taken at regular temporal intervals.

The presence of friction in the last two simulations causes the ball to start rolling on the plane surface. In addition, the increment of the μ value does not changes significantly the evolution of the simulation. This is in accordance with reality in the case of stiff bodies. In fact, after the body started rolling, the relative velocity and the tangential forces between the contacting points remain small and the increment of the friction coefficient μ leads to small differences in the simulation.

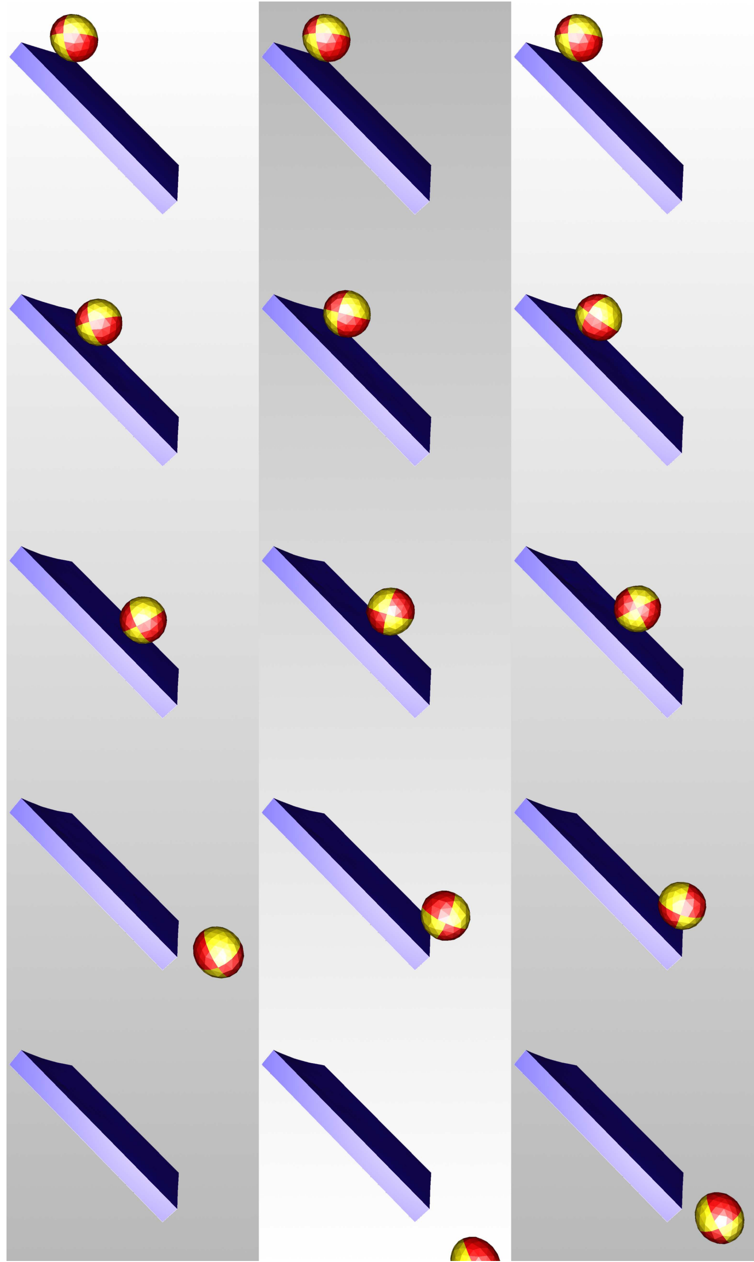


Fig. 5.17. Three simulations of a ball falling on an inclined plane. The left column presents some screenshots of the simulated environment without friction. The central column represents the same environment with friction modeled with Karnopp model. The right column presents the environment with a Karnopp model instantiated with a value of μ that is twice the one used in the central column. The images cover ten seconds of simulation, and are taken at regular temporal intervals.

5.5 Conclusions

To enrich the simulation of deformable environments with frictional contact we analyzed some of the most used friction models in physical simulation and in control. Two classes of model have been considered: static models (Coulomb and Karnopp models) and dynamic models (Dahl, LuGre and Elasto Plastic models). All of these models allow the simulation of dry contact, in addition LuGre and Elasto Plastic models also provide the ability to simulate lubricated contacts.

The analysis was conducted by isolating some dynamic aspects of friction, and by identifying the requirements and the features needed by our scenario: basically the requirement for fast computations and reduced interest in behaviors that cannot be perceived in the common life experience. The theoretical background of the different models have been presented along with their response to a sinusoidal input. The responses highlighted the difference between the models.

In addition, the models have been evaluated in a realistic scenario, i.e. integrated into a mass spring system undergoing a tangential force. The input force has been chosen to highlight the stiction component of model response. Results clearly show the difference between the analyzed models, in particular between the coulomb, Karnopp and Dahl models provides results that are qualitatively similar. LuGre and Elasto plastic models, on the other hand, show more complex behaviors but at the cost of higher computation time.

The choice of the friction model is strictly dependent on the phenomena of interest and on the condition of simulated environment. The reduced computational time of Dahl and Karnopp models make them suitable for environments where a lot of contacts are expected (very soft bodies or crowded environments) whereas LuGre and Elasto Plastic models are more suitable in presence of few contacts and when a higher realism is required. Due to the oscillations in the friction force computed by Coulomb model we do not integrate it in the simulation, as similar properties can be obtained by using the more stable Karnopp model.

The integration of the friction model into the deformable model simulation leads to realistic results, as shown by Figure 5.16. The use of different models lead to very similar results in the proposed case. This is mainly due to the simplicity of the simulated environment, nevertheless different values for the Coulomb force or for other parameters lead to great differences in the evolution of the scene. Figure 5.16 shows two antithetic cases: in the first sequence the falling model completely is completely stopped over the fixed one by the friction, whereas, in absence of friction it slides on the fixed model and falls.

Friction models require the tuning of one or more parameters. The identification of these parameters is not straightforward, but it should be carefully performed taking into account the different features of the colliding bodies such as: properties of the bulk materials of the bodies, properties of the surface materials of the bodies and the presence of lubricant or water.

The low frame rate obtained in the simulation of deformable models in contact with friction suggests that the overall computation requires great improvements. In particular, thanks to the nature of the computation, it can take advantage from the parallelization of the code. Thus, in Chapter 7 we will provide the details of a parallel implementation of a deformable environment with frictional contact that

exploits *GPU* computational power to obtain update frequencies that are suitable for interactive application with haptic feedback.

6

Anisotropic Mass Spring Models

In many scenarios, material behavior is more complex that the one described by an isotropic model. Many materials does not behave in the same way when deformed along different directions, and this requires the introduction of anisotropic deformable models. In Chapter 2 we limited our study to isotropic deformable models: in fact Equation 2.5 validity is restricted to this kind of tissues. This limitation can be too restrictive in many applications. In the simulation of anatomical structures, for example, many tissues are *transversally isotropic*, i.e. they have different behaviors along one direction and in the plane orthogonal to this direction. This is the case of muscles, tendons, ligaments or blood vessels and more generally fiber reinforced composites where all the fibers are parallel. For this reason it is important to introduce in the simulation deformable models that are able to mimic non isotropic materials.

A fibrous material usually offers more resistance when compressed or stretched along fiber directions. The modelization of such tissues requires a more complex approach. While an isotropic material has only two independent parameters that defines the elasticity matrix, for a transversally isotropic tissue the number of parameters is five. Supposing that the fiber are oriented along the z-axis and thus the anisotropy direction coincides with the z direction the five parameters can be defined as follows:

- E_p: Young's modulus in the x-y symmetry plane;
- ν_p: Poisson's ratio in the x-y symmetry plane;
- E_{pz}: Young's modulus in the z direction;
- ν_{pz}: Poisson's ratio in the z direction;
- G_{zp}: shear modulus in the z direction.

These parameters can be determined experimentally with uniaxial tension tests (E_o, ν_p, E_{pz}, $\nu_p z$) or with pure shear solicitations (G_{zp}).

In this chapter we will review some of the techniques developed to simulate anisotropic tissues in general and for transversally isotropic tissues in particular. Then we will propose an innovative method to introduce anisotropy in *MSM*s. We will compare the method with *FEM* formulation in the 2D case and provide results of simulations to prove that the method is particularly suited for interactive simulations because of its simplicity and limited computational requirements.

6.1 Related work

Because of the high importance of anisotropic behaviors many methods have been developed to introduce anisotropy in simulations. One of the most active area from this point of view is surgery simulation, because of the stringent requirements of realism.

Anisotropic material simulation using *FEM*s has been addressed in [110]. Two methods are introduced in this work: the first one expresses the material stiffness matrix in a frame of reference that is aligned with the material fiber direction. This approach is not suitable for interactive simulations because, even if the obtained realism is higher, since it requires to update the model every time that the direction of fibers change (i.e. the model rotates or bends). The second approach, followed also in [87] models anisotropy by using Saint Venant-Kirchhoff elasticity theory and adding to the isotropic elastic energy an anisotropic contribution that penalizes the material stretch along a desired direction. This method is based on the isotropic stiffness matrix that is defined by the two Lamé coefficients λ and μ and extends it by introducing two more parameters, called Lamé coefficients along the anisotropy direction: λ_{zp} and μ_{zp}. Results prove the feasibility of the methods: Figure 6.1 compares the behavior of an isotropic model (in pink, on the left) and of two transversally isotropic models obtained with the proposed method (in light blue and blue, the rightmost one has twice the stiffness of the central one along the anisotropy direction). The number of total independent parameters that rule the deformations is four, thus this method has some limitations in the realism of the simulation. In particular it neglects the shear modulus G_{zp} thus ignoring the dependency of deformations along different directions.

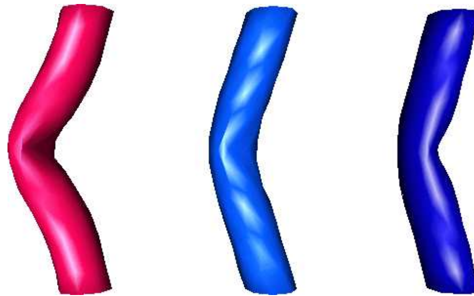
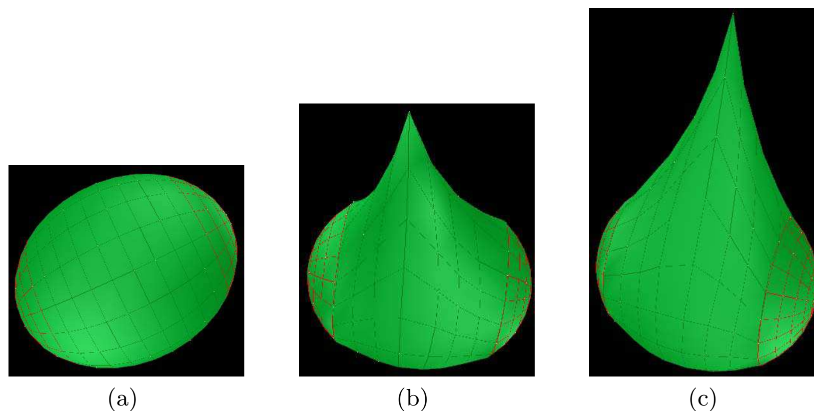


Fig. 6.1. Deformation of tubular structures with nonlinear transversally isotropic elasticity, from [87].

Due to the limitations of *FEM*s to model anisotropic behavior in interactive simulations ad-hoc methods have been developed. In [60] a chain linked model is extended to provide anisotropy in simulations at frequencies that are suitable for haptic feedback. The model is composed of springs and cylinders: springs represent material elastic properties whereas cylinders are used to activate springs.

The model deformations can be computed separately along each direction, this ensures the ability to introduce different behaviors along different directions. The deformed configuration is used to compute the object internal potential energy that is used to update the model and is reflected to the user. The method proved to be suitable for interactive simulation with force feedback, but the decoupling of deformations along different directions reduces the realism of the simulation. In particular this model completely decouples the different directions, thus a deformation along one axis does not affect the model configuration in the plane orthogonal to that axis. Another method is detailed in [123] where the authors propose an approach based on the analogy between heat conduction and elastic deformation. The potential energy stored in an elastic body is propagated among mass points following the principle of heat conduction. The method allows the simulation of large deformations and handles anisotropic behaviors by changing the thermal conductivity constants between a node and its neighbors. The simulation presented of anisotropic materials fails to provide different behaviors along different directions, instead they only show models with areas of different stiffness (see Figure 6.2). Thus it is not clear how the proposed method behaves in transversally isotropic tissue modeling. A tensor based approach is proposed in [88]. This approach extends the method introduced in [87] and discretize the deformable body volume with a tetrahedral mesh. Each tetrahedron is characterized by its four vertices and six edges, by four Lamé coefficients introduced in Section 2.1, a direction of anisotropy and four shape vectors. These shape vectors store the relative position of each vertex with respect to the other vertices of the tetrahedron. The obtained model allows to include the anisotropic contribution to the internal force computation. The computational complexity of the model is not discussed in the paper, but is is stated that the obtained physical simulation runs at $30Hz$, and forces fed to the user are extrapolated from the graphical simulation.



(a) (b) (c)

Fig. 6.2. Deformations of anisotropic materials. The part highlighted in red corresponds to anisotropic tissue.

The introduction of anisotropic behavior in mass spring based simulation is more difficult. The simulation of cloths is the more common area of application for this techniques. The simulation of cloths is indeed very challenging but the required anisotropic behavior is in two dimensions only and they are usually targeted at graphical simulations. In [4] a model of cloth using a triangular mesh is derived, it computes in-plane forces from a continuum formulation. This approach supports anisotropic behavior but it is not guaranteed to converge. In [9] the authors enhance *MSM*'s for tetrahedral and hexahedral meshes to include anisotropic material behavior. The volumetric elements they use (tetrahedra or hexahedra) include a frame of reference placed at their barycenter. The barycenter is connected to element faces by springs, in addition, angular springs are introduced at each vertex to the element, to oppose the deformation of connected edges. The frame of reference is used to assign the anisotropy direction to the single elements and used during internal force computation to obtain the desired behavior. This approach allows the simulation of anisotropic behaviors at the cost of increased computational complexity. Simulation results obtained with this method are shown in Figure 6.3.

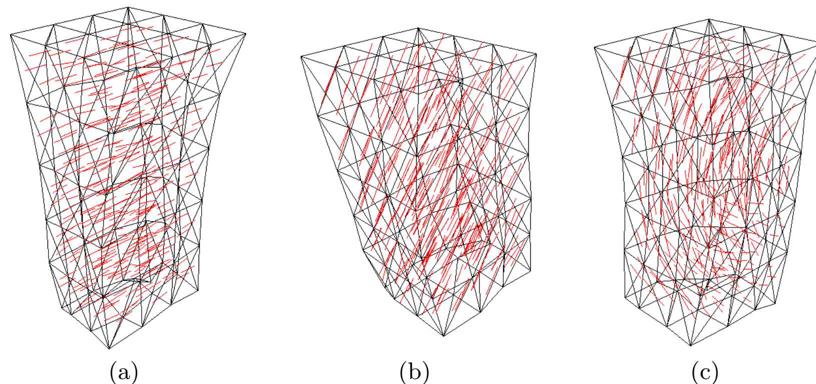


Fig. 6.3. Different anisotropic behaviors were obtained using the same tetrahedral mesh undergoing gravity force. Anisotropy is tuned by changing the stiffest direction in the material. This direction is: (a) horizontal (as a result, the material tends to get thinner and longer), (b) diagonal (with angle of $\pi/4$, which constrains the material to bend in this towards left), (c) hemicircular (as a C shape, which causes a snake-like undulation of the material.

A naive approach to anisotropy simulation is described in [17] where a regular lattice is generated. Springs are aligned with the anisotropy direction and their elastic coefficient depends on the orientation. This approach leads to realistic results when applied to regular structures but it is difficult to generalize it to irregular tetrahedral meshes or to varying anisotropy directions. Moreover, to reduce computational burden, the authors limit the propagation of forces and deformation to a subset of model nodes computed with a breadth-first search, this limits the realism of simu-

lation results. An innovative and interesting method is described in [116] where the authors generate a set of ellipsoids that cover the volumetric domain. The shape and orientation of the ellipsoids allow to keep track of the direction of anisotropy: spring rest length and forces are computed as a function of the ellipsoid shapes to maintain the anisotropy during deformations of the mesh. This method does not simulate deformations, but it can be used, in conjunction with a proper deformable model, to simulate anisotropic behaviors. The main drawback of the methods is the high computational cost introduced by the remeshing phase.

6.2 Method Description

The goal of the work described in this section is to obtain an effective way to simulate transversally isotropic tissues using *MSM*s. To introduce our approach we will first present it in two dimensions and then we will extend it to three and more dimensions.

6.2.1 Analytical Description

For a 2D model, or when a flat thin sheet of material is loaded with “in plane force” (i.e a force that lies in the model plane), the behavior of an anisotropic material can be described by the material elasticity matrix D, that has the form:

$$D = \frac{1}{1 - \mu_{xy}\mu_{yx}} \begin{bmatrix} E_x & E_x\mu_{xy} & 0 \\ E_y\mu_{yx} & E_y & 0 \\ 0 & 0 & G_{xy} \end{bmatrix} , \tag{6.1}$$

where E_x and E_y are Young's moduli and μ_{xy} and μ_{yx} are the Poisson's ratios in the x and y directions respectively and G_{xy} is the shear modulus. Young modulus is a measure of tissue stiffness and the difference between E_x and E_y control the amount of anisotropy in the model. Geometric considerations introduce the constraint $E_x\mu_{xy} = E_y\mu_{yx}$ and leads to a symmetric stiffness matrix.

The method we propose extends standard *MSM* springs with one or more parameters to obtain the anisotropic behavior. Each parameter is used to encode the initial orientation of the spring with respect to an anisotropy direction and is used to compute spring response. The method can handle any number of anisotropy axes and any model dimensionality. We will introduce our approach with an explanatory example in two dimensions and then we will present the analytical description of the behavior of extended springs.

We start by considering a 2D tissue whose principal axes are aligned with the coordinates axes (i.e. the difference in the behavior of the tissue is maximum for stimuli along the x and y directions) and that has a lower stiffness along the x direction. To obtain the needed anisotropic behavior we initialize the model as a standard, isotropic, mass spring model thus considering only one stiffness parameter k that represent the material stiffness along the stiffer direction (in our example the y axis). In addition we define a parameter j for each spring; this parameter is proportional to the cosine of the initial angle α between the spring and the softer direction (x axis in our case):

$$j = hl \cos \alpha$$

where γ is a global parameter that controls the magnitude of anisotropy (i.e. the difference of stiffness along the two directions) whereas l is the rest length of the spring. Using this method, springs that are aligned with the softer axis will have a j value equal to hl whereas springs that lie along the y direction will have a value of j equal to 0.

During the computation of spring length, needed to obtain the force exerted by the spring on connected masses, we add a contribution proportional to j to the actual spring length. Considering the 2D tissue previously defined we can consider, without loss of generality, a 2D spring that connects a mass located in the origin $O = (0, 0)$ of the frame of reference to a point $P = (x, y)$. The parameter j associated to the spring and the extended spring length are:

$$j = \gamma \sqrt{x^2 + y^2} \frac{x}{\sqrt{x^2 + y^2}} = \gamma x$$

$$l_{aug} = \sqrt{x^2 + y^2 + j^2} = \sqrt{l^2 + j^2}$$

where l is the spring length computed in the standard way and l_{aug} its augmented length. This new method of computing the spring length leads to a useful property of the extended springs. In the previous example, if the spring end in P is moved to P' by an amount d in a direction that forms an angle β with the original spring orientation, the force due to the length variation computed with the standard and the extended methods are, respectively:

$$F = k \left(\sqrt{l^2 + d^2 + 2dl \cos \beta} - l \right) \frac{P'O}{\|P'O\|}$$

$$F_{aug} = k \left(\sqrt{l^2 + d^2 + 2dl \cos \beta + j^2} - \sqrt{l^2 + j^2} \right) \frac{P'O}{\|P'O\|}$$

the modulus of the force exerted by the extended spring is always smaller than the force exerted by the regular spring, i.e:

$$\left| \sqrt{l^2 + d^2 + 2dl \cos \beta + j^2} - \sqrt{l^2 + j^2} \right| \leq \left| \sqrt{l^2 + d^2 + 2dl \cos \beta} - \sqrt{l^2} \right|$$

that is equivalent to say that the function F_{aug} has a global maximum in $j = 0$. The first derivative of F_{aug} can be written as

$$\frac{-1}{\sqrt{(l^2 + j^2)(l^2 + d^2 + 2dl \cos \beta + j^2)}} \cdot \frac{\left(\sqrt{l^2 + d^2 + 2dl \cos \beta + j^2} - \sqrt{l^2 + j^2} \right)^2}{\left| \sqrt{l^2 + d^2 + 2dl \cos \beta + j^2} - \sqrt{l^2 + j^2} \right|} \cdot j.$$

Since the modulus, square root and square are positives, the only term that influences the sign of the function is $-j$. Thus F_{aug} has a global maximum in $j = 0$. This mean that springs that are oriented along the y-axis, with associated $j = hl \cos \pi/2 = 0$, result stiffer than springs oriented along the x-axis for which $j = hl \cos 0 = hl$. This allows us to simulate the required anisotropy, moreover, the parameter γ defines the difference between the two extreme behaviors thus controls the magnitude of anisotropy.

6.2.2 Geometrical Interpretation

This method offers also a simple geometrical interpretation that can be exploited in the actual computation to avoid computing complex functions (such as $\cos(\cdot)$) thus leading to better performance. We can consider a standard spring with one end in the origin of the frame and the other on a point P in the xy plane. To obtain the anisotropic behavior we then lift the spring along the z-axis (thus introducing a new dimension to the model) of an angle that is proportional to the angle between the spring and x axis. The second end of the spring is thus raised of a certain amount that remains fixed for the overall simulation and that is a function of the orientation of the spring with respect to softer direction, in this example the x-axis. In this simple case the amoount is the difference between the x coordinates of the spring original ends. Applying the same procedure to all the spring of a model the whole model will be lifted from the xy plane and will lie in an inclined plane. Displacements will still lie in the starting xy plane but computed spring forces will be in the skewed plane and rotated back into the starting space.

The behavior of springs that are aligned with the y axis will remain unchanged, whereas the springs along the x axis will result softer. In fact the ratio between displacement and rest length changes when the spring rest length computed in the augmented space is higher than the rest length in the starting space. The behavior of springs at intermediate angles will be a combinations of the two described behaviors. Figure 6.4 represents a spring in the 2D space and the correspondent spring in the augmented space.

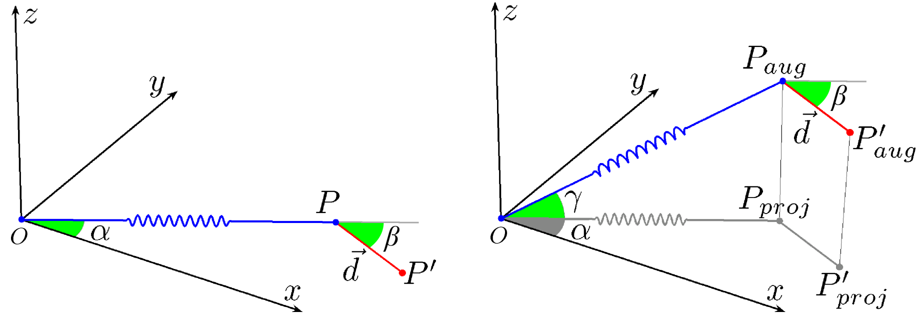


Fig. 6.4. Construction of the augmented model, starting from a standard 2D model we add a third dimension and obtain the augmented model (its projection is shaded).

To prove the correctness of this approach we can apply simple geometrical considerations to obtain the force exerted by a spring as a function of spring rest orientation, displacement intensity and orientation and slope of the plane in which the augmented model lies. Without loss of generality we can consider a spring of unit length that connects the origin O with a point $P = (\cos\alpha, \sin\alpha)$, where α is the angle between spring direction at rest and the x axis (refer to Figure 6.4 for a graphic representation). If we move the point P to a new position P' by displacing it of an amount d at an angle β with the spring orientation we measure a force F_{std} along the spring direction:

$$\vec{F_{std}} = \left(\sqrt{1 + d^2 + 2d\cos(\alpha - \beta)} - 1 \right) \vec{P'O} / \|\vec{P'O}\| \ . \tag{6.2}$$

Using the augmented model the point P is lifted to a new position $P_{aug} = (\cos\alpha \sin\gamma, \sin\beta \sin\gamma, z_{aug})$ where $\gamma = \tan^{-1} z_{aug}$ is the angle between the spring and the plane xy. If we apply to P_{aug} the same displacement than in the previous example and obtain the new position P'_{aug} the force F_{aug} we obtain can be expressed as:

$$\vec{F_{aug}} = \left(\sqrt{1 + d^2 + 2d\cos\gamma\cos(\alpha - \beta)} - 1 \right) \vec{P'O} / \|\vec{P'O}\| \ , \tag{6.3}$$

where the rightmost fraction is needed to align the force to the spring direction in the starting space.

It can be noted that in the second case the force exerted depends on the value of $\cos\gamma$ that is a function of z_{aug}. If $z_{aug} = 0$ the augmented spring behaves as a standard spring, if z_{aug} increases the behavior of the spring changes. Figure 6.5 plots the behaviors of the two springs, for $\alpha = \{0, \pi/4, \pi/2\}$, $d = 0.1$ and $z_{aug} = \cos\alpha \cos\pi/3$. From the figure it is clear that the augmented spring is softer

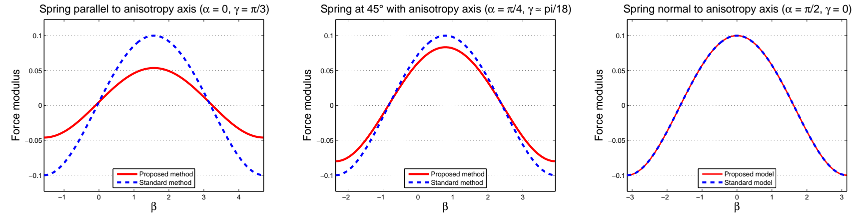


Fig. 6.5. Comparison of spring behaviors for three cases: $\alpha = \{0, \pi/4, \pi/2\}$ for $d = 0.1$ and $z_{aug} = \cos\alpha \cos\pi/3$.

when it is oriented along the x-axis ($\alpha = 0$) and becomes stiffer as it rotates toward the y-axis ($\alpha = \pi/4$). When the spring is aligned with the y-axis ($\alpha = \pi/2$) its behavior is identical to standard springs. Increasing the value of γ (i.e. if the plane containing the extended model moves toward the vertical) the difference between standard and extended behavior increases, augmenting the anisotropic behavior. To simplify the notation the spring length in these examples is one for both the standard and augmented spring, but in the actual application of the method we require that the projection of the spring along the augmented dimension has length equal to the length of the original spring. This ensure that the size of the projection of the augmented model match the size of the original model.

Extended springs can be used to simulate different stiffness along arbitrary directions. For a material in 2D that shows a softer behavior along an arbitrary direction $\vec{a}$ we compute, for each spring of the model, the angle α between the spring at rest and the direction $\vec{a}$ and lift one of its ends by a quantity $\tan\gamma\cos(\alpha) l_{std}$ where γ defines the anisotropy of the material and l_{std} is the rest length of the spring in the original model. This results in an extended model lying on a plane inclined of an angle γ along the direction $\vec{a}$. So when augmenting a model to add anisotropy only one value of γ should be defined. Our method ensures that the

projection of the undeformed augmented model along the augmented dimension coincides with the original model and that the projection of the deformed model represents the deformation of the original model with anisotropic behavior. Forces and displacements applied by the user are defined in a plane parallel to the original space. During the simulation we never explicitly compute the angle γ nor we use the $\cos(\cdot)$ function that has super linear computational complexity instead we handle augmented springs as regular springs defined in the augmented space.

The last constrain leads to non linear spring response. In fact in Eq. 6.3 the value of γ is not constant during the simulation: if a spring is lengthened the value of γ decreases whereas if it is compressed the angle γ augments. The extent of the introduced non linearity is discussed in Section 6.3. Looking at Eq. 6.3 it can be noticed that, for small values of γ the variation of $\sin\gamma$ is negligible while for bigger values of γ (i.e. the augmented spring is very steep and much longer than the original one) the ratio of spring length and its projection slightly changes if the spring is compressed or stretched. Thus we expect the variation of angle γ to be negligible.

The described method can easily be extended to handle 3D solids. By adding one augmented dimension to a 3D solid it is possible to obtain a transversely isotropic material, that is a material whose response in one plane S is different to its response in the direction $\vec{n}$ orthogonal to the plane. By using only one extra dimension and by defining the direction $\vec{a}$ as parallel to $\vec{n}$ allows to obtain a material that is softer in the direction $\vec{n}$. To model a body that is stiffer along $\vec{n}$ it is enough augmenting the model along two extra dimensions aligned with the two vectors orthogonal to $\vec{n}$ and lying on the plane S. In a similar way all kinds of anisotropic materials can be modeled, by associating up to two extra dimensions to the model each of them aligned with an axis of anisotropy.

6.3 Results

We analyzed two key aspects of the proposed method: the realism of the results and the additional computational time required to model the anisotropic behavior. To prove the effectiveness of the proposed method we compared its behavior with an anisotropic finite element models in MatLab, whereas to test the computational requirements we compared the C implementations of a standard *MSM* and the proposed method.

To obtain a reference behavior we have constructed an anisotropic dynamic *FEM* of a square with $100mm$ edge starting from a uniform mesh composed by 200 triangles, using the matrix obtained by (6.1) using $E_x = 100$, $E_y = 500$, $\mu_{xy} = 0.09$ and $G = 100$. Then we choose two model surfaces, corresponding to the square edges $x = 5$ and $y = 5$. We applied the same force to the face middle points. We have instantiated the two models to obtain the same behavior, i.e. tuning *MSM* stiffness and anisotropic parameter γ. The spring stiffness used is $300\,Pa$ and the slope is $\gamma = 1.1607\pi \approx 66°$. The comparisons of the simulation of anisotropic *FEM* and the proposed approach are depicted in Figure 6.6.

Figure 6.6 shows that the behavior of the proposed *MSM*s is clearly anisotropic and is qualitatively similar to the anisotropic *FEM*. We evaluated the relative

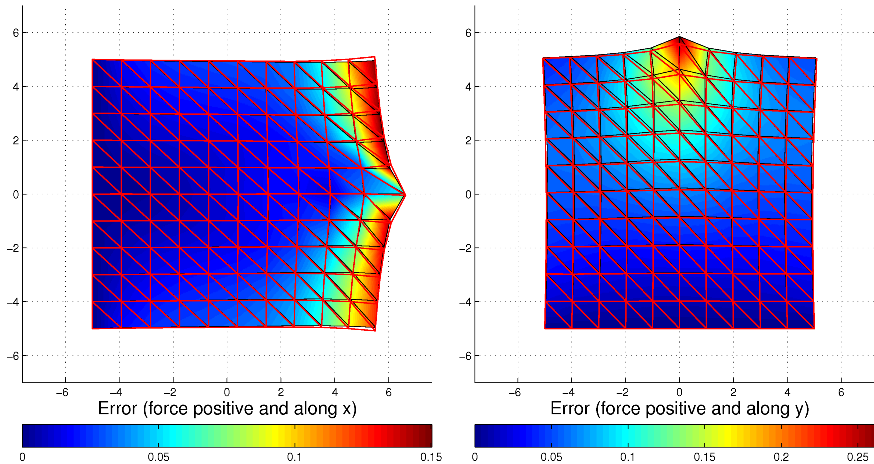


Fig. 6.6. Comparison between anisotropic *FEM* and proposed model: the shaded triangular mesh represents the *MSM* and the color indicates the error between the *MSM* and the *FEM*. red wire frames represent *FEM* models.

error in displacement between *FEM* and augmented *MSM* and it always stayed under 20% during tests. An error of 20% can be tolerated, as it is composed by to two contributions: the error due to mass spring model and the error due to the introduction of anisotropic behavior. In fact mass spring models can only approximate *FEM*'s [66], moreover triangular meshes are not really suitable to mimic *FEM*'s behavior, thus the error due to the introduction of anisotropy is far smaller than 20%. Another evidence of the different behavior of the model along the two directions is the angle between springs in the two test cases: springs along stiffer axis bend less with respect to springs in the softer direction.

To test the computational requirement of the proposed approach we instantiate three different *MSM*'s: a cylindric model composed of 988 springs, a finer cylindric model composed of 11321 springs and a spleen model with 19023 springs. We use them to simulate a standard model, with no anisotropy, and a model with anisotropy obtained with the proposed method. The key features of the models used during these test are summarized in Table 6.1 whereas mean computational time for a single frame of the simulation has been evaluated and is reported in Table 6.2.

Model	Points	Springs	Tetrahedra
Cylinder (coarse)	222	988	561
Cylinder (fine)	2579	11321	15092
Spleen	2700	19023	16383

Table 6.1. Key features of models used in tests

Model	Computational time standard model	Computational Time anisotropic model
Cylinder (coarse)	0.0463 ms (21.6 kHz)	0.0497 ms (20.1 kHz)
Cylinder (fine)	0.7042 ms (1420 Hz)	0.7564 ms (1322 Hz)
Spleen	0.8302 ms (1204 Hz)	0.9041 ms (1106 Hz)

Table 6.2. Computational times of standard and proposed *MSM*

The results obtained showed that the proposed method slightly slows down the computation. In the case of the simpler model the decrease of performance is about 6.8%. For the fine cylindric model the decrease in performance is 7.9% whereas when handling the more complex model the overall computation slows down of about 8.2%. The increment in computational complexity is justified by the possibility of modeling anisotropic behavior.

In Figure 6.7 we show different anisotropic models deformed by gravity acting along the z-axis. In the first column the direction of anisotropy is along the x-axis, in the second column the anisotropy is along th y-axis and in the third column it is parallel to the force direction (z-axis). In the first row the coefficient γ used is 4, in the second row it is 8 and in the last column used γ is 12, corresponding to a ratio between the stiffness along the stiffer and softer axis of 1.32, 1.44 and 1.48 respectively. It is clear from images that the choice of the anisotropy direction greatly affects the response of the deformable model. In particular, when the anisotropy is orthogonal to the acting force direction (first and second columns of Figure 6.7) the obtained deformations are similar, whereas when the anisotropy axis is aligned with the force the tissue results softer and deforms more.

To visually compare the results of our method with other methods proposed in literature we simulated a pinched tube, under different anisotropy conditions. Figure 6.3 shows the results of such simulations. In (a) the tissue is modeled with a standard isotropic model and is proposed as a reference for the other simulations. In (b) the tissue is modeled with the proposed method and presents a softer behavior along the vertical direction z (this required just one added dimension), finally, in (c) the model presents a higher stiffness along the x axis. To obtain this behavior we extended the model along two dimensions. As discussed, in fact, our method allows to make the tissue softer along desired directions. To obtain a 3D model with higher stiffness along one direction we need to soften the orthogonal directions, in this case, x and y. The forces are applied to slightly different heights. Model in (b) squeezes less since the reduced stiffness along the z direction allows it to deform along that direction and to stretch along z. In addition it also slightly bends because of the difference in the force application point. Model in (c) squeezes more because of its higher stiffness along the vertical axis.

As discussed in Section 6.2 one drawback of the proposed method is that the behavior of obtained springs is not linear because the angle γ between the spring and the xy-plane changes when the spring is deformed. However, the amount of non linearity is negligible. In fact for spring deformations that stay below 100% of spring rest lenght (computed in the original, non augmented, model) the difference

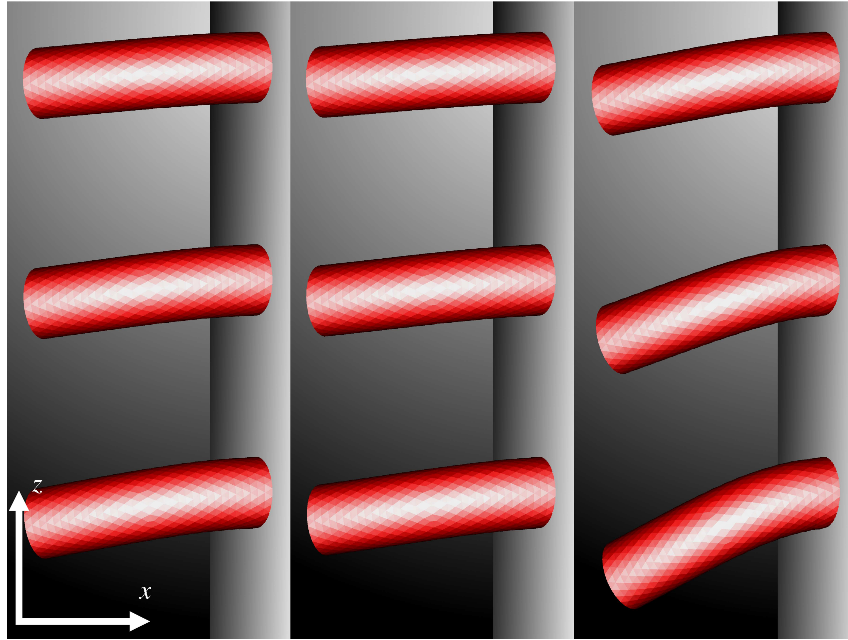


Fig. 6.7. Different anisotropic models undergoing gravity. In the first column model anisotropy (softer) axis is aligned with x-axis, in the second column it is aligned with y-axis and in the last column it is along the z-axis, and parallel to gravity direction. In first row a value of 4 for γ is used to instantiate the model, in the second row γ is 8 and in the third γ is 12, corresponding to a ratio between the stiffness along the stiffer and softer axis of 1.32, 1.44 and 1.48 respectively.

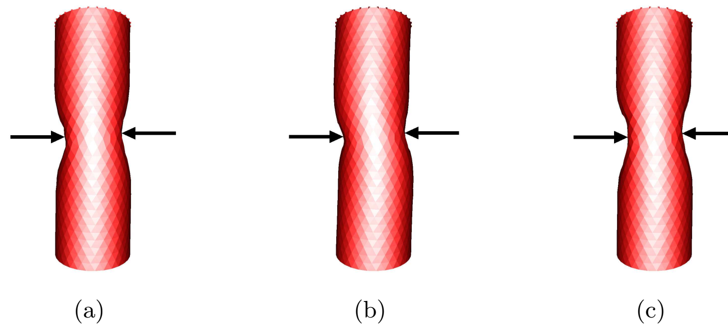


Fig. 6.8. Different models pinched by the same forces: in (a) an isotropic model, in (b) a transversally isotropic model, softer along the vertical direction and in (c) a transversally isotropic model, stiffer along the vertical direction.

between the linear behavior and the behavior of anisotropic spring is always smaller than 1% (measured as the relative error between the ideal, linear spring response and the anisotropic spring response).

6.4 Conclusions

In this chapter we described a novel method that allows to simulate anisotropic *MSM*s with reduced computational overhead. In the proposed approach *MSM* points are augmented with new dimensions that skew the model springs, thus obtaining different behaviors in different directions. Starting from a geometrical description of the method we have analyzed the response of the anisotropic springs, showing that they actually change their behavior along with their orientation. To test the realism of our method we compare the simulation results with anisotropic *FEM*.

Results prove that the proposed method provides an effective way to reproduce anisotropic behavior. Some differences have been identified in the behavior of *FEM* and augmented *MSM* that can be due to the lack of a correct calibration procedure or to coarser model resolution.

The behavior of the tested model clearly differs when the tissue is compressed or stretched along different directions moreover spring configuration during the deformation resembles anisotropic, fibrous tissue behavior. We have also tested the computational overhead introduced by the use of our anisotropic model and found that it is less than 10% of the computational time of a standard *MSM*. Therefore the model is suitable for interactive surgical simulations, where a proper trade off between realism and computational complexity is required. In fact the computational overhead introduced by our method is moderated, and it does not influence the nature, intrinsically parallel of *MSM*. This makes the proposed approach suitable for multicore and parallel implementations. Moreover the method is not only useful in modeling transversally isotropic materials, but it can be also used to model materials with different behaviors along the three axis.

One drawback of the model that has been identified during the tests is the introduction of a non linear behavior, however experimental results prove that the non linearity is negligible because its contribution stays under 1% of the total spring response.

One main limitation of the proposed method is the lack of a physical meaning for the γ parameters, that results in difficulties when tuning the model. This is, indeed, a common characteristic of all *MSM* parameters. For this reason proper calibration techniques have been developed to define spring and damper values that allow to mimic desired behavior [119]. These methods can easily be extended to handle the γ parameter introduced by our method.

7

Implementation

As described in Chapters 1 and 2 the implementation of an interactive, physically based environment requires particular attention to the computation time. In fact, the simulation of deformable models should satisfy Equation 2.3 that relates the temporal step used in the numerical integration to the properties of the simulated models, in particular its stiffness and its spatial resolution. The introduction of haptic feedback into the simulation imposes another tight constraint on the computational time. In fact, to ensure a smooth and realistic force feedback to the user, the rendered force should be updated every millisecond [60].

To avoid the computational burden due to the update of complex models at the frequency of 1 kHz, methods based on extrapolation have been developed. But, with the diffusion of parallel architectures they have been abandoned in favor of parallel implementations. The parallelization of the computation allows to update the physics of interactive environments at the proper frame rate, even in presence of deformable models. For most applications, a frame rate of 1 kHz provides a good trade off between computation time and realism of the simulation, in terms of model resolution and physical parameters of the modeled tissue. Thus it is widely accepted that a physical simulation with force feedback should run at a frequency above 1 kHz.

The current trend in *CPU* design integrates two or four cores on a single chip to improve the performance of the processor. The parallelism offered by these architectures, in fact, allows the execution of different applications at the same time. A similar approach led the development of the Cell BE processor, where eight (or more) synergistic processing elements (*SPE*) are coordinated by a single power processor element (*PPE*) to increase the level or parallelism. Graphic processing units (*GPU*), on the other hand, are an example of massively multi core processor, in fact they can embed more than 100 cores.

Thanks to the low cost of modern architectures the use of parallel processors is gaining popularity in many fields, from gaming to scientific computation: some libraries integrates in Matlab the support for the execution of basic operations such as FFT, matrix multiplication or element-by-element multiplication directly on the *GPU* [118].

We chose to focus on the use of *GPU* because of its wide diffusion on commercial personal computers that ensures both a lower cost and a good future

development for the hardware architecture. The objective of our work is the implementation of a physical based simulator that allows the user to freely interact with soft tissues feeling the force feedback. In the following we will discuss the aspects related to the physical engine underlying the virtual environment simulation. The engine exploits *MSM*'s to handle deformations and performs temporal integration, collision detection and solution, and computes the forces that should be rendered to the user. Then we will present a method that has been developed to reduce the impact of rendering on the performance of the simulator. This method is based on the encoding and transmission of a minimal description of the scene, that allows a remote machine to perform the actual graphics rendering and visualization.

7.1 *GPU* Implementation

The graphical processing unit (*GPU*) is the vector processor of current video cards. It consists of several embedded computational units which, given their inherent parallel structure, can be programmed using tools and languages originally designed for graphics rendering such as *OpenGL*. The computational model is based on a kernel function f_s, called *fragment shader* that is a, usually simple, code that is applied independently to each (u, v) element in a subregion of a bidimensional array (or texture in the following). The result of the computation is a vector obtained such as:

$$C_i(u, v) = f_s(u, v, p, C_{i-1}(u, v)) \tag{7.1}$$

where u and v are the coordinates of the processed element, p is a set of parameters defined at the beginning of the computation process (such as scalar constants or references to external arrays, called *textures*), $C_i(u, v)$ is the resulting vector and $C_{i-1}(u, v)$ is the previously stored result.

A non-trivial *CPU* algorithm cannot be directly ported to *GPU* due to some hardware limitations (such as limited number of simultaneously accessible resources, *CPU*-*GPU* synchronization issues, lack of function recursion support, conformance issues to IEEE-754 floating point numbers). The most complex phase in porting is the required adaptation of the original code in terms of arrays, interpolators and kernel functions.

Many works focused on the implementation of physical based deformable environments on *GPU*. One of the most challenging and investigated topics is the simulation of surgical scenarios. This is due to the level of realism required and to the benefits that such a software can bring to the medical field. In [24] the authors present an approach that works on *GPU* and does not rely on physics, instead it approximates the deformation of a soft body by a local displacement field thus obtaining frame rates suitable for haptic feedback. The main drawback of this and other not physically based techniques, is the difficulty in the calibration of the model. Moreover surface based methods usually do not allow to perform cuts in the simulated body.

In [39] the authors propose to allocate masses on a 2D texture and springs on a group of 2D textures where each element stores a spring connected to a mass stored on the same position. Since spring valence is not constant over the model,

it is necessary to store invalid elements that leave the result arrays unaltered: this reduces the computational efficiency and increases memory requirements. In order to limit the number of invalid elements, a sorting algorithm is applied before storing masses.

A different approach is presented in [95] where the physical simulation is not applied to the set of masses, but to a regular cubic grid of points interconnected to the 26 nearest ones. Since some of these points are external to the body's volume, the efficiency of the method is somewhat reduced. The advantage of this approach is that it does not require to store an additional spring array since connections are implicit in the uniform grid structure. When rendering of the body is required, the surface mesh has to be updated: for each vertex the position has to be interpolated from the positions of the 8 nearest points at rest instant and the normal has to be recomputed or approximated. Force rendering is based on the download on the system memory of all simulated data: this approach introduces synchronization issues and reduces global performance.

Currently, a lot of different implementation of surgical simulations are using *FEM*, such as in [113]. Recent researches have shown the possibility of taking advantage of the *GPU* for the necessary matrix multiplications. However, computational time does not allow to reach the desired haptic frequency of $1kHz$.

The use of *GPU* demonstrates to be effective in handling topological changes in interactive simulations of deformable models. In [33] and [34] a method is proposed that allows removing tetrahedra from a manifold mesh preserving the manifoldness property. The described strategy consists of a phase of mesh refinement in the area interested by the cut, followed by a phase of removal of the elements located near the surface cut. The approach has the advantage of generating high-quality elements but with the drawback of creating cuts that are not very smooth. Moreover, when applied to interactive simulations, the presented results show a big variance: the average time for removing one tetrahedron is 0.8 ms, but it can rise to 5 ms in some cases. This is not acceptable in interactive simulations with force feedback, as it will compromise the realism and stability of the simulation. Another method to handle topological changes in tetrahedrized deformable models is proposed in [93]. The approach keeps two different descriptions of the model, the first is a coarse representation of the physical model, based on tetrahedral mesh, the second is a triangular mesh that represents the surface and is embedded in the previous one. The method provides good graphical results, but it is not clear how it can handle possible haptic feedback and the time requirements of the method.

An important aspect in *GPU* based interactive simulation with haptic feedback is the computation of forces acting on surgical tools. Identifying contacts and computing forces introduces some important issues on *GPU* simulators, since downloading data from the graphics device requires *CPU*-*GPU* synchronization. The literature about this topic is not wide and the proposed solutions (such as [95]) imply a great performance loss. For this reason we developed an approximated collision detection algorithm (described in Section 7.3.1) that exploits the features of *GPU* to speed up the computation and provide reasonable results.

As we already discussed, one of the main issues of physical based simulations with haptic feedback is the requirement of 1 msec update frequency. However it is not enough to ensure that the mean computational time required for each temporal

step simulation is less than $1msec$. The delay between two contiguous simulation step should also be kept as constant and close to 1 msec as possible. An irregular delay in the rendering of the forces results, in fact, in a loss of realism of the simulation. In terms of graphical rendering a variation in the delay of the frame translates in not smooth scenes and in irrealistic variations of object perceived velocities, in haptic rendering it results in non smooth interaction with virtual object and in instabilities in the simulation. Thus, to further improve the performance of the proposed method we use a distributed approach for the graphical rendering.

If the same *GPU* is in charge of physical computation and graphic rendering it is difficult to ensure the necessary constant timing. When evaluating the performance of simulators, computation mean times are usually considered and exhaustive analysis of the computation timing is often neglected. We have investigated both measures in our simulator and we found that the graphic rendering phase can take ten times the time required for the physics update. Even splitting the graphic rendering task in simpler sub tasks the computational time dedicated to rendering introduces delay and discontinuities to the computation of physics and rendering of forces because of the use of the graphic card for the visual rendering computation.

Some works addressed the simulation of high demanding or collaborative environments and proposed the development of algorithm for remote rendering. The basic idea is to remotely perform the rendering computation of the scene. In [48] a network architecture that provides adaptive level of detail rendering is presented. In this architecture the machine in charge of rendering downloads a representation of the part of the scene that the user is exploring, and the server provides the representation with a progressive encoding. The main drawback of this architecture is the requirement of precomputing a sequence of approximations of the scene objects, from a coarse representation to a fine one. This requirement can not be satisfied when the environment is populated by deformable models, because of the difficulty of precomputing representation with different levels of detail.

In [96] a generic method for remote rendering of computed scene is proposed: the method basically draws the scene on the machine that performs the computation, and then transmits the drawn frame buffer through the network to the remote viewer. The network bandwidth requirement of this method is quite high, as the whole rendered scene has to be transmitted to the viewer at each frame. In addition this does not reduce the computational load on the *GPU*. In [31] a comparison between streaming of rendered scenes and streaming of rendering commands is performed. Moreover an innovative approach to *OpenGL* commands streaming is proposed and evaluated with games. The main limitation of the latter method is its bandwidth requirements, as, for complex scenes it exceeds 26 $Mbit/s$.

To address most of the limitations discussed, in the following sections we will present the data structures we developed to fully exploit the *GPU* computational power, then we will describe how the physics of *MSM* is implemented on *GPU* with simple collision detection between rigid bodies and virtual tools. Then we will explain how basic interactions can be rendered with this approach and how topological changes can be handled. At the end we will present a framework that allows exploiting remote computation to speed up the simulation and to obtain remote rendering of the virtual environment.

7.2 Physics Simulation

To fully exploit the *GPU* computational power it is necessary to encode data in a format that allows parallel processing and that agrees with the limitations imposed by *GPU* memory management. In addition to proper data structures, *GPU* computational model imposes limitations on the code, such as limited number of simultaneously accessible resources, *CPU*-*GPU* synchronization issues, lack of function recursion support and conformance issues to IEEE-754 floating point numbers. The most complex porting phase is the required adaptation of the original code in terms of arrays, interpolators and kernel functions. In the following sections we describe the data structures we designed to store *MSM* data and the algorithms we developed to compute the physics of the deformable models.

7.2.1 Physical Model Representation

Our model representation extends the one proposed in [39]. The mass data structure is composed by the position vector $\vec{x}$ for three contiguous time instants, a force vector $\vec{F}$ accumulating internal and external forces, and a set of constant values such as mass value and damping factor. Since current *GPU* arrays are limited to 4096 elements for each dimension and to 4 scalars for each element, the physical allocation of n masses requires a set of bidimensional arrays with a size of $w \times h$, with $w \cdot h \geq n$. This set, successively called mass array, is composed by 5 independent arrays: 4 to store $\vec{x_{t+1}}$, $\vec{x_t}$, $\vec{x_{t-1}}$, $\vec{F}$ in homogeneous coordinates, 1 to store other constant mass information. To load the entire data structure of the i-th mass the *fragment shader* needs to access each array with the same (u, v) coordinates, where $u = i$ div w and $v = i$ mod w.

The spring representation requires a set of S arrays of size $w \times h$ each, where S is the maximum spring valence. Each spring data structure is stored into two independent elements, at the coordinates used by the two connected masses (u_1, v_1) and (u_2, v_2). The structure stored in element (u_1, v_1) (alternatively (u_2, v_2)), is composed by the rest length, the elastic coefficient and the (u_2, v_2) coordinates (alternatively (u_1, v_1)). There is no need to store the coordinates of the first mass because they are equal to those where the actual spring structure is located. The spring second end (u_2, v_2) is encoded in one single integer $u_{v_2} = v_2 + \frac{u_2}{w}$ to save a scalar value, therefore we store the spring damping coefficient in that field. Since the spring valence s_i is not constant over the set of masses, the spring arrays will contain $\sum_i (S - s_i)$ empty elements. During the computation process the *fragment shader* will detect and discard these elements with a loss in global performance.

To reduce the impact of empty elements on system performance, we propose an innovative policy in the mass allocation phase aiming at reducing the number of empty elements processed. Our method is based on the possibility of applying the *fragment shader* not only to bi dimensional subregions but also to mono dimensional ones. Initially we set the mass array dimensions: width is set to the maximum *GPU* array size, usually 4096, while height is set accordingly to accommodate the entire mass set. Before allocating the mass set, we sort the masses by decreasing value of s_i and we compute the corresponding (u, v) coordinates as

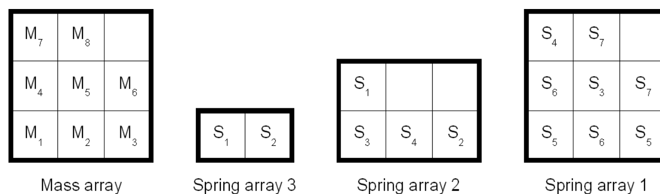


Fig. 7.1. Allocation of masses and springs of a dummy model: spring S_1 links masses M_1 and M_4, spring S_2 links masses M_2 and M_3 and so on.

described previously. At runtime every *fragment shader* is applied to the bidimensional array by horizontal lines, by the following pseudo-code:

```
current = 0
row = 0
while (current<total)
  count = max(width,total-current)
  line(0,row)(count,row)
  current = current+count
  row = row+1
```

where `width` is the width of the mass array, `total` is the number of total elements to process, `current` is the number of element processed, `count` is the number of elements to process in a single `line` call. A simple example is depicted in Fig. 7.1. This dummy model is composed by 8 springs and 8 masses (M_1 and M_2 with $s_i = 3$, M_3 and M_4 with $s_i = 2$, and M_5, M_6, M_7 and M_8 with $s_i = 1$). In this model, the maximum spring valence S is equal to 3, thus we created 3 spring arrays.

Along with the mass static information we store information about the state of springs acting on each mass. We put this information in bit masks that are stored in a texture of the same size of the mass texture and that is used to update the model topology at run time. We chose to store data in a half precision floating point texture to speed up the computation. *GPU*'s store half precision floating point values with one bit for the sign, ten bits for the mantissa and 5 bits for the exponent so integers in the range $[0, 2^{11}]$ can be stored without loss of precision. Since each texel has up to four values we get 4 positive integers of 11 bits each. In each bit we store the state (active/cut) of each spring, for a total of 44 springs for each mass, which we found to be more than enough for all models we used.

In Figure 7.2 we show an example of one texture used to encode the spring state. Two springs of the model are cut: spring 22 that links mass 30 and mass 18 and spring 27 that links mass 30 and mass 31. For mass 30 the encoded sequence of bit is 110 (since the second and the third springs are cut), and we store the sequence as a 6 in the spring state texture. When decoding this information we lookup the spring texture stack, and in the position corresponding to mass 30 we consider as cut the springs in the texture at level 1 and 2. Since springs are stored twice in the spring texture stack a total of 4 bit in the whole spring state matrix are set to 1.

As spring valence varies over the mass set, the spring textures will contain elements marked as empty: at runtime these elements must be detected and dis-

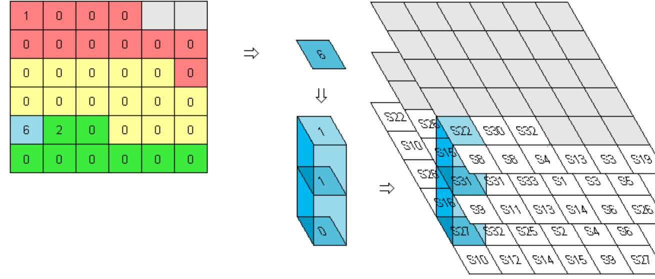


Fig. 7.2. Encoding of cut spring in the spring state texture: spring 22 and 27, linking masses 30 and 18 and 30 and 31, respectively, are cut.

carded. To access all springs connected to a specific mass stored at (u, v) the *fragment shader* has to fetch each element stored in position (u, v) in the spring textures, decompress each c_{uv} member and finally use the obtained coordinates to fetch the mass textures. The spring textures for a dummy model are depicted in Fig. 7.3: gray texels are marked as empty and should be discarded during runtime computation.

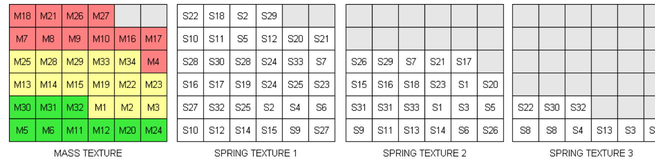


Fig. 7.3. Allocation of masses and springs of a dummy model: gray texels are marked as empty and represent springs that should be discarded during runtime computation

To better understand the proposed method is useful to think about the spring textures as a stack of horizontal slices (i.e. a 3D matrix or a 3D texture), where the first two indexes u and v identify the first mass of each spring, and the third index t is the height of spring inside the texture stack. By keeping u and v fixed, it is possible to scroll along the matrix to find all the springs that act on the mass (u, v). So each pile in the 3D texture can be seen as an ordered set of springs. When handling cuts or topological changes we use this implicit order to identify springs in the model and we represent cuts by enabling or disabling springs in a fast and optimized way.

7.2.2 Elastic Force Computation

The elastic force that acts on the mass i of the model is computed in accordance with the theory presented in Section 2.2, thus it is obtained by:

$$\vec{F}_i^{el} = \sum_n^{s_i} \vec{F}_n = \sum_n^{s_i} \left(k_n \frac{\left|\vec{l_n}\right| - r_n}{\left|\vec{l_n}\right|} \cdot \vec{l_n} \right) \tag{7.2}$$

where s_i is the number of springs connected to the i-th mass, called *spring valence*, $\vec{F_n}$ is the elastic force due to the n-th connected spring, $\vec{l_n}$ is the distance vector between the two connected masses, r_n is the spring rest length and k_n is the elastic coefficient. This equation is used to update the array where $\vec{F}^{el}$ vectors are stored. Each element accumulates $\vec{F_i}$ vectors computed in different iterations by the following pseudo-code:

```
#on cpu:
reset totalForce array
for each spring array i from S to 1
  springTexture = springArray(i)
  massTexture = massArray
  apply fragmentShader on masses with s>=i

#on gpu:
for each (u,v) selected by cpu:
  spring = fetch(springTexture,(u,v))
  break_if_null_spring(spring)
  break_if_cut_spring(spring)
  v1 = fetch(massTexture,(u,v))
  v2 = fetch(massTexture,expand(spring.uv2))
  l = v2-v1;
  f = spring.k*(l-spring.rest*normalize(l))
  totalForce(u,v) = totalForce(u,v)+f
```

To evaluate the function `break_if_cut_spring(spring)` the shader needs to fetch the element (u, v) from the spring state textures (i.e. a single floating point value that stores the state of all the springs connected to the spring first mass) and to detect whether the current spring is cut or not by checking the i-th bit in the spring state. In the dummy example of Fig.7.1 the algorithm saves only three useless calculations, but in real cases this technique significantly reduces the number of processed elements for texture lines that are partially full: if volume preservation is considered (see next Section), our method is 20% faster compared to [39].

7.2.3 Volume Preservation

One of the biggest issues using mass-spring systems is the inability to model some material volumetric properties, for example incompressibility. This limitation can be overcame by the introduction of volumetric entities, tetrahedra, and by considering additional force contributions, as suggested in [63]. The tetrahedron representation is similar to the spring one. The data structure stored in (u, v) is composed by the rest volume and 3 indexes, compressing the coordinates of the 3 masses connected to the one stored in (u, v). The computation of volume preservation force is more expensive than the computation of the elastic force: each tetrahedron is processed and stored four times. For this reason, we sort the mass set by a new key: $t_i \cdot S + s_i$ where t_i is the number of tetrahedra connected to the i-th mass.

The computation process of the volume preservation is similar to the elastic force method. The additional force contribution $\vec{F}_i^v$ is obtained by the following:

$$\vec{F}_i^v = \sum_j^{t_i} (v_r - v_j) \cdot \vec{n}_{ij} \tag{7.3}$$

where v_j is the volume of the j-th tetrahedron, v_r is the tetrahedron rest volume and $\vec{n}_{ij}$ is the normal vector for mass i facing outside the tetrahedron j.

This approach improves the simulation realism for deformable bodies with low elasticity springs but it does not provide any noticeable difference otherwise.

In the next section we will discuss how all the computed forces are used to evolve the state of the system.

7.2.4 Temporal Integration

A new computational phase starts after all the internal forces have been computed. In this phase the state of the model is updated to move its configuration towards an equilibrium state through the numerical integration of mass positions.

Temporal integration of mass position is performed by a *fragment shader* that works on mass positions and data and uses the computed internal force vector to update the position of each mass. The computation is based on a Verlet scheme[1] [106]. The choice of Verlet scheme is motivated by its good trade off between stability and computational complexity. In fact it is a two step explicit integration scheme and it only requires the position of the mass in the two previous temporal steps and the force currently applied to the mass. This makes the method easily parallelizable and suitable for *GPU* implementation: the following pseudo-code performs the temporal integration as described.

```
#on cpu:
massProperties = massStaticData
massCurrentPosition = massArray(t)
massPreviousPosition = massArray(t-1)
massForces = totalForce
apply fragmentShader on masses

#on gpu:
for each (u,v) selected by cpu:
  massData = fetch(massProperties,(u,v))
  break_if_fixed_mass(massData)
  currPos = fetch(massCurrentPosition,(u,v))
  prevPos = fetch(massPreviousPosition,(u,v))
  force = fetch(massForces,(u,v))
  newPos =
    2*currPos - prevPos + (force/massData.mass)*pow(dt,2)
  massArray(t+1,(u,v)) = newPos
```

The value `dt` that appears in the pseudo-code represent the temporal step used in the simulation. The three arrays storing the mass positions are handled as a circular buffer, to minimize the operations needed to update model state. Since all

[1] Verlet integration is a numerical method used to integrate Newton's equations of motion. The Verlet integrator offers greater stability than the much simpler Euler method, as well as time-reversibility and area preserving properties.

the data are stored on *GPU* memory, the *CPU* only binds some *GPU* memory areas to the shader variables and thus no complex data exchange takes place.

7.3 Deformable Model Interaction

In the simulator developed to demonstrate the methods described, deformable models interact with fixed structures and with two virtual tools controlled by the user. These two kinds of interaction are handled separately to optimize the computation and to reduce computational time. In the following we explain how we handle collisions between the nodes of the deformable model and some structures that are fixed during the whole simulation. Then we detail three techniques that provide the user with the ability to interact with the deformable models present in the scene.

To increase the realism of the simulation we focused on a surgical scenario, where the user controls surgical tools that act on a human abdomen, reconstructed from real patient CT data [8] . The user can interact with organs in the virtual environment by probing, grabbing or cutting them. All the tests involved in the detection and solution of the interactions are performed in a frame of reference relative to the tool, to simplify the computation. This introduces a small overhead to the computation: in fact it only requires a multiplication of each point by an affine transformation matrix, operation that can be performed very efficiently by the *GPU*. The shape of the tool is approximated with simple geometrical primitives, that can be expressed in analytical form. This reduces the complexity of the collision detection and allows computing the projection of each point to the surface of the tool.

The surface of the model that is used in collision detection and in frictional force computation is obtained as the triangularization of the tetrahedral mesh that composes the *MSM*. This provides us the sets of surfels and triangles needed to perform the graphical rendering of the scene but it also gives us the connectivity information between the surface points that are used to handle the changes in model topology. In fact each edge of the surface mesh is stored and used to improve the results of cuts and grabbing gestures. During each of the described interactions the user feels the forces acting on the virtual tool that he/she handles.

7.3.1 Collision Detection With Fixed Structures

Collision detection and response are the two main aspects of dynamic simulation. Due to problem complexity and hardware limitations, we cannot define an efficient general algorithm. We propose a method to detect body's collisions against rigid bodies with a high complexity mesh in a fixed space. Even if this is a really specific configuration, it is common in a surgical environment. This method is not as general as the methods described in Chapter 4, but it is suitable to detect collision between deformable and static structures at the very high frame rate required in haptic enabled interactive simulations.

Our approach requires discretization of the workspace in a uniform grid composed by cubic voxels. This structure can be stored in a 3D array where each

element stores a boolean value that indicates whether the identified voxel is taken up by some rigid body. Collision detection can be realized in the *fragment shader* used for the final time integration by identifying intersections between each mass motion vector $\vec{d} = \vec{x}_{i+1} - \vec{x}_i$ and voxels marked as *filled*. Due to the small temporal simulation step used, it is usually sufficient to fetch the 3D array at $\vec{x}_{i+1}$ coordinates, applying a standard collision detection algorithm to see whether the corresponding voxel is occupied. The geometry of a skeleton and the corresponding voxelization are depicted in Fig. 7.4.

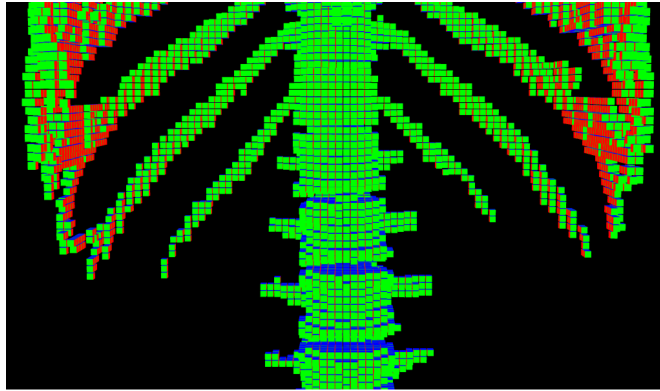


Fig. 7.4. Geometry voxelization.

7.3.2 Probing

The probing gesture allows the user to interact with objects by touching their surface. The implementation of this action is performed in two distinct phases.

The first step detects all the surfels that have collided with the tool during the last simulation step. To identify these points we check the position of each surfels with respect to the tool position in the current and in the previous temporal step. This allows detecting points that lie inside the tool at the end of the simulation step and points that, during the evolution of the scene, “jumped” from one side of the tool to the other. Points that collided with the tool are marked as active and are processed during the second phase.

In the second phase we perform the actual collision response: for each point that during the last temporal step has collided with the tool we compute the projection of its position on the tool surface. This operation is very effective because we use the analytical approximation of the tool. The point is displaced to the projected position to solve the collision, moreover, the projected position is also used to compute the relative velocity between the current point and the tool. The relative velocity is then used to compute the frictional force that is added to the forces acting on the point and used to update the physics of the environment. The results of probing gesture are shown in Fig. 7.5.

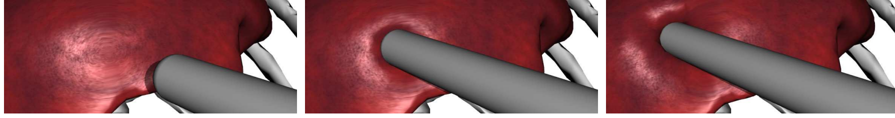


Fig. 7.5. Probing gesture

Fig. 7.6. Grabbing gesture

7.3.3 Grabbing

The grabbing gesture allows the user to manipulate deformable model parts with tweezers. The implementation of this action requires two steps.

In the first step we detect which masses are grabbed by the tools. Grabbed masses will be considered blocked and will follow tool movements. This phase is only needed at the instant when the tool actually grabs the model (e.g. when virtual tweezers close). Its impact on the overall computation time is thus reduced and this allows using complex detection tests. As introduced in Section 7.2.1 this phase takes into account the connections between surface points to increase the realism of the simulation. When the tool performs the grab action we check the edges that cross the tool and we mark them and memorize their position with respect to the tool.

During the second phase, that is performed at each simulation step, we move the grabbed edges to make them follow the motion of the tool. To force this movement we place the marked edges in the stored position with respect to the current tool position. Since this method works on two dimensional elements we are able to apply twist motion to model surface: this would be in general impossible to obtain with only surface points. Since this step is required only if at least one mass is grabbed, we use a query to count the number of marked masses at closing instant but we start using its results only after one simulation step to avoid synchronization issues. The results of this gesture are shown in Fig. 7.6.

7.3.4 Cutting

During the simulation, we apply a *fragment shader* to every edge of the model surface to check if it should be cut or not. We implemented two different *fragment shader*s for cut detection: the first one (*shader1*) is the simplest and it just checks if the segment associated to the current edge intersects the cutter's approximated geometry (an ellipsoid). This can be useful in surgical simulations, to mimic the behavior of an electro cutter. The second *fragment shader* (*shader2*) is more complex and checks whether the triangle approximating the cutter's blade intersects the segment associated to the spring and can be used to simulate the behavior of a knife. For example: when checking the i-th spring of the mass (u, v), de-

tector shader computes the following pseudo-code (the difference between *shader1* and *shader2* is how `detect_cut_spring(spring)` is actually computed):

```
#on cpu:
for each spring array i from S to 1
  springTexture = springArray(i)
  springState = springState
  apply fragmentShader on masses with s>=i

#on gpu:
for each (u,v) selected by cpu:
  spring = fetch(springTexture,(u,v))
  oldState = fetch(springState,(u,v))
  cut = detect_cut_spring(spring)
  newState = cut | oldState
  springState(u,v) = exp2(i)*newState
```

The results of the computations are summed up exploiting standard blending capabilities provided by the *GPU* and the updated state of model springs is obtained. The method we use to encode cut spring data ensures that the results of the sum on the floating point values is consistent with the bitwise OR operator that is commonly used to merge bit masks (i.e. only the *i*-th bit mask can have the *i*-th bit equal to 1). This data representation allows the computation to be performed completely on the *GPU* and does not require data to be exchanged with the *CPU* to update the physics of the model ensuring that the whole computation is fast and that the delay between user input and corresponding force output is kept low.

7.3.5 Interaction Forces Computation

The innovative method presented here to compute interaction forces requires to identify all masses in contact with a virtual tool and to accumulate acting forces and torques. This operation has a really simple implementation on *CPU*s but not on *GPU*s, due to current hardware limitations. Other implementations, as in [95], transfer all masses' positions and forces to the system memory and realize the whole computation on *CPU*. This approach is really simple but significantly degrades simulation performance since it introduces time latencies due to limited bandwidth and synchronization issues. Our approach exploits the *GPU* for all computations, so that only final force and torque vectors have to be transferred. The implementation of our approach is composed of three phases.

The first phase is aimed at computing force and torque vectors for each mass of the system. We store all masses' force F and torque T into two distinct arrays obtained by:

$$\begin{aligned} \vec{F} &= b \cdot M^{-1} \cdot f \\ \vec{T} &= b \cdot M^{-1} \cdot (f \times x) \end{aligned} \tag{7.4}$$

where b is the boolean mark that identifies if a mass collides with a tool and M^{-1} is the inverse matrix of tool position.

In the second phase we accumulate the content of the two new arrays $\vec{F}$ and $\vec{T}$, obtaining the force and torque vectors needed for haptic feedback. This is

realized in the third phase with an iterative process based on a *reduction operator*. The reduction is performed by updating the content of an array by computing the element (u, v) as the sum of elements $(2u, 2v)$, $(2u + 1, 2v)$, $(2u, 2v + 1)$ and $(2u + 1, 2v + 1)$ stored in the array by the previous iteration. The resulting vector will be stored at element $(0, 0)$ of the last destination array. This process is shown in Fig. 7.7 and can be easily implemented by using the classic ping-pong technique [21]. In the third phase we start the asynchronous transfer of results from the *GPU*

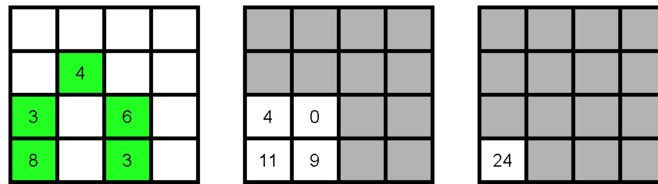


Fig. 7.7. Reduction steps

memory to the *CPU* one.

This method dramatically improves the global simulation process since it scales with *GPU* computational power and does not depend on system bandwidth and *CPU*. Our implementation is about 10 times faster than the one proposed in [95].

7.4 Results

In this section we will present and evaluate the results obtained by simulating models of different complexity and topology and cutting them using the proposed method with the two different cut detector shaders we described in Section 7.2.1. To test the performance of the method we integrated it into a deformable model simulator based on *OpenGL Shading Language* that exploits current graphics card computational power. Each step of physical simulation is composed of five phases:

- approximate collision detection between the two virtual tools and the deformable model;
- cut detection (only for the cutter tool);
- collision response;
- internal forces update;
- numerical integration of mass positions.

Obtained simulator represents a human abdomen composed by rigid, fixed structures, and one or more deformable organs. It provides different kind of virtual tools to allow the user performing actions such as grabbing, probing or cutting. The user handles up to two tools and receives graphic and haptic feedback. Since we model the physics of the whole deformable model, the user perceives also the interaction between the tools. A screenshot of the simulation can be seen in Figure 7.8.

The main goal of this work is the development of a method to simulate cutting of deformable models at haptic frequencies. Therefore the principal aspect we take

Fig. 7.8. A screenshot of the simulator showing the interaction between two tweezers and a virtual liver.

into account during tests is the physics update frequency, but we also present and discuss the graphical rendering, as it is an important part of any simulation with haptic feedback. In all tests discussed here the graphical rendering was performed at 30 fps, using per-pixel lighting and normal mapping. The tests were performed on a desktop equipped with an Intel Core 2 Duo E6600 with 2 GB of ram and an Nvidia GeForce 8800 GTX.

To obtain a meaningful evaluation of our algorithm we instantiated three models that differ for element (masses and springs) number and topology. The first model (*cube*) is obtained by tetrahedrization of a cube: the resulting model is composed by 21544 masses, 139236 springs and 452340 triangles (internal and external) and shows a regular disposition of masses and springs. The second and third models (*liver1* and *liver2* respectively) were reconstructed from surface representation of a liver with different levels of detail. The second model is composed by 7977 masses, 50115 springs and 158960 triangles, whereas the third one is composed by 20227 masses, 130754 springs and 425112 triangles: due to the nature of the initial surface the two liver models show an irregular mass arrangement that leads to an irregular mesh. In Figure 7.9 we present a wire frame representation of the models used during the tests and in Table 7.1 we summarize their features.

Model name	number of masses	number of springs	number of triangles
cube	21544	139236	452340
liver1	7977	50115	158960
liver2	20227	130754	425112

Table 7.1. Main features of the models we used during tests

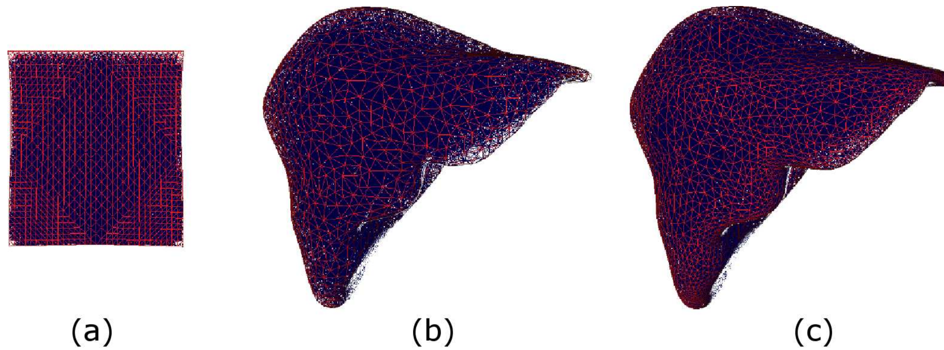


Fig. 7.9. Wire frame representation of models used during tests: in (a) the model cube, in (b) liver1 and in (c) liver2.

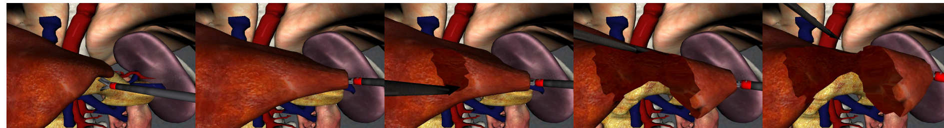


Fig. 7.10. Evolution of liver2 model undergoing deformation and resection.

We apply our method with the two different cut detector shaders to the three models and compute the update frequency of the model physics with the graphics rendering. During the test we allow the simultaneous use of two tools able to perform cut, so the shader in charge of handling the cut was applied twice for each simulation step. Table 7.2 summarizes the results we obtained.

Model	*shader1*	*shader2*
cube	0.91 msec (1099 Hz)	0.92 msec (1080 Hz)
liver1	0.74 msec (1351 Hz)	0.70 msec (1428 Hz)
liver2	0.90 msec (1111 Hz)	0.94 msec (1063 Hz)

Table 7.2. Update times of cutting method applied to the three models (in brackets the update frequency)

The proposed methodology has proven to work efficiently. Moreover, the mass-spring model used for deformation is also used transparently for the cut, so that there is no data/code duplication and the integration is full and natural. Synchronization between the surface mesh and the physics model is thus granted (see Figure 7.10 and Figure 7.11). This approach to the cut is the simplest possible, but relies on a series of specific solutions to avoid the down-sides.

As mentioned above, the cut has been studied in [33] and [34], concluding that the "remove tetrahedra from list" principle is considered simple but limited, so manifolds were preferred. Our implementation contradicts this point: the use of *GPU* introduced the possibility of doing calculation in parallel, increasing the

speed of computation on conventional PC with mid-end video cards. A wise use of the *GPU* allows our algorithm to perform at rates far over 1 kHz with an enormous number of elements. Therefore, no mesh smoothing/resampling is performed, as each tetrahedron is small: a cube with 113085 tetrahedra still manages to be simulated within the haptic threshold. No new points are added, and the movement of vertexes is managed independently by the underlying mass-spring model, so that coherence is still preserved. Given the need of haptic feedback, it is important to preserve the volume of data in use, as adding elements make structures and memory usage grow, thus increasing the computation complexity and leading to unpredictable update frequency. On the contrary, our approach guarantees a constant amount of data and makes the computation times constant and reliable. Interference detection, both to find collision and detect masses for probing/grabbing/cut scales depending on the number of vertexes, and can be tuned by setting the dimensions of the cutting tool. Along with the huge number of tetrahedra, the number of springs is also very high. This means that the behavior of the physics is still very realistic and precise. The distinction between operative and non operative zones is not present due to the fine definition of the external and internal mesh. A distinctive feature of the presented implementation is its generality, so that no constraints are imposed on the way meshes are built. So, whilst not needed, it is still possible to build models with different non-homogeneous resolutions whenever is necessary. Our method offers a solution easy to understand and to use. Graphically, we bypassed the limits of the tetrahedra removal approach by taking advantage of the *GPU* to compute fine meshes, thus getting a better rendering and a more precise and realistic physics, difficult in traditional removal-based algorithms. The need of smoothing seen in [33] and [34] is avoided, and so is the need of deep topology changes, as point addition and mesh resampling. The speed of calculations still allows using up-to-date graphical effects as per-pixel lighting and normal mapping to enhance the visual quality. The physics used for the cut is designed to blend with existing models and data structures, with few additions.

Fig. 7.11. A cube of cheese (the cube model described in Section 7.4) is deformed and cut.

In the whole, results reported in Table 7.2 show that this method scales well with the increase of model complexity, as the high parallelism provided by the *GPU* makes it works proportionally better with more complex models. Moreover it has been implemented to scale well with the *GPU* computational power, and, as video cards quickly increase, the speed of computation will significantly increase without any additional modification to the code. Despite FEM and other advanced methods may have higher precision, the high resolution of our models still grants a high level of realism, relying on the high number of elastic elements. The tricks associated to tetrahedral removal have been avoided, and the model is dependent neither on local models nor on pre-computations.

7.5 Optimized Graphical Rendering

As described in Section 7.1 irregular update of the force feedback in physical simulation leads to the perception of non realistic physical behavior of the environment. Moreover a common assumption in deformable model simulation is that the velocity of objects in the scene (human operated tools and environment) stays below a certain threshold because fast movements or sudden interactions may lead to instability in the simulation. Changes in the perceived speed of the environment cause faster reactions by the user and thus instability in the simulation. The physical simulation and the haptic rendering should then have the highest priority in the computation. On the other hand the visual rendering is slower, as 30 frame per second are enough to ensure a good visual quality, and can tolerate higher delays without causing loss of realism. For these reasons *GPU* based physical simulation can take advantage of the following method for remote visual rendering.

7.5.1 Remote Rendering Overview

We analyzed the behavior of the proposed interactive simulator that is based on *MSM* and exploits the *GPU* power to speed up the computation. We found that on a NVIDIA GeForce 7900 graphics card the mean time for a single frame update (made up of elastic and damping forces computation, collision detection and resolution followed by force summation) is 0.87 ms, while a single step of graphic rendering take approximately 8 ms. The switch between physics computation and graphics rendering leads to an unsynchronized update of the configuration of deformable models and to a non linear force feedback rendering.

To solve this problem we decided to perform the simulation using two computers arranged in a distributed system where a PC equipped with a powerful programmable *GPU* (the simulator) perform the physical simulation and the force summation for haptic rendering, while the other PC with no particular requirements (the viewer) is only in charge of the visual rendering. To limit the delay in the graphics rendering the coding, transmission and decoding process should be fast, moreover, to avoid jitter the process should always take the same time. From these considerations we decided not to use any data compression algorithm as it will increase the computational time and it will change the length of the coding during the simulation. We implemented instead a lightweight remote rendering protocol.

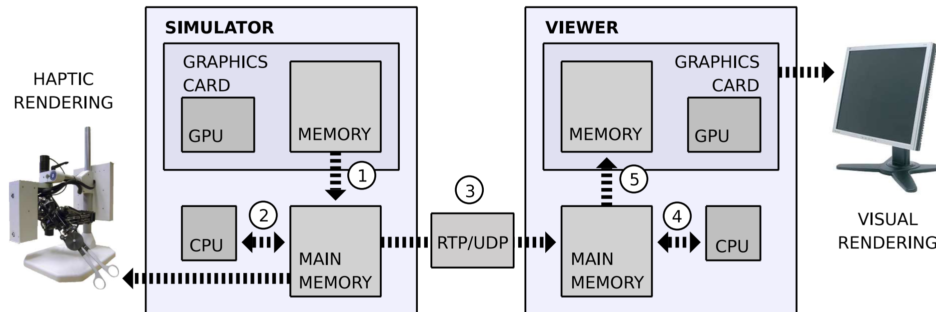


Fig. 7.12. Remote rendering protocol: (1) Asynchronous data download, (2) Data encoding, (3) Data transmission, (4) Data decoding, (5) Rendering

The network protocol chosen for the transmission is RTP (Real-time Transport Protocol) over UDP. We choose UDP because it guarantees speed and low delay in the transmission, and RTP because it delivers the necessary data to make sure that the viewer can put the received packets in the correct order, moreover RTP is a standard protocol for real time communications, and this makes it a good choice for future development.

Our protocol makes some assumptions about the bodies of the environment:

- The scene can be described two ordered sets: the first describing fixed or floating rigid bodies and the second composed by deformable bodies.
- Both PCs (simulator and viewer) have the initial representation of the scene (object models and placements).
- Deformable models can be described by a cloud of point potentially linked in a polygonal mesh.
- The simulator can asynchronously download the surface representation of deformable models from the graphics card memory (asynchronous download does not lock the *GPU* and thus introduces negligible delay in the physics computation).

Remote rendering is composed of five phases (graphically summarized in Figure 7.12):

1. *Asynchronous data download*. For each deformable model of the virtual scene its surface representation and edge connectivity data are copied into the main memory of the simulator (i.e. on the *CPU* memory). The use of asynchronous transfer allows the *GPU* to continue working and thus it does not stop the physical simulation.
2. *Data encoding*. For each point of the deformable models the simulator computes the difference between its current position and the initial position. Each coded point is represented by three displacement in three directions. Coded points are stored in memory ordered by model and then by their index inside

the model (this order must be the same on the simulator and the viewer). Edge data are not encoded, as they can be considered as an effective, compact representation of the model state since only one bit is used to store the state of each edge.

3. *Data transmission.* Data encoded by the previous step are sent to the viewer using RTP over UDP. Each RTP packet contains a time stamp that indicates the moment in the simulation at which the data has been encoded, a sequence number, which is a progressive packet number that goes from 0 to the total number to transmit one frame (this number is fixed during the whole simulation, as the encoding does not change the data length and the number of transmitted points remains constant). The last packet of the stream contains matrices that define positions and orientations of rigid bodies of the scene and, if necessary, other useful information for the viewer. The last packet has the 9-th bit in the header set to 1, to indicate the end of the coding.
4. *Data decoding.* At the beginning of the computation the viewer pre-allocates and zeroes a memory area that will contain the ordered displacement of sur-

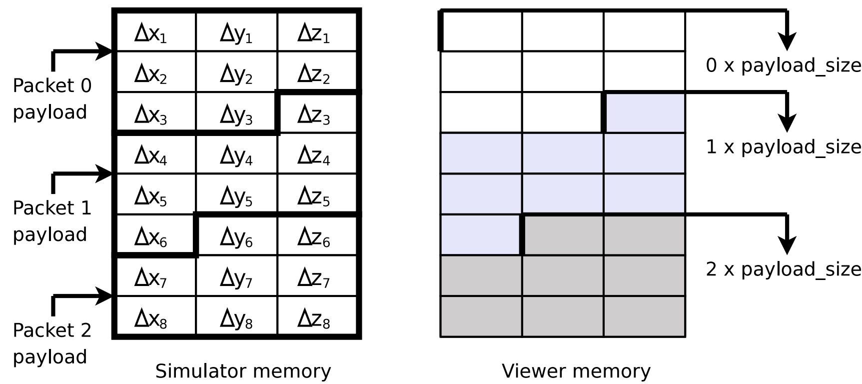


Fig. 7.13. Graphical representation of the simulator and viewer memory alignment. The viewer stores received packet with offset equal to the product between packet sequence number and RTP payload

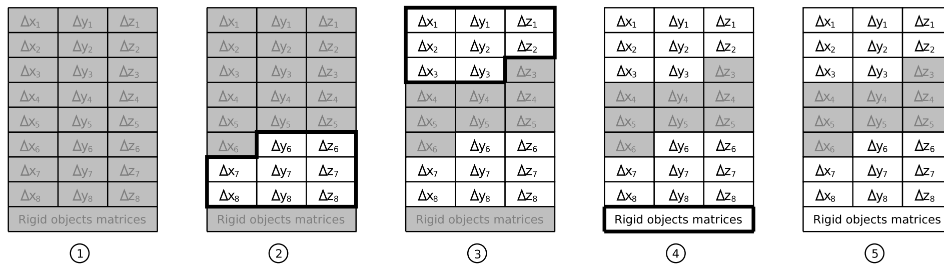


Fig. 7.14. Evolution of the viewer memory during a frame rendering: (1) Memory contains previous frame data, (2) Packet 2 is received, (3) Packet 0 is received, (4) Last packet is received, (5) Rendering is performed (packet 2 is lost or delayed: previous data is used)

fels and the edge state of deformable models. During the simulation received packets are discarded if their time stamp is lower than the higher time stamp received (i.e. if the packet belongs to an old frame). If the packet is not marked as last its data is copied into the memory, using the packet sequence number to compute the offset (Figure 7.14 (2) and (3)). The offset is computed as the product between packet sequence and RTP packet payload size (Figure 7.13 shows graphically the alignment of simulator and viewer memories). This ensures that the packet arrival order is not important in the decoding of a frame. Moreover, since the memory area that contains stored data is never zeroed (Figure 7.14 (1)), if a packet is lost or delayed previous data is automatically used (Figure 7.14 (5)). Since rendering is carried out at 30 frame per second and movements are usually slow, the introduced errors are negligible. When the last packet is received its data is processed: matrices are extracted and useful information are used accordingly (Figure 7.14 (4)).

5. *Rendering.* When the last packet has been received and decoded the viewer adds the updated displacements to the initial configuration of the model to obtain the current representation and update the state of the model edges accordingly to decoded data. Received matrices are used to update the position and orientation of rigid bodies in the scene. The described method only updates the scene, it does not trigger the graphical rendering. Instead the graphical rendering is performed at $30fps$ to ensure realistic perception and to allow the viewer to control some rendering parameters (mainly the camera position).

7.6 Integration in the Simulation

We implemented and integrated the proposed method into the GPU-based framework described before. In the implementation we choose to put in each RTP packet 1000 bytes of data (corresponding to about 83 encoded points), so the whole packet results composed by 1012 byte. This ensure a good trade off between speed and reliability.

To prove the correctness and test the scalability of this method we tested it on different configurations: we tried the simulator/viewer communications on different architectures, with different network connections and with models of different sizes, results of these tests are provided in Section 7.6.2 and following.

7.6.1 Deformable Models Rendering

During the graphical rendering the viewer should update the scene description. The scene is composed by three kinds of models:

- Rigid, still bodies, like chest or other bones, do not move during the simulation and thus do not require any update in their data structures.
- Virtual tools, controlled by the user, needs only few data to define their state, as we discussed, the matrices that define their position and orientation are sent in the last packet of the encoded scene, along with few bits that describes their state (closed/open tweezers, ...).

- Deformable bodies. The state of those models is described by the position of their points and by the state of their edges.

The rendering of the first two classes of models is trivial, even when performed remotely, but the handling of deformable models is more complex, also because of the changes in topology that can happen during the simulation. To render changes in edge connectivity in the model the viewer has to update the graphical representation of the model to reflect the changes in physics. During the physics simulation, springs (or edges) are marked as cut and their contribution to the internal force is neglected. In graphics it is necessary to discard the triangles that use that spring to represent the new physical state. Even if visual rendering has less strict requirements than haptic feedback all the model updates and visual rendering operations must run in real-time at 30 Hz. To carry out this task respecting those constraints we take again advantage of the GPU. We defined some optimized data structures to ensure the required frame rate.

Starting from a complete description of the model triangles (i.e. we have a list of every triangle of every tetrahedron of the model) the goal is to remove the triangles in which at least one side is matching with a cut spring in the physical model. Another requirement is to render in a different way the inside and the outside of the object in order to make the cut visual rendering more realistic.

To store information about points and the complete model triangulation we rely on Vertex Buffer and Index Buffer, respectively, that never change during the whole simulation, and thus does not need to be transmitted by the simulator to the viewer.

To efficiently handle data stored in these structures we developed a method to store and access triangle data that is both simple and fast, and is based on a list of incident triangles for each vertex. This structure is an array of lists whose length equals the number of element in the Vertex Buffer. At position i of this array there is a list whose elements are four integers, as presented in Figure 7.15: the first three elements are the indexes of the three vertices referenced by the triangle, one of those (marked in red in the figure) will be the vertex i. The last element of the list is the index of the triangle they belong to, which is the pointer to the first index in the Index Buffer. The vertex i can be omitted for what concerns the cut, but it is necessary if disabled triangles should be restored for any reason. In that case, in fact, the order of the vertexes will be needed. The fourth value is kept because it allows a fast access to the triangles that need to be cut, avoiding an expensive search. As we do not change the triangulation in real-time but we only delete triangles of the model representation, this data structure can be stored in the off line initialization phase, without affecting performance during the graphical rendering.

In this way we can retrieve the list of triangles that share a vertex in constant time. Using this structure we get the triangles in which two vertices are present simply by finding the triangles that are in both the lists. The described data structures is based on information available on two different buffers, the Vertex and the Index Buffers on the GPU. The need for fast access, considering that those buffers can contain huge amount of data, is then satisfied by using indices that build a map of the memory. If we need to find triangles relying on a certain vertex with index i, we just scan the array at position i.

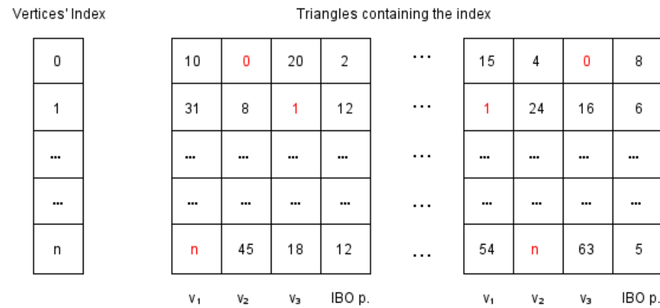


Fig. 7.15. The representation of the data structure that store and access efficiently the lists of triangle

Once that those data structures have been initialized at the beginning we can efficiently use them to update the graphical representation of the model. To do this, we obtain the cut edges from the simulator and find the triangles to remove. As explained in Section 7.2.1, during the physical simulation the spring states are stored for each vertex. The method we use to store them allows fast computation on the *GPU* and can be used as a 44 bit wide unsigned integer that, read bit by bit, represents the state of each spring connected to the point, each bit being a flag. Spring corresponding to bit with value 1 are the cut springs. So, when we do the cut we first get a texture of *half* which will be seen as a texture of bit masks. The texture is downloaded from the *GPU* memory to the *CPU* memory and decoded, then the obtained bit mask is used to find, through the indexes, the springs affected by the cut. From the edge index we can fetch its two ends indexes and use them together with our structure to find the triangles that should be cut. Those triangles are then turned into degenerate elements by setting their three indexes to the same value, so that the culling phase during the rendering will discard them. The consequence is that the triangle is not visible in the rendering. It is possible to use one single value to degenerate the triangles, e.g. 0, so all the cut triangles have 0, 0, 0 as indexes. By using instead different numbers it is possible to restore cut triangles.

With this method, we can handle all the internal triangles (that will not be visible until we cut the surface) and the surface triangles of our model in different structures and treat them in different ways (e.g. different textures) to improve the rendering realism.

After the model representation have been synchronized with the physical state the rendering is performed with the possibility, as mentioned above, to use different textures to best represent the discontinuity between original external surface and internal exposed surfaces as can be seen in Figure 7.16.

7.6.2 Architecture Scalability Test

The first test aimed at proving the scalability of the method to more powerful hardware, so we compared the performance of the simulator/viewer architecture when the underlying architecture varies between a Nvidia 7 series *GPU* and a

Fig. 7.16. The model of an orange cut in slices that fall under the effect of gravity.

Nvidia 8 series *GPU*. The first machine on which we tested the simulator was a laptop with an Intel Core 2 Duo T7200 (2.16 *GHz*), 2 *GB* of ram and a Nvidia GeForce 7900 GS. The second was a desktop equipped with an Intel Core 2 Duo E6600 with 2 *GB* of ram and a Nvidia GeForce 8800 GTX. The viewer run on a desktop with an Intel Xeon 2.80 *GHz*, 2 *GB* of ram and an Nvidia Quadro FX 3450. During these tests the PCs running the simulator and the machine running the viewer where connected through cable to the laboratory LAN. The deformable model used during these tests was composed by 7750 points, 38077 tetrahedra and 48254 springs, the scene was composed by some fixed rigid bodies (that does not affect the transmission), two virtual tools and a floating camera. Table 7.3 summarize results of the tests.

Table 7.3. Simulation and visualization on different architectures

Obtained frame rate			
Architecture	only physics simulation	physics and local rendering	physics and remote rendering
7900 GS	1299 *Hz*	1018 *Hz*	1288 *Hz*
8800 GTX	3412 *Hz*	3409 *Hz*	3167 *Hz*

Computational times			
Architecture	physics	local rendering	remote rendering
7900 GS	0.769 *ms*	7.211 *ms*	0.282 *ms*
8800 GTX	0.293 *ms*	2.394 *ms*	0.029 *ms*

Computational times reported in Table 7.3 show that the classic graphic rendering of a scene takes about ten times the computational time required to update the physics of one step, while the proposed method for remote rendering requires less than half of the time required for one physics step on a GeForce 7900 GS and a tenth of the time required to update the physics on a GeForce 8800 GTX. The big difference in the ratio between physics update and remote rendering on the two

architectures can be explained considering that the first computer is a laptop and that laptop performance is usually worse than a desktop is. These tests highlight the ability of the proposed method to save the graphics card from the burden of the rendering and confirm that the method is useful in guaranteeing a smoother physics simulation that leads to better realism of the simulation and numerical stability in the temporal integration.

7.6.3 Network Performance Test

Then we tested the proposed protocol with different network connections using the first computer and the environment described for the previous test. The connections we considered during this test were: direct connection via crossed cable, connection through wired LAN and connection through wireless LAN. The scene used during these test was the same scene used in previously described tests.

The use of a crossed cable between the simulator and the viewer allows the simulation to run at 1289 Hz whereas the use of cabled laboratory LAN leads to an update frequency of 1288 Hz. Last, the use of wireless connection from the simulator to the cabled LAN yield an update frequency of 1255 Hz for the physics simulation.

In these tests we intentionally avoid to consider transmission over Internet as our protocol is currently intended to work only on local networks. As expected the choice of UDP protocol ensures that the transmission speed is independent on the wired LAN configuration, as UDP provides no guarantees for message delivery. However the architecture performance depends on the network interface speed, as shown by the last test, where wireless LAN has been used. These results clearly show that the method is suitable to be used in local network to transmit the virtual scene from the simulator to the viewer, moreover the use of the wireless connection does not influence the quality of the rendering even if UDP protocol has been chosen for the transmission.

7.6.4 Model Complexity Test

For the last test we changed the size of the model used in the simulation. This allows to change the quantity of locally rendered and transmitted data for each graphic rendering step. We developed two deformable models representing the same deformable object. The first low resolution model, is composed of 1244 points, with 599 surface points, 5413 tetrahedra and 7253 springs. The second one with higher resolution, is composed of 7977 points 2401 of them marked as surface points, 39740 tetrahedra and 50115 springs. The two models simulation were tested on the PCs used for the first test, for each model and machine three modalities were evaluated: physical rendering only (without any visual rendering), classical local rendering and the proposed method for remote rendering. Results are presented in Table 7.4.

From the tests we obtained the overhead due to the download and the transmission of the scene description: in the case of the GeForce 7900 GS with the low-res model, the overhead is about 0.058 ms while with the high-res model the

Table 7.4. Effects of model resolution on the simulation

Intel Core 2 Duo T7200 with GeForce 7900 GS (Laptop)			
Model	only physics simulation	physics and local rendering	physics and remote rendering
Low-res model	2282 Hz	1872 Hz	2278 Hz
High-res model	1253 Hz	991 Hz	1244 Hz
Intel Core 2 Duo E6600 with GeForce 8800 GTX			
Model	only physics simulation	physics and local rendering	physics and remote rendering
Low-res model	3381 Hz	3229 Hz	3377 Hz
High-res model	3413 Hz	3228 Hz	3404 Hz

overhead is 0.239 ms. When using the GeForce 8800 GTX the overhead is reduced to 0.039 ms for the low-res model and to 0.088 ms for the high-res model.

Tests demonstrate that the on the first architecture our method scales linearly with the data transmitted. In fact, the ratio between transmitted data for the first and the second model is 0.249 and the ratio between time required for the encoding and transmission of the two models is 0.243. On the second architecture the ratio between the overheads is 0.443 that indicates a sub linear dependence on the data size. This indicates that the proposed method is suitable to handle complex scenes composed by many deformable models or for scene with high level of detail. It can be noticed that in the presented case the physic simulation performs better with the more complex model, this is not due to the remote rendering protocol but to the dependence of simulation performance on the model topology (i.e. the maximum number of springs connected to each point).

7.7 Conclusions

The objective of the presented methods is to propose a solution that is both graphically appealing and realistic, real-time (with no pre calculated data), general (with no limitations to user interactions), high performance and haptic compliant (frequency must be higher than 1kHz). Despite the severe requirements the results matched the expectations and can be considered good. Graphics, depending on the resolution of the models, has proved to be realistic. Shaders have been used to improve and enhance the rendering, including per-pixel lighting, normal mapping, shadowing and other effects. Stereo rendering and 3D goggles are supported by our current implementation.

The physics of the cut has also proved to have a realistic behavior. The cut can be both superficial and deep, with the possibility to cut the object into several pieces. Even complete resections of an object preserves the physics of all the parts. The results of the computations of the physics have been tested thoroughly with high-end haptic devices, proving to be perfectly working with real time constraints and to give a realistic sensation.

The implementation has proved to match the speed requirements, taking advantage of the power of the video cards. Programming the computation with

Fig. 7.17. The simulator experimental setup: haptic feedback is provided through two MPB Freedom 7s connected to the simulator, graphic rendering is obtained with the described method

shaders has shown to give very high computational rates, impossible with traditional *CPU* programming. Physics is updated at rates higher than 1 kHz, well beyond the requirement for the haptic devices. The implementation allows the cut to be completely generic, and no constraint is present about the way it is managed. Furthermore, the implementation uses *OpenGL Shading Language* shaders, highly compatible and cross platform, as well as *OpenGL*. The novelty of the approach is focused on realism, performance, generality and portability. The implementation, designed to be the state-of-the-art solution, has confirmed theoretical results and proved to be suitable for a wide range of fields, especially real-time, high-frequency applications.

With our remote rendering procedure we can increase the complexity of the simulated environment with deformable or rigid, fixed or floating, bodies. The method relies on the asynchronous non-blocking download of data from the graphics to the main memory to decrease the *GPU* computational burden, downloaded data is then sent through the network to the viewer with a protocol that automatically corrects packet losses. The method relies on the possibility to asynchronously download data from the *GPU* and on the relatively slow refresh rate of the visual rendering to obtain updated representation of deformable models from the graphics card memory without slowing down the *GPU* computation. Tests performed on the implementation of the method prove that the method can effectively save the *GPU* from the burden of rendering the scene and that it does not introduce any noticeable overhead on the computation. On the contrary, the reduction of

computational requirements for graphics rendering produces a more regular physical simulation with benefits to the realism and the stability. Remote rendering can be enriched with advanced techniques since it does not slow down the simulation. The method has been used to provide stereo rendering of a virtual environment with deformable models, thus allowing the user to navigate into the scene in a more intuitive way.

Our simulator still lacks a proper self collision algorithm that will allow to make the cut parts of the model to interact with each other, and the realistic computation of the physics of the forces acting during the cut. We foresee to include a *GPU* based collision detection algorithm into the simulation that will handle both collision between virtual tools and models and self collision in a unified way.

8

Conclusions and Future Work

In this thesis we analyzed the problems related to the simulation of the physics of deformable environments for interactive applications with force feedback. As discussed in Chapter 2, physic simulation requires very small steps in the temporal integration phase, in addition the introduction of interactive force feedback imposes real time constraints to the simulation. For these reasons the computational speed of the simulation represents a key aspect of the whole research.

To cope with the reduced computational time available in the evaluation of one single step of the simulation the different parts of the simulation needs to be carefully evaluated. For this reason we deeply investigated the features of different soft tissue modeling techniques, from the point of view of the aspects of the deformation that they can represent but also from the point of view of the computational complexity they impose to the simulation.

The results of this analysis show that the evaluated models (*FEM* *MSM* and meshless) provide very different behaviors and offers different advantages. In particular *FEM* and *MSM* can be simulated at very reduced computational cost thus allowing the simulation of a greater number of physics elements for an increased realism. Point based models, on the other hand, are computationally mode expensive but they handle changes in topology with ease and speed. For this reason it is useful to include these three model classes into the virtual environment. The problem with this approach is that the three models have very different structures and that their integration with other parts of the simulation requires to write codes to adapt these structures.

To reduce the effort spent in integrating the different parts of the simulation we developed a common representation framework, described in Chapter 3, that provides a unified interface between the different deformable models and the other components. Our approach defines a surface that wraps the physic node of the models and that moves with them. This framework handles the distribution of forces and displacements between deformable models and the environment. In addition it provides a surface that embodies some physical properties. In fact this surface representation is successfully integrated with collision detection algorithm and with friction models. Presented results show the difference in the behavior of different modelization techniques, but they also prove the effectiveness of the proposed framework in handling the displacements. Images reported in the chapter

clearly show that the use of triangles in the definition of the model allows the rendering of smooth surface as well as sharp details. In addition, the precomputation of the interdependence between surface points and internal, physics, points, limits the computational overhead introduced into the simulation.

The subsequent step in physical simulations requires to detect the interactions between bodies in the scene. To this extent we decided to integrate in our work a collision detection library. In Chapter 4 we evaluated some of the state of the art algorithms that perform collision detection between rigid models and deformable models. These algorithms have been compared to the requirements of interactive simulations of deformable environments and we identified in V-collide [50] the most suitable to our needs. This library is not targeted to deformable models, but it correctly handles polygonal soups and thus it is suitable for collision detection between soft bodies. In addition it detects interferences between multiple bodies and returns for each pair of colliding bodies the list of overlapping triangles. The introduction of collision detection in the physics simulation greatly reduces the frame rate of the simulation. For this reason the chosen library, even if it satisfies the requirements for the realism, is not appropriate for our needs.

In Chapter 5 we compared some of the most common methods used in friction simulations. Except for the Coulomb and Karnopp models, that provide similar behaviors, the differences between other models are very important, also from the point of view of the computational time. The comparison between complexity of simulations allows us to split the models in two classes. The first class comprises Karnopp and Dahl models and identifies the models that are suitable to handle big number of collision in the scene. The second class, composed by LuGre and Elasto Plastic models, is most suitable in the simulation of complex behaviors, but only for simple scenes, where there is a reduced number of interacting points.

We avoid restricting the simulation to one particular friction model, instead we decided to provide a framework that allows the integration of different friction models in the environment. The framework is in charge of computing the variables involved in the contact: it computes the tangential and the normal forces that acts in the contact and the relative velocity of the surfaces interested by the collision. These values are used by friction model to compute the friction force. Then the framework distributes the friction force among contacting surface points. This framework proved to work well in association with the presented friction models but thanks to its generality it can also operate with different models, as long as they only requires the cited variables and at most one variable state.

Chapter 6 describes an innovative approach to the simulation of anisotropic tissues with mass spring models. To evaluate the correctness of the method we compare it with *FEM* that are considered as the ground truth in soft tissue simulation. Comparisons have been carried on for bidimensional models to simplify the evaluation. Results show a good match between the simulations obtained with *FEM* and the ones obtained with our approach. To further prove the correctness of the method we provided some screenshots of results obtained in the simulation of anisotropic bodies. The method is very fast and realistic in the simulation of transversally isotropic materials, that are the most common among biological tissues.

Chapter 7 provides the details of the implementation of a physically based simulator. It describes an interactive deformable environments with haptic feedback and complex user actions. Figure 8.1 provides an example of the interaction with the simulation. The method exploits *GPU* parallelism and computational power to ensure fast update times and provides the ability to grab, probe and cut deformable tissues. The use of the *GPU* imposes some limitations on the developed code: by arranging required informations in proper data structures and by decomposing the whole process in smaller, parallelizable tasks, we are able to increase the speed of the simulation and to obtain the required frame rate to give the user haptic feedback during the interaction with complex models. Our method allows the user to handle two virtual tools and to have force feedback on them both.

Fig. 8.1. The complete simulation setup: two haptic devices allow the user to interact with the virtual environment that is perceived through haptic feedback and with graphical rendering.

To further optimize the computation we defined an innovative remote rendering protocol that increases the realism of the physical simulation. With this method we save computational time and thus we increase the complexity of the models that can be simulated in real time. This, in turn, increases the simulator realism: in fact the physics realism increases as more physical elements are used. The overhead introduced by the update of model surface representation is transferred to a remote machine via the proposed remote graphical rendering method. The method efficiently transmits the virtual scene from the simulator to the viewer. As expected, the choice of UDP protocol ensures that the transmission performance is independent on the network configuration, as UDP provides no guarantees for

message delivery. This required us to develop an implicit error recovery protocol. The proposed method has been used to increase the realism of the scene by using stereo rendering techniques. In the near future it will be extended to handle multiple viewers and frictional contacts between the virtual tools and deformable models will be introduced. Remote rendering technique will be improved to reduce the effect of data transmission delay on the physical simulation.

Our work proved that with commercial hardware it is possible to simulate complex scenes with physical realism and to provide the user with both advanced graphical rendering and force feedback. We implemented a prototype of a surgical simulator that can be used to train surgeons to laparoscopic interventions. A first validation of the simulations have been obtained by developing a real synthetic model, and by modeling it in the virtual environment. Forces measured during the interaction with the real model were compared to interaction forces computed by the simulation and we obtain a good similarity. Moreover the simulation allows the user to perceive and locate some stiffer inclusions under the surface of the synthetic model.

The main limitation of the implemented simulator is represented by the lack of a complex collision detection phase. As observed in Chapter 4, collision detection phase is still computationally expensive, moreover we found no algorithm that performs collision detection between deformable models and runs on GPU. The main extension to the obtained simulator is thus the introduction of proper collision detection routines, which will allow the simulation of several deformable models in frictional contact. Another possible extension to the simulator is the integration of different soft tissue modeling techniques and the integration of more friction models to allow the simulation of lubricated contacts without the complexity of LuGre or Elasto Plastic models.

The current simulator can be extended with more friction models. Suitable friction models should provide realistic behavior at a reduced computational cost, to prevent slowing down the simulation. The introduction of friction model that depends on values different from the relative velocity of contacting point or the force exerted during the contact will require the extension of the collision solution method.

References

1. Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):3–15, 2003.
2. Ron Alterovitz, Kenneth Y. Goldberg, and Allison M. Okamura. Planning for steerable bevel-tip needle insertion through 2d soft tissue with obstacles. In *ICRA*, pages 1640–1645, 2005.
3. Brian Armstrong-Hlouvry. *Control of Machines with Friction*. Springer, 1st edition, 1991.
4. David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1998. ACM.
5. Jernej Barbič and Doug L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):982–990, August 2005.
6. Cagatay Basdogan, Mert Sedef, Matthias Harders, and Stefan Wesarg. Vr-based simulators for training in minimally invasive surgery. *IEEE Computer Graphics and Applications*, 27:54–66, 2007.
7. Klaus J. Bathe. *Finite Element Procedures*. Prentice Hall, 1995.
8. Debora Botturi, Francesca Pizzorni Ferrarese, Giulia Angela Zamboni, and Davide Zerbato. Preoperative workflow for lymph nodes staging. *Int J Comput Assist Radiol Surg*, 4(1):99–104, 2009.
9. David Bourguignon and Marie-Paule Cani. Controlling anisotropy in mass-spring systems. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, Springer Computer Science, pages 113–123. Springer-Verlag, aug 2000. Proceedings of the 11th Eurographics Workshop, Interlaken, Switzerland, August 21–22, 2000.
10. W. F. Brace and J. D. Byerlee. Stick-slip as a mechanism for earthquakes. *Science*, (3739):990 – 992, 1966.
11. Morten Bro-Nielsen. Surgery simulation using fast finite elements. In *VBC '96: Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, pages 529–534, London, UK, 1996. Springer-Verlag.
12. Joel Brown, Stephen Sorkin, Jean-Claude Latombe, Kevin Montgomery, and Michael Stephanides. Algorithmic tools for real-time microsurgery simulation. *Medical Image Analysis*, 6(3):289 – 300, 2002.
13. Marie-Paule Cani and Mathieu Desbrun. Animation of deformable models using implicit surfaces. *IEEE Trans. Vis. Comput. Graph.*, 3(1):39–50, 1997.

14. C. Canudas. A new model for control of systems with friction. *IEEE Trans. on Automatic Control*, 40(3):419–425, 1995.
15. J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Computer Graphics (SIGGRAPH 01 Conf. Proc.), pages 6776. ACM SIGGRAPH*, pages 67–76. Springer, 2001.
16. M. A. Padilla Castaneda and F. Arambula Cosio. Improved collision detection algorithm for soft tissue deformable models. In *ENC '05: Proceedings of the Sixth Mexican International Conference on Computer Science*, pages 41–49, Washington, DC, USA, 2005. IEEE Computer Society.
17. Kup-Sze Choi, Hanqiu Sun, and Pheng-Ann Heng. Interactive deformation of soft tissues with haptic feedback for medical learning. *Information Technology in Biomedicine, IEEE Transactions on*, 7(4):358–363, Dec. 2003.
18. Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):91–101, 2005.
19. Yoo-Joo Choi, Min Hong, Min-Hyung Choi, and Myoung-Hee Kim. Adaptive surface-deformable model with shape-preserving spring. *Comput. Animat. Virtual Worlds*, 16(1):69–83, 2005.
20. Robert D. Cook, David S. Malkus, Michael E. Plesha, and Robert J. Witt. *Concepts and Applications of Finite Element Analysis.* John Wiley & Sons, 2007.
21. J. L. Cornwall. Efficient multiple pass, multiple output algorithms on the gpu. In *2nd European Conference on Visual Media Production*, 2005.
22. Jr. D. A. Haessig and B. Friedland. On the modeling and simulation of friction. *Journal of Dynamic Systems, Measurement, and Control*, 113(3):354–362, 1991.
23. P. R. Dahl. A solid friction model. Technical report, The Aerospace Corporation, El Segundo, CA, 1968.
24. M. de Pascale, G. de Pascale, and D. Prattichizzo. Exploiting gpus for visuo-haptic modelling of deformable tissues. *BioRob 2006*, pages 467–470, 2006.
25. Gilles Debunne, Mathieu Desbrun, Alan H. Barr, and Marie-Paule Cani. Interactive multiresolution animation of deformable models. In Nadia Magnenat-Thalmann and Daniel Thalmann, editors, *Eurographics Workshop on Computer Animation and Simulation'99, September, 1999*, Computer Science, pages 133–144, Milan, Italie, September 1999. Springer.
26. Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. Dynamic real-time deformations using space & time adaptive sampling. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 31–36, New York, NY, USA, 2001. ACM.
27. Hervé Delingette, Stéphane Cotin, and Nicholas Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *CA '99: Proceedings of the Computer Animation*, page 70, Washington, DC, USA, 1999. IEEE Computer Society.
28. Vincent Duindam and Stefano Stramigioli. Modeling the kinematics and dynamics of compliant contact. In *IEEE International Conference on Robotics and Automation, ICRA*, volume 3, pages 4029–4034. IEEE, 2003.
29. Pierre Dupont, Brian Armstrong, and Vincent Hayward. Elasto-plastic friction model: Contact compliance and stiction. In *American Control Conference*, pages 1072–1077, 2000.
30. Nra Dyn, David Levin, and John A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9:160–169, 1990.

31. Peter Eisert and Philipp Fechteler. Remote rendering of computer games. In *Proceedings of the International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, July 2007.
32. Gerald Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann, 5 edition, November 2001.
33. C. Forest, H. Delingette, and N. Ayache. Cutting simulation of manifold volumetric meshes. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI02). Volume 2489 of LNCS*, pages 235–244. Springer, 2002.
34. C. Forest, H. Delingette, and N. Ayache. Removing tetrahedra from manifold tetrahedralisation : application to real-time surgical simulation. *Medical Image Analysis*, 9(2):113–122, April 2005.
35. Sarah F. Frisken-Gibson. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *IEEE Transactions on Visualization and Computer Graphics*, 5:333–348, 1999.
36. Antonio Frisoli, Luigi Borelli, and Massimo Bergamasco. Modeling biologic soft tissues for haptic feedback with an hybrid multiresolution method. *Medicine Meets Virtual Reality 13: The Magical Next Becomes the Medical Now*, 111:145–148, 2005.
37. Y.C. Fung. *Biomechanics Mechanical Properties of Living Tissues*. Springer, Berlin, 1993.
38. Nico Galoppo, Miguel A. Otaduy, Paul Mecklenburg, Markus Gross, and Ming C. Lin. Fast simulation of deformable models in contact using dynamic deformation textures. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 73–82, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.
39. Joachim Georgii, Florian Echtler, and Rüdiger Westermann. Interactive simulation of deformable bodies on GPUs. In *In proc. SimVis 05*, pages 247–258. SCS Publishing House e.V, 2005.
40. Thomas Di Giacomo and Nadia Magnenat-Thalmann. Bi-layered mass-spring model for fast deformations of flexible linear bodies. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*, page 48, Washington, DC, USA, 2003. IEEE Computer Society.
41. Sarah Gibson, Christina Fyock, Eric Grimson, Takeo Kanade, Rob Kikinis, Hugh Lauer, Neil McKenzie, Andrew Mor, Shin Nakajima, Hide Ohkami, Randy Osborne, Joseph Samosky, and Akira Sawada. Volumetric object modeling for surgical simulation. *Medical Image Analysis*, 2(2):121–132, 1998.
42. S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180, 1996.
43. Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection among deformable models using graphics processors. *Comput. Graph.*, 31(1):5–14, 2007.
44. Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph.*, 24(3):991–999, 2005.
45. Kris K. Hauser, Chen Shen, and James F. O'Brien. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, pages 247–256. CIPS, Canadian Human-Computer Commnication Society, A K Peters, June 2003.
46. M. Hauth and W. Strasser. Corotational simulation of deformable solids. In *WSCG*, pages 137–144, 2004.
47. Martin Held, James T. Klosowski, and Joseph S.B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *In Canadian Conference on Computational Geometry*, pages 205–210, 1995.

48. Gerd Hesina and Dieter Schmalstieg. A network architecture for remote rendering. Technical report, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, April 1998.
49. D. P. Hess and A. Soom. Friction at a lubricated line contact operating at oscillating sliding velocities. *Journal of Tribology*, 112:147–152, 1990.
50. Thomas C. Hudson, Ming C. Lin, Jonathan Cohen, Stefan Gottschalk, and Dinesh Manocha. V-collide: accelerated collision detection for vrml. In *VRML '97: Proceedings of the second symposium on Virtual reality modeling language*, pages 117–ff., New York, NY, USA, 1997. ACM.
51. M. Hughes, C. DiMattia, M. C. Lin, and D. Manocha. Efficient and accurate interference detection for polynomial deformation. In *CA '96: Proceedings of the Computer Animation*, page 155, Washington, DC, USA, 1996. IEEE Computer Society.
52. INRIA: institut national de recherche en informatique et en automatique. Sofa, an open source framework targeted at real-time simulation. `http://www.sofa-framework.org`.
53. Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Trans. Graph.*, 22(3):879–887, 2003.
54. Doug L. James and Dinesh K. Pai. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 582–585, New York, NY, USA, 2002. ACM.
55. V. I. Johannes, Green M. A., and Brockley C. A. The role of the rate of application of the tangential force in determining the static friction coefficient. *Wear*, 24:381–385, 1973.
56. Dean Karnopp. Computer simulation of stick-slip friction in mechanical dynamic systems. *Journal of Dynamic Systems, Measurement, and Control*, 107(1):100–103, 1985.
57. Autar K Kaw and Egwu Eric Kalu. *Numerical Methods with Applications*. http://www.autarkaw.com, 2008.
58. John Keyser, Shankar Krishnan, and Dinesh Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic. In *SMA '97: Proceedings of the fourth ACM symposium on Solid modeling and applications*, pages 42–55, New York, NY, USA, 1997. ACM.
59. Chang E. Kim and Judy M. Vance. Collision detection and part interaction modeling to facilitate immersive virtual assembly methods. *Journal of Computing and Information Science in Engineering*, 4(2):83–90, 2004.
60. Sang-Youn Kim, Jinah Park, and Dong-Soo Kwon. The real-time haptic simulation of a biomedical volumetric object with shape-retaining chain linked model. *IEICE - Trans. Inf. Syst.*, E88-D(5):1012–1020, 2005.
61. Rolf M. Koch, Markus H. Gross, Friedrich R. Carls, Daniel F. von Büren, George Fankhauser, and Yoav I. H. Parish. Simulating facial surgery using finite element models. *Computer Graphics*, 30(Annual Conference Series):421–428, 1996.
62. Rynson W.H. Lau, Oliver Chan, Mo Luk, and Frederick W.B. Li. Large a collision detection framework for deformable objects. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 113–120, New York, NY, USA, 2002. ACM.
63. Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Realistic modeling for facial animation. In *In SIGGRAPH 95*, pages 55–62, 1995.
64. Ming C. Lin. Fast and accurate collision detection for virtual environments. 1999.
65. Ming C. Lin and Stefan Gottschalk. Collision detection between geometric models: A survey. In *In Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

66. Bryn A. Lloyd, Gbor Szkely, and Matthias Harders. Identification of spring parameters for deformable object simulation. *IEEE Transactions on Visualization and Computer Graphics*, 13:1081–1094, 2007.
67. Jean-Christophe Lombardo, Marie-Paule Cani, and Fabrice Neyret. Real-time collision detection for virtual surgery. In *Computer Animation*, May 1999.
68. Müller M, Keiser R, Nealen A, Pauly M, Gross M, and Alexa M. Point-based animation of elastic, plastic, and melting objects. *ACM SIGGRAPH/Eurographics Symp. Computer Animation*, pages 141–151, 2004.
69. Pauly M, Keiser R, Adams B, Dutré P, Gross M, and Guibas LJ. Meshless animation of fracturing solids. *ACM Transactions Graphics*, 24(3):957–964, 2005.
70. M. Mahvash. Novel approach for modeling separation forces between deformable bodies. *IEEE Transactions on Information Technology in Biomedicine*, 10(3):618–626, 2006.
71. Nuno N. Maia and Julio M. M. Silva. *Theoretical and Experimental Modal Analysis*. Research Studies Press, 1998.
72. Maximo De Mero and Antonio Susin. Deformable 3d objects for a vr medical application. In *3es. Jornades de Recerca en Enginyeria Biomdica*, pages 264–269, 2002.
73. Maximo De Mero and Antonio Susin. Deformable hybrid model for haptic interaction. In *3rd. Workshop in Virtual Reality, Interactions, and Physical Simulations (VRIPHYS'06)*, pages 8–16, 2006.
74. Shinya Miyazaki, Mamoru Endo, Masashi Yamada, Junichi Hasegawa, Takami Yasuda, and Shigeki Yokoi. A deformable fast computation elastic model based on element reduction and reconstruction. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds*, pages 94–99, Washington, DC, USA, 2004. IEEE Computer Society.
75. M.B. Mohr, L. Blumcke, F.B. Sachse, G. Seemann, and O. Dossei. Hybrid deformation model of myocardium. In *Computers in Cardiology, 2003*, pages 319–322, Sept. 2003.
76. Wouter Mollemans, Filip Schutyser, Johan Van Cleynenbreugel, and Paul Suetens. Tetrahedral mass spring model for fast soft tissue deformation. In *IS4TH*, pages 145–154, 2003.
77. Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*, 2(1):21–28, 1997.
78. Andrew Mor. *Progressive Cutting with Minimal New Element Creation of Soft Tissue Models for Interactive Surgical Simulation*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2001.
79. M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 141–151, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.
80. Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54, New York, NY, USA, 2002. ACM.
81. Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, 2005.
82. Ken Museth, David E. Breen, Ross T. Whitaker, and Alan H. Barr. Level set surface editing operators. In *ACM TRANSACTIONS ON GRAPHICS*, pages 330–338, 2002.

83. Matthieu Nesme, Maud Marchal, Emmanuel Promayon, Matthieu Chabanas, Yohan Payan, and François Faure. Physically realistic interactive simulation for biological soft tissues. In *Recent Research Developments in Biomechanics, 2005*, volume 2 of *Transworld Research Network*, pages 117–139. Research signpost, 2005.
84. Fries T P and Matties H G. Classification and overview of meshfree methods. In *Technical Report*, volume 3. TU Brunswick, Germany, 2003.
85. Celine Paloc, Alessandro Faraci, and Fernando Bello. Online remeshing for soft tissue simulation in surgical training. *IEEE Comput. Graph. Appl.*, 26(6):24–34, 2006.
86. A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *SIGGRAPH Comput. Graph.*, 23(3):207–214, 1989.
87. G. Picinbono, H. Delingette, and N. Ayache. Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1370–1375 vol.2, 2001.
88. Guillaume Picinbono, Jean Christophe Lombardo, Herv Delingette, and Nicholas Ayache. Improving realism of a surgery simulator: linear anisotropic elasticity, complex interactions and force extrapolation. *Journal of Visualization and Computer Animation*, 13:147–167, 2002.
89. William Press, Saul Teukolsky, William Vetterling, and Brian Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.
90. S. Quinlan. Efficient distance computation between non-convex objects. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3324–3329 vol.4, May 1994.
91. Alec R. Rivers and Doug L. James. Fastlsm: fast lattice shape matching for robust real-time deformation. *ACM Trans. Graph.*, 26(3):82, 2007.
92. Li Shaofan and Liu Wing Kam. Meshfree particle methods and their applications. *Applied Mechanics Review*, 54:1–34, 2002.
93. Eftychios Sifakis, Kevin G. Der, and Ronald Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 73–80, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.
94. Russel Smith. Ode, open dynamics engine. `http://www.ode.org`.
95. T. Soresen and J. Mosegaard. Haptic feedback for the GPU-based surgical simulator. *In Medicine Meets Virtual Reality 14*, pages 523–528, 2006.
96. Simon Stegmaier, Marcelo Magallón, and Thomas Ertl. A generic solution for hardware-accelerated remote visualization. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 87–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.
97. R. Stribeck. Die wesentlichen eigenschaften der gleit- und rollenlager (the key qualities of sliding and roller bearings). *Zeitschrift des Vereines Seutscher Ingenieure*, 46(38,39):1342–1348, 1432–1437, 1902.
98. Chuan-Jun Su, Fuhua Lin, and Lan Ye. A new collision detection method for csg-represented objects in virtual manufacturing. *Comput. Ind.*, 40(1):1–13, 1999.
99. Belytschko T, Krongauz Y, Organ D, Fleming M, and Krysl P. Meshless methods: an overview and recent developments. *Computation Methods Application Mechanical Engineering*, 139:3–47, 1996.
100. Belytschko T, Lu Y Y, and Gu L. Element free galerkin methods. *International Journal for Numerical Methods in Engineering*, 37:229–256, 1994.
101. Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. Interactive continuous collision detection between deformable models using connectivity-based culling. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 25–36, New York, NY, USA, 2008. ACM.

102. Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21, pages 205–214, New York, NY, USA, July 1987. ACM Press.
103. Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.
104. Gino Van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *J. Graph. Tools*, 4(2):7–25, 1999.
105. Allen Van Gelder. Approximate simulation of elastic membranes by triangulated spring meshes. *J. Graph. Tools*, 3(2):21–42, 1998.
106. Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159(1):98, Jul 1967.
107. Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. Geometric collisions for time-dependent parametric surfaces. *SIGGRAPH Comput. Graph.*, 24(4):39–48, 1990.
108. Clemens Wagner, Markus A. Schill, and Reinhard Männer. Collision detection and tissue modeling in a vr-simulator for eye surgery. In *EGVE '02: Proceedings of the workshop on Virtual environments 2002*, pages 27–36, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.
109. Feei Wang, Terril Hurst, Daniel Abramovitch, and Gene Franklin. Disk drive pivot nonlinearity modeling part ii: Time domain, 1994.
110. J.A Weiss, B.N. Maker, and Govindjee S. Finite element implementation of incompressible, transversely isotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering*, 135(1):107–128, August 1996.
111. Wingo Sai-Keung Wong and George Baciu. Robust continuous collision detection for interactive deformable surfaces: Research articles. *Comput. Animat. Virtual Worlds*, 18(3):179–192, 2007.
112. Wen Wu and Pheng Ann Heng. A hybrid condensed finite element model with gpu acceleration for interactive 3d soft tissue cutting: Research articles. *Comput. Animat. Virtual Worlds*, 15(3-4):219–227, 2004.
113. Wen Wu and Pheng-Ann Heng. An improved scheme of an interactive finite element model for 3D soft-tissue cutting and deformation. *The Visual Computer*, 21(8-10):707–716, 2005.
114. Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Computer Graphics Forum*, pages 349–358, 2001.
115. Guo X and Qin H. Point-based dynamic deformation and crack propagation. In *Technical Report*. Stony Brook University, 2004.
116. Soji Yamakawa and Kenji Shimada. Anisotropic tetrahedral meshing via bubble packing and advancing front. *International Journal for Numerical Methods in Engineering*, 57(13):1923–1942, 2003.
117. Che Yinghui, Wang Jing, and Liang Xiaohui. Real-time deformation using modal analysis on graphics hardware. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 173–176, New York, NY, USA, 2006. ACM.
118. GP you Groupe. Gpumat: Gpu toolbox for matlab. `http://www.gp-you.org`.
119. D. Zerbato, S. Galvan, and P. Fiorini. Calibration of mass spring models for organ simulations. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 29 2007-Nov. 2 2007.
120. Liangjun Zhang, Young J. Kim, and Dinesh Manocha. A fast and practical algorithm for generalized penetration depth computation. In *Robotics: Science and Systems Conference (RSS07*, 2007.

121. Xinyu Zhang and Young J. Kim. Interactive collision detection for deformable models using streaming aabbs. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):318–329, 2007.
122. Deng-Hua Zhong, Ming-Chao Li, Ling-Guang Song, and Gang Wang. Enhanced nurbs modeling and visualization for large 3d geoengineering applications: An example from the jinping first-level hydropower engineering project, china. *Comput. Geosci.*, 32(9):1270–1282, 2006.
123. Yongmin Zhong, B. Shirinzadeh, G. Alici, and J. Smith. A new methodology for deformable object simulation. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1902–1907, April 2005.

Curriculum Vitæ of Davide Zerbato

Davide Zerbato was born in Verona on the 30th of March 1979. He received a master's degree in Computer Science from University of Verona in March 2006. In his thesis he addressed the problem of deformable model calibration. He obtained his PhD in Computer Science at University of Verona with a thesis about the simulation of frictional contact in interactive deformable environments.

Since 2006 he is a member of ALTAIR Laboratory (Verona) where he worked for AccuRobAs European project and he is currently involved in SAFROS European project, the focus of his work is the implementation of a surgical simulator with haptic feedback based on general purpose graphics processing unit programming. The focus of his work is the development of highly parallel algorithms for the simulation of interactive environments with haptic feedback.

His expertise includes deformable tissues modeling and haptic simulations.

Education

2007 - 2009: **Dottorato di ricerca in Informatica, Università degli Studi di Verona** with the thesis "Frictional Contact in Interactive Deformable Environments" on soft bodies frictional contact simulation in virtual interactive environments with haptic feedback.

2004 - 2006: **Laurea specialistica in Sistemi intelligenti e multimediali, Università degli Studi di Verona** with a thesis entitled "Deformable models calibratiion for organ simulation", about a method to calibrate mass spring model in an automatic way to obtain patient-specific organ models (graded 110/110 *cum laude*).

2001 - 2003: **Laurea Triennale in Tecnologie dell'informazione: multimedia, Università degli Studi di Verona** with a thesis entitled "Multiresolution surface parametrization", about a procedure to parametrize meshes that allow to simplify or modify them in an easy and intuitive way (graded 109/110).

1993 - 1998: **Diploma di Maturità Scientifica: Istituto "Alle Stimate", Verona** (graded 54/60).

Work Experience

2008 & 2009:	**University of Verona, teaching assistant.** Provided instructional support for the courses "System and signals" held by Prof. Paolo Fiorini.
Since April 2006:	**Altair, robotic Laboratory of University of Verona, internship.** Development of a prototype of a surgical simulator with haptic feedback: graphical interface and deformable models.
Winter 2002:	**Intercomp, Verona, technician**. Technical assistance and stock list for ULSS 20, Legnago.
Summer 2001:	**Banca Popolare di Verona BSGSP, Verona, bank teller.**

Publications and Patents

1. D. Zerbato, D. Baschirotto, D. Baschirotto, D. Botturi, P. Fiorini GPU based physical cut in interactive haptic simulations, accepted to Int. J CARS
2. D. Zerbato, D. Baschirotto, D. Baschirotto, D. Botturi, P. Fiorini GPU based physical cut in interactive haptic simulations, CARS 2010
3. D. Zerbato, D. DallAlba, L. Giona, M. Vicentini, D. Botturi, P. Fiorini Enhancing maxilofacial implantology with virtual fixtures, CARS 2010
4. D. Dall'Alba, L. Giona, D. Zerbato, D. Botturi, P. Fiorini, G. Schiroli, Image based accuracy analysis in dental implantology applications. CARS 2010
5. D. Botturi, F. Pizzorni Ferrarese, G.A. Zamboni, D. Zerbato, Preoperative workflow for lymph nodes staging, Int. J CARS (2008)
6. D. Botturi, F. Pizzorni Ferrarese, D. Zerbato, G.A. Zamboni, Automatically Segmented CT Model for Preoperative Lymph Nodes Characterization and Staging, RSNA 2008
7. M. Altomonte, D. Zerbato, D. Botturi, Fiorini, Simulation of Deformable Environment with Haptic Feedback on GPU, IROS 2008
8. F. Pizzorni Ferrarese, D. Botturi, G.A. Zamboni, D. Zerbato, Preoperative non-invasive staging of lymph nodes, CARS 2008
9. D. Zerbato, S. Galvan, P. Fiorini, Calibration of Mass Spring Models for Organ Simulation, IROS 2007
10. D. Zerbato Metodo per la simulazione interattiva di immagini (Method for the interactive simulation of images), domanda di brevetto.

Sommario

L'uso di simulazioni garantisce notevoli vantaggi in termini di economia, realismo e di flessibilità in molte aree di ricerca e in ambito dello sviluppo tecnologico. Per questo motivo le simulazioni vengono usate spesso in ambiti quali la prototipazione di parti meccaniche, nella pianificazione e nell'addestramento di procedure di assemblaggio e disassemblaggio inoltre, recentemente, le simulazioni si sono dimostrate dei validi strumenti anche nell'assistenza e nell'addestramento ai chirurghi. Ciò risulta particolarmente vero nel caso della chirurgia laparoscopica.

La chirurgia laparoscopica, infatti, viene considerata lo standard per molte procedure chirurgiche. La principale differenza rispetto alla chirurgia tradizionale risiede nella notevole limitazione che ha il chirurgo nell'interagire e nel percepire l'ambiente in cui sta lavorando, sia nell'aspetto visivo che in quello tattile. Questo rappresenta una forte limitazione per il chirurgo a cui è richiesta una lunga fase di addestramento prima di poter ottenere la necessaria destrezza per intervenire in laparoscopia con profitto. Queste limitazioni, d'altra parte, rendono la laparoscopia il candidato ideale per l'introduzione della simulazione nell'addestramento.

Attualmente sono disponibili in commercio alcuni software per l'addestramento alla chirurgia laparoscopica, tuttavia essi sono in genere basati su modelli rigidi, o modelli che comunque mancano del necessario realismo fisico. L'introduzione di modelli deformabili porterebbe a migliorare notevolmente l'accuratezza e simulazioni più realistiche. Nel caso dell'addestramento alla laparoscopia il maggior realismo potrebbe permettere all'utente di acquisire non solo le conoscenze motorie basilari ma anche capacità e conoscenze di più alto livello. I corpi rigidi, infatti, rappresentano una buona approssimazione della realtà solo in situazioni particolari ed entro intervalli di sollecitazioni molto ristretti. Quando si considerano materiali non ingegneristici, come accade nelle simulazioni chirurgiche, le deformazioni non possono essere trascurate senza compromettere irrimediabilmente il realismo dei risultati della simulazione. L'uso di modelli deformabili tuttavia introduce notevole complessità computazionale per il calcolo della fisica che regola le deformazioni e limita fortemente l'uso di dati precalcolati, spesso utilizzati per velocizzare la fase di identificazione delle collisioni tra i corpi. I ritardi dovuti all'uso di modelli deformabili rappresentano un grosso limite soprattutto nelle applicazioni interattive che, per consentire all'utente di interagire con l'ambiente, richiedono il calcolo della simulazione entro intervalli di tempo molto ridotti.

In questa tesi viene affrontato il tema della simulazione di ambienti interattivi composti da corpi deformabili che interagiscono con attrito. Vengono analizzati e sviluppati differenti tecniche e metodi per le diverse componenti della simulazione: in particolare la simulazione di modelli deformabili, gli algoritmi di identificazione e soluzione delle collisioni e la modellazione e l'integrazione degli effetti dell'attrito nella simulazione.

In particolare vengono valutati i principali metodi che rappresentano lo stato dell'arte nella modellazione di materiali deformabili. L'analisi considera i fondamenti fisici su cui i modelli si basano e quindi sul grado di realismo che possono garantire in termini di deformazioni modellabili e la semplicità d'uso degli stessi (ovvero la facilità di comprensione del metodo, la calibrazione del modello e la possibilità di adattare il modello a situazioni differenti) ma viene considerata anche la complessità computazionale di ciascun metodo in quanto essa rappresenta un fattore estremamente importante nella scelta e nell'uso dei modelli deformabili nelle simulazioni.

Il confronto dei differenti modelli e le caratteristiche identificate hanno motivato lo sviluppo di un metodo innovativo per fornire un'interfaccia comune ai vari metodi di simulazione dei materiali deformabili. Tale interfaccia ha il vantaggio di fornire dei metodi omogenei per la manipolazione di tutti i differenti modelli deformabili e ciò garantisce la possibilità di scambiare il modello usato per la simulazione delle deformazioni mantenendo inalterati le altre strutture dati e i metodi della simulazione.

L'introduzione di tale interfaccia unificata si dimostra particolarmente vantaggiosa in quanto permette l'uso di un solo metodo per l'identificazione delle collisioni su tutti i differenti modelli deformabili. Ciò semplifica molto l'analisi e la definizione dei requisiti di tale modulo software. L'identificazione delle collisioni tra modelli rigidi generalmente pre computa delle partizioni dello spazio in cui i corpi sono definiti oppure sfrutta la suddivisione del corpo analizzato in parti convesse, per velocizzare i calcoli durante la simulazione. Nel caso di modelli deformabili non è possibile applicare tali tecniche a causa dei continui cambiamenti nella configurazione dei corpi.

Dopo che le collisioni tra i corpi sono state riconosciute e che i punti di contatto sono stati identificati e necessario risolvere le collisioni tenendo conto della fisica sottostante i contatti. Per garantire il realismo fisico è necessario assicurare che i corpi non si compenetrino mai durante la simulazione e che nella simulazione delle collisioni tutti i fenomeni fisici di interesse coinvolti nel contatto tra i corpi vengano considerati: questi includono le forze elastiche che si esercitano tra i corpi e le forze di attrito che si generano lungo le superfici di contatto. L'innovativo metodo proposto per la soluzione delle collisioni garantisce il realismo della simulazione e l'integrazione con l'interfaccia proposta per la gestione unificata dei modelli.

Una caratteristica importante dei tessuti biologici è il comportamento anisotropico, dovuto, in genere, alla loro struttura fibrosa. In questa tesi viene proposto un nuovo metodo per aggiungere l'anisotropia al comportamento dei modelli massa molla. Il metodo ha il vantaggio di mantenere la velocità computazionale e la semplicità di implementazione dei modelli massa molla classici e riesce a differenziare efficacemente la risposta del modello alle sollecitazioni lungo le differenti direzioni.

Le tecniche descritte sono state integrate in due applicazioni che forniscono la simulazione della fisica di ambienti con corpi deformabili. La prima delle due implementa tutti i metodi descritti per la simulazione dei modelli deformabili, identifica le collisioni con precisione e le risolve fornendo la possibilità di scegliere il modello di attrito più adatto, dimostrando così la fattibilità dell'approccio proposto. La limitazione principale di tale simulatore risiede nell'alto tempo di calcolo richiesto per la simulazione dei singoli passi di simulazione. Tale limitazione è stata superata in una seconda implementazione che sfrutta il parallelismo intrinseco delle simulazioni fisiche per ottimizzare gli algoritmi e che, quindi, riesce a sfruttare al meglio la potenza computazionale delle architetture hardware parallele. Al fine di ottenere le prestazioni richieste per la simulazione di ambienti interattivi con ritorno di forza, la simulazione è basata su un algoritmo di identificazione delle collisioni semplificato, ma implementa gli altri metodi descritti in questa tesi. L'implementazione parallela sfrutta le capacità di calcolo delle moderne schede video munite di processori altamente paralleli e ciò permette di aggiornare la scena ogni millisecondo. Questo elimina ogni discontinuità nel ritorno di forza reso all'utente e nell'aggiornamento della grafica della scena, inoltre garantisce il realismo necessario alla simulazione fisica sottostante.

Le applicazioni implementate provano la fattibilità della simulazione della fisica di interazioni complesse tra corpi deformabili. Inoltre, l'implementazione parallela della simulazione rappresenta un promettente punto di partenza per la realizzazione di simulazioni interattive che potrà essere utilizzato in ambiti di ricerca differenti, quali l'addestramento di chirurghi o la prototipazione rapida.