THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE
Spécialité : **Mathématiques-Informatique**

Arrêté ministérial : 7 Août 2006

Présentée par

Pierre-Luc Manteaux

Thèse dirigée par **François Faure**
et codirigée par **Marie-Paule Cani**

préparée au sein **Laboratoire Jean Kuntzmann (LJK)**
et de **EDMSTII**

Simulation et contrôle de phénomènes physiques

Thèse soutenue publiquement le ,
devant le jury composé de :

Joëlle Thollot
Professeur, Université de Grenoble-INP, Présidente
Maud Marchal
Maître de conférences, INSA Rennes, Rapporteur
Christian Duriez
Directeur de recherche, INRIA Lille, Rapporteur
Florence Zara
Maître de conférences, Université de Lyon 1, Examinatrice
Paul G. Kry
Maître de conférences, McGill University, Examinateur
François Faure
Professeur, Université de Grenoble, Directeur de thèse
Marie-Paule Cani
Professeur, Université de Grenoble-INP, Co-Directeur de thèse

Remerciements

Je remercie tout d'abord François Faure et Marie-Paule Cani pour m'avoir donné la chance de réaliser ce doctorat, pour m'avoir encouragé tout au long de ces quatre ans et avoir partagé avec autant d'enthousiasme leur passion pour la recherche. Je remercie François pour m'avoir tant appris sur la simulation physique et pour m'avoir aidé à détricoter méticuleusement tant de questions scientifiques. Je remercie Marie-Paule pour sa joie de vivre à chacune de nos réunions, ses formidables intuitions et la générosité avec laquelle elle transmet ses connaissances, ses idées et ses conseils.

J'ai eu la chance de collaborer avec de nombreuses personnes d'horizons très différents pendant les projets de cette thèse. Merci à Stéphane Redon pour sa patience et son implication sur mon premier projet de recherche et sur la rédaction au long cours de l'état de l'art sur les méthodes adaptatives. Merci à Weilun Sun et James O'Brien pour ces trois superbes mois passés à Berkeley, c'est certainement l'expérience de recherche la plus forte que j'aurais connue pendant cette thèse. Merci à Paul Kry pour son dynamisme, son retour si pertinent sur mes travaux et son aide inestimable sur mes fautes d'anglais. Merci aussi à Rahul Narain et Chris Wojtan pour les nombreuses discussions autour des méthodes adaptatives. Je tiens également à remercier Chris pour m'avoir accueilli à l'IST pendant trois semaines, d'avoir été aussi ouvert et attentif sur toutes les pistes que l'on envisageait pour le contrôle de liquide. J'ai été très impressionné par le dynamisme qu'il insuffle à son équipe de recherche, ce fut un plaisir de partager leur quotidien. Merci à Ulysse Vimont, pour son petit grain de folie et toutes ses heures passées à faire avancer notre projet de sculpture de liquides. Je n'aurais pas pu le réaliser sans toi et le passage à l'IST aurait été très différent. Enfin, j'ai été très heureux que l'on puisse confronter autant d'idées toujours dans un esprit de collaboration.

La thèse m'a permis de découvrir le plaisir d'enseigner. Je souhaiterais remercier tous les étudiants que j'ai rencontrés au cours de ces années. Ils m'ont permis de trouver un équilibre quand la thèse n'avançait pas, de découvrir d'autres pistes de recherche et de garder le lien avec ces années que j'ai passées à l'ENSIMAG. Je tiens tout spécialement à remercier les groupes de projet de spécialité, c'est toujours une expérience forte de suivre un petit groupe d'étudiants pendant quatre semaines. Merci en particulier à Thibault Lejemble,

Amélie Fondevilla, Thibault Blanc-Beyne et Nicolas Durin, qui ont su pousser leur projet jusqu'à obtenir une publication dans une conférence internationale. Merci également à Mickaël Ly, Ilyes Kacher, Mathieu Stoffel et Maxence Hammen pour leur motivation sans failles, leur bonne humeur et pour avoir su garder le contact une fois le projet terminé. Merci à Camille Schreck pour sa combinaison très personnelle de l'humour et du sérieux et son aide dans l'encadrement d'un projet de spécialité. Merci à Thomas Delame pour son implication et sa détermination à donner un nouveau souffle aux TP du cours de Graphique 3D.

Merci à tous les membres de IMAGINE et MAVERICK avec qui j'ai partagé tant de bons moments à discuter, troller, jouer au baby-foot ou au ping-pong. Merci à Léo pour sa présence indéfectible que ce soit pour des questions de mathématiques, des relectures d'articles, du soutien moral ou des bières : je n'imagine même pas comment ce serait passé ces quatre ans sans toi. Merci à Hugo pour sa folie et son incroyable capacité à rendre réel ce que l'on n'ose imaginer. Je pense que je me souviendrais encore très longtemps de notre rapide passage aux studios Disney de Los Angeles et à ces quelques minutes passées à regarder des rushs avec une des légendes de l'animation. Rien de tout cela ne serait arrivé sans toi. Merci à Quentin et Ali pour tous ses bons moments passés pendant notre viré en Californie après SIGGRAPH. Merci à Damien pour sa sagesse dont j'ai tant aimé me moquer mais sans laquelle je me serais senti bien démuni. Merci à Antoine pour sa bonne humeur et son écoute, plus d'une fois ça m'a permis de décompresser et de repartir à l'attaque. Merci à Matthieu, Romain et Thomas pour toutes les conversations Sofaïenne.

Merci aux membres du jury pour leur relecture du manuscrit et leur implication tout au long de la soutenance.

Je remercie enfin mes amis et ma famille pour m'avoir supporté et encouragé pendant ces quatre ans. Mon épouse, Nassima, pour avoir été à mes côtés à chaque instant, qu'il soit bon ou mauvais. Mes parents, Marie-Hélène et Gabriel, et mes sœurs, Anne-Elisabeth et Claire-Lucie, pour leur soutien et toutes ses bouffées d'air qui m'ont aidé à poursuivre ce doctorat.

Résumé

Simulation et contrôle de phénomènes physiques

En informatique graphique les phénomènes physiques simulés pour la création d'animations, de jeux vidéos ou la conception d'objets sont de plus en plus complexes : tout d'abord en terme de coût de calcul, l'échelle des simulations étant de plus en plus importante ; ensuite en terme de complexité des phénomènes eux-mêmes qui requièrent des modèles permettant de changer d'état et de forme. Cette complexité grandissante introduit de nouveaux défis quand il s'agit d'offrir à un utilisateur un contrôle sur ces simulations à grande échelle. Dans de nombreux cas, ce contrôle est réduit à un cycle d'essais et d'erreurs pour déterminer les paramètres de la simulation qui satisferont au mieux les objectifs de l'utilisateur.

Dans cette thèse, nous proposons trois techniques pour répondre en partie à ces défis. Tout d'abord nous introduisons un nouveau modèle adaptatif permettant de réduire le temps de calcul dans des simulations Lagrangiennes de particules. À l'inverse des méthodes de ré-échantillonnage, le nombre de degrés de liberté reste constant au cours de la simulation. La méthode est ainsi plus simple à intégrer dans un simulateur existant et la charge mémoire est constante, ce qui peut être un avantage dans un contexte interactif. Ensuite, nous proposons un algorithme permettant de réaliser la découpe détaillée d'objets fins et déformables. Notre méthode s'appuie sur une mise à jour dynamique des fonctions de forme associées à chaque degré de liberté, permettant ainsi de conserver un nombre de degrés de liberté très faible tout en réalisant des changements topologiques détaillés. Enfin, nous nous intéressons au contrôle d'animations de liquide en s'inspirant des méthodes d'édition interactive de formes en modélisation 3D. Dans ce système, l'utilisateur travaille directement avec le résultat d'une simulation, c'est-à-dire une suite de maillages représentant la surface du liquide. Des outils de sélection et d'édition spatio-temporelle inspirés des logiciels de sculpture de formes statiques lui sont proposés.

Abstract

Simulation and control of physical phenomena

In computer graphics, the physical phenomena simulated for the creation of animations, video games or the design of objects are more and more complex: First, in terms of the computational cost, the scale of the simulations can be extremely important; Then, in terms of the complexity of the phenomena themselves, which require the models to be able to change their state and shape. This growing complexity introduces new challenges in order to offer control on these large scale simulations to the user. In many cases, this control is reduced to a trial-and-error process in order to determine the parameters of the simulation which best meet the objectives of the user.

In this thesis, we propose three techniques to tackle these challenges. First, we introduce a new adaptive model which allows the reduction of the computational cost in Lagrangian simulations of particles. In contrast with re-sampling strategies, the number of degrees of freedom remains constant throughout the simulation. Therefore, the method is simpler to integrate into an existing simulator and the memory consumption remains constant, which can be an advantage in an interactive context. Then, we propose an algorithm which allows the detailed cutting of thin deformable objects. Our method relies on a dynamic update of the shape functions associated to the degrees of freedom, which therefore allows the use of a very low number of degrees of freedom while performing detailed topological changes. Finally, we focus on the control of animations of liquid and take inspiration from interactive methods of shape editing in the field of 3D modeling. We introduce a system where the user directly edits the result of the simulation: a sequence of meshes representing the surface of the liquid. We propose selection and editing spatio-temporal tools inspired from static shape sculpting software.

Contents

Chapter 1

Introduction

1.1 A short story of physics-based animation

In 1937, the Walt Disney company presented its first full length animated film, *Snow white and the seven dwarfs*. Based on traditional animation techniques, every single image of the movie was drawn by hand. Years and years of learning and expertise were necessary to the animators to tackle this tremendous amount of work. The movie was a success and others followed still involving more complex animations. Among them, the animation of natural phenomena was certainly one of the most difficult to achieve due to its visual complexity. Specialized artists would draw each frame of smoke, water or dust animations.

In the middle of the eighties, the Pixar company started to present short computer generated movies at the SIGGRAPH conference. Even though *The adventures of André and Wally B.* was a success in 1984, it is *Luxo Jr.* in 1986 that showed that computer generated movies could compete with traditional animations in conveying emotions. For this short movie, the pipeline of traditional animation had been adapted to computers. Drawing was replaced by 3D modeling. Animators defined key-framed positions that the computer would interpolate to produce the in-betweens. Finally, the computer would also compute the shading of the scene. Modeling, animation and rendering were presented as the pillars of computer graphics movies.

In this context, physics appeared as a way to automatically generate the animation of complex natural phenomena. It would be able to handle the high level of details expected from large scale smoke clouds, the complex deformations arising from a twisting rope or the numerous interactions occurring between colliding objects. Researchers focused on studying physics with respect to computer graphics' purposes. Aside from films, physics-based animation was also developed for games, medical applications, education and craft prototyping.

1.2 Classical mechanics & physics-based animation

Physics-based animation has a great big brother: Classical mechanics. Classical mechanics group centuries of studies of how objects behave in the real world. This experience was invaluable and certainly allowed physics-based animation to progress extremely fast. The great strength of physics-based animation is not being restrained by this heritage but being able to inspire from it. The variety of applications resulting from physics-based animation and the interaction with a user created a wide number of research prospects.

As to know if the relationship between classical mechanics and physics-based animation is unilateral, the answer is definitely no. Physics-based animation is not doomed to simply transpose advances in classical mechanics to computer graphics. Neither it is only a subset of classical mechanics that cheats to get fast, inaccurate but compelling results. First of all, the relationship between the two fields is surely bilateral. Classical mechanics inspires physics-based animation and physics-based animation inspires classical mechanics. This is confirmed by the growing number of works from physics-based animation which are published in physics and mechanics conferences and vice versa. For instance, our work on adaptive particle simulation [Man+13] was inspired by the work of Artemova and Redon [AR12] which was published in the *Physical Review Letter* journal. We could also cite the work Qiu et al. [QLF16] on moving cartesian grids which was published in *Journal of Computational Physics* and was inspired by the work of English et al. [Eng+13] published at *SIGGRAPH*. Second, physics-based animation interacts more and more with other fields such as machine learning or biology. This increasing interdisciplinary brings new problems that may not have been investigated by classical mechanics.

1.3 Three challenges in physics-based animation

1.3.1 Adaptive physics-based animation

One could say that it is only a matter of time before computers be powerful enough to make intractable simulations run in real-time. Until now, this prediction has always failed because of the growth of the required level of details in physics-based animation. Finding the right model to describe a phenomena is not sufficient, both efficient and accurate simulations are needed.

Several approaches have been proposed to reduce the computational time, thus transforming off-line simulations into real-time ones and making intractable simulations possible to compute. Among these approaches, we can mention the use of reduced models, boundary only simulations and adaptive techniques. Reduced models consist of precomputing a carefully chosen small subset of degrees of freedom of the simulated object and running the simulation only on this subset. Thus the computational cost can be reduced by several orders of magnitude. This approach, however, imposes constraints on the range of phenomena that can be simulated, for instance topological changes are not handled. Boundary only simulations consist of simulating volumetric objects only via their surfaces. Finally, adaptive techniques propose to adapt in space and time the representation of the deformable model in order to find the best match between efficiency and accuracy.

In this manuscript we focus on this last method and present an extensive study of adaptive models for physics-based animation in Chapter 3.

Adaptive techniques have a great history in classical mechanics and computer graphics. The most common form of adaptivity consists of re-sampling a new set of degrees of freedom along the simulation in order to concentrate computational time where and when it is most needed based on accuracy and visual criteria. Among the most common criticisms concerning adaptive techniques, we retained two of them. First, adaptive techniques are notoriously hard to integrate in existing simulation frameworks: tey may require to *bend* the technique to the framework or vice-versa. Even if they can bring great speed ups, the amount of implementation work makes them unattractive. Therefore, there is a great need for more general approach of adaptivity producing less intrusive techniques. Second, adaptive methods are often subject to popping artifacts when updating the resolution of the simulation. Especially in computer graphics, these artifacts should be prevented.

In Chapter 4, we propose a new adaptive model to address these challenges.

1.3.2 Detailed topological changes

Complex simulations are often characterized by objects undergoing strong changes of state and shapes. Material can flow, break and deform irreversibly. This requires a robust handling of topological changes.

In the context of cutting and fracture, handling detailed topological changes interactively remains a challenge. Games and surgical simulators need a detailed and efficient representation of these topological changes. Because of restrictions on the amount of memory and computational time for the simulation, they cannot always afford adaptive techniques to handle the increasing computational cost due to the changes.

Aside from the computational challenge, the discontinuities induced by topological changes remain hard to faithfully represent. However, they are a key ingredient to predict how an object will react under cutting or tearing. This is particularly important when the user interaction needs to be taken into account precisely, such as in surgical simulators.

Our second contribution, presented in Chapter 5 tackles this problem.

1.3.3 Simulation control

In computer graphics, controlling a simulation is of great importance in order to meet artistic choices when designing an animation. Aside from the animation field, prototyping is also in needs of techniques to control mechanical systems so that it fulfills an objective.

Unfortunately, controlling a simulation to match specific goals is a hard problem. Simulations are formulated as an initial value problem, meaning that the whole behavior is determined by the parameters of the simulation such as initial and boundary conditions or material parameters. Finding the right set of parameters is often the result of a tedious trial and error process.

In contrast with animation, there is still no strong pipeline that leave a user design a physics-based animation from scratch. The main challenge remains to build high level control tools that would help the user to intuitively design animations or prototypes. In the case of prototyping, these tools should produce viable objects that respect the underlying mechanics. In the case of animation, they should provide a way to adjust between what makes the animation looks realistic and the artistic constraints which convey emotions.

Our last contribution, presented in Chapter 6 falls in this domain.

1.4 Contributions

The contributions of this work are as follows:

- First, we transfer and extend a new adaptive physically-based animation technique from nanosystems simulation to computer graphics. In contrast with classical methods, our technique is less intrusive, it requires minimal changes in an existing simulator and retains physical accuracy. We demonstrate its use in the case of particle-based fluid simulation and extend it by proposing an implicit integrator for the simulation of elastic solids animations.

- Second, we propose an algorithm for the efficient cutting of thin deformable objects using the frame-based deformable model. By dynamically adapting the shape functions associated with the different degrees of freedom, we take into account detailed topological changes in the dynamics while keeping a very low number of degrees of freedom.

- Finally, we present a new technique to design an animation of liquid. In contrast with previous work which focus on the control of the simulation, we propose to directly sculpt liquid simulation results. The input of our method is a sequence of meshes representing the surface of the liquid over time. In our system the user can select sub-parts of the animation and copy, cut, paste them at different space-time locations, in different target animations.

Aside from these technical contributions, a part of this thesis was dedicated to the study of adaptive models for physics-based animation. This work led to a state of the art review published in *Computer Graphics Forum* that we present in Chapter 3.

1.5 Structure of the document

The document is divided into five main parts.

In Chapter 2, we introduce the basics of continuum mechanics that are necessary for the whole understanding of the manuscript and we provide a survey of methods allowing to control a physics-based animation.

In Chapter 3, we present a detailed description of existing adaptive models for computer graphics.

In Chapter 4 we describe our adaptive technique for the non-intrusive and efficient simulation of liquids and elastic solids.

In Chapter 5 we describe our method to perform detailed cutting of thin deformable objects while keeping a very low number of degrees of freedom.

In Chapter 6 we describe our system to sculpt an animation of liquid by using high level tools inspired from 3D modeling.

Finally, we conclude this work in Chapter 7 and discuss limitations and possible future work.

1.6 Publications by the author

- [Man+13] *Exploring the Use of Adaptively Restrained Particles for Graphics Simulations*
  Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS), 2013 (see Chapter 4)
  Pierre-Luc Manteaux, François Faure, Stephane Redon, Marie-Paule Cani
  A video of our method is available here: `https://youtu.be/RpJjGAoqp50`.

- [Man+15] *Interactive detailed cutting of thin sheets*
  ACM SIGGRAPH Conference on Motion in Games (MIG), 2015 (see Chapter 5)
  Pierre-Luc Manteaux, Wei-Lun Sun, François Faure, Marie-Paule Cani,

James F. O'Brien
A video of our method is available here: https://youtu.be/coA_tcomWlE.

- [Lej+15] *Interactive Procedural Simulation of Paper Tearing with Sound*
ACM SIGGRAPH Conference on Motion in Games (MIG), 2015
Thibault Lejemble, Amélie Fondevilla, Nicolas Durin, Thibault Blanc-Beyne, Camille Schreck, Pierre-Luc Manteaux, Paul G. Kry, Marie-Paule Cani
A video of our method is available here: https://youtu.be/EiP3fHqtZnk.

- [Man+16] *Adaptive Physically-based Models in Computer Graphics*
Computer Graphics Forum, 2016
Pierre-Luc Manteaux, Chris Wojtan, Rahul Narain, Stephane Redon, François Faure, Marie-Paule Cani (see Chapter 2)

- The work presented in Chapter 6 has been submitted to the conference *Motion In Games* 2016. A video of our method is available here: https://www.dropbox.com/s/0cob2nuztdimjol/fluidSculpting_MIG2016.mp4?dl=0.

Chapter 2

Physics-based animation and control: State of the art

This chapter presents a state of the art on physics-based animation and control. As physics-based animation is a huge field of research, we cannot cover it all. We choose to focus our review on two topics which are directly related to the following chapters. First, we propose a short introduction to continuum mechanics and to the different physically-based models used through this thesis. Then, we describe various approaches for the control of physically-based animations.

2.1 Continuum mechanics

Continuum mechanics allow the definition of general equations of motion for a wide range of phenomena from liquids to deformable solids. In this section, we propose a progressive introduction to the basics of continuum mechanics and provide details about the physical models used in the following chapters. Firstly, we describe how to formulate general equations of motion and we present the different concepts and numerical tools used to solve them. Secondly, we present a constitutive law for fluid mechanics leading to Navier-Stokes equations and we detail their discretization using the Smoothed-Particle Hydrodynamics (SPH) model that will serve as background for our contribution in Chapter 4. Finally, we present a constitutive law for solid mechanics which allows the simulation of elastic solids and we detail the discretization of the equations of motion using the frame-based model that we use in Chapter 5. If the reader looks for a wider presentation of the existing deformable models in computer graphics, we suggest the survey of Nealen et al. [Nea+06].

2.1.1 Equations of motion

The equations of motion describe the behavior of an object over time. They are generally derived from conservation laws such as mass and momentum conservation. A constitutive law is also used to depict the intrinsic behavior of the simulated object. The remainder of this section first describes the conservation laws used to formulate general equations of motion. Then we detail the concepts and numerical tools used to solve these equations.

2.1.1.1 Conservation of mass

The conservation of mass states that, whatever the physical material which is studied, mass cannot be created or destroyed. More precisely, if we look at a small volume of the simulation domain, the variation of mass in that volume should be equal to the flux of mass going through its border. In Figure 2.1, we illustrate this law with the example of a glass of water. At a time t_1, the mass of the water should be equal to its mass at time t_0 plus the mass of the inflow and minus the mass of the outflow which occurred between t_0 and t_1.

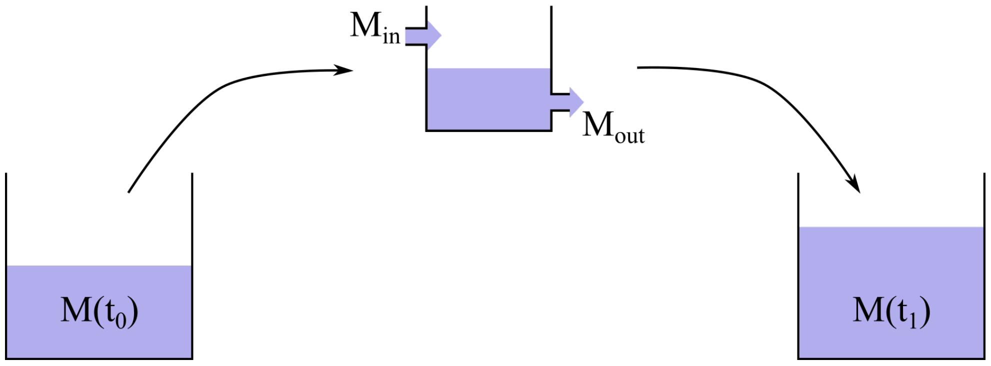


Figure 2.1: Mass conservation. $M(t_1) = M(t_0) + M_{in} - M_{out}$.

Mathematically, we can write the conservation of mass as

$$\frac{d}{dt}\left(\int_{\mathcal{V}} \rho(t,\mathbf{x})dv\right) = -\int_{\partial\mathcal{V}} \rho(t,\mathbf{x})\mathbf{v}(t,\mathbf{x})\cdot\mathbf{n}(\mathbf{x})ds \tag{2.1}$$

where

- $\rho(t,\mathbf{x})$ is the density at a point $\mathbf{x}$ and at a time t.
- $\mathbf{v}(t,\mathbf{x})$ is the velocity at a point $\mathbf{x}$ and at a time t.
- $\mathcal{V}$ is a small volume of the simulation domain Ω.
- $\partial\mathcal{V}$ is the border of $\mathcal{V}$.
- $\mathbf{n}(\mathbf{x})$ is the normal outward of $\mathcal{V}$ on a point $\mathbf{x}$ of $\partial\mathcal{V}$.

This formulation involves an integral over the volume and its boundary. When numerically solving this equation, it is simpler not to have to make the distinction between a small element of $\mathcal{V}$ and a small element of $\partial\mathcal{V}$. By using Stokes' theorem, we have

$$\int_{\partial\mathcal{V}} \rho\mathbf{v}\cdot\mathbf{n}ds = \int_{\mathcal{V}} \nabla\cdot(\rho\mathbf{v})\,dv \tag{2.2}$$

and we can rewrite Equation (2.1) as a single integral over $\mathcal{V}$

$$\int_{\mathcal{V}} \left(\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v})\right) dv = \mathbf{0}. \tag{2.3}$$

2.1.1.2 Conservation of momentum

The conservation of momentum is implied by Newton's second law which states that the forces applied on an object result into an acceleration which is inversely proportional to the mass of the object. Figure 2.2 illustrates this law in the case of two balls on which a same force is applied. The acceleration induced by the force is more important for the small ball which has a mass less important than the big ball.

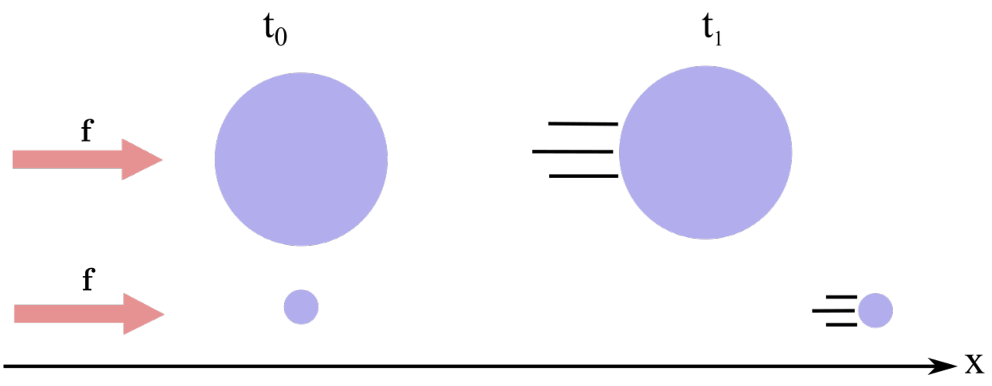


Figure 2.2: Momentum conservation. The same force $\mathbf{f}$ is applied on two balls. The resulting acceleration is inversely proportional to their mass.

Mathematically, this law can be written as

$$\int_{\mathcal{V}} \rho(t,\mathbf{x})\mathbf{a}(t,\mathbf{x})dv = \int_{\mathcal{V}} \mathbf{f}(t,\mathbf{x})dv \tag{2.4}$$

where

- $\mathbf{a}(t,\mathbf{x})$ is the acceleration at a point $\mathbf{x}$ and at a time t.
- $\mathbf{f}(t,\mathbf{x})$ is the force applied on a point $\mathbf{x}$ and at a time t.

By performing a Taylor-Young expansion on the acceleration, we can re-write the momentum conservation as

$$\int_{\mathcal{V}} \rho(t,\mathbf{x}) \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) (t,\mathbf{x}) dv = \int_{\mathcal{V}} \mathbf{f}(t,\mathbf{x}) dv. \tag{2.5}$$

Two kind of forces are generally applied on an object, the *external* forces and the *internal* forces. External forces describe the action of the surrounding environment on the object. The simplest example is the weight, i.e. the force applied by the constant gravity $\mathbf{g}$ on the object.

$$\int_{\mathcal{V}} \mathbf{f}_{ext}(t,\mathbf{x}) dv = \int_{\mathcal{V}} \rho(t,\mathbf{x}) \mathbf{g} dv \tag{2.6}$$

Internal forces describe the reaction of the object to an external deformation. Generally, they are defined by using a stress tensor $\sigma(t,\mathbf{x})$ as

$$\int_{\mathcal{V}} \mathbf{f}_{int}(t,\mathbf{x}) dv = \int_{\partial\mathcal{V}} \sigma(t,\mathbf{x}) \mathbf{n}(\mathbf{x}) ds. \tag{2.7}$$

Basically, the stress relates the deformation of the object to its material properties by using a constitutive law. A constitutive law is specific to the phenomena which is simulated. In Section 2.1.2 and Section 2.1.3, we will respectively describe the most common constitutive laws for incompressible fluids and solids. For a detailed and intuitive definition of the stress tensor we refer the reader to the *SIGGRAPH* course about real-time physics [Mül+08] by Müller et al.

Same as for the conservation of mass, we can use Stokes' theorem to get a single integral over $\mathcal{V}$.

$$\int_{\mathcal{V}} \mathbf{f}_{int}\left(t,\mathbf{x}\right) dv = \int_{\mathcal{V}} \nabla \cdot \sigma \left(t,\mathbf{x}\right) dv \tag{2.8}$$

Then, we can rewrite Equation (2.5) as

$$\int_{\mathcal{V}} \left(\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) - \nabla \cdot \sigma - \rho \mathbf{g} \right) dv = \mathbf{0}. \tag{2.9}$$

2.1.1.3 Eulerian vs. Lagrangian formulations

Eulerian and Lagrangian formulations are two different ways of interpreting the equations of motion. To understand the key difference, we take the example of the simulation of a river and use Figure 2.3 to illustrate it. Let us suppose that we want to measure fluid properties such as velocity, density or temperature of the river. A first possibility would be to measure it at a fixed location, similarly to a buoy that would remain at the same spot and measures data at regular intervals of time. This is the Eulerian viewpoint. A second possibility is to measure it along a path that would follow the flow of the river, similarly to an object floating on the river or a particle of water moving accordingly to

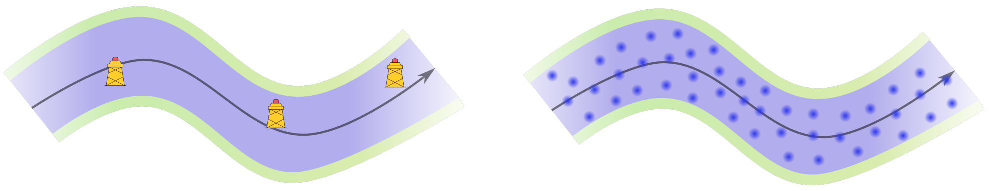


Figure 2.3: Eulerian vs. Lagrangian viewpoint. On the left, buoys are put at fixed positions on a river and measure properties such as velocity, temperature, etc. The Eulerian approach takes a similar approach. On the right, we represent the river with particles of water which follows the flow of the river and carry properties with them. This is similar to the Lagrangian viewpoint.

the flow. This is the Lagrangian viewpoint. Equation (2.3) and Equation (2.9) are described for fixed locations. Put together, they define the equations of motion from an Eulerian viewpoint:

$$\left\{\begin{aligned} &\int_{\mathcal{V}} \left(\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}\cdot\nabla\mathbf{v}\right) - \nabla\cdot\sigma - \rho\mathbf{g}\right) dv = \mathbf{0} \\ &\int_{\mathcal{V}} \left(\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v})\right) dv = 0 \end{aligned}\right. . \tag{2.10}$$

In order to adopt the Lagrangian viewpoint, let us suppose that the fluid properties are carried by particles and that $q(t,\mathbf{x})$ denote one of this property at a time t for a particle which is at a position $\mathbf{x}$. If we want to compute the derivative of q for the particle at position $\mathbf{x}$, we have to use the total derivative:

$$\frac{dq(t,\mathbf{x})}{dt} = \frac{\partial q}{\partial t}\cdot\frac{dt}{dt} + \frac{\partial q}{\partial \mathbf{x}}\cdot\frac{d\mathbf{x}}{dt} = \frac{\partial q}{\partial t}(t,\mathbf{x}) + \left(\nabla q(t,\mathbf{x})\cdot\mathbf{v}\right)(t,\mathbf{x}). \tag{2.11}$$

In Equation (2.9) which describes the momentum conservation, we can directly use Equation (2.11) to adopt a Lagrangian viewpoint. For the mass conservation described in Equation (2.3), we first need to develop the divergence of the product between density and velocity and then Equation (2.11) can be used. Finally, we can write the equations of motion from a Lagrangian viewpoint:

$$\left\{\begin{aligned} &\int_{\mathcal{V}} \left(\rho\frac{d\mathbf{v}}{dt} - \nabla\cdot\sigma - \rho\mathbf{g}\right) dv = \mathbf{0} \\ &\int_{\mathcal{V}} \left(\frac{d\rho}{dt} + \rho\nabla\cdot\mathbf{v}\right) dv = 0 \end{aligned}\right. . \tag{2.12}$$

In the following sections, we will keep Equation (2.12) which will be used by the deformable models used in Chapter 4 and Chapter 5.

2.1.1.4 Numerical solution

Once the equations of motion are stated, they are discretized in space and time in order to be numerically solved.

Spatial discretization The spatial discretization consists of approximating the object to simulate using a finite number of samples. These samples carry the *degrees of freedom* used to numerically solve the equations of motion. Then, by using an *interpolation method*, it is possible to continuously approximate quantities such as position, velocity, density and so on, at any location on the domain. Finally, an integration rule, also called *quadrature rule*, is needed in order to integrate these quantities over the domain. These are the main components for solving the equations of motion: degrees of freedom, an interpolation method and a quadrature rule for the numerical integration over the simulation domain.

There are multiple possibilities for these components. It is crucial to choose them based on the goals of the simulation: What do we want to measure? How will boundaries be represented? Will any topological changes occur? Are there restrictions in terms of computational or memory cost? In the following we briefly introduce the most commonly used solutions in computer graphics.

Degrees of freedom In Eulerian simulation, velocities are the most common independent degrees of freedom while Lagrangian simulations generally use positions and velocities. We can also mention the use of affine frames to capture translations, rotations and shearing. They are used in the frame-based model which will be detailed in Section 2.1.3.2.

Usually, we distinguish two types of sampling of the degrees of freedom: mesh-based and mesh-less (see Figure 2.4).

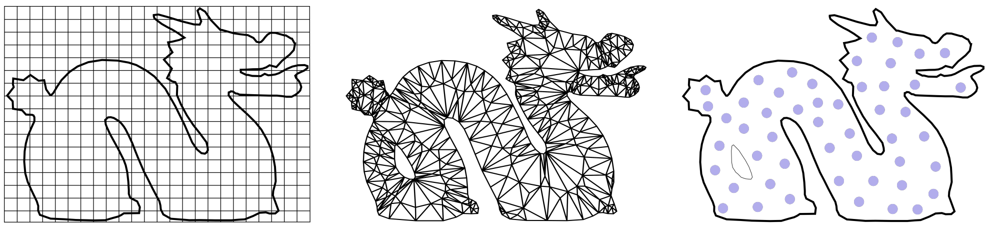


Figure 2.4: On the left a grid discretization, commonly used in Eulerian simulations. In the middle an unstructured mesh discretization, commonly used in mesh-based Lagrangian simulations. On the right, a point-based discretization, commonly used in mesh-less Lagrangian simulations.

In mesh-based methods, the vertices of a mesh are used to sample the degrees of freedom. For instance, Eulerian simulations usually use a cartesian grid which allows the accurate computation of derivatives. In contrast, Lagrangian simulations are mostly based on unstructured triangular meshes which allows the handling of complex boundaries. In mesh-less methods, the samples are uniformly distributed over the domain. Depending on the simulated phenomena, the structure may be quasi-nonexistent which brings a lot of flexibility. For fluids, the neighborhood of the samples will change every time whereas for elastic solids their neighborhood will remain the same as long as the object does not undergo topological changes such as fracture or cutting. Each of these samplings can benefit from adaptivity. We will detail the different possibilities of spatial adaptivity in Section 3.2.

Interpolation Interpolation is used to approximate the physical quantities over the domain such as density, displacement, pressure and so on. In computer graphics, we can distinguish two major interpolation methods: polynomial interpolation and kernel interpolation. For each of them, different weights are used, as illustrated in Figure 2.5. The weights are also called *shape functions* and can be seen as the region of influence of a sample.

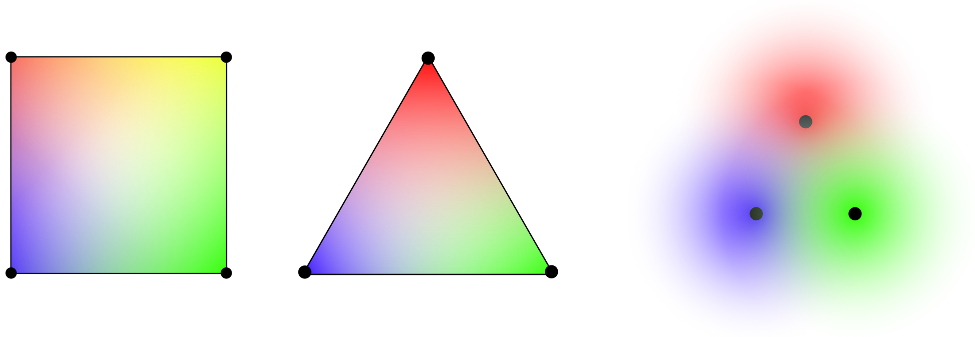


Figure 2.5: Three examples of shape functions in 2D. Each color represents the shape function associated to one sample point (black circle) carrying the degrees of freedom. On the left, shape functions for bilinear interpolation are illustrated. In the middle, barycentric shape functions are illustrated. On the right, kernel-based shape functions that are used in SPH and MLS interpolation are illustrated.

The choice of the interpolation method mainly depends on how the material samples have been distributed. For a sampling based on cartesian grid, trilinear interpolation is often used, for instance in Eulerian simulations [Bri08]. For unstructured triangle meshes, linear interpolation with barycentric weights is the most popular choice to simulate elastic solids [Mül+08]. For mesh-less samplings, the two most common kernel interpolation methods are Smoothed-Particle Hydrodynamics (SPH) interpolation and Moving Least Squares (MLS) interpolation. SPH interpolation has been used to simulate a wide range of phenomena from fluid [DC99] to elastic solids [BIT09]. MLS has been introduced by Müller et al. to simulate elastic and plastic deformations [Mül+04]. It was later extended to solids fracture by Pauly et al. [Pau+05] and interactive cutting by Steinemann et al. [SOG09]. Both methods require a dense sampling of th object and can be used using a cubic kernel as shape function. We will provide more details about SPH and its use for the simulation of incompressible fluid in Section 2.1.2.2 and how to save computational time by using an adaptive model in Section 4.2.

Mesh-based and mesh-less methods can be combined to get the best of both worlds. In this case, two interpolation methods are used, one for the mesh-based side and one for the mesh-less side. We refer the reader to the recent *SIGGRAPH* course on the material point method [Jia+16]. In this course, existing hybrid models are compared and the interpolation methods to transfer data from one representation to another are described.

A common drawback of the methods mentioned above is that they require a dense sampling. In Section 5.3, we will detail how to interactively handle topological changes with a sparse sampling by using the Voronoi-based interpolation method used in the frame-based model.

Spatial Integration Over the simulation, different physical quantities such as density or internal forces, need to be integrated over the domain. There is need for a quadrature rule. Many exist, most of the time the simple midpoint rule is chosen (see Figure 2.6).

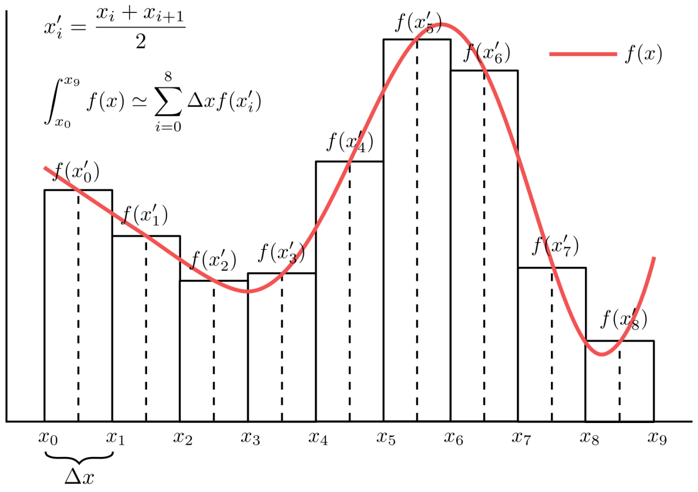


Figure 2.6: Illustration of the midpoint rule for a one dimensional function. The integration domain is partitioned into uniform regions $[x_i, x_{i+1}]$, an integration point x'_i is sampled at the center of each partition and the function is evaluated at the location of the integration points.

The domain is decomposed in a set of partitions, where each partition has an associated volume V_i. *Mid*-Points $\mathbf{x}'_i$ are sampled at the center of each partition. They are called *integration points*. Then the integral of a function f of class C^{k+1} over a domain Ω is approximated by

$$\int_\Omega f(\mathbf{x})dv \simeq \sum_i V_i f(\mathbf{x}'_i). \tag{2.13}$$

In mesh-based methods, it is common to consider one integration point at the center of each element and integrate over the volume of the element. In mesh-less methods, when the sampling is dense, integration points are often co-located with the material samples and integrated over their associated volume. However, when the sampling is sparse, an independent sampling of integration points can be used to get a finer integration. This is the case in the frame-based method that will be described in Section 2.1.3.2.

Time integration Let us assume that we spatially discretized the Lagrangian equations of motion, Equation (2.12), using a finite number of samples N which carry positions and velocities as degrees of freedom. For the sake of simplicity, we also assume that each sample has a fixed mass through the simulation which implies that mass conservation is ensured. Then, we can solely focus on the conservation of momentum and its integration over the time range $[0, T]$ of the simulation.

As for spatial integration, the temporal domain is discretized. Ideally, this discretization would adapt to the time scale of the simulation. For instance, fast motion would require a fine discretization while a coarse discretization would be sufficient for slow motion. We will detail the different techniques for an adaptive discretization of time in Section 3.1. In the following, we simply assume a uniform discretization of $[0, T]$ defined by a time step Δt. The integration between two consecutive time steps t_n and t_{n+1} can be written as

$$\int_{t_n}^{t_{n+1}} M \frac{d\mathbf{v}}{dt} dt = \int_{t_n}^{t_{n+1}} \mathbf{f} dt \tag{2.14}$$

where

- $\mathbf{v} \in \mathbb{R}^{3N}$ concatenates the velocity of each sample.
- $\mathbf{f} \in \mathbb{R}^{3N}$ concatenates the forces applied on each sample.
- $M \in \mathbb{R}^{3N \times 3N}$ concatenates the mass of each sample.

Many integration schemes can be used. Most of them can be explained using the Taylor expansion of a function f:

$$f(x) = \sum_{\alpha=0}^{k} \frac{(x-a)^\alpha}{\alpha!} f^{(\alpha)}(a) + \int_a^x \frac{(x-t)^k}{k!} f^{(k+1)}(t) dt. \tag{2.15}$$

By applying Equation (2.15) on $\mathbf{v}$ with $k = 0$, we have

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \int_{t_n}^{t_{n+1}} \frac{d\mathbf{v}}{dt} dt \tag{2.16}$$

which allows one to rewrite Equation (2.14) as

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \int_{t_n}^{t_{n+1}} M^{-1} \mathbf{f} dt. \tag{2.17}$$

This expression can be further expanded in order to get more accurate results. In computer graphics, this is the most used expansion. Finally, the integral term in Equation (2.17) is generally computed with the rectangle quadrature rule which we illustrate in Figure 2.7.

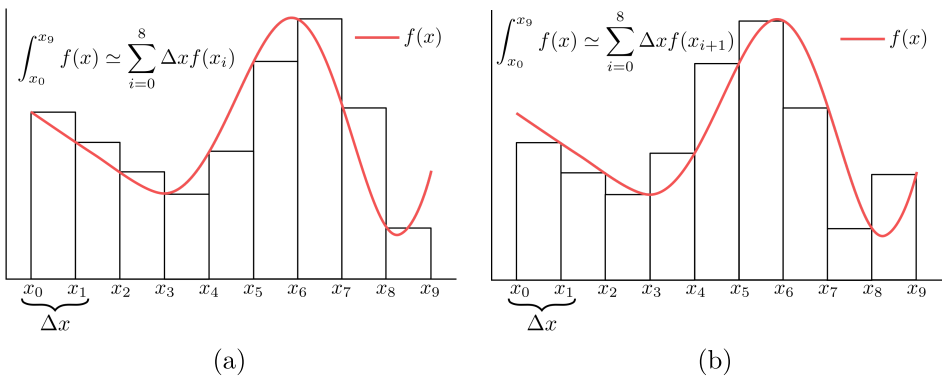

$\int_{x_0}^{x_9} f(x) \simeq \sum_{i=0}^{8} \Delta x f(x_i)$
$f(x)$
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
Δx
(a)
$\int_{x_0}^{x_9} f(x) \simeq \sum_{i=0}^{8} \Delta x f(x_{i+1})$
$f(x)$
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
Δx
(b)

Figure 2.7: Illustrations of the rectangle integration rules. A function f can be integrated over a regularly sampled domain using either the left rectangle rule (a) or the right rectangle rule (b).

For a left rectangle method, we get an explicit integration of the velocity:

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t M^{-1}\mathbf{f}(t_n). \tag{2.18}$$

For a right rectangle method, we get an implicit integration of the velocity:

$$\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t M^{-1}\mathbf{f}(t_{n+1}). \tag{2.19}$$

We can apply the same rule for the integration of the positions. Depending on which quantity is integrated first and which integration rule is used, we can distinguish three main schemes used in computer graphics: forward Euler, symplectic Euler and backward Euler. Forward Euler is the explicit integration of both position and velocity. Figure 2.8 illustrates the integration of position in 1D.

$$\begin{gathered}\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \Delta t\mathbf{v}(t_n)\\ \mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t M^{-1}\mathbf{f}(t_n)\end{gathered} \tag{2.20}$$

Symplectic Euler is the explicit integration of velocity and implicit integration of position:

$$\begin{gathered}\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t M^{-1}\mathbf{f}(t_n)\\ \mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \Delta t\mathbf{v}(t_{n+1})\end{gathered} . \tag{2.21}$$

Backward Euler is the implicit integration of both position and velocity:

$$\begin{gathered}\mathbf{v}(t_{n+1}) = \mathbf{v}(t_n) + \Delta t M^{-1}\mathbf{f}(t_{n+1})\\ \mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + \Delta t\mathbf{v}(t_{n+1})\end{gathered} . \tag{2.22}$$

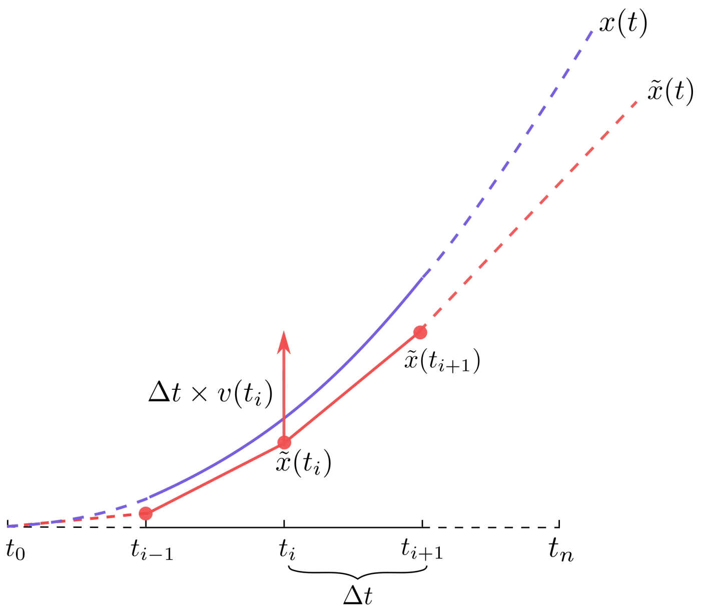


Figure 2.8: Illustration of a 1D forward Euler time integration. The solution is described by $x(t)$ and the numerical approximation by $x'(t)$.

On one hand, forward and symplectic Euler are easy to implement and cheap to compute but stability is guaranteed for a restricted range of time steps. On the other hand, backward Euler is more expensive as it requires the solution of an equation but this is greatly compensated by the larger time steps which can be used while ensuring stability. However, this speed-up comes at the price of numerical damping which might be undesired. A nice way to solve the non-linear equation of backward Euler is to expand the force expression and linearize in order to fall back to solving a linear system. There, efficient iterative methods such as the conjugate gradient can be used. In Section 4.3, we describe the linear system resulting from the use of backward Euler and detail how to combine it with the adaptive method that we present in Chapter 4.

2.1.2 Fluid mechanics

In this section, we describe a constitutive law for incompressible fluids such as water and we detail how to solve the equations of motion by using the Smoothed-Particle Hydrodynamics (SPH) model that we will use in Section 4.2.

2.1.2.1 Constitutive Law

Fluids, for instance smoke, mainly react to pressure and viscosity. The stress tensor that we introduced in Section 2.1.1.2 is used to relate the deformation of the fluid to this two properties in the following constitutive law:

$$\sigma = -pI_{3\times 3} + \eta\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right) \tag{2.23}$$

where

- $I_{3\times 3}$ is the identity.
- p is the pressure applied on the fluid.
- η is the viscosity of the fluid.

By injecting this equation into Equation (2.8), we get the internal forces of the fluid.

$$\mathbf{f}_{int} = \nabla\cdot\sigma = \nabla\cdot\left(-pI + \eta\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right)\right) = -\nabla p + \eta\Delta\mathbf{v} \tag{2.24}$$

If we want to describe an incompressible fluid such as water, we need to specify that the mass should not vary over time:

$$\frac{d\rho}{dt} = 0. \tag{2.25}$$

By using Equation (2.24) and (2.25) in the equations of motion described in Equation (2.12), we now describe an incompressible fluid. These new equations of motion are called the Navier-Stokes equations for an incompressible fluid:

$$\begin{cases} \int_{\mathcal{V}} \left(\rho\frac{d}{dt}\mathbf{v} + \nabla p - \eta\Delta\mathbf{v} - \rho\mathbf{g}\right) dv = \mathbf{0} \\ \int_{\mathcal{V}} \nabla\cdot\mathbf{v}dv = 0 \end{cases}. \tag{2.26}$$

2.1.2.2 Smoothed-Particle Hydrodynamics model

Smoothed-Particle Hydrodynamics (SPH) is an interpolation method that can be used to approximate Navier-Stokes equations in a Lagrangian way. SPH was initially proposed by Monaghan [Mon92] and introduced in graphics by Desbrun et al. [DC99]. The fluid is discretized into particles which represent small volumes of the whole fluid and each quantity carried by the particle is interpolated using SPH. In Chapter 4, we will describe how to easily combine SPH with an adaptive model in order to save computational time.

SPH interpolation The interpolation of a function f and its derivatives of order α at a position $\mathbf{x}$ using SPH are defined by these equations:

$$\begin{cases} f(\mathbf{x}) = \int_V f(\mathbf{x}')W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}' \\ D^\alpha f(\mathbf{x}) = \int_{\mathcal{V}} f(\mathbf{x}')D^\alpha W(\mathbf{x}-\mathbf{x}', h)d\mathbf{x}' \end{cases}. \tag{2.27}$$

where

- W is a function called *kernel*.
- h is the smoothing radius, also called length scale, which represents the support of W.

Let assume that the fluid is discretized using N particles. These particles play two roles: First they carry the degrees of freedom and the fluid properties; Second their positions are used as integration points. By applying the midpoint rule from Equation (2.13), we get a discretized version of Equation (2.27):

$$\begin{cases} f(\mathbf{x}) = \sum_{i=1}^{N} f(\mathbf{x}_i) V_i W(\mathbf{x} - \mathbf{x}_i, h) \\ \\ D^{\alpha} f(\mathbf{x}) = \sum_{i=1}^{N} f(\mathbf{x}_i) V_i D^{\alpha} W(\mathbf{x} - \mathbf{x}_i, h) \end{cases} \tag{2.28}$$

where

- $V_i = m_i / \rho_i$ is the volume of particle i.
- m_i is the mass of particle i.
- ρ_i is the density of particle i.

How to choose the kernel The choice of the kernel varies depending on the function to interpolate. In practice, the cubic kernel from Monaghan [Mon92] (Figure 2.9) that we use in Chapter 4 works well. This kernel meets properties which are generally required from W:

- W is normalized. Thus, constants are interpolated exactly.

$$\int_{\mathcal{V}} W(\mathbf{x}, h) dx = 1 \tag{2.29}$$

- W has a compact support.

$$\parallel \mathbf{x} \parallel \geq h \implies W(\mathbf{x}, h) = 0 \tag{2.30}$$

- W tends to the delta function when the length scale h tends to 0.

$$\lim_{h \to 0} W(x, h) = \delta(x) \tag{2.31}$$

- W should be symmetric to enforce invariance under rotation.

$$W(-x, h) = W(x, h) \tag{2.32}$$

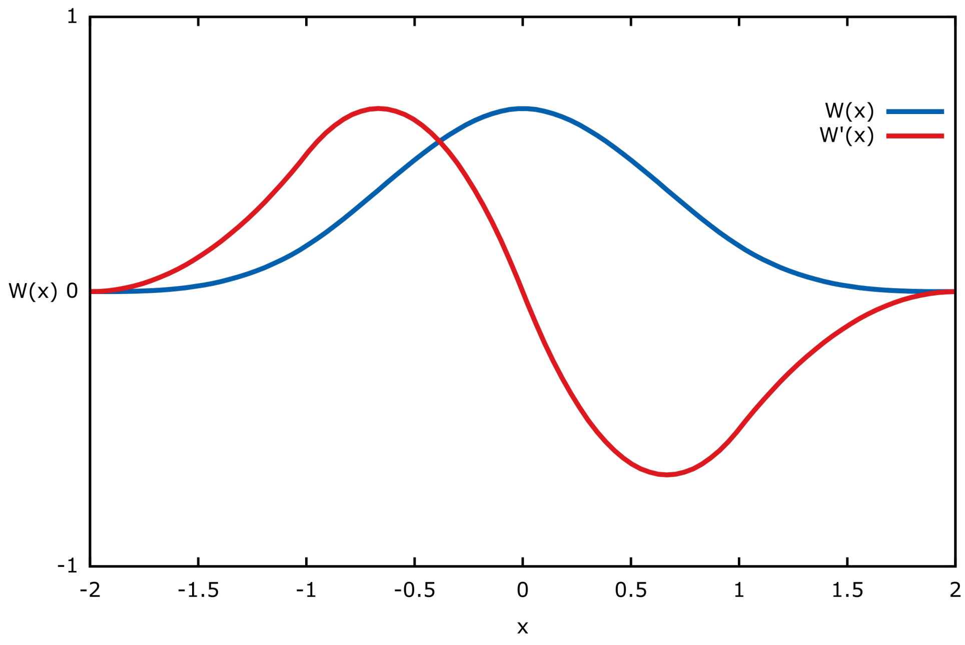


Figure 2.9: Illustration of the 1D cubic kernel (in blue) used by Monaghan et al. [Mon92] and its derivative (in red) for $h = 1$. Note that the support of W is $2h$.

- Depending on the function to interpolate the kernel should be positive to prevent unphysical interpolated value.

$$W \geq 0 \tag{2.33}$$

Here we can notice a first limitation of SPH: For a constant function f, the approximation of its derivatives using Equation (2.28) will not necessarily vanish depending on W. In practice, we only need the first derivative. A common practice to fix that is to consider the derivative of the product of f with an arbitrary differentiable function that we note Φ:

$$\nabla f = \frac{1}{\Phi}\left(\nabla(f\Phi) - f\nabla\Phi\right). \tag{2.34}$$

In this case, we can approximate $\nabla(f\Phi)$ and $\nabla\Phi$ using Equation (2.28). If f is constant, we will get $\nabla f = 0$. In practice the density ρ is often used for Φ. In the following paragraph, we will see another usage of this technique.

Application to Navier-Stokes equations First of all, let us discretize Navier-Stokes equation (Equation (2.26)) on the sampling of the particles using the midpoint rule:

$$\begin{cases} \sum_{i=1}^{N} \left(\rho_i \frac{d}{dt}\mathbf{v}_i + \nabla p_i - \eta \Delta \mathbf{v}_i - \rho_i \mathbf{g} \right) V_i = \mathbf{0} \\ \sum_{i=1}^{N} \nabla \cdot \mathbf{v}_i V_i = 0 \end{cases} . \tag{2.35}$$

We can omit the mass conservation as we did before, by assuming that the particles have a fixed mass through the simulation. Recently, Bender and Koschier [BK15] demonstrated that this simplification prevents from using larger time steps. But for the sake of simplicity, we will keep this approximation in the remainder of this manuscript. Now, we can discretize each term of the equations for a particle i using the SPH technique from Equation (2.28).

Density

$$\rho_i = \sum_{j=1}^{N} m_j W(\mathbf{x_i} - \mathbf{x_j}, h) \tag{2.36}$$

Pressure

$$p_i = k\left(\rho_i - \rho_0\right) \tag{2.37}$$

where

- ρ_0 is the rest density of the fluid ($1000\text{kg}/\text{m}^3$).
- k is a stiffness parameter.

Equation (2.37) is a simple and cheap computation of the pressure. This equation of state acts like a spring in order to enforce the incompressibility of the fluid. However, a high stiffness is generally needed to get close to incompressibility. Therefore, very small time steps are required to ensure stability. Recently, new techniques were proposed to ensure incompressibility while using larger time steps. We do not detail these methods in this manuscript but refer the reader to the work of Ihmsen et al. [Ihm+14a] and Bender and Koschier [BK15].

Pressure gradient

$$(\nabla p)_i = \sum_{j=1}^{N} \frac{m_j}{\rho_j} p_j \nabla W(\mathbf{x_i} - \mathbf{x_j}, h) \tag{2.38}$$

Here we can notice that the resulting pressure force between two particles i and j is not symmetric and therefore does not conserve linear and angular momentum:

$$\mathbf{f}_{ij}^{pressure} = -\frac{m_i m_j}{\rho_i \rho_j} p_j \nabla W(\mathbf{x_i} - \mathbf{x_j}, h). \tag{2.39}$$

To remedy this problem, we can use Equation (2.34) with $\Phi = \frac{1}{\rho_i}$:

$$(\nabla p)_i = \rho_i \left(\nabla \left(\frac{p_i}{\rho_i} \right) + p_i \frac{\nabla \rho_i}{\rho_i^2} \right). \tag{2.40}$$

Finaly, we re-use SPH interpolation to get a new approximation of the pressure gradient:

$$(\nabla p)_i = \rho_i \sum_{j=1}^{N} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(\mathbf{x_i} - \mathbf{x_j}, h). \tag{2.41}$$

This results into symmetric pressure forces between two particles i and j

$$\mathbf{f}_{ij}^{pressure} = -m_i m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(\mathbf{x_i} - \mathbf{x_j}, h). \tag{2.42}$$

Velocity Laplacian

$$(\Delta \mathbf{v})_i = \sum_{j=1}^{N} \frac{m_j}{\rho_j} \mathbf{v}_j \Delta W(\mathbf{x_i} - \mathbf{x_j}, h) \tag{2.43}$$

Same as for the pressure gradient, this would result in a non-symmetric inter-particle viscosity force:

$$\mathbf{f}_{ij}^{viscosity} = \eta \frac{m_i m_j}{\rho_i \rho_j} \mathbf{v}_j \Delta W(\mathbf{x_i} - \mathbf{x_j}, h). \tag{2.44}$$

If we assume that the density is constant, which is the case in theory, we could obtain symmetric forces by using Equation (2.34) with $\Phi = \rho_i$. However, in practice, the density is not constant and the evaluation of the Laplacian of the kernel is sensitive to particle sampling, which makes this solution inadequate. Actually, it is quite hard to correctly handle viscosity using SPH and this is still an area of research, especially for liquids exhibiting complex viscous behaviors such as coiling or buckling. We refer the reader to the recent work of Peer et al. [Pee+15] and Takahashi et al. [Tak+15] about this topic. For fluid with a low viscosity, Monaghan [Mon05] proposed a gradient-based formulation of the Laplacian which results in symmetric forces:

$$(\Delta \mathbf{v})_i = \frac{1}{\rho_i} \sum_{j=1}^{N} m_j \Pi_{ij} \nabla W(\mathbf{x_i} - \mathbf{x_j}, h) \tag{2.45}$$

where

$$\Pi_{ij} = -\frac{2hc_s}{\rho_i + \rho_j} \frac{\mathbf{v}_{ij}^T \mathbf{x}_{ij}}{|\mathbf{x}_{ij}|^2 + \epsilon h^2} \tag{2.46}$$

and c_s is the speed of sound in the media and ϵ is a numerical constant to avoid singularities. In practice, $\epsilon = 0.01$ works well. We will use this formulation in Section 4.2.

Time integration If we assume a uniform discretization of the time based on a time step Δt, symplectic Euler is common choice to integrate the equations of motion over time and for a particle i we get the following equations.

$$\begin{aligned} \mathbf{v}_i(t_{n+1}) &= \mathbf{v}_i(t_n) + \frac{\Delta t}{m_i}\left(\sum_{j=1}^{N}\left(\mathbf{f}_{ij}^{pressure}(t_n) + \mathbf{f}_{ij}^{viscosity}(t_n)\right) + m_i\mathbf{g}\right) \\ \mathbf{x}_i(t_{n+1}) &= \mathbf{x}_i(t_n) + \Delta t \mathbf{v}_i(t_{n+1}) \end{aligned} \tag{2.47}$$

We described the key ingredients of the SPH model and how to use them to discretize Navier-Stokes equations. There is not enough space to cover the exciting challenges related to the building of a full SPH simulator. For a robust handling of static and dynamic boundaries, a surface tension model and an efficient surface reconstruction pipeline, we refer to the work of Akinci et al. [Aki+12b; AAT13; Aki+12a]. For a state of the art of optimization techniques for SPH, we refer the reader to the work of Ihmsen et al. [Ihm+11]. For other references related to the handling of viscosity, multiphase simulations and other problems, the state of the art report on SPH by Ihmsen et al. [Ihm+14b] is a safe starting point.

2.1.3 Solid mechanics

In this section, we focus on the simulation of elastic objects. When submitted to external forces, an elastic object reacts so that it comes back to its rest shape. In contrast with fluids, internal forces are history dependent, they depend on how much the object deformed compared to its rest shape. It becomes crucial to be able to describe the deformation of an object in order to express its reaction.

The deformation is modeled by a mapping Φ between the undeformed configuration Ω_0 and the deformed configuration Ω (see Figure 2.10). Φ is called the *displacement field*:

$$\begin{array}{cccccc} \Phi & : & \mathbb{R}\times\Omega_0 & \longrightarrow & \Omega \\ & & (t,\mathbf{X}) & \longrightarrow & \mathbf{x} \end{array} \tag{2.48}$$

where

- t is the time.
- $\mathbf{X}$ is a point in the undeformed configuration.
- $\mathbf{x} = \Phi(\mathbf{X}, t)$ is the mapping of $\mathbf{X}$ into the deformed configuration at time t.

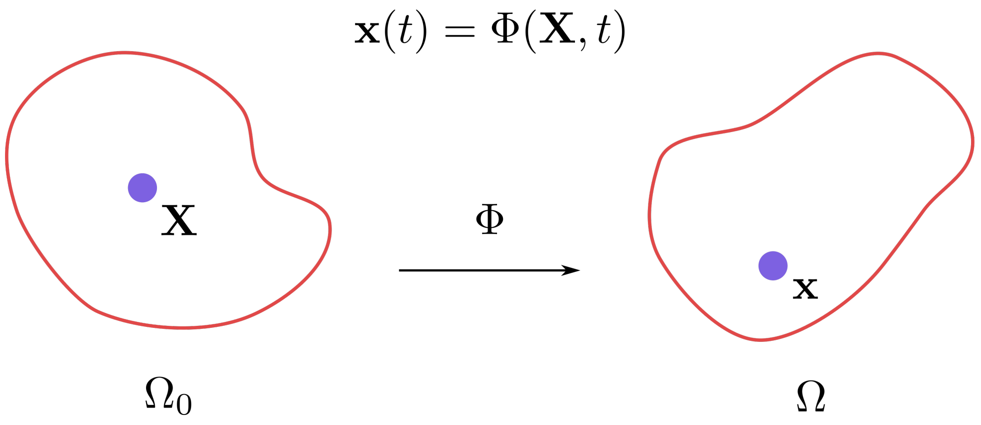


Figure 2.10: The displacement field Φ maps each point $\mathbf{X}$ from the rest configuration Ω_0 to a point $\mathbf{x}$ in the deformed configuration Ω.

The deformation gradient F describes the local state subject to rigid and/or deformable displacement with respect to the undeformed configuration. It is defined by the following equation.

$$F = \frac{\partial \Phi}{\partial \mathbf{X}} \tag{2.49}$$

The strain tensor ϵ measures the deformation. There are many strain measures such as the Green-Lagrange strain:

$$\epsilon = \frac{1}{2}\left(F^T F - I\right) \tag{2.50}$$

or its linearized version, the Cauchy strain:

$$\epsilon = \frac{1}{2}\left(F + F^T\right) - I. \tag{2.51}$$

In contrast with the Green-Lagrange strain, the Cauchy strain does not capture well large rotations and therefore is mainly used for small deformations. In Chapter 5, we use the Green-Lagrange strain for the simulation of elastic thin sheets.

The displacement field, deformation gradient and strain tensor are the main components of the constitutive law that relates the deformation to the material properties of the object.

2.1.3.1 Constitutive Law

For elastic materials, the stress tensor σ can be described using constitutive density energy Ψ that is derived with respect to the strain tensor ϵ:

$$\sigma = \frac{\partial \Psi}{\partial \epsilon}. \tag{2.52}$$

Different forms of energy exist. For an elastic material, also called Hookean material, the density energy is the following equation,

$$\Psi = \frac{1}{2} H \epsilon^2 \tag{2.53}$$

where H is called the stiffness tensor and is a $3 \times 3 \times 3 \times 3$ tensor. By injecting Equation (2.53) into Equation (2.52), we get the following constitutive law:

$$\sigma = H\epsilon. \tag{2.54}$$

For isotropic materials, the number of material parameters in H can be reduced to two, the Young's modulus E and the Poisson's ratio ν. These parameters respectively describes the resistance of the object to extension and to shearing. Moreover, the strain and stress tensor are symmetric which simplifies Equation (2.54):

$$\sigma = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{bmatrix} = \tilde{H} \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{bmatrix} \tag{2.55}$$

where

$$\tilde{H} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix}. \tag{2.56}$$

Instead of computing internal forces as the divergence of the stress, they can be computed as the derivative of the density energy with respect to the degrees of freedom:

$$\mathbf{f} = -\int_{\mathcal{V}} \frac{\partial \Psi}{\partial \mathbf{x}}^T dv. \tag{2.57}$$

2.1.3.2 Frame-based model

There are many models to discretize the equations of motion for an elastic solid: Moving Least Squares [Mül+04], finite element method [OH99], etc. Here we choose to discretize these equations using the frame-based model, as we will use it in Chapter 5 to handle interactive and detailed cutting of thin elastic objects.

The frame-based model was introduced by Gilles et al. [Gil+11] to simulate deformable objects. In contrast to other deformable models, it allows the simulation of complex object with very few degrees of freedom and to handle heterogeneous materials easily [Fau+11]. Additionally, this work formalized the concept of multi-layer physical framework. In the following, we first describe what is a multi-layer framework and illustrate it in the case of the frame-based model. Then we detail a standard choice for the different components of the frame-based model: degrees of freedom, interpolation and integration. As in the previous section, this section is a high level overview. Detailing collision detection and response processes and giving a more accurate formulation of viscosity via the strain rate are out of the scope of this chapter.

A multi-layer physical framework Most of the time, the different components of a physics-based model are described as a monolithic framework: degrees of freedom, interpolation, integration and constitutive law are put together in one formula which computes the forces applied on the degrees of freedom. On one hand, this provides a compact and implementation-friendly expression. On the other hand, it requires assumptions on each component and make it hard to distinguish what should be changed in order to integrate collisions, to embed a visual model or to test variations of the initial model.

An interesting alternative is to build a multi-layer framework where each component of a physics-based model represents a layer which is able to communicate with other layers through mappings. We distinguish three main advantages: Firstly, the framework has then a great modularity, as different components can be implemented separately, re-used and mixed together. A direct consequence is the ease at prototyping. State of the art methods can be implemented in hours instead of days. Comparisons of different models is much easier. Secondly, as each layer can be discretized at its own resolution, the granularity of the simulation can be easily controlled and computational tasks better distributed. For instance, in the case of the frame-based model, the sampling of the degrees of freedom is sparse to accelerate the time integration while a denser sampling is used for the integration points in order to accurately compute the deformation. Finally, embedding techniques, used to display a fine visual model or to handles collision with a coarse representation, fits well in this framework by using the displacement field to map these models to the degrees of freedom.

In their work, Gilles et al. [Gil+11] proposed such an alternative by decomposing the computation of force using the derivation chain rule:

$$\mathbf{f} = -\int_V \left(\frac{\partial \Psi}{\partial \mathbf{x}}\right)^T dv = -\int_V \left(\frac{\partial F}{\partial \mathbf{x}}\right)^T \left(\frac{\partial \epsilon}{\partial F}\right)^T \left(\frac{\partial \Psi}{\partial \epsilon}\right)^T dv. \tag{2.58}$$

The three different layers are now visible: the degrees of freedom, the deformation gradient, the strain tensor and the constitutive density energy. Moreover, we can distinguish the Jacobian of the mappings that link degrees of freedom to deformation gradient and deformation gradient to strain. Notice that the derivative of the constitutive density energy with respect to strain is actually the stress tensor presented in Equation (2.52). In Figure 2.11, we illustrate this framework in the case of the frame-based model.

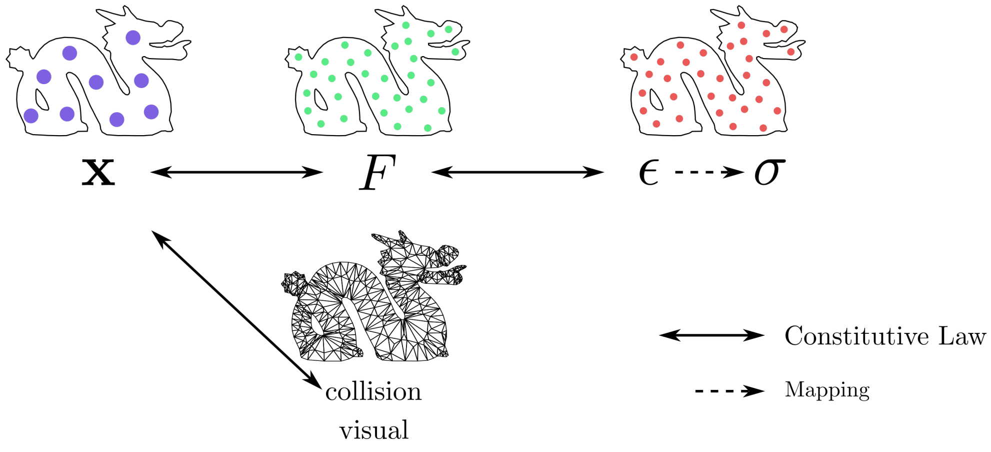


Figure 2.11: Each component of the simulation is isolated and communicates with other components through mappings. By doing so, the framework allows fast prototyping and comparison of a wide range of deformable models.

Degrees of freedom Let assume that N samples have been uniformly distributed over the object. Each sample carries an affine frame $T = (A, \mathbf{t})$ which represents 12 degrees of freedom: 3 for translation $\mathbf{t}$, 9 for the matrix A combining rotation, scaling and shearing. Affine frames are expressed with respect to their initial configuration $T_0 = (A_0, \mathbf{t}_0)$.

Interpolation Linear blend skinning is used to interpolate the displacement field and other quantities of the simulation. A deformed position $\mathbf{x}$ can be interpolated as a weighted sum of the affine transformations applied to the rest position $\mathbf{x}_0$:

$$\begin{aligned} \mathbf{x} &= \Phi(\mathbf{x}_0) = \sum_{i=1}^{N} w_i(\mathbf{x}_0) \left(\mathbf{t}_i + A_i \mathbf{x}_{0,i}^{rel} \right) \\ \mathbf{x}_{0,i}^{rel} &= A_{0,i}^{-1} \left(\mathbf{x}_0 - \mathbf{t}_{0,i} \right) \end{aligned} \tag{2.59}$$

where

- $\mathbf{x}_{0,i}^{rel}$ is the relative position of $\mathbf{x}_0$ in the frame defined by $T_{0,i}$.
- w_i is the shape function associated to the frame i.

Different shape functions can be used. When using linear interpolation, three properties are important in order to represent a physical behavior. First, the shape function should linearly decrease with respect to distance in the material. Otherwise, the deformation will not be uniform with respect to the distance from the frame. Second, the shape function should be positive. Third, the shape functions should form a partition of unity. In practice, we use the Voronoi-based shape functions. In Section 5.3, we will detail the computation of Voronoi-based shape functions and how to dynamically update them to take into account topological changes.

From the description of the displacement field (Equation (2.59)), the deformation gradient can then be derived:

$$F\left(\mathbf{x}\right) = \frac{\partial \mathbf{x}}{\partial \mathbf{x}_0} = \sum_{i=1}^{N} \nabla w_i(\mathbf{x}_0)\left(\mathbf{t}_i + A_i \mathbf{x}_{0,i}^{rel}\right) + w_i\left(A_i A_{0,i}^{-1}\right). \tag{2.60}$$

Spatial integration Any quadrature rule can be used. Here, for the sake of simplicity, we suppose that we use the midpoint rule (see Equation (2.13)) briefly described in Figure 2.6. In contrast with the SPH model that we described in Section 2.1.2.2, the sampling of the frames is very sparse. On one hand this allows the integration of the equations of motion over time very efficiently. On the other hand, there are not enough samples to accurately measure the deformation. To remedy this, an additional denser sampling is used for the integration points. Let assume that we sampled the object with M integration points, then we can use the midpoint rule to compute the internal forces for one frame,

$$\mathbf{f}_i = -\sum_{j=1}^{M}\left(\frac{\partial F}{\partial \mathbf{x}_i}\right)^T \left(\frac{\partial \epsilon}{\partial F}\right)^T \left(\frac{\partial \Psi}{\partial \epsilon}\right)^T \left(\mathbf{x}_j\right) V_j \tag{2.61}$$

where

- $\mathbf{x}_j$ is the position of the integration point j.
- V_j is volume associated to integration point j.

Temporal integration Assuming the mass matrix is lumped and that we use symplectic Euler, the time integration of one frame i is the following.

$$\begin{aligned}\begin{pmatrix}\mathbf{t'}_i(n+1)\\ A'_i(t_{n+1})\end{pmatrix} &= \begin{pmatrix}\mathbf{t'}_i(t_n)\\ A'_i(t_n)\end{pmatrix} + \Delta t M_i^{-1}\left(\mathbf{f}_i(t_n) + M_i \mathbf{g}\right)\\ \begin{pmatrix}\mathbf{t}_i(n+1)\\ A_i(t_{n+1})\end{pmatrix} &= \begin{pmatrix}\mathbf{t}_i(t_n)\\ A_i(t_n)\end{pmatrix} + \Delta t \begin{pmatrix}\mathbf{t'}_i(n+1)\\ A'_i(t_{n+1})\end{pmatrix}\end{aligned} \tag{2.62}$$

Where M_i is the mass matrix of the frame i.

$$M_i = \int_V w_i^T \rho w_i dv \tag{2.63}$$

and ρ is the density of the object.

We described the key ingredients to simulate elastic solids using the frame-based model. As for fluid mechanics, there are many other related topics that we choose not to detail here. For the formulation of implicit Euler and a survey of collision detection techniques, we respectively refer the reader to the course of Witkin et al. [WBK01] and the work of Teschner et al. [Tes+05].

2.1.4 Conclusion on continuum mechanics

In this section, we briefly presented the basics of continuum mechanics. First, we described how to design equations of motion and which numerical tools are used to solve them. Then, for fluids and solids mechanics, we mentioned their specificities and detailed a deformable model. We illustrated the fact that continuum mechanics allows the automatic computation of realistic motion from a wide range of phenomena. This strength mainly explains why hand-made animations of physical phenomena have been replaced by physics-based animation. However, controlling a physics-based model remains a tedious task. It usually consists of tweaking parameters whose understanding require advanced knowledge of the underlying model. In the end, the user apply a trial-and-error process to find the parameters which produce a result close enough to its expectations. In the following section, we present different methods for the control of physics-based animation.

2.2 Control of physics-based animation

As we mentioned in the introduction, computer were used for producing animations in two main ways. First, principles of traditional animation were adopted, leaving the animator to describe the key-frames that would bring life and style to characters. Second, physics was used to animate objects whose complexity in terms of scale and behavior would have been intractable for a single animator. Nowadays, these two use cases are only the extremities of a large spectrum. In between lies the control of physics-based animations which tends to take into account user directions, that will bring life and style, and physics, that will handle the exciting complexity and dynamics of the physical behavior. In this section, we illustrate the problem of simulation control and describe the main solutions proposed so far.

2.2.1 Problem: The trial and error process

How to control a physics-based animation is an old problem in computer graphics. In order to understand the different problems that arise, it is good to start from the most naive way of controlling a physics-based animation: trial and error.

Let us say we have to design an animation of a ball launched on the ground that bounces twice before hitting the center of a target on the ground.

The elastic behavior of the ball and the changes of speed make it a hard animation for a key-framing animator. Using physically-based simulation methods, the behavior takes some time to compute but can be easily solved. As the simulation is an initial value problem, the whole behavior of the ball is dictated by the initial and boundary conditions of the simulation, the material parameters of the ball and the external forces that can be set to help guiding the animation. The trial and error process consists of setting these conditions and parameters, running the simulation and correcting the parameters until the ball reaches the desired target (see Figure 2.12).

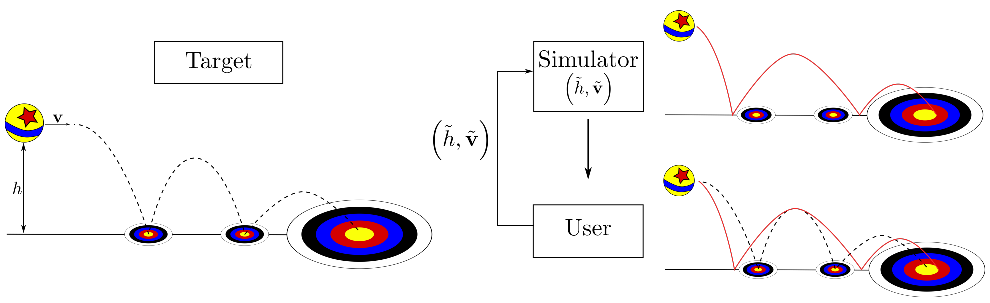

Target
v
h
$(\tilde{h}, \tilde{\mathbf{v}})$
Simulator
$(\tilde{h}, \tilde{\mathbf{v}})$
User

Figure 2.12: Trial and error process. The user runs successive simulations with different parameters such as the initial height h and velocity $\mathbf{v}$ of the ball until it reaches a desired target.

There are mainly three constraints which make this task a nightmare:

- Firstly, the computational time plays a major role. This is obvious but still important to state. A real-time simulation will allow the user to quickly explore parameters whereas an offline simulation might require days and days of tuning. Here, we can mention procedural tools which are generally much more efficient than simulation and therefore enable faster editing loops. Unfortunately they are often limited to overly restrictive models such as large open ocean surfaces [HNC02; Tes04b; JW15; Hor15].

- Secondly, the control is indirect and, most of the time, based on unintuitive parameters which require some expertise about the underlying physical model. The user cannot directly control the trajectory nor the shape of the object.

- Thirdly, many physics-based animations describe a non-linear behavior. Fluid animations are among them for instance. This non-linearity makes it very hard to choose the right parameters. Small changes can produce very different results making tedious to explore the range of possible behaviors and almost impossible to respect specific artistic directions such as timing, key positions, trajectories or shapes. Also, it prevents the user from interactively controlling a low-resolution simulation and then achieve a similar behavior with the same parameters at a higher resolution.

2.2.2 Space-time constraints paradigm

A general approach for controlling a physics-based animation is to formulate it as an optimization under constraints: "*Find the value of the parameters such that the physical behavior and user constraints are respected over the animation*" (see Figure 2.13).

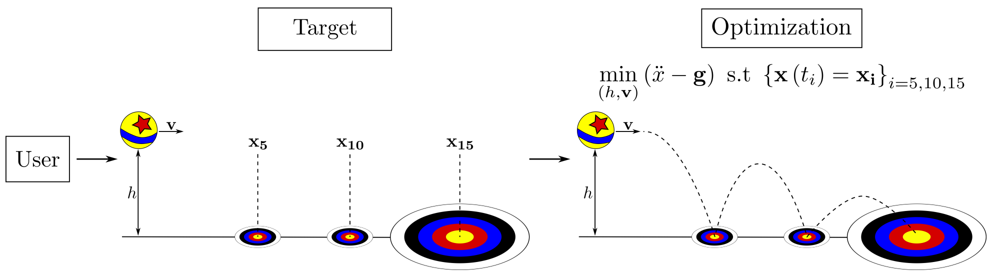

Target
Optimization
$\min_{(h,\mathbf{v})} (\ddot{x} - \mathbf{g})$ s.t $\{\mathbf{x}(t_i) = \mathbf{x_i}\}_{i=5,10,15}$
User
$\mathbf{v}$
h
$\mathbf{x_5}$
$\mathbf{x_{10}}$
$\mathbf{x_{15}}$
$\mathbf{v}$
h

Figure 2.13: Space-time constraints. An optimization problem is defined to find the value of the parameters which solve the equations of motion while respecting constraints defined by the user. In this example, the parameters are the initial height h and velocity $\mathbf{v}$ of the ball and the constraints are the position of the ball at different time. The ball has a mass of 1kg and is only submitted to the gravity $\mathbf{g}$.

Four challenges arise: What are the parameters we want to control, what are the user constraints, how to formulate them and finally how to numerically solve the problem? Witkin and Kass were among the first to introduce this formulation to the computer graphics community in their pioneer work *Space-time constraints* [WK88].

2.2.2.1 Parameters

The most common parameters are the positions of the material samples and the external forces. A common drawback of using external forces is that the necessary changes may become highly unrealistic. As an alternative, Coros et al. [Cor+12] proposed to adapt the rest shape of the object from one frame to another in order to induce internal forces that would match the user goals. When large deformation occurs it is possible that the computed solution is at the limit of the deformation that the object can reach. In order to enforce the optimization, Li et al. [Li+14] proposed to optimize material parameters as well.

2.2.2.2 Constraints

Position, velocity and density are among the constraints that are most often used for controlling an animation. Of course, they depend on the simulated object: rigids, deformable solids, smoke, liquids, etc. Certainly, position constraints are the most intuitive for the user as they can be specified using key-frames. However, designing a key-frame for a highly elastic object or viscous materials might be extremely hard to achieve. Here, there is a direct link to the works about surface modeling which propose to deform naturally an object [SA07; Hil+11]. In some cases, these deformation tools can be seen as a local static simulation of the object that computes its deformation from a displacement induced by the user. For elastic objects, such deformation tool has been proposed by Barbic et al. [BSG12]. For liquids, Pan et al. [Pan+13] propose an interactive method to deform wave shapes by sketching their profiles. Thus, direct spatial deformation is made possible. Both fit in the framework of surface modeling deformation. The only difference is that the functions which are minimized are directly derived from the internal forces acting on the object. These methods are particularly useful when interactively editing an animation, especially when they are combined with high level deformation tools such as sketching.

2.2.2.3 Numerical solution

In their work, Witkin and Kass [WK88] dealt with small systems and short simulation period. Therefore, they could afford to directly solve the optimization problem, meaning that at each iteration a whole simulation would be computed. For instance, small rigid bodies simulation could be interactively designed through this approach [Pop+00; PSE03]. For complex models and large simulation time, this approach is intractable. Windowing methods allowed to restrict the optimization to the space-time range of interests [Coh92]. In their work, [McN+04] proposed to use the adjoint method to efficiently compute gradients in liquid simulation thus improving the performance of the optimization. The approach was also used by Wojtan et al. [WMT06] for handling large particle systems. For elastic bodies, the use of reduced model allowed to achieve speed-ups of several order of magnitude, thus allowing to interactively edit a physics-based animation [BSG12; Hil+12; Hah+12]. Since, this approach has been further improved to be faster [Sch+14] and to deal with large deformations [Li+14].

2.2.3 Applications & Alternatives

The strength of the space-time constraints paradigm relies on its very general definition. Therefore, a large number of applications and methods can be seen as offsprings of this approach. In the following, we distinguish different applications of physics-based animation control and present alternatives to the above methods.

2.2.3.1 Enriching an animation with physics

Given a full animation, a simulation is run in order to enhance the input animation with detailed physically-based secondary motions. This approach was successfully applied to enhance character animation with wrinkles and folds of their skin by Bergou et al. [Ber+07]. In their work, they compute the dynamics of thin shells on top of the animation by using a multi-resolution approach. In fluid animation, details enhancing brings a lot of attention as it would be easier to set up low resolution simulation and add details on top of it without loosing the global behavior. A nice approach was proposed by Mercier et al. [Mer+15] where they solve a Lagrangian wave simulation only at the surface of the low resolution simulation.

2.2.3.2 Guiding a simulation with animation data

A large number of methods propose to guide a simulation without the need of an expensive optimization problem. Generally these methods propose to use external forces that are automatically computed from the user inputs such as key-frames. One of the first approach of this kind was proposed by Lamouret and Cani [LC96]. This approach was later extended to more involved inputs such as simulation data. In the case of fluid, user-defined velocity field [KMD06], distance fields [Yan+13] and control particles [Thü+06; MM13] were proposed to control the trajectory of fluid animation. Still in the same idea, there are many methods which propose to guide the behavior of an object using geometric proxies which are easier to control for an artist than simulation data. For example, artists can use a triangle mesh to specify a target shape which will act as an attractor by adding artificial attraction forces based on the distance to the mesh surface. Such approaches have been successfully developed to drive smoke [FL04; HK04; SY05b] and liquid simulations [SY05a; Rav+12]. Taking this strategy further, the geometric proxies themselves can be defined by a low-resolution fluid simulation. To achieve this, the artist quickly sets up a coarse simulation and uses the output geometry to guide the main features of a full resolution simulation. Several approaches modify a high-resolution smoke simulation using optimization [Nie+09; NC10], patterns extracted as skeleton [YCZ11], or sparse sampling [HK13]. For liquid simulations, Nielsen and Bridson [NB11] propose to restrict the high resolution simulation to a thin layer around a guiding coarse animation. Although each of these approaches are able to successfully guide an animation, they do not enable direct control of the resulting animation. Designing precise timing or feature scaling would therefore still require iterative trial-and-error steps to converge toward a desired animation.

2.2.3.3 Example-based simulation

Instead of controlling the trajectory of an object, one might want to control how it deforms when it collides with other objects. In exampled-based simulations, a set of examples that represent the desired deformations is used to build a space of preferred deformations from where internal forces will be deduced. This method was first introduced by Martin et al. [Mar+11] for elastic deformations. Jones et al. [Jon+16] propose a similar method to easily incorporate plastic deformations in rigid bodies simulations.

2.2.3.4 Animation sampling

In multibody systems, the space-time constraints paradigm can hardly be used. In cause, the large number of discontinuous contacts events which makes the optimization problem particularly difficult to solve. In contrast, multibody simulators are particularly fast, so fast that it is possible to run in parallel a large number of simulations. In their work, [CF00] and [TJ07] leverage this performance to sample the space of parameters of an animation and thus finding those which satisfy the user constraints.

2.2.3.5 Animation editing

In contrast with simulation control, animation editing consists of deforming in space and time the output of a simulator without the need to re-simulate. Closely related to surface modeling, it has to take into account the temporal dimension and its relation with space in order to propose new editing tools. In these approaches, finding a natural way to deform an animation and ensuring temporal consistency are amongst the main challenges. Very few methods have been proposed, however it represents a much faster approach to edit animations. Among the most interesting approaches, Pighin et al. [PCS04] propose to build a space-time parametrization of a smoke animation using advected radial basis functions. The parametrized data are then deformed using a trajectory-based editing tool. Schpok et al. [SDE05] proposed to extract and parametrize features such as vortices, uniform advection, sinks and sources to allow the user to modify the parameters in a smoke simulation. For animations of liquid, Raveendran et al. [Rav+14] propose a semi-automatic method to match two animations and smoothly interpolate between them. This approach allows a quick exploration of the parameter space such as boundary conditions or viscosity parameter and therefore to produce a large number of new animations without the need for re-simulation.

2.2.4 Conclusion on simulation control

All the methods mentioned above make a trade-off between realism and control. On one hand, realism requires a high computational cost and makes methods such as space-time constraints unattractive for interactive use. On the other hand, giving the user full control may introduce unrealistic and unpleasant behavior. Very few works explored the edit of an existing animation and how to extend classical modeling techniques to animated content. We think that this approach would allow the interactive design of physics-based animation without the need to re-simulate. In Chapter 6, we will present a system for the design of animations of liquid based on this idea.

2.3 Chapter conclusion

In this chapter, we proposed an introduction to continuum mechanics and described a variety of methods for controlling simulations. Most of the time, the practibility of these models and methods depends on the simulated phenomena and the required computational time. Adaptivity is a general strategy to achieve large scale simulations and resolve complex behaviors. In Chapter 3, we propose a detailed review of these models. In Chapter 4, we extend an adaptive model, initially proposed for nanosystems simulations, to speed-up particle simulations in computer graphics, Then, we study how to perform detailed topological changes with the frame-based model in Chapter 5, an interesting approach which ensures a very small number of degrees of freedom and therefore interactive performances. Finally, we explore a new way of editing fluid animations in Chapter 6, where the user can manipulate sub-parts of the animation interactively without the need to re-simulate.

Chapter 3

Adaptive techniques and models for physics-based animation

Complex real-world behaviors may exhibit multi-scale phenomena in space and time, large deformations or topological changes. Sophisticated physical models involving a tremendous number of degrees of freedom are required to accurately reproduce these phenomena. This comes at a high memory and computational price that can quickly become intractable or prevent from an interactive usage. Adaptive models provide a general paradigm to solve these efficiency goals.

We call a model or simulation method *adaptive* if it automatically adapts the underlying mathematical representation, data structure and/or algorithm at run time, based on the evolving state of the simulated system. The adaptation is designed to meet a given criteria which depends on the application. Examples of frequently used criteria include reducing the overall computational complexity without loss of quality, improving the quality of real-time simulation, or simulating more precisely the parts of the scene which the user is currently interacting with.

In this chapter, we review the different families of adaptive methods and present the way they have been applied in the different domains. In Section 3.1, we present time adaptive techniques such as dynamic time stepping and freezing techniques. Then, we focus on spatially adaptive techniques. Section 3.2 discusses the most popular approach of spatial adaptivity: geometric adaptivity(h-adaptivity where h classically refers to the size of elements in the finite element method), which refers to varying the discretization resolution via refinement and coarsening strategies. This chapter is subdivided according to the types of adaptive spatial discretization. Section 3.3 covers other spatial adaptivity approaches: basis refinement which adapts the number of bases, their order (p-adaptivity where p stands for polynomial), the basis functions themselves using enrichment, or, in subspace simulation, the deformation modes of the basis; moving grids methods (r-adaptivity where r stands for relocation) which relocate nodes without changing their connectivity; and mixed models which selectively apply a combination of different computational models. This organization reflects the taxonomy we propose in Figure 3.1. We finally conclude and sketch future research avenues in adaptive simulation.

This chapter was published in *Computer Graphics Forum, 2016* [Man+16].

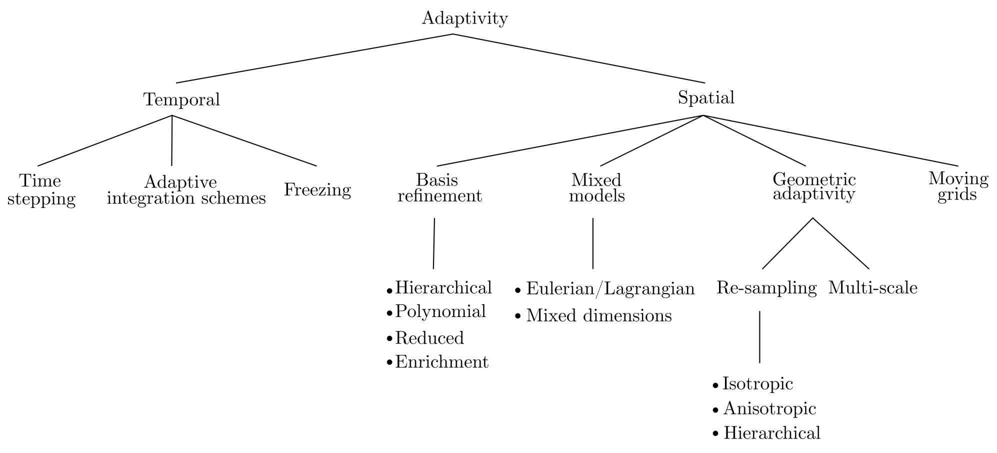


Figure 3.1: Taxonomy of adaptive physically-based models in computer graphics. Temporal adaptivity (see Section 3.1) and geometric adaptivity (see Section 3.2) are the two most common strategies. Basis refinement (see Section 3.3.1), moving grids (see Section 3.3.2) and mixed models (see Section 3.3.3) are less common but proved to be very promising.

3.1 Temporal adaptivity

Animating any simulated object requires integrating its equations of motion over time. There are many reasons why this integration procedure may need to adapt to the circumstances of the simulation, whether for accuracy, consistency, or stability. For instance, a system entering a highly nonlinear regime, such as during fracture, requires smaller time steps to maintain the desired degree of accuracy. Alternatively, adaptive time steps may be necessary to prevent the system from entering an invalid state: for example, many collision resolution schemes maintain the invariant that the system is never allowed to enter an interpenetrating configuration, which can require adjusting the time step according to the frequency of collisions. Finally, the stability of continuum-mechanical simulations is closely tied to the relationship between the time step length, the spatial resolution, and the speed of propagation of information in the system, so if either of the latter change (in the presence of *spatial* adaptivity or fast-moving flow) the time step must be adapted as well.

Techniques for temporal adaptivity fall into two main categories. One approach focuses on time resolution, that is, how to choose an appropriate step length for time integration. The second focuses on integration techniques themselves, seeking to switch between different integration schemes depending on the local context. This section explores both of these possibilities for temporal adaptivity.

3.1.1 Adaptive time step selection

Time step criteria First, let us focus on the simulation of continuous media, like fluids and elastic solids, whose models are governed largely by hyperbolic partial differential equations. Here the most prominent criterion for time step selection is the Courant-Friedrichs-Lewy (CFL) condition [CFL28]. To understand this condition, we first note that the solution of a partial differential equation at some point depends on a particular subset of initial or boundary data; we call this subset of data the *domain of dependence*. The CFL condition states simply that for any numerical scheme to converge to the true solution, the domain of dependence of the numerical scheme must, in the limit, contain the true domain of dependence of the underlying differential equation. (Otherwise, one could perturb the initial data in the region outside the numerical domain of dependence and change the true solution without affecting the computed one.) The CFL condition can also serve as a stability criterion thanks to the Lax-Richtmeyer equivalence theorem [LR56; Str04], which states that a consistent finite difference method for a well-posed initial value problem is convergent if and only if it is stable.

For example, for the classical wave equation $\partial_t^2 f = c^2 \nabla^2 f$, the domain of dependence of any point $(\mathbf{x}, t)$ includes only points $(\mathbf{x}_0, t_0)$ with $\|\mathbf{x} - \mathbf{x}_0\| \leq c(t - t_0)$, because information propagates only at the wave speed c. Consequently, the time step Δt must be small enough to prevent information from propagating outside the spatial stencil over a single time step. In particular, for the first-order explicit finite difference scheme applied to the wave equation, where over each time step a grid cell is only affected by adjacent grid cells (separated by Δx), the CFL condition requires that $c\Delta t \leq \Delta x$.

In general, the CFL condition takes the form

$$C \equiv \frac{c\Delta t}{\Delta x} \leq C_{\max}, \tag{3.1}$$

where c is the speed of information propagation, and $C_{\max}$ is a method-dependent constant which depends on the size of the finite-difference stencils. The dimensionless ratio C is known as the CFL number or the Courant number. Note that implicit methods are not restricted by the CFL condition because the solution at any point at the end of the time step depends on the values at *all* the points at the beginning of the time step, so the numerical domain of dependence is effectively infinite.

The CFL condition can be a useful heuristic even in situations where it does not directly apply. In computer graphics, semi-Lagrangian advection [Sta99] has been a popular scheme for solving the advection equation, as it is unconditionally stable and not restricted by the CFL condition [Bri08]. Nevertheless, excessively large time steps can lead to undesirable artifacts such as numerical dissipation and volume loss. Foster and Fedkiw [FF01] advocate limiting the time step size using (3.1) with c being the maximum of the flow speed $\|\mathbf{u}\|_\infty$ over the domain, and $C_{\max} = 5$.

In Smoothed-Particle Hydrodynamics (SPH), each particle is affected by other particles within its smoothing radius h, analogous to the grid separation Δx in finite-difference methods. Consequently, time step selection criteria based on the CFL condition can be applied. For pressure waves in compressible SPH, we continue to have

$$\Delta t \le C_{\max} \frac{h}{c}, \tag{3.2}$$

and values of $C_{\max}$ between 0.25 and 0.4 have been used [Mon92; DC99]. Fast relative motion of particles, which occurs especially during fluid-fluid or fluid-solid collisions, can also produce artifacts due to interactions not considered in (3.2). Therefore, one can introduce additional time step constraints based on the instantaneous acceleration $\mathbf{a}$ and divergence of velocity $\nabla \cdot \mathbf{v}$ [Mon92; DC99]:

$$\Delta t \le \Lambda \sqrt{\frac{h}{\|\mathbf{a}\|}}, \tag{3.3}$$

$$\Delta t \le \frac{\Gamma}{|\nabla \cdot \mathbf{v}|}. \tag{3.4}$$

Λ and Γ are dimensionless constants which Desbrun et al. [DC99] set to 0.5 and 0.005 respectively. Equation (3.3) can in fact be interpreted as a CFL-type condition comparing the size of the spatial neighborhood, h, to a particle's relative displacement $\frac{1}{2}\|\mathbf{a}\|\Delta t^2$ due to acceleration over time Δt. Finally, we also point out that these criteria (3.2)–(3.4) may either be applied globally by taking the minimum of all particles' allowed time steps, or locally on a per-particle basis (as we discuss below).

When adapting the time step based on higher-order derivatives such as acceleration (3.3), care must be taken because such quantities can be much noisier than the system variables themselves, causing large fluctuations in Δt. Ihmsen et al. [Ihm+10] have observed that in incompressible SPH simulations, these time step fluctuations can lead to spurious density shocks that destroy the convergence of the simulation. A solution is to change Δt gradually rather than instantaneously: if the system violates any of the desired time step criteria, Δt is decreased by a small amount ([Ihm+10] use 0.2%), otherwise, if it is well within all the criteria, Δt is increased by the same amount. This strategy causes the time step to change smoothly towards the ideal step length. On the other hand, it may also prevent the time step from changing quickly enough to resolve sudden shocks, such as those from high-velocity impacts. Therefore, Ihmsen et al. introduce an additional procedure that detects shocks if the density error increases suddenly; if so, the simulation is rewound two time steps and resumed with a sufficiently small Δt.

Shock detection can be considered an example of an *a posteriori* time step adaptation strategy, where undesirably large time steps are detected and rewound. This kind of approach is useful whenever it is costly or difficult to estimate the right time step length in advance. Instead, we simply perform a time step with the current estimate of Δt, and then test whether it adequately resolves the motion of the system. If not, we reject the step, decrease the time step by setting $\Delta t \leftarrow \Delta t/\alpha$ for some factor $\alpha > 1$, and try again. When several time steps in a row are successful, we increase the time step, $\Delta t \leftarrow \alpha \Delta t$.

Bridson et al. [BFA02] use this approach for efficient collision handling in cloth simulation, using a combination of inelastic repulsion forces and a robust collision resolution algorithm. Inelastic repulsions are much more inexpensive than full collision resolution, but as they only check proximity at discrete points in time, they can easily fail to prevent interpenetrations if the cloth moves too far in a single time step. If this happens, the time step is rejected and Δt is reduced; collision resolution is only triggered if after multiple failures the time step falls to a specified minimum size. The collision resolution step incorporates rigid impact zones [Pro97], which can be seen as a freezing technique and is discussed in Section 3.1.2. Bargteil et al. [Bar+07] simulate plastic flow using a finite element mesh, where excessive deformation of elements can be problematic. They reject a time step if any edge changes significantly in length, or a sudden acceleration takes place. Both methods discussed here use $\alpha = 2$, that is, time steps are halved or doubled as needed.

The main drawback of this *a posteriori* strategy is that the whole system must be globally rolled back to its previous safe state, even if it was caused by a localized event. In rigid bodies simulation, this challenge was addressed by Mirtich [Mir00] to efficiently handle collisions. Inspired by the work of Jefferson et al. [Jef85], he proposed a *time warp* algorithm to asynchronously handle collision events. Here, the integration of a rigid body is interrupted only when resolving an event that concerns it. This technique can be seen as a local time stepping technique, and inspired works on *asynchronous variational integrators* (AVIs) which we discuss later in this section.

Global time stepping The simplest way to perform adaptive time stepping is to choose the time step that is safe for the entire simulation domain, and perform integration for the entire system using that time step. That is to say, given a time step criterion (or criteria) such as (3.2)–(3.4) that can be evaluated locally, one evaluates the permissible time step Δt_i at all simulation points i, and steps the entire system forward by a time step of length $\Delta t = \min_i \Delta t_i$. For methods that use implicit integration or other globally coupled schemes, such as grid-based fluids with a global pressure solve, this is the only possible approach. For this reason as well as for its conceptual and practical simplicity, global time stepping is probably the most widely used form of temporal adaptivity in practice.

It is worth pointing out here that adaptive time stepping is not a free lunch for all time integration schemes. The Verlet, or leapfrog, scheme is second-order accurate, and has excellent energy conservation properties thanks to its symplectic nature, but both these features rely on the time step being fixed. To maintain second-order accuracy with a variable time step, Bridson et al. [BMF03] proposed a time integration scheme that combines a leapfrog scheme for position with an implicit trapezoidal rule for velocity:

```
1: $\tilde{v}^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, \tilde{v}^{n+1/2})$    ▷ implicit
2: $x^{n+1} = x^n + \Delta t \tilde{v}^{n+1/2}$    ▷ explicit
3: $v^{n+1/2} = v^n + \frac{\Delta t}{2} a(t^n, x^n, v^n)$    ▷ explicit
4: $v^{n+1} = v^{n+1/2} + \frac{\Delta t}{2} a(t^{n+1}, x^{n+1}, v^{n+1})$    ▷ implicit
```

Maintaining symplecticity is much more challenging, as naively varying the time step can lead to instabilities and inconsistent energy behavior [Har+09]. Much more elaborate time stepping schemes are needed to recover energy preservation, such as the asynchronous variational integrators discussed below.

Local time stepping In complex simulation scenarios, different regions of the simulation domain may have very different time step requirements. For example, resolving challenging collision and contact scenarios requires careful time stepping, but only for the parts of the system that are affected by the contact. Similarly, in simulations with adaptive spatial resolution, the CFL condition requires finer-resolution regions to take smaller time steps. It can become intractable to simulate the whole model in lockstep using the most conservative time step. Instead, it is desirable to perform local time stepping, integrating each element of the simulation at its own pace.

Explicit integration schemes can readily incorporate local time stepping. We illustrate this with a generic two-dimensional system,

$$q_1'(t) = f_1(t, q_1(t), q_2(t)), \tag{3.5}$$
$$q_2'(t) = f_2(t, q_1(t), q_2(t)). \tag{3.6}$$

If at time t, q_2 requires a very small time step Δt_2, one can still integrate q_1 with its own time step Δt_1, giving the following.

$$q_1(t + \Delta t_1) = q_1(t) + \Delta t_1 f_1(t, q_1(t), q_2(t)). \tag{3.7}$$

Meanwhile, as q_2 takes multiple steps to cover the same time interval, it will require values of q_1 at intermediate times $t+\Delta t_2$, $t+2\Delta t_2$, and so on; these can be linearly interpolated from $q_1(t)$ and $q_1(t + \Delta t_1)$. Equivalently, to simplify bookkeeping, one can take the same small time steps Δt_2 for both q_1 and q_2, but only evaluate q_1' at the first step and hold it fixed until a time Δt_1 has been covered. This approach has the same computational advantage because evaluation of f is the most expensive part in explicit methods.

Early work on SPH [DC96; DC99] recommended this approach for local time stepping, using the CFL condition as the time step criterion. The same technique was also used to simulate elastic bodies using spatially adaptive finite element meshes [Deb+01], discussed in more detail in Section 3.2.2. For a linear elastic material with density ρ and Lamé coefficients λ and μ, the upper bound on the time step for an element can be approximated by

$$\Delta t \leq h\sqrt{\frac{\rho}{\lambda + 2\mu}} \tag{3.8}$$

where h is a measure of the size of the element, such as its inradius: skinnier elements or stiffer materials require smaller time steps. To improve the parallelization of local time stepping for SPH fluids, Goswami and Batty [GB14] divide the computational domain into blocks and choose time steps independently per block.

Asynchronous variational integrators (AVIs) studied by Lew et al. [Lew+04] are a family of time integration schemes that provide excellent energy conservation behavior while allowing different elements of the system to use different time steps. Unlike the local time stepping model described above, AVIs associate a time step with each *force* rather than each variable. Thus each force term applies a series of impulses to its associated nodes. The time step for a particular term must remain constant throughout the simulation, but different forces may have very different time steps. This approach is implemented as an event-driven simulation loop, using a priority queue to schedule the updates for all the forces in order. Thomaszewski et al. [TPS08] applied this approach to cloth simulation with a finite element triangle mesh, choosing time steps independently for each element using the CFL criterion (3.8). AVIs are known to exhibit "resonance instabilities" that can potentially cause the energy to increase without bound; however, these instabilities tend to be extremely weak in solid mechanics problems [FDL07] and have not been observed to cause difficulties in computer graphics [Har+09].

The energy conservation properties of AVIs hinge on the regular spacing of the impulses applied by each force term, which makes collisions challenging to incorporate: naively applying contact forces at the moment of collision breaks the periodicity and destroys energy conservation, while applying contact forces at regular intervals risks missing collisions. Recent work [Har+09; Ain+12] addresses this problem by replacing each contact force with a sum of stiffer and stiffer penalty layers with smaller and smaller time steps, which together are guaranteed to prevent interpenetration. While the number of penalty layers is conceptually infinite, any given collision can only activate a finite number of penalty layers, so the actual amount of computation is finite. Initial work by Harmon et al. [Har+09] used kinetic data structures to detect all collision events in advance. Ainsley et al. [Ain+12] instead adopt a speculative approach based on the time warp algorithm [Jef85; Mir00], analogous to the *a posteriori* time step adaptation techniques discussed previously. An interval of time is first simulated without considering any new collisions; then, collision detection over the simulated interval is performed, and if new collisions are found, those force terms are activated and the interval is re-simulated. Speculative simulation greatly reduces the computation time spent in bookkeeping and collision detection, and also allows for easy parallelization.

3.1.2 Adaptive integration

Adaptive choice of time integration scheme In some cases, such as when the system involves highly stiff modes or poorly conditioned elements, an adaptive choice of time step is no longer the most efficient strategy. The time step requirements for explicit integration may become extremely restrictive. Instead, one may locally change the integration scheme to deal with the problematic components, for example by switching to an implicit method or a nonlinear one. By doing so adaptively, one can continue to use an inexpensive explicit integration scheme for the remainder of the system.

In the finite element method, ill-shaped elements impose severe restrictions on the allowed time step for explicit integration (3.8). When the object undergoes topological changes such as cutting or fracture, it can be difficult to avoid introducing such elements. As an alternative to local time stepping, where ill-shaped elements would have to be simulated with extremely small time steps, Fierz et al. [FSH11] propose an element-wise implicit-explicit (IMEX) scheme. Here the same time step Δt is used for the entire system, but ill-shaped elements that would be unstable if explicitly integrated over Δt are instead simulated with an unconditionally stable implicit scheme. Nodes that are not adjacent to any ill-shaped element are integrated explicitly, then held fixed as boundary conditions for implicit integration of the remaining nodes. As long as the number of ill-shaped elements is low, this relaxes the time step restriction faced by explicit integration, while minimizing the computational cost and numerical dissipation associated with implicit integration.

Thin materials such as hair exhibit a high stiffness in their stretching modes, but pose the additional challenge that their collision response is highly nonlinear due to the presence of rotation. Therefore, depending on the amount of bending, an implicit first-order model for collisions may fail to capture the correct response and lead to instabilities. Kaufman et al. [Kau+14] propose an algorithm that adaptively chooses the *degree* of nonlinearity in each contact resolution step to safely resolve the collision. Specifically, they adapt the number of constrained Newton iterations used to solve the nonlinear contact model, terminating when the stretch over all affected edges is sufficiently reduced. This allows for large simulation time steps in the face of many energetic collisions, while efficiency is maintained because most collisions require only a single iteration (equivalent to a linearly implicit step).

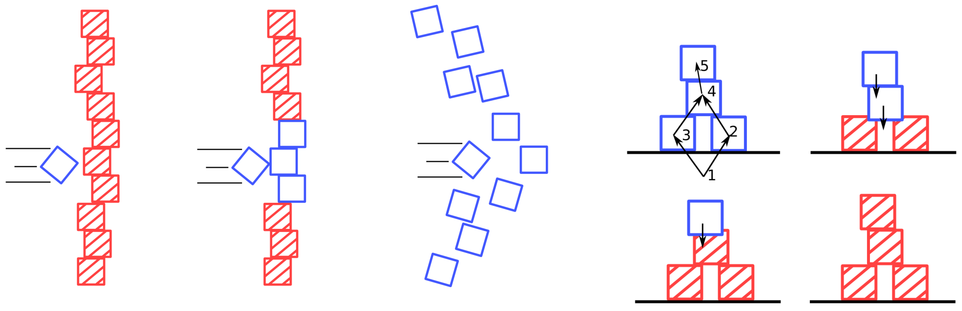
Freezing techniques Freezing techniques, also called sleeping techniques, lie in between temporal and spatial adaptability: the degrees of freedom that are considered unimportant in the simulation are kept constant in time for a specified duration. This can be seen as animating them at a much larger time scale, or as temporarily deactivating them. No memory is saved while doing so, but computation time is reduced. These techniques are useful in situations where the spatial domain is large and filled with many quiescent objects. Therefore, game environments and surgical simulations are perfect candidates for these methods. Conversely, freezing techniques may not be useful in highly dynamic situations where most of the degrees of freedom are active most of the time. In addition to defining good freezing criteria, the main challenge of freezing techniques is to design a reactivation process of the frozen degrees of freedom that ensures plausible subsequent motion.

In rigid objects simulation (see the survey of Bender et al. [Ben+12]), important computational resources are dedicated to solving contacts between the different objects of a scene. This process becomes unnecessarily expensive when stacking occurs and nothing moves. In these cases, freezing techniques prove to be useful for saving computational time without compromising the plausibility of the simulation.

Schmidl [Sch02] uses a heuristic based on kinetic energy to determine whether to freeze a body or not.

$$\frac{1}{2}m\mathbf{v}^2 + \frac{1}{2}\omega^T\mathbf{I}\omega < \frac{\mathbf{p}_g^2}{2m} \tag{3.9}$$

Where $\mathbf{v}$ and ω respectively denote the linear and angular velocity of the rigid body of mass m and inertia tensor $\mathbf{I}$, and $\mathbf{p}_g = m\mathbf{g}\Delta t$ is the momentum that the body accumulates during a time step Δt from gravity. If condition (3.9) is fulfilled by the body during a user-defined number of consecutive time steps, then it is frozen. Note that a single frozen body by itself does not save computation time. Time is saved when multiple neighboring objects become frozen, as their contact forces no longer need to be computed. The only way for a frozen rigid body to be reactivated is when it receives a large impulse during a collision. Once it is reactivated, the information is propagated to all its direct neighbors and a given number of indirect neighbors, potentially awakening them (see Figure 3.2a).



(a) (b)

Figure 3.2: Freezing techniques applied to rigid bodies stacking. At the stage of contact solving, those methods can save computational time while ensuring a plausible motion. In (a), quiescent rigid bodies are frozen (in hatched red) in a stacking configuration. A collision event then awakes one of the rigid bodies, and the active rigid body (in blue) propagates the information to its neighbors. In (b), a contact graph stores stack ordering. During shock propagation, a bottom-up traversal is performed and objects at each level are frozen by assigning an infinite mass.

Freezing techniques have been used as a failsafe procedure for resolving collisions and contact between rigid or deformable bodies. When many interacting contacts are present, iterative processes for collision response can require an excessively large number of iterations to converge, and early termination leaves the system in an interpenetrating state. Freezing procedures are attractive in this context as they can guarantee elimination of interpenetrations. However, they also introduce loss of kinetic energy by treating contacts as fully inelastic, and are therefore only invoked as a last resort after multiple iterations of a physically correct solver have failed to resolve the collisions. For rigid body contact, Guendelman et al. [GBF03] propose a shock propagation strategy which freezes bodies progressively from the ground up. They start by building a directed acyclic graph consisting of "levels" of objects that are resting on objects of lower levels (cyclic dependencies are grouped into the same level). A single bottom-up traversal of the graph resolves contacts level by level, assigning infinite mass to lower-level objects for which contact is solved (see Figure 3.2b). Rigid impact zones, described by Provot [Pro97] and Bridson et al. [BFA02], are extensively used to resolve collisions in cloth. Nodes involved in multiple interfering collisions are collected into disjoint sets called "impact zones", constructed by merging zones if their nodes are involved in the same node-face or edge-edge collision. Each impact zone is made rigid by replacing the motion of its nodes with a rigid body motion while preserving the total linear and angular momentum. This process eliminates collisions within the zone, but may introduce new collisions with nearby elements outside the zone. Therefore, one must perform collision detection again and grow the impact zones if new collisions are detected, iterating this process until no new collisions occur.

Freezing techniques were also used for articulated rigid bodies, both for dynamics [RGL05] and quasi-statics [RL06]: joints are activated or deactivated based on user-defined error metrics, leading to a simplification of the whole model and a sub-linear computational complexity.

Denoting by $\ddot{\mathbf{q}}^C = (\ddot{\mathbf{q}}_1, \ldots, \ddot{\mathbf{q}}_{N_C})^T$ the composite acceleration of an articulated body C, where N_C is the number of joints in C, the *acceleration metric value* of C is defined as

$$\mathcal{A}(C) = \sum_{i \in C} \ddot{\mathbf{q}}_i^T \mathbf{A}_i \ddot{\mathbf{q}}_i,$$

where $\mathbf{A}_i$, $i \in C$, are symmetric, positive definite $d_{J_i} \times d_{J_i}$ *weight matrices*, and d_{J_i} is the number of degrees of freedom of joint i in C. The weight matrices $\mathbf{A}_i$ are required to depend at most on joint positions, and the simplest choice is the identity matrix. The key to the sub-linear complexity is the demonstration that the acceleration metric of each sub-articulated body is a quadratic function of the forces applied to its handles (i.e. the free rigid bodies used to assemble sub-articulated bodies in Featherstone's methodology [RGL05]). As a result, the acceleration metric may be computed *before computing the joint accelerations themselves*. This is used during the top-down pass of Featherstone's divide-and-conquer algorithm to determine which joint accelerations should be computed because they are sufficiently large [RGL05]. For example, if the acceleration metric of the complete articulated body is small, then all joint accelerations are small and joint velocities are constant for the current time step. A similar metric is built for joint velocities to determine which joint positions should remain constant, and thus which inter-body forces and inertial terms should be updated. Once all position-dependent terms are up to date, the motion metrics are available for the next time step, and a new set of active joints can be determined.

It was shown that this approach could also help to speed up continuous collision detection for articulated bodies [KRK08a] and haptics rendering [MR07], as well as enable view-dependent articulated-body dynamics by combining

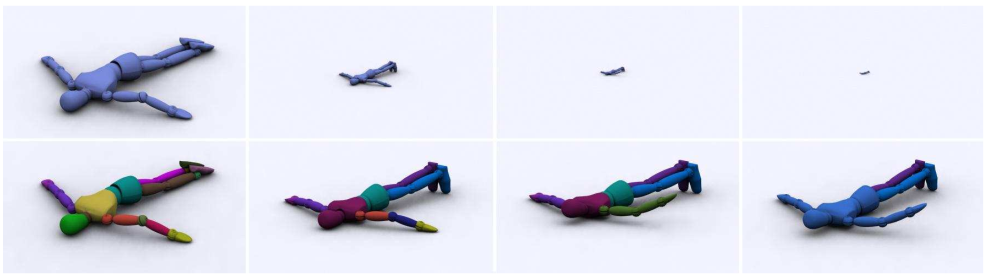


Figure 3.3: View-dependent dynamics of articulated bodies. Top: the algorithm proposed by Kim et al. [KRK08a] automatically simplifies the dynamics of a falling character as its distance to the viewer increases. Bottom: corresponding rigidification at this time step (one color per rigid group).

motion metrics with visibility estimates [KRK08b] (see Figure 3.3). Gayle et al. [GLM06] demonstrate how to perform contact handling with such an adaptive articulated-body method, which is used as the core of a physics-based sampling algorithm for highly articulated chains [Gay+07] and cable route planning [KGL07].

Recent work has sought to apply freezing techniques to accelerate SPH fluid simulation. In these methods, particles are divided into two sets: a set of *active* particles following a classical simulation step, and a set of *inactive* particles that are skipped in the simulation. As physical quantities in SPH are interpolated from neighbors, inactive neighbors of active particles also need to be updated to ensure that active particles compute correct physical quantities. Computation time is saved for inactive particles that only have inactive neighbors. The main challenge in these methods is the definition of the process to transform a particle from the inactive set to the active one and vice versa. Indeed, as SPH is sensitive to particle distribution, an inadequate transition of state can directly lead to instabilities. Additionally, a judicious choice of criterion to decide whether a particle should be active or not is essential.

Goswami and Pajarola [GP11] propose a simple method that evaluates the active status of particles at each time step. Particles are marked as active if they are close to the boundary or if their velocity exceeds a threshold. Inactive neighbors of such particles are also added to the active set, and continue with their last active velocity. All other particles are considered inactive. Unfortunately, the resulting method does not obey Newton's third law, resulting in some loss of momentum. In the context of molecular simulation, Artemova and Redon [AR12] propose a fundamentally different approach which ensures momentum conservation. In Chapter 4, we extend their work to computer graphics.

3.2 Geometric adaptivity

Geometric adaptivity describes various techniques that adapt the spatial resolution of a model by refining and coarsening its discretization. These techniques are also referred in the literature as *adaptive spatial refinement*. However, as they include coarsening as well, we adopted a more general term.

Geometric adaptive techniques have two major components: a refinement criterion that determines where higher resolution is needed, and a refinement/coarsening scheme that modifies the discretization to match the desired resolution. Both physical and visual criteria have been employed in existing work, and we discuss them in more detail below. The refinement scheme itself essentially depends on the type of spatial discretization. In the following subsections, we will deal with the three major kinds of discretization separately: structured meshes and grids, unstructured meshes, and meshless models.

Refinement criteria The choice of refinement criteria plays a major role in the quality of the resulting simulation. Many techniques use simple heuristics such as the distance to boundaries, surface curvature, and the presence of contacts. However, some authors have shown that employing criteria that are more closely tied to the dynamics of the system can be important in many contexts.

In elastic and plastic solids, the stress and strain in an element characterize the amount of local deformation. Therefore, the values and gradients of these quantities are often used to control refinement. Wu et al. [Wu+01] describe several different error estimators of this type and discuss their relative advantages, drawbacks, and performance. Going beyond estimating a scalar error, Wicke et al. [Wic+10] define a metric tensor $\mathbf{M}$ that approximates the spatial variation of strain by comparing deformation gradients of adjacent elements. The matrix $\mathbf{M}$ is defined in such a way that it has large eigenvalues in directions in which the deformation changes most rapidly, and can thus be used to control anisotropic refinement (see Figure 3.4).

If a multi-grid algorithm is used, the difference between the solution at the current level, $\mathbf{x}^j$, and the one prolonged from the next coarser level, $\mathbf{P}^j\mathbf{x}^{j+1}$, provides a natural measure of the quality of $\mathbf{x}^{j+1}$. Otaduy et al. [Ota+07] weigh the error by the local system matrix $\mathbf{A}$ to reduce possible popping in stiff scenarios, leading to the error metric

$$e = \|A^j(\mathbf{x}^j - \mathbf{P}^j\mathbf{x}^{j+1})\|, \tag{3.10}$$

and perform refinement if e exceeds a predefined threshold.

Lower-dimensional bodies, like wires, strands, cloth, and thin sheets, undergo not just stretching but also bending (and torsion, in the case of one-dimensional strands). While the stretching forces within the material are much stiffer than the bending forces, the bending deformation can often be more

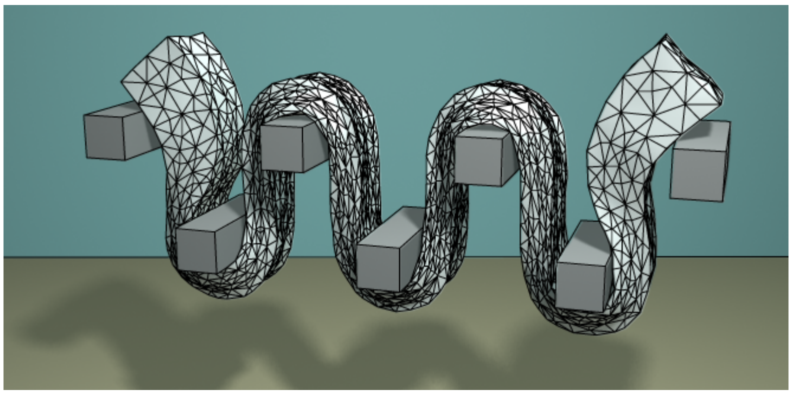


Figure 3.4: Elastoplastic simulation with dynamic local remeshing [Wic+10]. By using the strain gradient as the refinement criterion, regions undergoing severe deformation are refined locally.

visually important, and also introduces geometric nonlinearities that must be carefully resolved. In this context, geometrical curvature and the presence of contacts (which are likely to induce bending) are the most commonly used refinement criteria. However, the interaction between stretching and bending leads to additional considerations.

First, in the context of cloth simulation, Simnett et al. [SLD09] and Narain et al. [NSO12] point out that elements under compression are likely to buckle and should therefore also be refined, otherwise the wrinkles that would arise in subsequent time steps will fail to be represented on the coarse mesh. By considering the trade-off between bending and stretching energy, Narain et al. estimate that a sheet under compressive strain ϵ is likely to form wrinkles of width proportional to $\sqrt{k_b/(k_s\epsilon)}$ where k_b and k_s are the bending and stretching stiffnesses respectively, providing an estimate of the extent of refinement necessary. In the physics literature, Cerda and Mahadevan [CM03] have provided a detailed semi-analytical analysis of wrinkle geometry that is valid far from the small-deformation limit, and may be useful for future work in graphics.

Second, finer meshes allow higher-frequency modes of transverse oscillation, leading to time step constraints and stability problems that can be fatal for interactive applications. To ensure stability, Servin et al. [SLN08] propose a *coarsening* criterion, reducing the resolution of the mesh so that only those oscillations whose frequency is lower than the time stepping rate can be represented. For simulation of systems with stiff wires, they estimate the maximum frequency of oscillations in a wire discretized with n nodes as

$$\omega_{\max} \approx 2(n+1)\sqrt{\frac{f}{mL}}, \tag{3.11}$$

where f is the tension in the wire and m and L are its mass and length. Requiring this frequency to be lower than $1/\Delta t$ gives an upper bound on n for each wire.

In fracture simulation, the stress forms a singularity at the crack tip, which must be resolved accurately with a fine resolution mesh for realistic crack paths to be obtained. If the crack origin is known *a priori*, one can track the crack path as the fracture proceeds and refine the mesh based on the distance from the crack [BDW13]. However, if fracture is allowed to originate anywhere, it is necessary to refine the mesh wherever stress is sufficiently high, i.e. close to the material's strength τ, because such regions may generate new cracks. Koschier et al. [KLB14] refine elements whose tensile stress σ exceeds a specified fraction of τ. Pfaff et al. [Pfa+14] choose the desired resolution of the mesh to be proportional to tensile stress by requiring the length of each edge $\mathbf{e}$ to satisfy

$$\|\mathbf{e}\| \leq \max\left(\frac{\tau}{2\sigma}, 1\right) \ell_{\text{min}} \tag{3.12}$$

where ℓ_{min} is a user-specified refinement limit. This approach allows the mesh to be coarsened again after the crack tip has passed.

It is also possible to perform view-dependent adaptivity by modulating the refinement criteria based on visibility and distance from the camera. Such techniques have been applied to the simulation of wires [SLN08] and cloth [KNO14]. New issues arise in such contexts, such as preventing artifacts when coarser regions come into the field of view and must be refined: Koh et al. [KNO14] achieve this by smoothly increasing the resolution in advance based on the known camera path.

Many adaptive methods for simulation of liquids have relied on the distance to the free surface as the sole refinement criterion. However, much greater adaptivity is possible by observing that in regions where the surface is flat and does not exhibit detailed motion, it can also be coarsened without significantly affecting the flow. Adams et al. [Ada+07] propose a purely geometrical criterion based on an "extended local feature size" which measures the distance to the surface and to the medial axis of the fluid volume. This criterion assigns coarse resolution both far from the surface and near thick, flat surfaces. However, it does not take the motion of the fluid into account. To detect regions with significant flow detail, Hong et al. [HHK08] use a "deformation factor" that locally estimates the Reynolds number,

$$\text{Df} = \frac{(\mathbf{u} \cdot \nabla)\mathbf{u}}{\nu \nabla^2 \mathbf{u}}, \tag{3.13}$$

where $\mathbf{u}$ is the fluid's velocity field and ν is its viscosity. Ando et al. [ATW13] use a flexible sizing function that combines multiple criteria to set the desired resolution. Among them the depth of the liquid, the camera viewpoint, the fluid surface curvature and the norm of the strain rate, $e = \|\nabla \mathbf{u}\|_F$, in order to preserve detailed motions.

3.2.1 Structured meshes and grids

Spatially adaptive simulation techniques can often benefit from symmetry or structure, at the expense of flexibility. Techniques using hexahedral elements or finite differences can use quadtrees or octrees to add spatial adaptivity. Alternatively, techniques that use a volumetric tetrahedral mesh, like some finite element methods or mass-spring systems, can easily take advantage of structured meshes based on lattices. Special techniques can also combine grids with “tall cells” or far-field grid structures, as discussed at the end of this section.

The techniques described in this section are useful when one wishes to speed up simulations by using local spatial adaptivity without completely committing to an unstructured mesh technique. Structured meshes often allow more code to be re-used when converting from a regular grid to an adaptive one, and they are often more cache-coherent than fully unstructured meshes. However, structured meshes do not have as much flexibility as unstructured ones.

Quadtrees and Octrees The spatially adaptive schemes of Hutchinson et al. [HPH96] and Villard and Borouchaki [VB05] use quadtrees to directly connect masses and springs, introducing T-junctions in the process. Ganovelli et al. [GCS99] propose an octree-based multi-resolution method for determining connectivity in a mass-spring system. Debunne et al. [Deb+99] simulate elastic models using control nodes based on approximate finite differences operators, and they use an octree-based refinement to increase detail based on a Laplacian deformation metric. As discussed in several of the aforementioned works, the main difficulty with using mass-spring models instead of approaches based on continuum mechanics is the notion of *convergence*. Convergence is well-studied in the finite element method, and it is trivial to show that increasing spatial resolution will lead to a more accurate solution. Mass-spring models can also converge under refinement if spring stiffness parameters are chosen appropriately, but these stiffness values are not as straightforward to derive compared to the stiffness matrix in a finite element method.

The works of Dick et al. [DGW11] and Seiler et al. [Sei+11] use an octree to define a set of hexahedral finite elements in an elasticity simulation, which is specifically used for simulating cuts in a deformable body. Dick et al. also leverage the hierarchy provided by the octree in a geometric multi-grid method for solving the elastodynamics. The work was subsequently extended to interactively animate high resolution boundary surfaces [WDW11], to improve collision handling [WDW13], and to simulate at haptic rates [WWD14]. For more details on methods for cutting deformable bodies, please see the state of the art report by Wu et al. [WWD15].

Researchers have also developed octree-based discretizations of the Navier-Stokes equations, which lead to efficient animations of smoke and liquid [SY04; LGF04; LFO05]. These approaches also inspired spatially adaptive works that handle discontinuities across free surfaces [HK05], resolve extremely thin surfaces in bubbles and foam without losing volume [Kim+07], and animate multiphase fluids using regional level-sets [Kim10]. More recent work [FWD14] combines an adaptive hexahedral finite element method based on octrees with a multi-grid Poisson solver to animate highly detailed liquids. Bargteil et al. [Bar+06] use octree-based spatial adaptivity to track detailed deforming liquid surfaces using semi-Lagrangian contouring.

It is worth noting that quadtree and octree grid refinement can have subtle side-effects when discretizing partial differential equations. In particular, the regular staggered-grid discretization of the Poisson equation (which is used for enforcing incompressibility in fluid flows [Bri08]) happens to satisfy Stokes' theorem and exactly integrates fluid fluxes — it doubles as a "finite volume method" and can be alternatively derived using discrete exterior calculus [Cra+13]. When one replaces the regular grid with an octree, however, it is unsafe to assume that such useful properties will still hold in the presence of T-junctions. The previously-mentioned octree-based fluid simulation methods counteracted this particular problem by carefully designing a new divergence operator, and some researchers observed the emergence of spurious rotational flows when refining an octree near liquid surfaces.

These subtle problems help explain the large number of adaptive BCC-mesh-based liquid solvers discussed below, which do not exhibit T-junctions.

Adaptive BCC lattices A standard way to efficiently generate a tetrahedral mesh with spatial adaptivity is to combine a spatial hierarchy with predefined lattice-based stencils. One particularly popular strategy combines an octree with the body-centered cubic (BCC) lattice tetrahedralization. To give some context, a *regular* BCC lattice is defined by first inserting vertices at the corner of a regular cubic grid, inserting additional vertices at the center of every grid cell, and then creating a Delaunay tetrahedralization of these vertices. A *spatially adaptive* tetrahedral mesh is created similarly by first creating a weakly-balanced octree (instead of a regular grid), inserting vertices at the corners and centers of the octree cells, and then tetrahedralizing the set of vertices. Please see the work of Molino et al. [Mol+03] and Labelle and Shewchuk [LS07] for some examples of how to create such an octree-based adaptive BCC mesh. This particular meshing strategy has several benefits: the average tetrahedral element has a nearly optimal shape, the quality of the worst element is bounded and completely acceptable in practice, and the computation time required to build the mesh is orders of magnitude faster than unstructured meshes, because it makes use of trees and precomputed stencils.

Employing these ideas, Wojtan and Turk [WT08] use an adaptive non-conforming BCC tetrahedralization to simulate highly plastic materials while efficiently remeshing whenever element quality degrades (See Figure 3.5). Batty

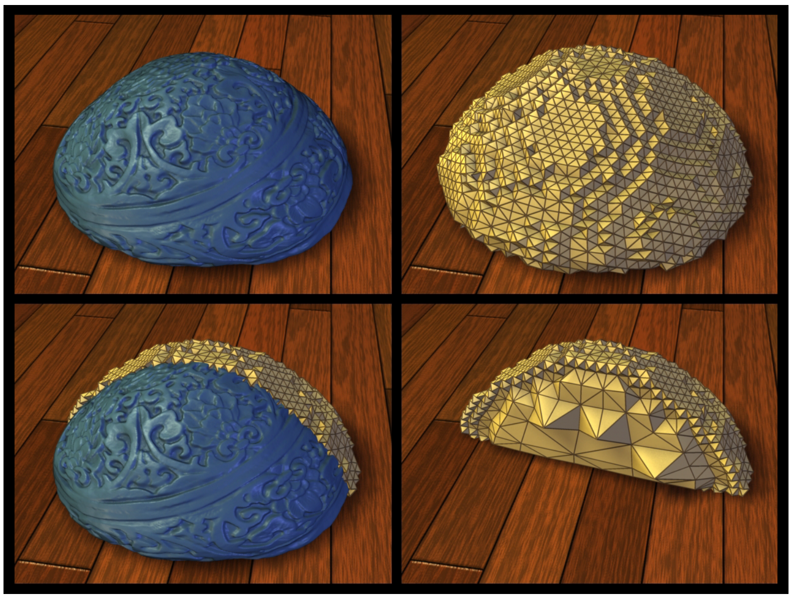


Figure 3.5: A high resolution surface mesh (blue) embedded into a deforming low resolution spatially-adaptive tetrahedral mesh based on a BCC lattice (gold). The bottom row shows the cross section of the tetrahedral mesh, illustrating the BCC structure. Image from Wojtan and Turk [WT08].

et al. [BXH10] use a similar meshing strategy to simulate inviscid fluids using a finite-volume discretization. The method also handles embedded solid and free-surface boundary conditions, even though the tetrahedral mesh does not conform to the domain boundaries. Batty and Houston [BH11] extend this work by adding an implicit viscosity model. As explained later in the Section 3.2.3, Ando et al. [ATW13] also use an adaptive BCC mesh for simulating liquids. Sifakis et al. [Sif+07] introduce the concept of “hard bindings” to create an adaptive BCC lattice with T-junctions for the purposes of elastic solid simulation.

Adaptively-refined BCC lattice is exceptionally useful in simulations requiring tetrahedral meshes, because the structured lattice makes the expensive operations of remeshing and re-sampling simulation data relatively insignificant. Although the structure of the mesh removes some control over the shapes and nature of the spatial adaptivity, the structure also eliminates typical issues, such as degenerate tetrahedra.

Tall cell grids Simulations of deep water also benefit from adaptive “tall cell” techniques [Irv+06; CM11], which use regular grid cells near the water surface (where detail and accuracy is important), and tall rectangular fluid cells farther below the surface. This strategy effectively assumes that the behavior in regions located deep underwater is simpler, in that the simulation variables cannot change arbitrarily with depth. While we specifically discuss tall cell grids here because of their use of geometric adaptivity, we will also address some related height-field methods in Section 3.3.3. These methods seem to work very well when the assumptions of the methods hold. However, there is reason to believe that artifacts (like spurious reflecting waves or dissipating vortices) can occur when the tall cells fail to properly resolve important dynamics.

Far-field grid structures Zhu et al. [Zhu+13] propose a method for fluid simulation that maintains an efficient Cartesian grid structure but allows non-uniform spacing between the nodes. This approach makes it easy to concentrate smaller cells where the domain is more interesting and retain larger cells farther away from areas of interest. In addition to concentrating detail in important regions, this method also approximates non-reflecting boundary conditions by greatly extending the boundaries of the simulation domain.

3.2.2 Unstructured meshes

Adaptivity on unstructured meshes is closely related to the problem of remeshing. Considered purely as a geometrical problem, remeshing has been studied extensively in computational geometry [CDS12] and in a modeling context in computer graphics [All+08]. Indeed, the techniques adopted for adaptive simulation often build on the framework of geometrical remeshing methods, and extend them to a simulation context. A number of challenges arise when performing remeshing during simulation, though. First, the mesh elements must remain well-conditioned to avoid degrading the stability of the simulation. Second, modifying the discretization on the fly risks introducing errors in transferring energy and momentum to the new mesh, such as numerical diffusion due to re-sampling, and discontinuous “popping” artifacts in thin strands or sheets. Third, removing degrees of freedom must be done with care, as a mesh with fewer DOFs cannot represent the previous system state exactly. Depending on the characteristics of the simulated material—whether it is elastic, plastic, or fluid; whether volumetric or lower-dimensional—different remeshing methods are found to be appropriate.

Hierarchical schemes The simplest remeshing strategy is to use a fixed hierarchical scheme for refinement, which provides guaranteed bounds on element quality. Two such schemes are illustrated in Figure 3.6a, 3.6b. In cloth simulation, triangle subdivision schemes such as 1-to-4 splits, $\sqrt{3}$ refinement, and edge bisection have been employed [LV05; SLD09; BD12]. A similar scheme for subdivision of tetrahedra was used by Koschier et al. [KLB14] for volumetric fracture. Wu et al. [Wu+01] build on the concept of progressive meshes [Hop96] to precompute FEM parameters, allowing adaptive simulation of deformable bodies at interactive rates.

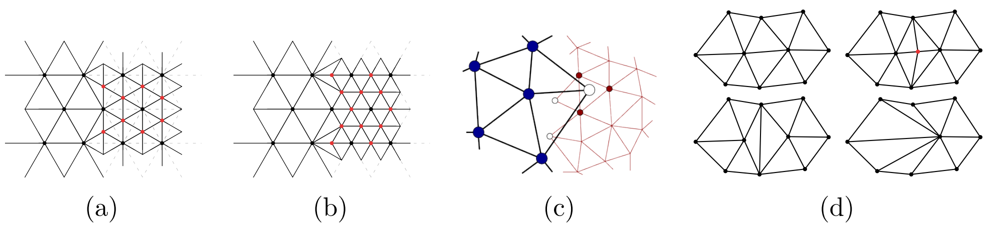


Figure 3.6: Overview of refinement schemes for unstructured meshes. In (a) and (b), the right half of a mesh is refined using (a) $\sqrt{3}$ refinement and (b) hierarchical edge bisection, with inserted nodes highlighted in red. In (c), we illustrate two levels of non-nested meshes. In (d), the three primitive operations of local remeshing are applied to the middle edge: in reading order, we show the original mesh, after an edge split, after an edge flip, and after one of two possible edge collapses.

However, subdivision schemes still degrade element quality by a moderate extent compared to an optimized mesh. When this is undesirable, an alternative is to use a hierarchy of "non-nested meshes" at different resolutions [Deb+00; Deb+01]; see Figure 3.6c. Each level of the hierarchy is a complete mesh that does not necessarily share any nodes with meshes at other levels, and can be independently optimized *a priori*. At run time, regions at different levels of detail use subsets of different meshes. Coupling is achieved by allowing the regions to overlap slightly; nodes in the overlap region in each mesh are treated as inactive "ghost" nodes that are embedded in the containing element of the other mesh. The work of Otaduy et al. [Ota+07] seamlessly integrates such adaptive non-nested meshes with a multi-grid algorithm and an adaptivity-aware collision detection technique.

Hair simulation can benefit from adaptivity, as the contact interactions between hair strands lead to the formation of emergent clusters. Bertails et al. [Ber+03] introduce a hierarchical structure called an adaptive wisp tree (AWT), which represents hair clusters that can progressively split into smaller clusters from the base to the tip. Refinement is performed by splitting a node if its size and acceleration are large, while coarsening is performed by merging sibling nodes if they have similar positions and velocities.

Nearly regular meshes Some techniques for liquid simulation use a structured mesh in the interior of the volume, but allow irregularities at boundaries to better capture the dynamics of the free surface. These techniques benefit from many of the advantages of structured meshes discussed in Section 3.2.1, while still retaining much of the flexibility of unstructured meshes, such as the ability to accurately match boundary conditions.

In such methods, one maintains a high-resolution representation of the surface as a triangle mesh that is updated at each time step. The surface is superimposed on a regular mesh structure such as an octree or a uniform grid, and elements near the surface are modified, or new elements inserted, to better conform to the surface. In particular, Chentanez et al. [Che+07] use the isosurface stuffing algorithm [LS07] that generates an adaptive BCC lattice whose surface tetrahedra are warped and possibly subdivided to conform to the surface geometry. Using an octree to construct the lattice allows for coarser resolution away from the free surface. Brochu et al. [BBB10] use a uniform background lattice and introduce additional pressure samples along both sides of the free surface to ensure that all surface features are resolved. The pressure projection is then performed on a mesh consisting of the Voronoi cells of these sample points.

Local remeshing In some contexts, it is desirable to allow the connectivity structure of the mesh to be modified freely during the course of the simulation. Such a requirement arises when anisotropic elements are needed to resolve strongly directional features, or when the material exhibits both elastic properties and unbounded deformation, such as in plastic flow.

While it is possible to perform global remeshing—that is, simply creating a new simulation mesh from scratch whenever needed [Kli+06; Bar+07]—this approach can lead to undesirable diffusion of stored physical quantities such as plasticity information. An increasingly popular alternative is to remesh locally using a set of local operations that refine, coarsen, and reshape existing elements. In local remeshing techniques, any elements in the mesh that do not satisfy the desired size and shape criteria are improved by careful application of these operations. This process is repeated until all mesh elements are satisfactory.

When performing remeshing, it is important to ensure that the mesh remains well-conditioned for simulation, through the use of various element quality measures [She02]. In 2D, the Delaunay triangulation optimizes several important notions of mesh quality, including the minimum angle and the maximum circumradius of any triangle, but the Delaunay tetrahedralization in 3D provides few such guarantees, and more sophisticated mesh improvement strategies may be required [Wic+10]. If anisotropic remeshing is desired, the remeshing criterion is often expressed through a spatially varying metric tensor $\mathbf{M}$, with the goal being that each edge $\mathbf{e}$ should have $\mathbf{e}^T\mathbf{M}\mathbf{e} \approx 1$. Equivalently, we desire $\|\mathbf{M}^{1/2}\mathbf{e}\| \approx 1$; that is, we want edges to be of unit length and elements to be equilateral in the transformed space of $\mathbf{M}^{1/2}$. This viewpoint allows element quality metrics and Delaunay properties defined for isotropic meshes to be carried over to the anisotropic setting.

For manifold triangle meshes, high-quality remeshing can be accomplished using only the three simple operations of edge split, edge collapse, and edge flip (see Figure 3.6d). Narain et al. [NSO12] use these operations in cloth simulation, generating an adaptive anisotropic mesh that resolves detailed

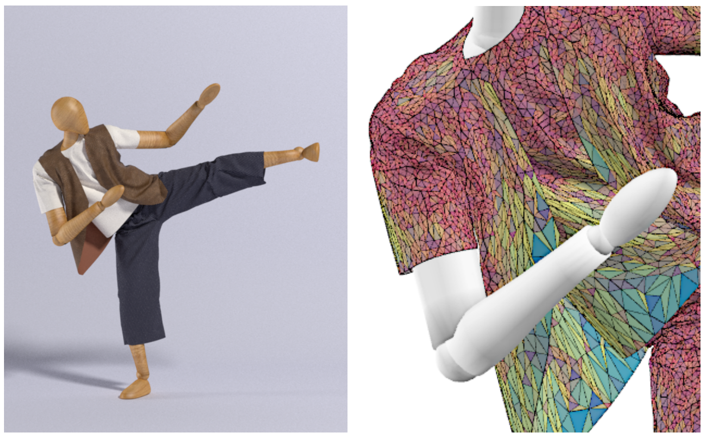


Figure 3.7: The anisotropic remeshing algorithm of Narain et al. [NSO12] allows detailed wrinkles in cloth to be resolved accurately with fine elements (red), while much coarser elements (blue/green) are used in flat regions. Long, narrow folds are best represented using anisotropic elements (yellow) aligned with the curvature direction.

wrinkles and folds (see Figure 3.7). First, all edges that are unacceptably long according to the refinement criterion are split, then edge collapses are attempted as long as they do not create new unacceptable edges. During both steps, edge flips are performed to maintain an approximately Delaunay mesh relative to the anisotropic metric.

Local remeshing of tetrahedral meshes is significantly more involved, requiring several different local operations and a complex schedule for the order in which to apply them [KS07]. This technique was first applied to simulation by Wicke et al. [Wic+10], who used it to minimize artificial diffusion in elastoplastic flow. Subsequent work has applied such remeshing techniques to simulation of incompressible liquids [MB12; Mis+12; Cla+13].

Misztal et al. [MB12; Mis+12] build a mesh over the entire simulation domain, with some elements belonging to the fluid and the rest to the exterior, while Clausen et al. [Cla+13] mesh only the fluid volume. The former approach allows topological changes like collisions to be handled automatically without special treatment, although at the cost of maintaining a mesh over a potentially much larger domain. Clausen et al. also describe techniques for guaranteeing incompressibility and momentum conservation that are applicable to both approaches. Two key advantages offered by these methods are that (i) the advection step causes no numerical diffusion, because physical quantities move with the mesh nodes, and that (ii) surface tension can be modeled accurately thanks to an explicit surface representation tied directly to the simulation mesh.

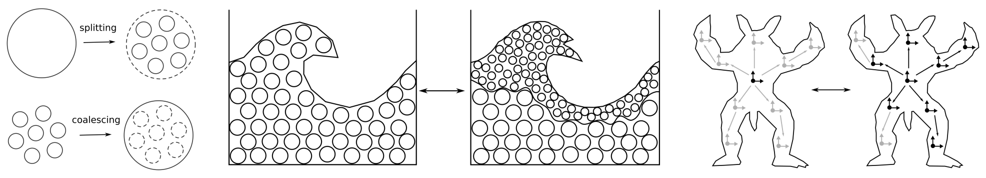
Apart from simply adding or removing vertices, surface tracking algorithms in fluid dynamics may also move vertices along the surface in order to optimize mesh shapes. A process called “null-space smoothing”, which slides vertices within the tangent space of a meshed surface, is used in several works [Jia07; BB09; BBB10; Wic+10; Cla+13]. This strategy improves the quality of simulation elements without changing the shape of the tracked surface.

Additional challenges and techniques When refinement is performed, the position of the newly inserted node has to be chosen carefully. Simply placing it at the midpoint of the original element can cause physical quantities such as bending to change discontinuously, injecting artificial energy into the system and leading to instabilities. Instead, it is better to adjust the mesh locally to bring it into an energy-minimizing configuration. Spillmann and Teschner [ST08] consider the positions of the new node $\mathbf{x}_i$ and its neighbors as variables and perform an optimization to minimize the total energy,

$$U(\mathbf{x}_1, \dots, \mathbf{x}_n) - \sum_{j=1}^{n} \mathbf{f}_j^T \mathbf{x}_j, \tag{3.14}$$

where U is the internal energy due to elastic forces, and $\mathbf{f}_j$ is the external force acting on node j. A similar approach has been used for simulating the behavior of stiff two-dimensional sheets such as paper and metal [NPO13; Pfa+14], but with $\mathbf{f}_j$ replaced with an acceleration-corrected term $\mathbf{f}_j - m_j\mathbf{a}_j$ to preserve the instantaneous acceleration of each node.
Liquids with surface tension may freely transition between volumes, thin films, filaments, and droplets; representing these transitions is a challenge for most mesh-based techniques. Zhu et al. [Zhu+14] address this problem using non-manifold meshes of mixed dimensionality, composed of tetrahedra, triangles, segments, and points. Beyond the traditional remeshing operations that work within a single dimensionality, they also provide operations for dimensionality transitions via element collapse (e.g. transforming a thin triangle to a segment) and merging (e.g. generating a tetrahedron to connect two adjacent triangles with small dihedral angle).

Finally, we point out the recent “power particles” technique of de Goes et al. [Goe+15], which builds an unstructured mesh at each time step using a Voronoi-style power diagram. This can be viewed as a global remeshing approach like the ones mentioned above [Kli+06; Bar+07], but this method stores physical quantities on Lagrangian particles without maintaining an explicit connectivity, and thus avoids numerical diffusion due to re-sampling. While this work is not specifically an adaptive strategy, it is a form of spatial discretization that makes adaptivity very easy to implement, and could be a fruitful basis for future work in adaptive simulation.



(a) (b) (c)

Figure 3.8: Overview of adaptive meshless techniques. (a) Dynamic local re-sampling applies splitting and coalescing operators to degrees of freedom in order to locally refine and coarsen regions of interests (b) Multi-scale methods couple several simulations with different resolutions. Coarser simulations are used as boundary conditions for the finer resolutions. Feedbacks from the finer resolutions are used to avoid divergences between two different resolutions. (c) Hierarchical refinement dynamically activates or deactivates levels of a precomputed hierarchy between degrees of freedom.

3.2.3 Meshless models

In the last two decades, numerous meshless models have been extended to perform adaptive physically-based animation. They include Smoothed-Particle Hydrodynamics (SPH) (see the survey of Ihmsen et al. [Ihm+14b]), fluid-implicit particle (FLIP) (see the seminal work of Zhu and Bridson [ZB05]), moving least squares (MLS) (see Muller et al. [Mül+04]) and frame-based models (see Gilles et al. [Gil+11]). These models were used to describe a wide range of phenomena, from fluids (SPH, FLIP) to solids (MLS, frame-based).

Due to the absence of fixed connectivity, meshless models are among the most flexible models for spatial adaptivity. This flexibility, combined with the variety of models, leads to an impressive number of re-sampling strategies, developed to resolve details near splashes, large deformations, viscoplastic flows and fractures. We classify these strategies into three categories: (1) dynamic local re-sampling, (2) multi-scale methods, and (3) hierarchical refinement (see Figure 3.8).

These different strategies make meshless models particularly useful for material that undergo large irreversible deformations such as the one cited above. However, it is important to keep in mind that the flexibility of meshless models come with expensive nearest-neighbor search algorithms to determine the connectivity between material samples at run time.

We inform the reader that complementary information about SPH adaptive techniques can be found in the state-of-the-art report by Ihmsen et al. [Ihm+14b].

Dynamic local re-sampling The idea is to dynamically subdivide or merge particles to fit a desired resolution (see Figure 3.8a). The success of this strategy mainly relies on the re-sampling scheme's ability to ensure stability, to accurately represent boundaries, and to prevent popping artifacts. Depending on the underlying model (SPH, FLIP, MLS) and its sensitivity to intense re-sampling, different strategies have been proposed.

As a full particle-based method, the SPH model is a perfect candidate to dynamic re-sampling. Yet, its sensitivity to particle distribution makes re-sampling strategies challenging. First, the interaction between particles with different sizes increases the error in the pressure term, leading to instabilities. Therefore smooth grading of resolution is required to minimize this error. Second, the change of positions during re-sampling can create a sudden change in density which will result in violent pressure forces, which again lead to instabilities (see Orthmann and Kolb [OK12]). Several methods that can be combined were proposed to avoid these local change in density. First, instead of computing the density based on positions, one can use the continuity equation as done by Desbrun and Cani [DC99]. In order to avoid integration error to be accumulated along the simulation, the density still needs to be re-computed based on positions at a user-defined interval. Then, one can perform position optimization to minimize errors during re-sampling [Ada+07] and use quantity blending over time to smooth out inevitable sampling error [OK12]. Also, it is important to keep in mind that another challenge is to efficiently retain the parallel nature of SPH in the adaptive scheme. Zhang et al. [ZSP08] and Yan et al. [Yan+09] propose two different methods to make splitting and merging operators parallel.

More recently, dynamic re-sampling has been applied to FLIP. As FLIP is a combination of grid and particles, two levels of adaptivity are possible: one on the grid resolution, and the other on the particle sampling. Also, only advection operations are performed on the particles, which results in a less position-sensitive simulation and allows much more flexibility than SPH. More precisely, FLIP does not apply density-based forces to the particles. Consequently, sudden density changes due to particle splitting or merging do not have the same catastrophic consequences as in SPH. However, damping is introduced once particles are merged. This can be taken into account by changing blending parameters of the FLIP simulator as suggested by Ando et al. [ATT12]. Early works on adaptive FLIP perform adaptivity only on particles based on a deformability criterion and the distance to surface [HHK08; ATT12]. Ando et al. use the flexibility of FLIP regarding the particles' positions in order to preserve fluid sheets by creating additional particles. In both methods, the largest particle size is bounded by the cell size of the underlying grid, which precludes aggressive adaptive sampling and the use of a fully adaptive FLIP simulator. Ando et al. [ATW13] combine an adaptive

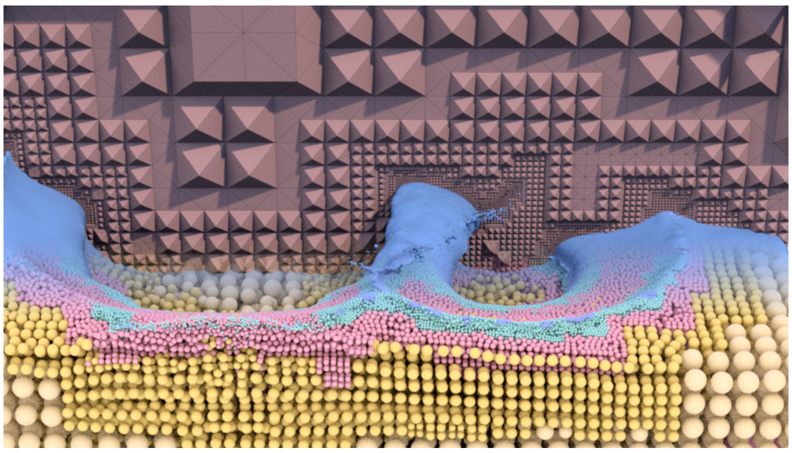


Figure 3.9: Ando et al. [ATW13] simulate liquid by combining adaptively-sized FLIP particles (bottom, front) with an adaptive tetrahedral mesh for the pressure solve (top, back).

BCC mesh (see Section 3.2.1) with adaptive particle sampling to handle highly different resolutions. (see Figure 3.9).

When simulating solids, local re-sampling is essential in describing phenomena such as large deformations and fracture. First, like mesh-based methods, large deformations create poorly sampled regions leading to ill-conditioned deformation gradients and instabilities. Second, as explained in Section 3.2, a challenge in fracture simulation is to reach a sufficiently fine discretization near the tip of the crack in order to obtain a realistic crack path. In these cases, local re-sampling can greatly improve accuracy and stability while retaining efficiency. However, there are two main challenges that require special care.

The first one consists of accurately describing material discontinuities. Most of the time, shape functions are spherical and they must be modified so that sharp boundaries can be represented. In their work on large viscoplastic deformation and fracture, Pauly et al. [Pau+05] model discontinuities using extended shape functions with transparency criteria, and locally modify a sparse neighborhood graph to update connectivity. Another strategy was proposed by Steinemann et al. [SOG09] to address the high cost of shape functions update. It consists of using a visibility graph to efficiently handle connectivity and approximate material distances used in computing shape functions.

The second one comes from rendering artifacts that can occur at the surface due to splitting. In this context, Jones et al. [Jon+14] propose a strategy to re-sample elastoplastic simulation while alleviating popping artifacts. They base their method on the evaluation of each particle neighbor's density. This evaluation is performed using a weighted covariance matrix computed in rest space for each particle i, where $\mathbf{u}_{ij}$ denotes the vector between particle i and particle j, its neighbor.

$$\mathbf{B}_i = \sum_j \frac{\mathbf{u}_{ij}\mathbf{u}_{ij}^T}{\|\mathbf{u}_{ij}\|^4} \tag{3.15}$$

If the maximum eigenvalue of $\mathbf{B}_i$ is too small, then there are too few particles in the neighborhood and the particle is split in two. New particles are positioned along each side of the eigenvector with the minimum eigenvalue. In order to prevent from rendering artifacts, splittings which are not tangent to the surface are rejected, and particles near the surface are split along the middle eigenvector whose direction is tangent to the surface. Conversely, if the minimum eigenvalue of $\mathbf{B}_i$ is too large, then there are too many particles in the neighborhood and the particle is merged with its closest neighbor. The new particle is positioned halfway between the two merged particles.

Multi-scale methods In multi-scale methods, several simulations with different resolutions are coupled in a hierarchical way (see Figure 3.8b). At the coarsest simulation level L_0, the whole domain is discretized. Then, each finer simulation level L_r discretizes a subset of L_{r-1} with a finer scale. This finer subset is defined to match regions of interests that can be physically or visually motivated. Transitions between two scales are bilateral: the coarsest simulation level L_r is used to build boundary conditions for the finer simulation level L_{r+1} and feedbacks from a finer simulation level L_{r+1} to a coarser simulation level Lr are applied, in order to prevent dynamics of two different levels from diverging. Solenthaler and Gross [SG11] and Horvath and Solenthaler [HS13] apply this idea to SPH fluid simulation. Compared to merging and splitting particles, the main advantage is that interactions between different resolutions are not direct anymore. Thus, stability can be more easily ensured and large differences in resolution can be handled. In Solenthaler and Gross's approach, a two-scale simulation is performed. High-resolution regions are defined based on the distance to the surface and the view frustum. In these regions, low-resolution particles emit finer particles according to a cubic pattern which ensures a uniform space sampling. Relaxation steps are performed when particles enter the high-resolution region in order to avoid large pressure forces. Horvath and Solenthaler extend the two-scale simulation to multi-scale simulation, and avoid previous artifacts such as mass loss due to particle removal and instabilities due to oversampling near boundaries.

Hierarchical refinement Dynamic re-sampling techniques were also used in frame-based methods to simulate elastic deformations, see [Gil+11] for a full description of the method. Tournier et al. [Tou+14] use a hierarchical approach to achieve simplifications during deformation without popping artifacts (see Figure 3.8c). The material is deformed using physically-based control frames organized in a generalized hierarchy. The model can be simplified by attaching frames to their parents at any time in their current relative positions. Activation and deactivation of nodes is performed based on relative velocity and user-specified metrics, while integration points are updated according to the hierarchy. These hierarchical techniques take advantage of their structure to improve efficiency, but may lack of flexibility, especially regarding topological changes.

3.3 Miscellaneous techniques for spatial adaptivity

By far the most popular approach to spatial adaptivity in computer graphics is to add more computational elements where more accuracy or detail is desired, as surveyed in Section 3.2. This type of spatial adaptivity is often called h-refinement, because the length of an edge in a mesh is typically indicated by the letter h. In addition to this tried-and-true strategy, there are other fundamentally different approaches for achieving spatial adaptivity. In the next sections, we will discuss strategies that refine the basis within a single element (a superset of p-refinement, which refers specifically to the order of a polynomial basis) and during a subspace simulation, strategies that use multiple grids that move and overlap to track locations where more detail is desired, and strategies that mix different reference frames in order to use the computational degrees of freedom optimally.

3.3.1 Basis refinement

If we wish to achieve spatial adaptivity without explicitly remeshing (perhaps because it is difficult to control element quality when remeshing, or because a particular application requires that we preserve the original mesh), then we can perform basis refinement instead. The concept of basis refinement can be a difficult one to grasp for newcomers to the field. One of the best ways to understand basis refinement is in the context of finite element methods (FEM). In generic terms, FEM attempts to approximate a function (typically the solution to a partial differential equation) with a very limited, very specific subset of all possible functions. Most methods in computer graphics use linear interpolation within each element, which essentially restricts the solution to a piecewise linear function. For the purposes of this discussion, we would say that the elements are using linear basis functions, and that the overall solution is expressed in a piecewise-linear basis. However, we can actually represent the solution more accurately (in the sense that the solution converges more quickly under refinement) by using more elaborate bases, like piecewise quadratic functions instead of piecewise linear ones.

This section discusses four types of basis refinement: hierarchical basis refinement, polynomial basis refinement, basis enrichment, and adaptive reduced basis functions. The first three topics discuss adaptivity at the level of basis functions, whereas the fourth is about the adaptive creation of reduced basis in reduced model simulations.

Hierarchical basis refinement In computer graphics, hierarchical basis refinement has mainly been applied to finite element simulations (FEM) of solids and shells [Cap+02; GKS02]. The idea is to refine computational basis functions instead of elements. From a theoretical point of view, there are no differences between hierarchical basis refinement and hierarchical element refinement. Both adaptively add more degrees of freedom with increasingly local support in order to improve accuracy where needed. Both use hierarchical schemes in order to efficiently sample the simulation domain. The main differences are practical. By refining basis functions instead of elements, compatibility between regions with different resolutions are implicitly handled. This makes adaptivity much easier and general.

For instance, hierarchical basis refinement allows a simple handling of *T-junctions*. In FEM, each element's node carries a basis and builds a local stiffness matrix from its node's stiffness which are then assemble into a global stiffness matrix. During this process, only independent degrees of freedom should add their contribution to the global matrix. However, when using hierarchical element refinement, non-independent degrees of freedom are added during the subdivision of the simulation mesh. They are called *T-junctions* or *T-nodes* (see Figure 3.10) and require specific handling. Suddenly, a simple subdivision scheme becomes dependent on the dimensionality, the element type and the basis order, thus requiring important implementation work. In contrast, hierarchical basis refinement handles *T-nodes* at the basis level by making sure that no bases are redundant. The hierarchical structure makes this process simple and thus offers a more general framework for adaptivity which can handle arbitrary resolution differences.

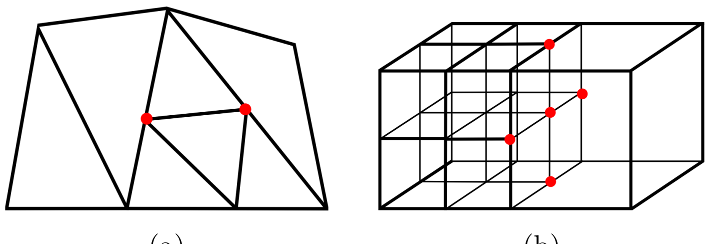


Figure 3.10: Refinement by element subdivision is attractive by its simplicity for 2D and 3D meshes, however it introduces T-junctions (in red) at interface between resolutions with different sizes. Bookkeeping or extra-remeshing operations are required to take care of these non-independent degrees of freedom.

Capell et al. [Cap+02] embed a high-resolution mesh in a hexahedral complex and precompute a hierarchy of bases up to a given level of refinement.

During the simulation, depending on the amount of deformation, each level of the hierarchy will refine or coarsen, thus updating the current set of active bases. Basically, it is the same idea of [Deb+01] but from a basis point of view.

The Conforming Hierarchical Adaptive Refinement Methods (CHARMS) framework of Grinspun et al. [GKS02] generalizes the idea of spatially refining the bases instead of the elements. They provide an in-depth explanation of the concept and describe numerous results of basis refinement applied to shells, solids and electrocardiography simulations. In fact, it is quite surprising that this method was not more studied or extended in the last decade. A possible explanation is the fact that, in the last few years, adaptivity proved to be essential for large and complex deformations such as visco-elastic, visco-plastic flows. In those cases, hierarchical refinement is not sufficient anymore to ensure well-conditioning of the system matrices.

Polynomial basis refinement Polynomial basis refinement methods, also called *p-adaptivity*, increase or decrease the order of the basis functions. For a given spatial resolution, this allows the improvement of the quality of the deformation without remeshing. In computer graphics, using high order approximation to resolve fine details is not new. However mixing different orders of approximation to adaptively resolve details was only recently applied in the context of fluid simulation and elastic deformations.

For fluid simulation, the smoothness of velocity and pressure makes *p-adaptivity* potentially much more efficient than geometric adaptivity, because the error per degree of freedom decreases exponentially with the approximation order but only geometrically with the spatial resolution. In this favorable context, Edwards and Bridson [EB12; EB14] use polynomial basis refinement in a Discontinuous Galerkin FEM framework in order to simulate detailed water with coarse grids. They use low-order bases deep inside the liquid and increase the basis order closer to the liquid surface, where more visual detail is desired (see Figure 3.11a). By using basis refinement instead of element refinement, they can keep the simple structure of a low resolution Cartesian grid while pushing back the limit on the scale of details in one cell. In terms of cost, their method is approximately as expensive as a classical high spatial resolution simulation but it provides much more details such as extremely thin sheets.

Bargteil and Cohen [BC14] animate deformable bodies by combining linear and quadratic Bézier elements. The main advantage of their method is the ability to locally increase degrees of freedom and to simulate nonlinear geometry without remeshing (see Figure 3.11b). To decide whether an element should be linear or quadratic, they compare the linear and quadratic predicted positions of the midpoint of each edge of the element. If the difference between the two positions is larger than a threshold then the edge becomes

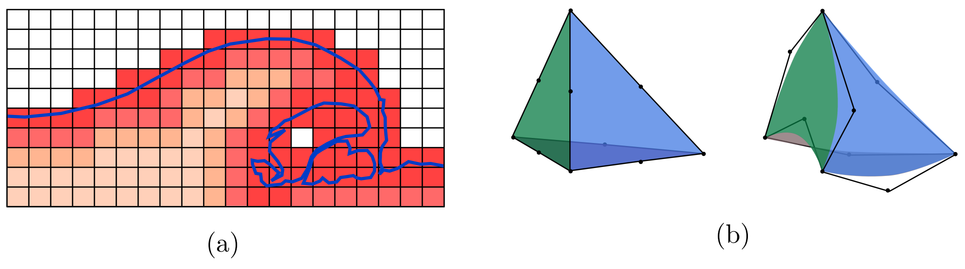


Figure 3.11: Illustrations of p-adaptive techniques for (a) fluids and (b) deformable solids. In (a) Each fluid cell uses a different approximation space depending on its distance to surface (blue line). Surface cells (in red) use fourth order polynomial to precisely approximate pressure. A smooth grading of the approximation is performed inside the fluid (lighter cells) that allows one to save computational time. In (b) the canonical tetrahedron with quadratic control points and next to it the quadratic deformation induced by the deformed control mesh. Such a deformation would require many linear elements.

quadratic. If the difference is less than another threshold then it becomes linear. Thus, some elements can have linear and quadratic edges, usually in transition regions. Bargteil and Cohen observe that, as the number of degrees of freedom increases, visual differences between linear and quadratic elements become difficult to discern. Moreover, the additional cost remains important and local deformations on the surface of a quadratic element due to collisions still cannot be resolved without element refinement. Therefore, their method is particularly efficient on low-resolution models, where it provides smoother geometry and better dynamics quality.

In both cases, the differences of resolution between two regions that can be achieved only using *p-adaptivity* are limited. An important avenue for research would be to combine those methods with geometric adaptive techniques. Such methods have been well studied in engineering fields and are called *hp-adaptive* methods.

Basis enrichment Another way to add spatial detail to a physical model without remeshing is by using "basis enrichment." The main idea behind basis enrichment is to adaptively add carefully-chosen basis functions that are specifically designed for the phenomena being modeled. (This is in contrast to p-refinement, which is restricted to polynomial functions, regardless of the phenomena being simulated.) For an example of basis refinement, consider an object is being fractured; some material that used to be connected together will have to be split in two. A simple linear basis function could be split into two functions that are linear on one side of the fracture and zero on the other, as illustrated in Figure 3.12.

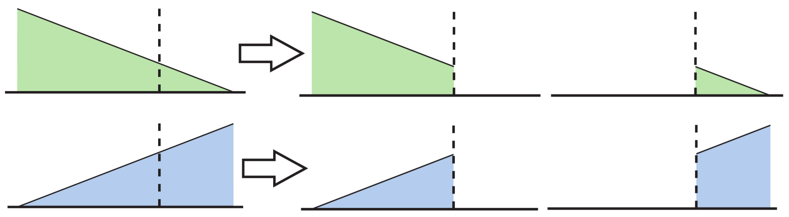


Figure 3.12: A one-dimensional element with linear basis functions can simulate a fracture by enriching its basis. Here, the new basis functions drop off to zero on the other side of the fracture site indicated by the dashed line, directly encoding the severed connection.

This type of basis replacement effectively avoids remeshing by inserting a fracture directly into the basis itself. Note that in this case, we opted to enrich the basis with these particular "step" functions which drop to zero after a certain point, instead of trying to fit some polynomial that might overshoot or otherwise imperfectly capture the desired physics. Such basis enrichment techniques have been termed the "general finite element method" (GFEM) or "extended finite element method" (XFEM) [BGV09].

Adaptive basis enrichment methods have been recently applied in many computer graphics contexts. The virtual node algorithm uses basis enrichment to stabilize fracture and cutting simulations [MBF04; Heg+13]. Instead of remeshing and potentially inserting poorly-conditioned elements, the virtual node algorithm essentially copies the entire element and adapts its basis functions to the fracture site. The virtual node algorithm has recently been improved to robustly allow multiple cuts in a single element [SDF07; Wan+14]. One drawback to adaptive basis enrichment is that complicated cuts can require arbitrarily complicated basis functions that may not be simple to compute. In the case of cutting thin shells, Kaufmann and colleagues [Kau+09] note that the most physically-appropriate enrichment functions require the solution of a Laplace equation with time-varying boundary conditions. Similar basis enrichment techniques may not be computationally feasible for the adaptive simulation of more complicated volumetric phenomena.

Adaptive reduced bases As pointed out by Bargteil and Cohen [BC14], volumetric elastic effects are usually low-resolution in space, because we often care about longer time scales in computer graphics. Model reduction methods, also called subspace simulation, exploit this fact to turn extremely costly nonlinear finite element simulations into interactive ones. The idea is to solve equations of motion in a reduced basis that is usually computed using modal analysis or from a database using principal component analysis (PCA). Thus, instead of having a complexity dependent on the simulation mesh resolution, it only depends on the size of the reduced basis (the number of deformation modes) which is much smaller. A cubature scheme is used to perform integration on only a reduced sets of well-chosen samples. In the end, this allows simulation of nonlinear deformable models with orders of magnitude speed-up.

The drawbacks are that the creation of the basis requires heavy precomputation, and that the motion of the model is constrained to lie in this basis. Furthermore, while the full finite element simulations often have local compact basis functions and sparse system matrices (because each degree of freedom is a vertex that only influences its neighbors locally), reduced simulations tend to have global basis functions and dense system matrices (because each degree of freedom impacts all points in space simultaneously). Small dense systems are often more efficient than large sparse ones, but the increased storage requirement limits how many modes can be feasibly included in model-reduced simulations, and it limits the use of model reduction in settings where large numbers of modes are essential for visually-plausible behavior (like cloth and fluids). Therefore model reduction, *alone*, is truly efficient for smooth global deformations and predictable scenarios where it is possible to build a basis that remains constant over time. It is also only useful in situations where a limited number of modes can adequately describe the system.

Several strategies have been proposed to overcome some of these drawbacks by adaptively improving the reduced basis. Common components to those strategies are the different processes to update the basis and the criteria used to decide when the basis should be adapted. Moreover, a common challenge is to ensure temporal coherence while adapting the basis. Kim and James [KJ09] combine a full nonlinear simulation with subspace simulation in order to exploit coherence in the global motion of the model. They incrementally build a reduced-order nonlinear deformable model as the full nonlinear simulation progresses. When possible, full nonlinear steps are skipped with subspace steps resulting in significantly cheaper steps. The challenge of this strategy is to provide efficient operators to update the reduced-basis and to robustly choose which steps can be reduced. We briefly describe the two main operations that compose the incremental construction of the basis: the updating operation and the downdating operation. The updating operation adds a new vector to the basis only if it is significant. To do so, a displacement vector received from a full simulation step is orthogonalized against the existing basis. If its norm is above a given threshold then it is concatenated to the basis. The downdating operator is applied when the basis reached its maximal size r and a new significant vector needs to be added. The basis is then modified so that the $r/2$ most significant directions are preserved. Finally, as the cubature (reduced integration) is dependent on the basis, it also needs to be updated. The updating operation is triggered through different criteria depending on the application. Kim and James describe criteria for quasi-static and dynamic simulations. The dynamic case is particularly challenging as the error is history dependent, which means that taking full steps after reduced steps do not correct errors from the subspace simulation. This method presents impressive speed up for nonlinear simulations. However, the performance is highly dependent of the rate of expansion of the basis.

As subspace simulation tends to reproduce the global motion of an object, it is particularly challenging to produce very local deformations in this framework. A direct consequence is a simplification of the dynamic behavior. Harmon and Zorin [HZ13] tackle this problem by including *a priori* knowledge in the building of the basis. More precisely, they augment a standard precomputed basis with a dynamic basis. This dynamic basis is built with custom functions derived from analytic solutions to static load. Unpredictable local deformations that arise due to collisions and contacts can then be handled. Moreover, as the change in the basis is very local, they can ensure temporal coherence by projecting the current subspace coordinate vector into the new basis whenever it changes. However, a limitation of the addition of local modes is a restriction on the time step size in order to properly represent the dynamics. Furthermore, the size of local displacements is limited.
Pushing further the idea of using *a priori* knowledge in the building of a reduced basis, Hahn et al. [Hah+14] perform adaptive subspace simulation of cloth. They start with a large amount of high-resolution simulation data from multiple training animations, and convert it into a database of low-dimensional bases associated with poses. Then at each time step, they adaptively choose a subset of low-dimensional bases in the data base depending on the pose of a clothed character. Highly nonlinear folds and wrinkles, which are very hard problems for subspace simulation, can then be reproduced. Dynamics is damped near tightly constrained regions such as sleeves, but this may be acceptable in practice for animating tight clothing.

Recently, Teng et al. [Ten+15] propose the use of subspace condensation to locally switch between subspace and fullspace simulation at run time. When dealing with localized deformations, this allows the behavior not to be limited by *a priori* knowledge such as in [HZ13] and [Hah+14].

3.3.2 Moving grids

As mentioned in Section (3.2.1), grids are an efficient and simple data structure compared to unstructured meshes. However, this simplicity is counterbalanced by a severe lack of flexibility. Firstly, simulating fluids on very large domains requires a prohibitive amount of memory. Secondly, focusing computational resources on regions of interest remains a challenge. While octrees and other adaptive structured meshes discussed in Section 3.2.1 address these challenges, they lose the cache-coherent structure that makes uniform grids so efficient. Moving grids methods, also called Chimera grids, allow more flexibility while keeping the advantages of Cartesian grids. The main idea is to use one or more computational grids and allow them to move at each time step to follow the region(s) of interest.

Shah et al. [Sha+04] propose a simple approach in which a single grid is used whose location and size changes to enclose the region containing significant flow. This strategy is useful when there is only a single region of interest in the fluid, such as when simulating explosions.

A more versatile approach is to use multiple independently moving grids, centered at each moving object in the scene. The grids may undergo pure translation [CTG10], or may also rotate with the object [Dob+08; Eng+13]; in the latter case, centrifugal and Coriolis forces may also need to be taken into account. A coarser background grid can also be used to represent the global flow in the remainder of the domain not covered by any local grid. The major question that arises in these approaches is how to couple the degrees of freedom in regions where two or more grids overlap.

For interactive smoke simulation, Cohen et al. [CTG10] omit the coupling step entirely. Instead, smoke particles that lie within multiple overlapping grids are simply advected with a weighted average of the flow velocities indicated by different grids. The resulting motion is not physically valid, but works well for interactive applications.

To perform correct coupling for globally incompressible flow, there are two possible strategies: solve for incompressibility as usual in each grid and transfer data between them in an outer loop, or build a single discretization that couples all the grids together. Dobashi et al. [Dob+08] use the former approach to efficiently simulate interaction between smoke and rigid objects. Pressure is solved using a modified Gauss-Seidel solver where each iteration follows three steps. First, data from coarser grids is copied to the boundary cells of finer grids for use as boundary conditions. Then, pressure is computed independently on each grid. Finally, data from the interior cells of finer grids is copied to overlapping coarser grids. This process is repeated until it converges; unfortunately, convergence can be slow. More recently, English et al. [Eng+13] developed a full moving Cartesian grids model. Instead of solving for pressure on each grid separately, they combine all grids in a single discretization. Coarse grid cells that contain the cell center of a finer cell are removed, and a Voronoi mesh is built using the remaining cell centers. An monolithic Poisson solver then computes the pressure over the entire mesh.

In summary, moving grids are a good solution for accelerating Eulerian fluid simulation. In the applied math vocabulary, they belong to dynamic domain decomposition methods. These methods tackle with success the challenge of increasing the local accuracy of Cartesian grids methods while keeping their natural efficiency. However, while such methods are extremely efficient for environments where regions of interests are known or easy to compute, they are not well suited for interactive scenarios where any number of new regions of interest may pop up at any location and time, and where adaptive particle simulation remain more appropriate.

3.3.3 Mixed models

Another important strategy for adaptively focusing computation in computer animation is to selectively apply a mixture of different computational models. The motivation is that every model has its own strengths and weaknesses, and some are better suited to some situations than others. In particular, many methods combine Eulerian reference frames (which describe motions relative to a fixed point in space) with Lagrangian reference frames (which describe motions relative to a physically important moving trajectory). The driving goal is to judiciously combine techniques in a way that leverages the strengths of each model and suffers none of the drawbacks.

While most of these mixed models represent a clever combination of techniques whose whole is greater than the sum of its parts, the mixed models which could not be classified as "adaptive" have been omitted from this work. This section only discusses mixed models that adaptively change from one model to another when the situation calls for it. This section is separated into methods used to simulate solid objects and methods used to simulate fluids.

We note that numerous other techniques use two-way coupling between different phenomena ([CMT04; Rob+08; SSF08; RK13], to name a few). However, while these approaches are adaptive in the sense that they modify computation depending on the phenomena being simulated, we feel these two-way coupling methods are outside of the scope of this section. Instead of surveying all possible combinations of different phenomena-specific discretizations, we only discuss here methods that combine different discretizations of the *same phenomena* in order to gain a computational speedup.

3.3.3.1 Solids

While most mixed models for simulating solid dynamics do not quite adapt their models to their environment, both Sueda et al. [Sue+11] and Servin et al. [Ser+11] successfully address the challenging problem of simulating stiff elastic strands in a collision-heavy scenario. They accomplished this by introducing Eulerian nodes into a largely Lagrangian strand simulation. The Eulerian nodes sit still at important contact points, while the standard Lagrangian nodes sample the strands as normal. These models are "adaptive" under our definition, because the Eulerian nodes add local detail and their location is decided during run-time.

3.3.3.2 Fluids

The large memory and computation requirements of 3D fluid discretizations are undesirable, so 2D simplifications are often preferred when applicable. In addition, an Eulerian reference frame is popular for guaranteeing a uniform mesh-spacing, maintaining cache-coherence, avoiding remeshing, and describing swirling flows without explicitly sampling complicated trajectories.

However, Eulerian methods are often inferior to their Lagrangian counterparts when sampling fine individual features like droplets and bubbles, or for explicitly tracking many individual vortices.

This section describes many techniques that adaptively combine 2D/3D and Eulerian/Lagrangian techniques in order to get the most out of a fluid simulation. We first survey various methods that combine Eulerian techniques with Lagrangian particles (droplets, bubbles, vortices, etc.). Next, we discuss how some Eulerian models also use Lagrangian particles to couple directly with SPH solvers. After that, we review some methods which combine 3D solvers with 2D techniques or with surface physics.

Eulerian simulation & Lagrangian particles Several early methods combined Eulerian fluid simulations with Lagrangian particles to animate splashing droplets. O'Brien and Hodgins [OH95] combine a 2D pipe-based fluid model with particle-based droplets. Holmberg and Wünsche [HW04] create an Eulerian waterfall model and used Lagrangian particles to animate spray, and Kim et al. [Kim+06] used particles from a 3D surface tracker to fill in missing splash details. Chentanez and Müller [CM10] combine an Eulerian discretization of the shallow water equations with a Lagrangian simulation of spray, splash, and foam particles. The particles add important missing details to the simulations and are allocated dynamically at run-time.

Researchers also use Lagrangian particles to capture bubble behavior in Eulerian simulations. Mould and Yang [MY97] augment a height-field model with Lagrangian particles for droplets and bubbles. Many simulation methods [GH04; Hon+08; Pat+13] compute a 3D Eulerian fluid simulation, and they represent bubbles that are too small to be resolved on the grid with Lagrangian particles. The differences in these methods lie in the varying bubble dynamics and the subtleties of how to transition between the Eulerian grid bubbles and the Lagrangian particle bubbles.

The main concept for these methods is to use an Eulerian representation for the bulk of the flow, but to adaptively turn to a Lagrangian particle representation whenever the Eulerian model is insufficient. This switching point is often easy to detect, because it occurs exactly when the diameter of a water droplet or bubble falls below the Eulerian grid resolution. Not only does this strategy conserve mass and momentum better than simply deleting small features, but it fills in visually important information by animating sprays as a collection of small Lagrangian droplets and foams as a collection of Lagrangian bubbles. Lagrangian droplets and bubbles are practically indispensable in a production workflow, because the small expense of adding additional point geometry with simple physics pays off with enhanced visual realism.

Eulerian simulation & SPH Eulerian methods can also be combined with Lagrangian particles in other ways beyond droplets and bubbles. Losasso et al. [Los+08] combine SPH with a FLIP simulation, Wang et al. [Wan+13] combine SPH with a Lattice-Boltzmann simulation and Chentanez et al. [CMK14] combine SPH with 3D Eulerian grid. This idea of adaptively switching between Eulerian simulations and SPH is still an active research topic. Using SPH instead of simple passive or ballistic particles is clearly more realistic, but it comes with the expense of additional neighborhood operations and more delicate numerical calculations in general. It is not clear yet whether the realism gained by augmenting an Eulerian simulation with SPH particles is worth the computational expense.

Combining 2D & 3D Several techniques like discretizing the shallow water equations [LP02; Hag+05] or linearized wave equations [KM90; Tes04a; KB14] are useful for reducing computational degrees of freedom, but we do not believe they are inherently *adaptive* by themselves, and we do not discuss them in detail in this document. However, several techniques utilize these 2D discretizations in ways that we would classify as an adaptive "mixed-model" approach.

The work of Thürey et al. [TRS06] combines a 3D Lattice-Boltzmann simulation with a 2D simulation of the shallow water equations, allowing a local region that to adapt to 3D phenomena while distant motion remains a simple height field. Chentanez et al. [CMK14] combine a 3D Eulerian solver with both a 2D shallow water solver and a particle-based fluid simulation. Mixing three models allows them to simulate extremely detailed water interactions at efficient frame rates. These methods fit our definition of adaptivity, because they both locally increase the computational degrees of freedom in interesting regions, and these decisions of where to place the new degrees of freedom are decided at run-time as the simulation progresses. As these papers suggest, adaptively switching simulation dimensions will clearly make animations more efficient, because the computational complexity plummets as the simulation transitions from 3D to 2D. The only reservations here are that there is a significant implementation expense to maintaining two or more solvers (one for each dimension), and the seamless coupling between dimensions is a sensitive process that is still being actively researched.

3.4 Discussion and concluding remarks

Adaptive physically-based models are becoming ubiquitous in computer graphics. In the last decade, various techniques for almost all types of deformable models have been proposed and extended. In this chapter, we have classified adaptive techniques into five different categories: (1) temporal adaptivity, (2) geometric adaptivity, (3) basis refinement, (4) moving grids and (5) mixed models. For each category, we have described the variants that were developed for different applications and have discussed their strengths and weaknesses.

Among those different categories, geometric adaptive techniques are the most studied, and are perhaps the most intuitive due to their geometrical nature. The many variations in application contexts such as dimensionality and discretization have led to a proliferation of different innovative techniques. Yet, as we have tried to show, there are also many commonalities between aspects of disparate techniques, for example in terms of refinement criteria, and there is potential for consolidation of the many approaches in this area.

In the opposite direction, mixed models represent an important area of work. Unfortunately, as they rely on the specific characteristics of each model, it is more difficult to extract general patterns and strategies. Even so, they perfectly represent the idea and the versatility of adaptive techniques by combining the strengths of different approaches as the simulation evolves.

For now, polynomial basis refinement represents only a small fraction of the methods studied. In computer graphics, this is a very recent topic, but which can build on strong foundations from engineering and applied mathematics where it has been extensively studied. Results in solid and fluid simulation show that polynomial basis refinement can indeed produce impressive animations. One of the most exciting avenues of future research is the combination of this technique with geometric adaptivity.

In subspace simulation, adaptive reduced bases can greatly extend the range of deformation that can be achieved by introducing local and non-linear deformations such as wrinkles. Nevertheless, there is still a large room for improvement and innovation, especially regarding the possibility to handle topological changes and couple different subspace simulations such as deformable solids and fluids.

Even if there are only a few works that focus on temporal adaptivity, these methods are widely used and play a crucial role in ensuring stability and efficiency. Their importance is due to two main reasons. First, spatial and temporal resolution are often strongly related. Secondly, the necessary temporal resolution is inherently dependent on the events occurring during a simulation and cannot always be predicted in advance, necessitating adaptive techniques. As we seek to resolve details at increasingly finer time scales, further research will be required to capture them without paying an exorbitant computational cost.

Adaptive methods are not without limitations, and we briefly summarize the major ones. At present, setting up a new adaptive method is quite difficult: adaptive methods have often been application-specific so far, which makes the study of existing solutions quite intricate. Adaptivity usually makes the implementation of a model much more complex and may ruin the regularity of computations, causing incompatibilities with GPU implementations. Evaluating the future overhead due to the online adaptation process is often difficult, which may make such techniques unusable in performance-constrained contexts such as interactive applications. There are also potential concerns relating to simulation fidelity. If not tackled with care, popping between different spatial and temporal resolutions may cause instabilities and visual artifacts. Furthermore, the energy diffusion that necessarily occurs when permanently adapting a model may be an issue when an accurate simulation is required.

Nevertheless, the space of adaptive simulation techniques is vast and fruitful, and many compelling benefits have been uncovered so far. There is much room for future work in developing new adaptive methods which are both easier to implement and still generic enough to be used in different applications. This challenge requires methods which, for example, minimize the overhead due to additional structures while making it possible to integrate different adaptation criteria. Many other avenues of future research remain, including combinations of different forms of adaptivity and techniques that adapt between different dimensionalities, different formulations, and other characteristics that have traditionally remained separate.

Chapter 4

Extending Adaptively Restrained Particle Simulation to Graphical Simulations

Combining efficiency with visual realism has been one of the main goals of computer graphics research in the last decade. In Chapter 3, we presented adaptive models, a general strategy to concentrate the computational time on the most interesting parts of an animated scene. We observed that the two main approaches consist of adapting time or spatial sampling. Although they can achieve impressive results, they are often difficult to implement, they may be restricted to specific applications and they sometimes generate discontinuity artifacts due to sudden simplifications.

A different approach for adaptive simulation [AR12] was proposed in the context of molecular dynamics (MD). Contrary to most of the methods reviewed in Chapter 3, Adaptively Restrained Particle Simulations (ARPS) do not adapt time or spatial sampling, but rather switch the positional degrees of freedom of particles on and off, while letting their momenta evolve. The key idea is that since most of the computation time is spent in computing interaction forces based on positions, particles with low velocity could be considered fixed in space - and the corresponding interaction forces constant - until they accumulate enough momentum to start moving again. Therefore, inter-particles forces do not have to be updated at each time step, in contrast with traditional methods that spend a lot of time there.

While freezing objects to gain computation time has been used in video games (see Section 3.1.2), the question of when and how to release them has not been extensively studied, and has mainly relied on *ad hoc* heuristics.

Adaptively Restrained Particle Simulations (ARPS), in contrast, introduce a physically sound approach with proven correctness, and has been successfully used in the context of predictive, energy-and momentum-conserving particle simulation.

In this chapter, we explore the use of Adaptively Restrained (AR) particles for graphics simulations. Our experiments show that this new, simple strategy for adaptive simulations can provide significant speed-ups more easily than traditional adaptive models. The key contributions of our work are as follows:

- We adapt ARPS to particle-based fluid simulations and propose an efficient incremental algorithm to update forces and scalar fields.
- We introduce a new implicit integration scheme enabling to use ARPS for stiff objects simulations such as cloth.

The remainder of this chapter is organized as follows: Section 4.1 presents the initial formulation of ARPS that was introduced for molecular dynamics simulations and explores its potential for computer graphics applications. Section 4.2 and 4.3 respectively introduce our extension of ARPS to fluid and stiff objects simulations. Section 4.4 deals with the practical implementation and parameters tuning. Section 4.5 concludes and gives some perspectives of future works.

The works described in this chapter were presented at the conference *VRIPHYS 2013* [Man+13]. A video illustrating the method and the results is available here: `https://youtu.be/RpJjGAoqp50`.

4.1 Adaptively Restrained Particles

Basic ideas: Adaptively Restrained Particle Simulations (ARPS) [AR12] were recently developed to speed up particle simulations in the field of Molecular Dynamics. They rely on Hamiltonian mechanics, where the state of a particle system is described by a position vector $\mathbf{x}$ and a momentum vector $\mathbf{p}$, and its time evolution is governed by the following differential equations:

$$\begin{aligned}\frac{d\mathbf{p}}{dt} &= -\frac{\partial \mathcal{H}}{\partial \mathbf{x}} \\ \frac{d\mathbf{x}}{dt} &= +\frac{\partial \mathcal{H}}{\partial \mathbf{p}}\end{aligned}$$

Here, the Hamiltonian $\mathcal{H}$ is the total mechanical energy given by

$$\mathcal{H}(\mathbf{x},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p} + V(\mathbf{x}) \tag{4.1}$$

where the first term corresponds to the kinetic energy, while the second represents the potential energy. In [AR12], an *adaptively restrained* (AR) Hamiltonian is introduced.

$$\mathcal{H}_{AR}(\mathbf{x},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\Phi(\mathbf{x},\mathbf{p})\mathbf{p} + V(\mathbf{x}) \tag{4.2}$$

The matrix Φ is a block-diagonal matrix used to switch on or off the positional degrees of freedom of the particles during the simulation. Each 3x3 block corresponds to a particle i and is equal to $\Phi_i(\mathbf{x}_i,\mathbf{p}_i) = m_i^{-1}[1-\rho_i(\mathbf{x}_i,\mathbf{p}_i)]\mathbf{I_{3x3}}$. The function $\rho_i \in [0,1]$ is called the *restraining function*. When $\rho_i = 0$, $\Phi_i = m_i^{-1}$ and the particle is *active*: it obeys standard (full) dynamics. When $\rho_i = 1$, $\Phi_i = 0$ and the particle is *inactive* (not moving). When $\rho_i \in [0,1]$, the particle is in transition between the two states. The restraining function ρ_i of each particle is used to decide *when* to switch positional degrees of freedom on or off. In [AR12], ρ_i depends on the particle kinetic energy. The function uses two thresholds, a restrained-dynamics threshold ϵ^r and a full-dynamics threshold ϵ^f. It is defined as

$$\rho_i(\mathbf{p}_i) = \begin{cases} 1, & \text{if } 0 \le K_i(\mathbf{p}_i) \le \epsilon_i^r \\ 0, & \text{if } K_i(\mathbf{p}_i) \ge \epsilon_i^f \\ s(K_i(\mathbf{p}_i)) \in [0,1], & \text{elsewhere} \end{cases} \tag{4.3}$$

where $K_i = \mathbf{p}_i^2/2m_i$ is the kinetic energy, and s is a twice-differentiable function. In practice a 5th-order spline is used.

Adaptive equations of motion: The adaptive equations of motions are derived from the AR Hamiltonian (Equation (4.2)).

$$\begin{aligned} \frac{d\mathbf{p}}{dt} &= -\frac{\partial \mathcal{H}_{AR}}{\partial \mathbf{x}} = -\frac{\partial V(\mathbf{x})}{\partial \mathbf{x}} \\ \frac{d\mathbf{x}}{dt} &= \frac{\partial \mathcal{H}_{AR}}{\partial \mathbf{p}} = M^{-1}[I-\rho(\mathbf{p})]\mathbf{p} - \frac{1}{2}\mathbf{p}^T M^{-1}\frac{\partial \rho(\mathbf{p})}{\partial \mathbf{p}}\mathbf{p} \end{aligned} \tag{4.4}$$

While the momenta evolve as in classical Hamiltonian mechanics, the positions evolve differently. When a particle's momentum is small enough, the particle becomes inactive and stops moving. However, even if the particle is inactive, its momentum may change. Therefore its kinetic energy may become large enough again for the particle to resume moving. In general, particles switch between active and inactive states during the simulation.

Notice that $-\dfrac{\partial V(\mathbf{x})}{\partial \mathbf{x}}$ is equal to $\mathbf{f}(\mathbf{x})$, the forces applied on the particle system. In the initial formulation presented in Equation 4.2, these forces are conservative and only depend on positions. In Section 4.2 and Section 4.3, we detail how to use ARPS in dissipative systems where damping forces based on velocity are used.

Finally, we introduce a notion that we will use in the following sections: the *effective velocity*. It is the rate of position change of a particle i and can be expressed by the following equation.

$$\mathbf{v}_i^{eff} = \frac{1}{m_i}\left((1-\rho_i(\mathbf{p}_i))\mathbf{p}_i - \frac{1}{2} \parallel \mathbf{p}_i \parallel^2 \frac{\partial \rho_i(\mathbf{p}_i)}{\partial \mathbf{p}_i}\right) \tag{4.5}$$

A simple example: Consider a 1D harmonic oscillator: a particle attached to the origin with a perfect spring. Figure 4.1 shows a phase portrait of the corresponding AR system.

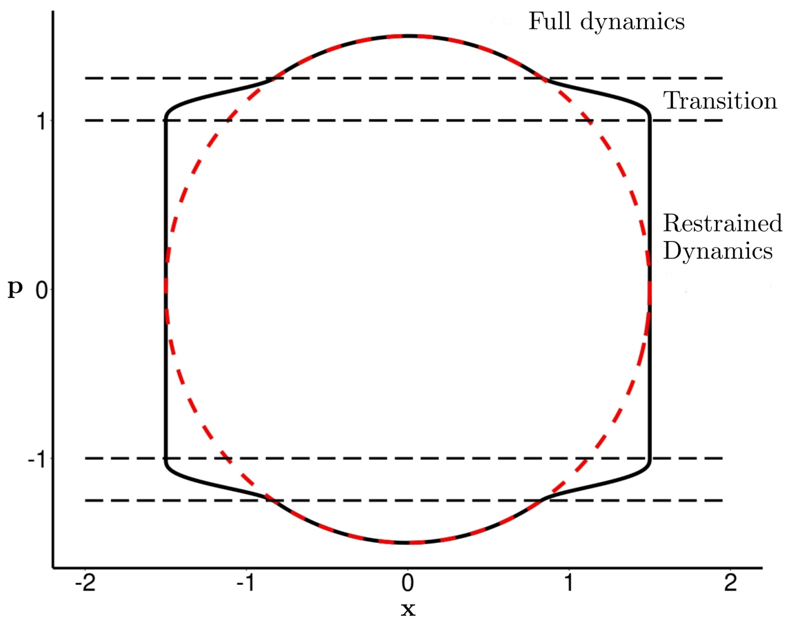


Figure 4.1: Phase portrait of a harmonic oscillator. The red dotted ellipse corresponds to standard Hamiltonian mechanics, while the solid black line corresponds to ARPS. During restrained dynamics, the particle's momentum is accumulated. When enough energy has been accumulated, a transition phase takes place leading the particle to switch back to full dynamics.

In classical mechanics, the trajectory of the state in this (position, momentum) space is an ellipse, the size of which depends on the (constant) energy of the system. Using ARPS, the position is constant (vertical straight parts) as long as the kinetic energy is small enough, while it is an ellipse as long as the kinetic energy is large enough. These trajectories are connected by a transition corresponding to an energy between the two thresholds of Equation (4.3). The closed trajectory corresponds to a constant *adaptively restrained energy* H_{AR}.

Generalization: Due to the similarity of the adaptive kinetic energy with the standard kinetic energy, particle systems simulated using ARPS exhibit the expected properties of standard physical simulation, namely the conservation of momentum and (adaptive) energy. It is therefore possible to perform macroscopically realistic simulations with reduced computation time as illustrated in Figure 4.2.

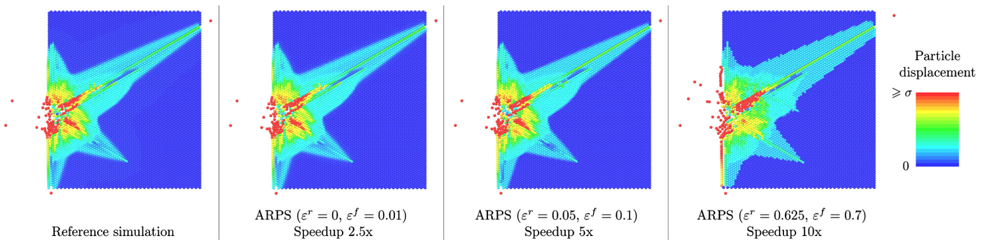


Figure 4.2: A particle is launched at high velocity toward a 2D static particle system made of 5390 particles, resulting into a collision cascade [AR12]. On the left, the result of a full dynamics simulation. The three other images are the result of ARPS with different thresholds which allow a smooth trade between efficiency and precision.

Computational performance: The authors of [AR12], Artemova and Redon, obtained significant speed-ups by exploiting the immobility of particles. Figure 4.2 illustrates one of their result where a 10× speed-up was achieved in a collision scenario while keeping a behavior close to the reference one. In this example, inter-particles forces were derived from a Lennard-Jones potential. To save time on inactive particles, they proposed an incremental method to update the particles' forces at each time step:

1. All forces that were acting on each active particle at the previous time step are subtracted based on previous position.

2. New forces based on current positions are added to each active particle.

The increase of computational performance comes from the absence of force computation between two inactive particles and the absence of neighbor search for inactive particles. As these two steps are common bottlenecks in particle simulation, significant speed-up can be achieved.

Potential benefits of extension to computer graphics: Molecular dynamics often inspired particle-based simulations in computer graphics. The same bottleneck, namely inter-particles forces computation based on neighbor search, is present in the two fields, so we can expect interesting performance for ARPS in graphics. The remainder of this chapter explores two applications of ARPS to graphical simulations:

1. Particle-based fluid simulation. Originally, ARPS has been applied to conservative system. While fluid simulation involves position-based forces, it also involves velocity-based viscosity forces that need to be taken care of. We propose a simple method to handle them. Additionally, we extend the incremental algorithm proposed by Artemova and Redon to update forces as well as scalar fields.

2. Stiff object simulation. Explicit integration of stiff objects such as cloth is expensive due to stability issues. A well known solution is to use implicit integration instead. We derive an implicit formulation of ARPS and propose a hybrid solver to exploit the inactivity of particles in cloth simulation.

It is clear that ARPS is not well-suited for simulations where all degree of freedom move: classical spatial adaptation is better suited in this case. In contrast, ARPS is best suited for simulations where most parts are immobile but may resume moving at any time. Even if these situations are not the most visually exciting, they are very common in computer graphics: they include simulation of characters clothing when many of the characters are at rest, surgical simulations with local-only user interaction, and the animation of large volumes of liquid, when most of it already came to rest.

4.2 Extension to SPH fluid simulation

As presented in section 2.1.2.2, SPH fluid simulation is widely used in computer graphics and many methods have been proposed [DC96], [MCG03], [SP09], [Ihm+14a]. SPH approximates fluid dynamics with a set of particles. The particles are used to interpolate the properties of the fluid anywhere in space. Each particle samples the fluid properties such as density, pressure or temperature. All these properties are updated based on the particle neighbors and are used in short-ranged inter-particle forces.

4.2.1 Incremental update

To integrate ARPS, we chose WCSPH (Weakly Compressible Smoothed Particle Simulation) [BT07], a standard SPH formulation [DC96], [MCG03]. For the sake of simplicity, we limit our discussion to the main inter-particles forces: pressure and viscosity. Algorithm 1 describes the classical simulation loop.

Algorithm 1 WCSPH simulation loop

```
for all particle $i$ do
    find neighbors $j$
end for
for all particle $i$ do
    compute density $\rho_i$ (e.g. Eq. 2.36)
    compute pressure $p_i$ using $\rho_i$ (e.g. Eq. 2.37)
end for
for all particle $i$ do
    $\mathbf{f}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$ (e.g. Eq. 2.41)
    $\mathbf{f}_i^{viscosity} = m_i \eta \nabla^2 \mathbf{v}_i$ (e.g. Eq. 2.45)
    $\mathbf{f}_i(t) = \mathbf{f}_i^{pressure} + \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{other}$
end for
for all particle $i$ do
    $\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{f}_i(t)/m_i$
    $\mathbf{x}_i(t+\Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t+\Delta t)$
end for
```

ARPS can save time on each computation step involving the position of the particles. In SPH fluid simulation, every property of a particle is computed as a weighted sum of its neighbors' property. And the weights are computed based on the position of the particle and its neighbors. Therefore, time can be saved as soon as a particle is close to immobility. This makes SPH a perfect candidate for ARPS.

As long as a particle is inactive and is surrounded by inactive particles, its properties and neighborhood do not need to be updated. In practice, this includes pressure and viscosity forces as well as density and pressure scalar field.

To gain computational time from the inactive particles, we extend the incremental algorithm proposed in [AR12] (see Algorithm 2). In this algorithm we denote as *active* every *active* or transitive particle. From one step to another, only the contributions to physical properties from active particles are updated. Thus, even inactive particles whose neighbors are active are kept up-to-date.

Algorithm 2 WCSPH+ARPS simulation loop

```
if step=1 then
    Perform Algorithm 1
else
    for all active particle and its neighbors $i$ do
        Subtract old density contribution from neighbor $j$
        Subtract old pressure/viscosity force contribution from neighbor $j$
    end for
    for all active particle neighbors $i$ do
        find neighbors $j$
    end for
    for all active particle and its neighbors $i$ do
        Add new density contribution from neighbor $j$
        Update pressure $p_i$
        Add new pressure/viscosity force contribution from neighbor $j$
        Add new density contribution from neighbor $j$
    end for
    for all particle $i$ do
        $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t \mathbf{f}_i(t)/m_i$
        $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i^{eff}(t + \Delta t)$
        Update particle's state based on $\rho(\mathbf{p_i})$.
    end for
end if
```

4.2.2 Viscosity

In SPH, viscosity forces play an important role in the stability of the simulation and in the range of phenomena that can be simulated. However, ARPS was designed for conservative systems and no damping terms involving the velocity of the particle is involved, as we can see in the equation of motion (see Equation 4.4). In ARPS, velocity can be represented in two different ways. We may define it based on the momentum and set $\mathbf{v}_i = \mathbf{p}_i/m_i$, or based on the *effective velocity* $\mathbf{v}_i^{eff}$ defined in Equation 4.5. In the first case, we can get time-varying forces even for inactive particles, which we want to avoid. We therefore use the effective velocity of the particle, as defined in Equation(4.5). Applied to a harmonic oscillator, this results in the behavior illustrated in Figure:4.3. The more the particle is damped the longer it remains inactive, which is an intuitive behavior.

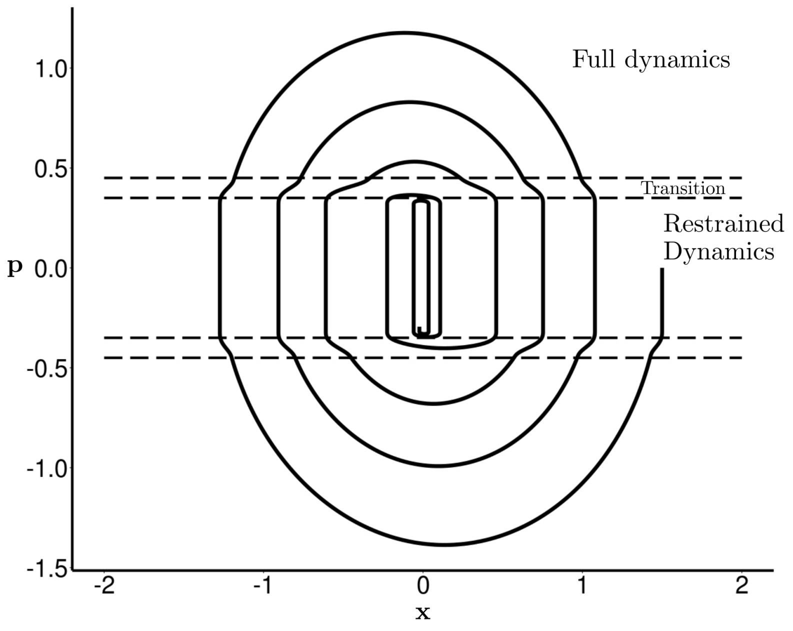


Figure 4.3: Phase portrait of our viscosity approach in ARPS. As with a classic damped oscillator we obtain a spiral phase portrait.

4.2.3 Modified inactivity criterion

Since our viscosity force vanishes along with the effective velocity of the particle, it drags down the kinetic energy asymptotically close to the inactivity threshold, without ever reaching it. Consequently, particles only subject to viscosity forces never become inactive, and we do not spare computation time, even when the particles get nearly static. To remedy this problem, we consider inactive the particles which effective velocity fall below a user-defined threshold.

4.2.4 Performance

We performed two experiments to measure computation time. The first one is a classical dam break simulation made of 5000 particles. We can see in Figure 4.4 that during fast movements most of the particles are active and the adaptive simulation stays close to reference simulation. Therefore small scale details like splashes are preserved.

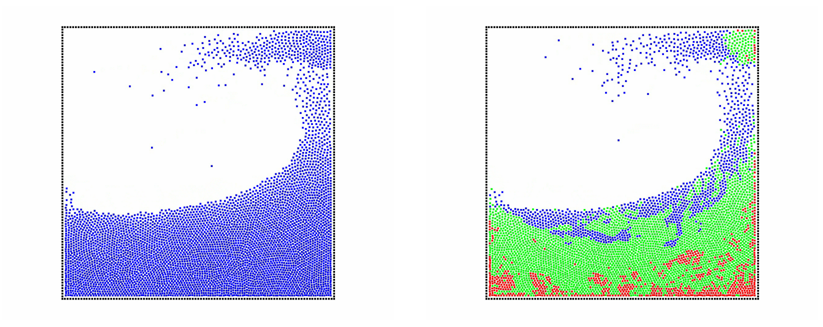


Figure 4.4: A dam break simulation with 5000 particles simulated with WCSPH (on the left) and with our adaptive method (on the right). On the right image, blue corresponds to full-dynamics particles, green to transition particles and red to restrained particles.

As soon as most particles come to rest and become inactive the speed-up can be significant (see Table 4.1). For 15s, the mean speed-up is 3.8. The speed-up can locally reach 25.7.

Simulation Time	SPH	ARPS	Speed-up
15s	893s	232s	{0.91, 25.73, 3.85}

Table 4.1: Dam break - Computation time and speed-up {min, max, mean}

The second experiment is the creation of a permanent flow with 4240 particles. As we can see in Figure 4.5, once the permanent flow is installed a large amount of particles are restrained. We reach an interesting speed-up while keeping a motion close to the reference (see Table 4.2).

Simulation Time	SPH	ARPS	Speed-up
30s	2166s	814s	{0.83, 3.99, 2.66}

Table 4.2: Permanent flow - Computation time and speed-up {min, max, mean}.

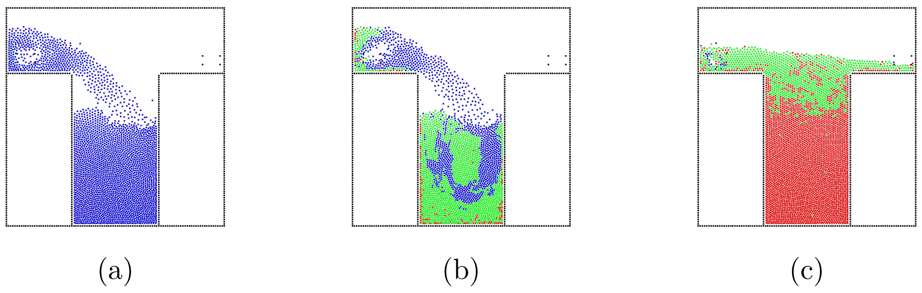


Figure 4.5: A permanent flow simulation with 4240 particles. (a) is a classic WCSPH simulation. (b) is our adaptive method at the same time step as (a) with restrained particles in red. (c) is our adaptive method once the permanent flow is installed.

These examples show that ARPS can effectively be extended to speed up liquid simulation in computer graphics. The next section consider its application to cloth animation.

4.3 Extension to stiff objects: Implicit Integration

In this section we explore the application of ARPS to stiff objects simulations, such as clothes, and propose an implicit integration scheme which saves computation time for particles at rest. Implicit integration for cloth simulation was introduced by Baraff and Witking [BW98]. An introduction to implicit integration is proposed by Witkin et al. [WBK01]. While originally formulated on velocity, it can be straightforwardly expressed on momentum. Instead of integrating the momentum using the forces at the current time step, implicit integration uses the forces at the end of the current step. As we do not know these forces we end up with a non linear function. We can linearize this function and solve the resulting linear system to obtain the next momentum

$$(I - \Delta t^2 K M^{-1})\Delta \mathbf{p} = \Delta t(\mathbf{f} + \Delta t K M^{-1}\mathbf{p}) , \tag{4.6}$$

where $\mathbf{f}$ are the forces applied on the system, $K = \dfrac{\partial \mathbf{f}}{\partial \mathbf{x}}$ is the stiffness matrix and M is the mass matrix. Solving the linear system is more costly than explicit integration, but it allows the use of larger time steps without any loss of stability, enabling to advance much faster. For our experiments, we used simple spring forces between particles.

4.3.1 ARPS Implicit Integration

We derive an implicit integration scheme from Adaptively Restrained equations of motion. The linear system has to take into account the state of the particles.

The discrete equations of motions for implicit Euler are

$$\begin{aligned} \Delta \mathbf{p} &= \Delta t \mathbf{f}(\mathbf{x}_{n+1}, \mathbf{p}_{n+1}) \\ \Delta \mathbf{x} &= \Delta t \Big(M^{-1}(1-\rho(\mathbf{p}_{n+1}))\mathbf{p}_{n+1} \quad . \\ &-\frac{1}{2}\mathbf{p}_{n+1}^T M^{-1} \frac{\partial \rho(\mathbf{p}_{n+1})}{\partial \mathbf{p}} \mathbf{p}_{n+1} \Big) \end{aligned} \tag{4.7}$$

We perform a Taylor-Young expansion of $\mathbf{f}(\mathbf{x}_{n+1}, \mathbf{p}_{n+1})$ and introduce $\Delta \mathbf{x}$ in the expended momenta equation. We then perform a Taylor-Young expansion of $\rho(\mathbf{p}_{n+1})$ in the momentum equation, which gives us the following equation system.

$$(I - \Delta t^2 KRM^{-1})\Delta \mathbf{p} = \Delta t(\mathbf{f} + \Delta t K M^{-1} s) \tag{4.8}$$

R is a block-diagonal matrix where each 3×3 block R_{ii} is

$$\begin{aligned} R_{ii} &= I - \rho(\mathbf{p}_n^i) - \mathbf{p}_n^i \frac{\partial \rho(\mathbf{p}_n^i)}{\partial \mathbf{p}_i}^T \\ &-\frac{1}{2}\mathbf{p}_n^i \mathbf{p}_n^{iT} \frac{\partial^2 \rho(\mathbf{p}_n^i)}{\partial \mathbf{p}_i^2}^T - \frac{\partial \rho(\mathbf{p}_n^i)}{\partial \mathbf{p}_i} \mathbf{p}_n^{iT} \ , \end{aligned} \tag{4.9}$$

while s is a $3N$ vector where N is the number of particles, and each s_i is

$$s_i = \mathbf{p}_n^i - \rho(\mathbf{p}_n^i)\mathbf{p}_n^i - \frac{1}{2}\mathbf{p}_n^{iT} \mathbf{p}_n^i \frac{\partial \rho(\mathbf{p}_n^i)}{\partial \mathbf{p}_i}. \tag{4.10}$$

Note that if all particles are inactive then we have $R = 0$ and $s = 0$ and we get an explicit formulation.

$$I\Delta \mathbf{p} = \Delta t \mathbf{f} \tag{4.11}$$

Conversely, if all particles are active then $R = I$ and $s = \mathbf{p}$ and we get the classical implicit formulation of Equation(4.6). In the general case, we loop over time using algorithm 3.

Algorithm 3 Implicit integration scheme

```
for each time step do
    compute $\rho, R, s, \mathbf{f}$.
    compute $A = I - \Delta t^2 KRM^{-1}$
    compute $b = \Delta t\mathbf{f} + \Delta t^2 KM^{-1}s$
    solve $A\Delta \mathbf{p} = b$
    compute $\mathbf{p}_{n+1} = \mathbf{p}_n + \Delta \mathbf{p}$
    compute $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta t M^{-1}(R\Delta \mathbf{p} + s)$
end for
```

Figure 4.6 shows the phase portrait of a harmonic oscillator simulated using our implicit formulation. As expected, the well-known numerical damping

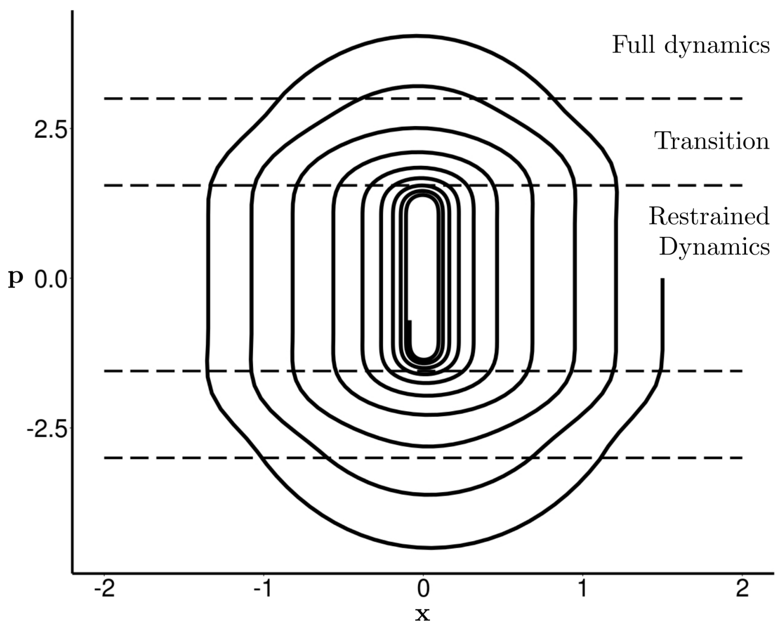


Figure 4.6: Phase portrait of an implicit ARPS harmonic oscillator.

effect of implicit Euler provides us with the same behavior we could observe with a damped harmonic oscillator.

To include a more controllable damping term in the physical model, we derived an implicit formulation which includes a damping term $\mathbf{f}_d = -\gamma \mathbf{v}^{eff}$.

$$(I + \Delta t \gamma M^{-1} R - \Delta t^2 K R M^{-1}) \Delta \mathbf{p} = \Delta t (\mathbf{f} + \Delta t K M^{-1} s + \mathbf{f}_d) \qquad (4.12)$$

Solving the equation: We exploit inactive particles to save computation time. As discussed earlier, inactive particles can be handled using explicit integration, which is much simpler. When a particle is inactive and has no active neighbors we do not need to include it in the linear system. We thus build the minimal linear system, which only contains active particles and their neighbors. These particles are implicitly integrated, while the others are explicitly integrated.

Figure 4.7 shows a hanging cloth with active and inactive particles. At the beginning all the particles become active. Then a moving front of inactivation/reactivation traverses the cloth at decreasing frequency. The cloth finally finds a rest position, where all the particles are inactive and simulated explicitly, saving computation time. The particles can become active again if external forces or imposed motion are applied. Table 4.3 shows performances we achieved with our hybrid solver. As soon as a large number of particles become inactive the simulation is explicitly integrated and interesting speed-up can raise.

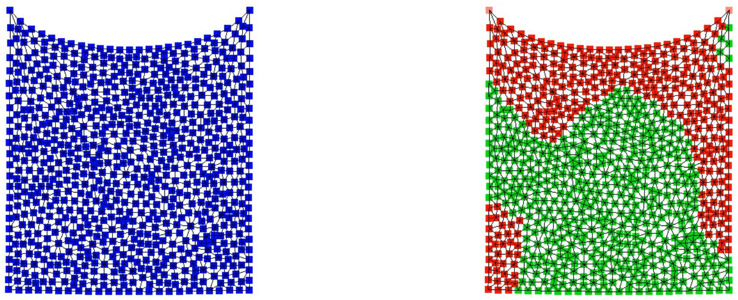


Figure 4.7: Hanging cloth. Left: traditional implicit simulation. Right: implicit ARPS simulation with a varying set of active and inactive particles.

Simulation Time	Implicit	Hybrid	Speed-up
20s	16.9s	6.2s	{0.77, 15.16, 2.73}

Table 4.3: Implicit vs Hybrid solver. Computation time and speed-up {min, max, mean}.

However, while smoothly varying external forces are well handled by our simulator, we noticed instabilities when interacting strongly with the model. They seem to occur during the transition between the transitive and the full-dynamics states. A more thorough study of the influence of the transition function ρ on the stability of the system would be necessary to come up with robust implicit ARPS simulations. This transition should be really well taken to avoid any instabilities.

4.4 Implementation

4.4.1 Parameters

ARPS requires setting the two parameters, ϵ^r and ϵ^f of Equation 4.3. The main goal of ARPS in computer graphics is to save time when nothing happens. So we generally want a low ϵ^r not to miss interesting movements. When sudden movements occur, we want a standard reaction, so we want the inactive particles to quickly become active. This requires a short transition, *i.e.* ϵ^f close enough to ϵ^r. However, due to discrete time integration, a short transition may be stepped over, or not enough sampled, which may result in instabilities. In table 4.4 we refer the thresholds used in our simulations.

	ϵ^r	ϵ^f	Tolerance
SPH	1-e6	2-e5	8e-5
Cloth	0.05	1	1e-4

Table 4.4: ARPS thresholds for SPH and Cloth simulation

4.4.2 Linear solver

A linear equation solver is necessary in implicit integration, as presented in Section 4.3. In contrast with most formulations, implicit ARPS generally results in an unsymmetrical equation matrix, due to the matrix products in Equation(4.8). We currently use a sparse LU solver from the *umfpack* library, but it would be interesting to try a Conjugate Gradient method for unsymmetrical matrices to control the computation time, as it is usually done in implicit integration.

4.4.3 Choice of the restraining function and criterion

In ARPS the restraining function is a 5^{th}-order spline. The spline directly depends on particle kinetic energy which is the *restraining* criterion. The implicit solver involves second derivatives of the restraining function, which may have large values, leading to instabilities. We found that controlling the state of the particles based on momenta norm rather than kinetic energies seems to mitigate this and lead to more stable simulations. Investigating this issue would be an avenue for future work.

4.5 Discussion and concluding remarks

In this chapter, we have shown that ARPS, a new, simple approach to adaptive simulation, can effectively be applied to computer graphics. We have demonstrated that through two specific applications. For SPH simulations, we have obtained significant speed-ups with only minor changes to the original simulation method. For stiff materials, we achieved promising results for implicit integration. From this work, we distinguished two main directions of study. First, it is crucial to address the stability issues that we met. A first step in this direction would be to conduct a careful study of the restraining function. Second, we think that better results could be achieved by employing non-physically-based transition criteria. Indeed, the current one, based on kinetic energy, is well adapted to molecular dynamics simulation. However, in computer graphics, we are more interested in visual results. Therefore, transition thresholds, based on visibility or distance to camera, would certainly allow an even more focused computational power where it most contributes to the quality of the result.

While ARPS allows one to save computational time and to maintain a constant number of degrees of freedom, particle systems generally require a dense sampling in order to produce visually plausible behavior. The large number of degrees of freedom resulting from this sampling imposes a strict limit on computational time and memory consumption. Recently, new deformable models have been proposed to achieve complex elastic behaviors with interactive performance by using a very small number of degrees of freedom. The frame-based model introduced in Section 2.1.3.2 for the simulation of elastic solids belongs to this family. Unfortunately, these models are generally not able to handle topological changes without a large computational time, which makes them not interesting anymore. In the next chapter, we address this challenge by introducing a novel method for the interactive and detailed cutting of thin sheets.

Chapter 5

Detailed Cutting of Thin Deformable Models with Sparse Sampling

COMBINING interactive user actions and detailed convincing animations is crucial for the user's experience in simulation and games. Unfortunately, computational constraints limit the fidelity that can be achieved with physics-based animation in interactive simulations. Often, the simulated objects lack of details compared to the rest of the virtual environment. Furthermore, operations that modify the structure of the simulated objects, such as cutting, may be incompatible with faster simulation methods. When not prohibited, the latter generally exhibit strong limitations. Indeed, the level of sampling of a physically-based model usually depends on geometric complexity. Detailed cuts result in an increase of the sampling which directly impacts the performance. In practice, the number of samples is limited to ensure real-time performance. This limitation quickly prevents the user from applying detailed cuts.

In this chapter, we address the issue of enabling detailed cuts of thin deformable sheets at interactive rates. Our method is able to capture detailed cuts while using a relatively low number of control nodes for the physically-based model. Our approach to decoupling the sampling of the physical and of the geometric model, is to use a mesh-less simulation method called the frame-based model [Gil+11] that we presented in Section 2.1.3.2. In this method, the deformation field induced by animated frames is applied to the geometric model using skinning weights. As each frame can cover a large, detailed region of the geometric mesh, only a few of them are required.

To achieve user-driven cuts in a frame-based simulation, we allow cuts to be performed anywhere over the underlying mesh. We build a non-manifold grid that keeps track of the mesh topology at the simulation level and allows us to incrementally adapt the frames regions of influence in order to represent the cut. Although remaining low, the number of frames does increase during a cut. In particular, when a model is cut apart, at least one frame is needed to represent each disconnected component. Therefore, we detect crucial cutting events, enabling us to automatically insert new frames when and where they are needed. In order to reduce computations, we exploit the locality of the ongoing cutting gesture to incrementally update all the data used for the simulation.

Our contributions include:

- The building of a non-manifold grid to compute shape functions that faithfully represent the complex topology of the visual mesh while keeping a low number of control nodes (Section 5.3).
- The dynamic re-sampling of new frames into disconnected parts (Section 5.4).
- The incremental update of the different components of the simulation that were concerned by the cut (Section 5.5).

Our method can be used to simulate a wide variety of objects, such as stretchable cloth or pieces of paper. It features a very low number of frame nodes, high resolution mesh embedding, numerous and detailed cuts. Performance ranges from interactive to offline depending on the desired accuracy and the complexity of the cuts. We illustrate our method with examples inspired by the traditional Kirigami artform.

We motivate our work with respect to existing methods for the simulation of cutting and fracture in Section 5.1. We choose to not include this section in Chapter 2 for two reasons: Firstly, this is a high level presentation of related works and we think a more detailed presentation of the underlying mathematics would be needed to be a part of the state of the art chapter; Secondly, this configuration allows us to keep a more coherent discussion about our choices for the model. In Section 5.6, we illustrate our method in different scenarios and detail the computational results. Finally, we discuss limitations and future work in Section 5.7.

The works described in this chapter were presented at the conference *MIG 2015* [Man+15]. A video illustrating the method and its results is available here: `https://youtu.be/coA_tcomWlE`.

5.1 Related work on cutting and fracture

Cutting and fracture are both fascinating behaviors which can be simulated separately. In fracture, stress measurements predict how the material breaks. In cutting, the interaction with a tool defines the cut path. For more details about cutting we refer the reader to the recent survey of Wu et al. [WWD15]. Our review focuses on the modeling of topological changes in deformable models.

A first possibility consists of using the same model for physics simulation and visualization. Topological changes are then mostly modeled by remeshing operations. Simple and fast remeshing techniques such as element deletion or element splitting were proposed. The latter was used in the first simulation of brittle and ductile materials [OH99], [OBH02]. Methods that preserve element quality by local and global remeshing have also been developed. They recently led to stunning results in the simulation of multi-layered paper tearing [BDW13] and sheets tearing [Pfa+14]. These methods cause the number of simulation nodes to vary over the course of a simulation, and this variation can be problematic in a realtime game context. By limiting the scope of remeshing predictable realtime performance can be achieved [PO09]. An alternative to remeshing is to enrich elements with additional basis so that discontinuities can be represented. This is the core idea of the eXtended Finite Element Method (XFEM). It was successfully applied for offline cutting of discrete shells [Kau+09].

A second possibility is to separate the visual model from the physics model, this is known as embedding. Numerous embedding techniques have been proposed. The virtual node method [MBF04] embeds ill-shaped elements that arise after remeshing inside of well-shaped elements. This allows the robust simulation of detailed cuts [Wan+14]. However, the number of nodes increases substantially with the complexity of the cut. To reduce it, hierarchical methods were proposed and real-time cutting in medical applications has been achieved using composite finite element method [WDW11]. Still, the number of nodes grows quickly with the number of cuts and remains limited to ensure interactive frame rate. Meshless methods avoid the problem of element quality. However, boundary and discontinuities require extra effort to be sharply represented. Pauly et al. [Pau+05] proposed to use visibility criterion to perform fracture. Steinemann et al. [SOG09] used the visual model as a visibility graph to define nodes connectivity. Both methods rely on a dense sampling near the surface of the model and quickly impact performances as the number and the detail of cuts increases. There are also work to carry complex materials [Nes+09] and thin shells [RK13] in hexahedra elements.

Embedding techniques have inspired our work. They allow interesting trade-off and show impressive cutting and fracture simulations. However, the relation between the resolution of the physical model and the visualization model remains very strong. Complex cuts result in a fast increase of the number of nodes. We want to reduce this connexion as much as possible. Complex topologies could be simulated with a very low number of nodes. Then, interactivity and intuitive control would be at hand.

Few models have been proposed that simulate detailed deformable objects using a low number of nodes. Subspace simulations [BJ05] compute a small basis of deformation modes in order to achieve real-time performance on detailed models. However, the basis is acquired after heavy precomputations. Interactive scenario could not handle the update of the basis at each topological change. More recently, [Gil+11] and [Fau+11] proposed a physics-based skinning technique, called the frame-based method. Highly detailed meshes can be embedded in very coarse simulations. The control nodes are affine frames and the deformation field is described by a linear blend skinning. Classical continuum mechanics is then used to solve for the dynamics. Skinning weights, also called shape functions, are built on linear interpolation using discrete Voronoi regions. Thus, for each frame, they can represent a large region of influence with complex shape. Unfortunately, the current frame-based method does not allow the shape functions to reflect the topological changes of the embedded mesh.

5.2 Overview of the method

The goal of this work is to enable interactive detailed cutting of deformable thin sheets. The frame-based method exhibits some of the key features we are looking for: a very low number of nodes and a tunable separation between visual and physical models. We build on this framework and extend it to handle topological changes.

To transfer the cuts from the mesh to the frames, we continuously adapt the shape functions to the evolving mesh topology. This allows us to keep a constant number of nodes as long as there are no disconnected parts. In [Fau+11], the shape functions are computed on a uniform grid. The structure is simple and efficient. However, discontinuities that can be represented are very limited and strongly connected to the grid resolution. Instead, we build a non-manifold grid to compute topology-preserving shape functions (see Figure 5.1).

The main idea is that cut cells are duplicated and that each resulting instance stores different connectivities. Therefore, grid resolution depends much less on the mesh topology while keeping all topological informations. We summarize our simulation loop in Algorithm 4 and detail our remeshing algorithm in appendix A.

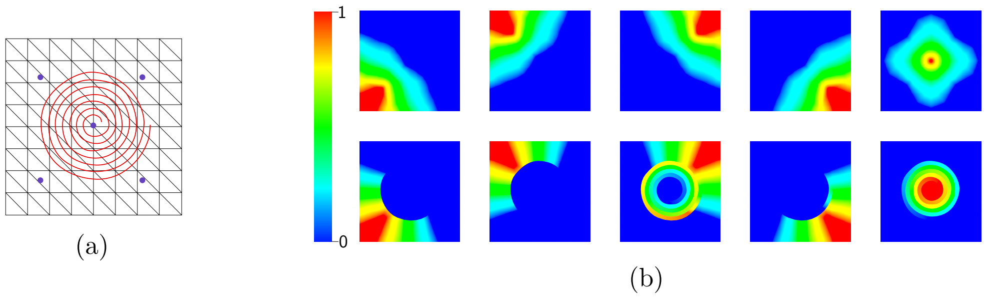


Figure 5.1: Comparison between shape functions computed on a uniform grid and on a non-manifold grid. (a) The underlying mesh (black lines) is cut by spiral (red line) and sampled with five control frames (blue circle). (b) The shape functions for each of the frame with a uniform grid (top row) and with a non-manifold grid (bottom row). Values range from 1 to 0 and are respectively depicted from red to blue. We can observe that shape functions computed on the non-manifold grid strictly preserve the details and topology of the underlying mesh.

Algorithm 4 Simulation loop

```
for each time step do
    perform a frame-based simulation step
    split the mesh along the cut
    embed the mesh in a non-manifold grid
    add new frames if required
    add new samples (collision, integration) if required
    compute shape functions on the grid
    incrementally update the samples
end for
```

5.3 Adaptive shape functions

In this section, we first summarize how Voronoi shape functions are traditionally computed. Then we detail why a non-manifold grid is necessary, how to build it and how to use it to compute the shape functions on complex topology.

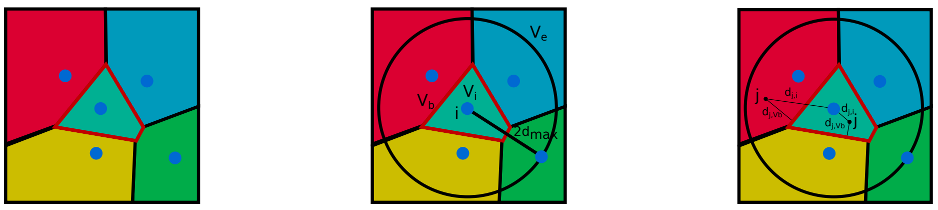
Let $w_i(x) : \Omega \to \mathbb{R}$ be the shape function for the i-th control frame, where Ω represents the domain. Starting from the Voronoi partition V of the set of control frames, we can independently compute w_i for each frame.

First, we compute the maximal distance d_{max} from the control node to its Voronoi boundary V_b. Then we extend its Voronoi region V_i to twice d_{max}. This gives a new region V_e which describes the final boundary of the shape function. Now, we can compute w_i inside V_e. We set w_i to be 1 at the frame position, 0 at the others and 0.5 on V_b. Finally, we linearly interpolate

w_i between V_b, the frame position and the boundary of V_e. We detail the interpolation in Algorithm 5 and in Figure 5.2.

Algorithm 5 Shapefunction computation

```
1: procedure COMPUTE_SHAPEFUNCTION
2:     for each frame $i$ do
3:         $V_i \leftarrow$ Voronoi region of $i$
4:         $V_b \leftarrow$ boundary of $V_i$
5:         $d_{max} \leftarrow$ maximum distance to $V_i$ boundary
6:         $V_e \leftarrow$ extend $V_i$ to $2.0 \times d_{max}$
7:         ▷ dist($A$,$B$) is the geodesic distance between $A$ and $B$
8:         for each grid cell $j$ in $V_e$ do
9:             if $j$ is inside $V_i$ then
10:                $w_i(j) = 0.5\left(1 + \frac{dist(j,V_b)}{dist(j,V_b) + dist(j,i)}\right)$
11:            else if $j$ is inside $V_e$ then
12:                $w_i(j) = 0.5\left(1 - \frac{dist(j,V_b)}{dist(j,i) - dist(j,V_b)}\right)$
13:            end if
14:        end for
15:    end for
16: end procedure
```



(a) (b) (c)

Figure 5.2: Illustrations of Voronoi shape function computation. (a) Starting from samples (blue circles), we build a Voronoi diagram using Dijkstra's shortest path algorithm. (b) Then, for each frame and its region V_i, we compute the maximum distance d_{max} to its Voronoi boundary V_b. We extend V_i to twice d_{max} which gives V_e. (c) Finally for each grid cell j in V_e we linearly interpolate using distance to the frame position and distance to V_b.

5.3.1 Voronoi shape function

In practice, the Voronoi diagram is computed using Dijkstra's shortest path algorithm on a grid in order to preserve geodesic distances. For each frame, the shape function is computed on the whole grid. As the grid resolution can be quite coarse, this is particularly fast. Negative values are clamped and weights are normalized to form a partition of unity. Then least-square approximation is performed to evaluate the shape function and its derivatives at specific positions.

Voronoi shape functions were designed in order to respect key properties that are particularly useful for physics-based animation [Fau+11] . First, they respect the Kronecker property, i.e $w_i(x) = \delta_i(x)$ where $w_i(x)$ is the shape function of node i, x is a spatial position and δ_i is Dirac function. Second, they form a partition of unity, i.e $\sum_i w_i(x) = 1$. Third, they are built to be as linear as possible in order to produce uniform deformations. Finally, they can easily be biased by material properties in order to represent heterogeneous material.

5.3.2 Non-manifold grid

As mentioned above, in [Fau+11], shape functions are computed on a uniform grid using Dijkstra's shortest path algorithm to compute geodesic distance. Starting from a uniform grid with a 8-neighbor connectivity, we could reflect topological change by changing the connectivity of the cut cells. Then, when we re-compute shape functions, the topology would automatically be taken into account as we use geodesic distance.

Unfortunately, this strategy is very limited for uniform grids and would only work in simple cases. For instance, several cuts that intersect or that create disconnected components inside one cell could not be represented. Even without cut, small gaps that lie inside one cell could not be correctly represented. Geodesic distances would be false and the object would behave as if there were no cuts or gaps. Augmenting the resolution would not solve the problem. We would fight the same issue as previous methods. Our grid resolution would be highly dependent on the complexity of the topology and the geometry of the object. It would directly impact performances.

We want each grid cell to be able to represent the connectivities of the different disconnected components that lie in the cell. To do so, each cut cell is duplicated as many times as it contains disconnected parts. Each duplicate has a specific connectivity built from the material connectivity. This results in a data structure called *non-manifold grid* (see Figure 5.3).

Non-manifold grids are used by many other cutting methods to embed fine geometric details in coarse finite element simulations. However, we make a completely different use of it. Instead of duplicating control nodes as the cells are cut, thereby increasing their number and the computation time, we use the grid to adapt the shape functions to the evolving topology of the mesh. Most of the time, the number of nodes can remain constant while representing detailed geometry and multiple cuts.

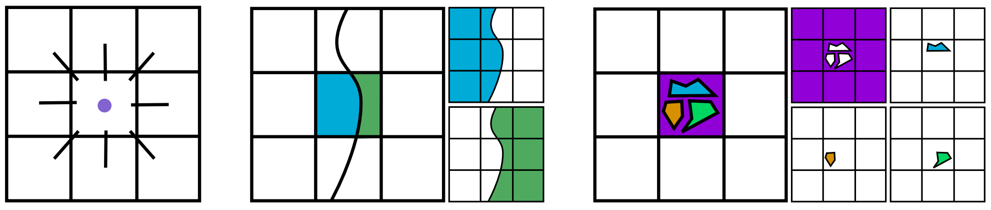

(a) (b) (c)

Figure 5.3: Illustrations of different possibilities for a non-manifold cell with eight connectivity (a). In (b), the cell is simply cut into two cells. Each duplicate of the cut cell has a specific connectivity that represent the cut topology. In (c), multiple disconnected components can be contained inside one cell. The cell is duplicated four times. Three of the duplicates have no connectivity. However they can embed complex geometry and then be simulated by adding new frames for each of the component. The fourth duplicate keeps its eight neighbors and remains independent from the three other.

There are several ways to compute this non-manifold grid. In our method, we start by embedding the mesh in a uniform grid. Mesh elements that overlap a grid cell are detected using intersections tests and are assigned to it. Then, for each grid cell, we use a flood fill algorithm to detect the disconnected parts of the mesh. This informs about how many duplicates need to be created for the cell. Finally, for each duplicate we establish its connectivity by comparing its geometry with the geometry of the neighbor cells duplicates. We summarize our method in Algorithm 6 and illustrate the main steps in Figure 5.4.

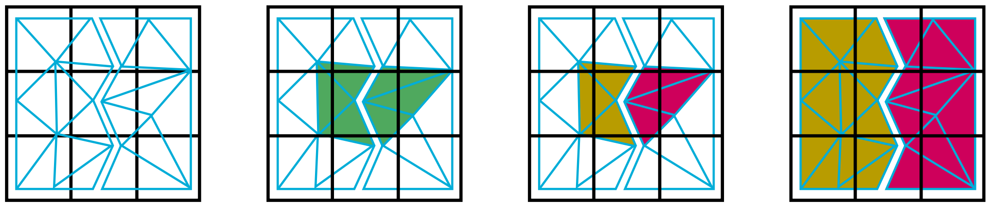

(a) (b) (c) (d)

Figure 5.4: We describe the building of the non-manifold grid for the center cell of the grid. (a) The mesh is embedded in a uniform grid. (b) First, we store the overlapping geometry in the cell. (c) Then we detect disconnected parts using a flood fill algorithm. (d) Finally the cell is duplicated. For each duplicate, we look for other duplicates that share geometry and establish its connectivity.

Algorithm 6 Non-manifold grid building

```
 1: procedure BUILD_NON_MANIFOLD_GRID(grid $G$, mesh $M$)
 2:     BUILD_GRID_GEOMETRY($G$,$M$)
 3:     DUPLICATE_GRID_CELL($G$)
 4:     BUILD_GRID_CONNECTIVITY($G$)
 5: end procedure
 6:
 7: procedure BUILD_GRID_GEOMETRY(grid $G$, mesh $M$)
 8:     for each cell $i$ of $G$ do
 9:         Store overlapping element of $M$
10:     end for
11: end procedure
12:
13: procedure DUPLICATE_GRID_CELL(grid $G$, mesh $M$)
14:     for each cell $i$ of $G$ do
15:         $C \leftarrow$ disconnected component of $M$ in $i$
16:         for each component j of $C$ do
17:             Duplicate the cell $i$
18:             Store $j$ in the duplicate
19:         end for
20:     end for
21: end procedure
22:
23: procedure BUILD_GRID_CONNECTIVITY(grid $G$)
24:     for each cell $i$ of $G$ do
25:         $N \leftarrow$ neighbor cells of $i$
26:         for each duplicate $j$ of $i$ do
27:             for each duplicate $k$ in $N$ do
28:                 if $j$ and $k$ shares geometry then
29:                     Create a link between $j$ and $k$
30:                 end if
31:             end for
32:         end for
33:     end for
34: end procedure
```

5.4 Frame re-sampling

As long as no parts of the model are disconnected, our method the use of a constant number of control frames. However, when parts are disconnected, we need to sample it with at least one frame in order to simulate it.

We start by detecting empty regions i.e lists of connected cells that are not influenced by any frame. This is done using a flood fill algorithm on the grid containing the shape functions values. These empty regions are then sampled using a farthest sampling algorithm. Finally, the samples are uniformly distributed by applying several Lloyd relaxation steps. For now, the number of frames which are sampled is user-defined but we would like to investigate for setting it automatically (see Section 5.7).

As a cut progresses, it may happen that only one frame influences a large region. Then this region can only express affine motion. Depending on the material properties, the size and the shape of the region, this can result in unconvincing behaviors. For rigid materials this is not a problem but for soft material this can quickly become unrealistic. We propose a simple strategy to solve some of these cases. For each frame, we look for regions where the shape function value is above a user-defined threshold w_{max}. Then if the volume of the region is above a maximal volume threshold v_{max}, we uniformly re-sample the region. This strategy allows the detection of large regions which are mostly influenced by only one frame and are the most likely to need re-sampling. For now, w_{max} and v_{max} are user-defined.

As regions of influence are very large, the popping artifacts induced by adding instantaneously one additional frame can be noticeable. In order to reduce them we propose a simple strategy. Once the position of the new frame in the undeformed, material space has been chosen, we use the previous deformation field to interpolate its new position, orientation and velocity.

5.5 Incremental update

The domain and the shape functions continuously change during cutting. Therefore, all the simulation data that are related to the domain or the shape functions need to be updated at each time step a cut occurs. Fortunately, cutting is often a local phenomenon. We exploit this locality to incrementally update only what is necessary and therefore save substantial computational time.

In our case, there are several simulation components that need to be updated. The first of this component contains the integration points that compute deformation gradients and transfers internal forces to the control frames. Then there is the collision component, a simple set of points, that transfers external forces to the control frames. Finally, there is the mesh that we visualize whose vertices positions are interpolated from the frame positions. Each of this component can have its own resolution. Their data are computed from the control frames using interpolation. This layer-based organization allows the separation of the resolutions of the physical simulation, the interactive model and the visual rendering to achieve a good trade-off between realism and performance.

In the following sections we describe the mechanisms we used to incrementally update the different components of the simulation.

5.5.1 Re-sampling

As for the frames, we always need to have at least one collision node and one integration point inside each part of the model. Otherwise, we cannot compute deformations or interact with these parts of the model. Usually, there are much more collision nodes and integration points than frames. Instead of adding new points only when we detect new empty regions, we perform a few Lloyd relaxation steps at each time step to always keep a uniform sampling of the domain. In a progressive cut scenario, only a small number of samples will need to be updated at each time step and will result in an efficient incremental update. However, if disconnected parts are created from a cut, we apply the re-sampling strategy discussed in Section 5.4. We detect the disconnected parts using a flood fill algorithm and uniformly re-sample them.

5.5.2 Integration point update

Integrations points are used to compute deformation gradients and transfer internal forces to the frames. To do so, each integration point are interpreted as a small volume of the domain and carries a position, a region's volume and the volume moments. As soon as a cut occurs, the region's volume of integration points close to the cut will change and it becomes necessary to update these integration points. This can be easily done by storing an explicit description of the region of the integration point i.e a list of cells. If the cut goes through one of these cells then we update the integration point data.

5.5.3 Local weights update

Weights and derivatives are interpolated from the grid to positions of the different samples: collision nodes, integration points and mesh vertices. At each cut, we need to update these values. In an interactive context, we cannot afford to perform interpolation for all these samples. Once again, we leverage the fact that a cut is very often a local event, sometimes progressive, and will impact only a small fraction of the different samples. Our idea is to perform incremental update of weights and derivatives by detecting the low number of samples that were impacted by the cut. At each time step, if a cut was performed, we compare the new shape functions with the previous ones and detect the grid cells which have been impacted by the cut. All the samples that are contained or are neighbors of these cells need to be updated. In the end, even if we have control frames that covers large regions of the domain compared to classical simulations, simulation data that need to be updated remains spatially local.

5.6 Results

We illustrate our method in a variety of simulations where a piece of paper undergoes progressive scripted cuts. As we use several layers of samples (frames, collision nodes, integration points), choosing a good trade off between accuracy and performance is essential. In all the examples, we used the minimum number of samples we could without compromising visual results. Frame re-sampling was required to simulate disconnected parts. However it appears that no additional collision nodes or integration points were required. We can deduce that our relaxation strategy is sufficient to keep the object uniformly sampled along the simulation.

Figure 5.5a shows a long spiral cut in a sheet of paper simulated with only 5 frames. The shape functions of the frames faithfully represent the cut as shown in Figure 5.1. To illustrate that our method can handle multiple cuts and still simulate complex deformations, the creation of a Kirigami is shown in Figure 5.5b. 48 cuts are performed and it only required 47 frames and 400 integration points to produce a plausible behavior.

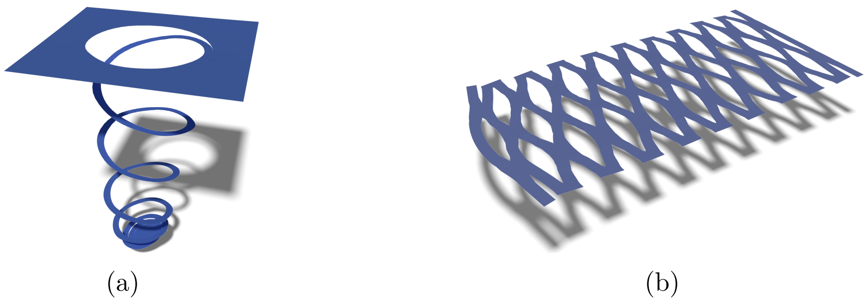


Figure 5.5: Progressive cutting of a spiral using only five control frames (a). Simulating complex deformations resulting from Kirigami cutting (b). Note that an horizontal stretching force results into a twist of the bands of material, as in real experiments using paper.

Detailed cuts can be performed and separated components can be handled as shown in Figure 5.6a. In a cloth sheet, we progressively cut bunny, teapot, dragon and armadillo shapes. Each time a new object is completely cut, it is automatically re-sampled with additional frames.

As we explained, the non-manifold grid can represent an arbitrary number of connectivity in one cell. This is particularly useful in order to represent intersecting cuts as shown in Figure 5.6c.

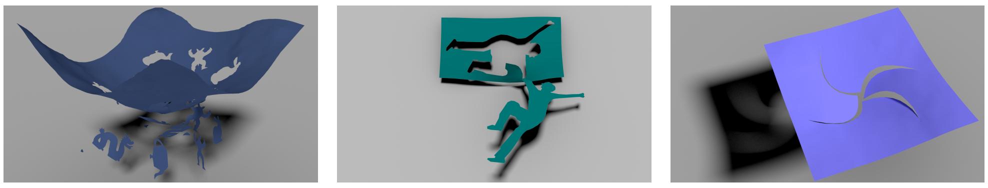

(a) (b) (c)

Figure 5.6: (a) Several highly detailed shapes are cut in a deformable sheet. Each disconnected part is automatically re-sampled with additional control frames. (b) Simulation of a highly detailed cut that falls under gravity and remains attached to the main part by a thin piece of paper. (c) Two cuts intersect to form a vortex shape. This illustrates the abilities of the non-manifold grid to handle multiple intersecting cuts.

All our examples run at interactive frame rate during the whole simulation (see Table 5.1). Frame rates were collected on a twelve-core 3.20 GHz Intel Xeon CPU with 15.6 GB RAM.

		#frame		#vertices				Lowest FPS		
Name	Grid Size	Initial	Final	Initial	Final	#collision	#integration	Before Cutting	During Cutting	After Cutting
Spiral (Fig. 5.5a)	40 × 40	5	5	81	2111	200	200	60	14.4	60
Kirigami (Fig. 5.5b)	68 × 68	47	47	4225	7453	600	800	11.3	3.2	10.9
Patchwork (Fig. 5.6a)	50 × 50	5	12	4225	8253	200	200	60	6.8	45
Vortex (Fig. 5.6c)	68 × 68	5	5	4225	4889	200	200	45	7.2	35.7
FallingGuy (Fig. 5.6b)	100 × 50	10	10	289	861	500	500	60	8.2	60

Table 5.1: Resolution of the different components of the simulation and timings.

We noticed that even if a cut only concern a few grid cells, the number of data to re-compute is much more important. This comes from the fact that each frame can cover a large region and changes arising from a local cut can be important. Fortunately, our incremental update mechanisms allows the saving of numerous unnecessary computations as shown in Table 5.2.

5.7 Discussion and concluding remarks

In this chapter, we presented a novel method to simulate highly detailed cuts with a sparse set of control nodes which allows interactive frame rates. This approach can be seen as a reduced simulation that handles topological changes without requiring expensive precomputations. Of course, our work is not without limitations and brings interesting directions for future work.

	Percentage of update for a cutting step				
Name	%grid cell	%shape function cell	%vertices	%collision nodes	%integration points
Spiral (Fig. 5.1a)	0.07	28.1	61.5	27.2	41.3
Kirigami (Fig. 5.5b)	1.06	15.9	17.8	15.8	20.3
Patchwork (Fig. 5.6a)	0.02	2.78	4.33	2.55	7.15
Vortex (Fig. 5.6c)	0.08	10.3	12.8	9.2	24.2
FallingGuy (Fig. 5.6b)	0.09	5.84	11.4	5.77	14.9

Table 5.2: Percentage of updated data in a cutting time step. We averaged the percentage for the whole cutting time. We notice that even if very few grid cell are affected, it implies important changes on the shape functions and the samples that are associated to these values.

Firstly, as very few frames are used, one cut may generate large changes in the weight distribution and produce popping artifacts that cannot be avoided using our interpolation strategy. This is particularly noticeable when simulating soft materials and can be seen in some of our *Patchwork* example. Strategies proposed by [NPO13] and [Tou+14] in the context of adaptive simulations could be used to limit this problem.

Secondly, for large deformations, the surface can look bumpy. There are several reasons for this problem. Linear blend skinning, used to approximate the displacement field, produces well known artifacts that could be solved using a better skinning approach such as dual quaternion skinning. Also, the shape functions derivatives are discontinuous and this is particularly noticeable during high deformations. One could easily change the shape functions and still use the non-manifold grid to depict the topology.

Thirdly, our implementation is far from being optimal. Currently the non-manifold grid and the shape functions are re-computed from scratch at each cut. We could enjoy a dynamic acceleration structure to incrementally update our non-manifold grid. Shape functions could also be incrementally updated. Finally, there are several parts of our method that could enjoy parallelization such as samples interpolation.

Finally, we would like to extend our work to 3D. The implementation of our current non-manifold grid would require a tetrahedron representation of the object. We would like to investigate the method of [RK13] to build this structure only from the object surface. We think that the frame-based framework can be used to produce interactive detailed fracture simulation. The main challenge is to accurately compute stress tensors which are then used to determine fracture direction. Instead of using a dense sampling of frames and integration points to compute the stress tensors, we would like to combine a low resolution stress tensor measurement with procedural detail generation as in the work of [Che+14] and [Lej+15]. In a similar direction, we would like to investigate advanced sampling strategies in order to automatically determine how many frames are required for a given region. This would involve the material property, the size and the shape of the region that needs to be sampled.

In this chapter and the previous one, we explored different simulation techniques which enable efficient computations and topological changes. When experimenting these two methods, a substantial amount of time was dedicated to adjust the parameters of the simulation: time step, damping, stiffness, initial and boundary conditions, in order to achieve the results we had in mind. As we explained in Section 2.2, controlling a physics-based simulation is a hard problem, especially when it exhibits complex topological changes, such as in liquid simulations. In the next chapter, we address this problem by proposing a novel method to sculpt animations of liquid using intuitive tools inspired from shape modeling.

Chapter 6

Sculpting of Liquid Animations

DUE to advances in fluid simulation methods over the last two decades, animations of liquid has become commonplace in 3D animation productions. The animations can be either highly realistic — for example showing plausible fluid dynamics and interactions with obstacles — or they can exhibit a more expressive behavior to convey specific artistic intentions. In both cases, it is essential for the artist to be able to control the simulation in order to achieve their goals.

As mentionned in Section 2.2, the simulation control is generally achieved through the careful setting of a large number of parameters such as initial conditions, boundary conditions, viscosity, or external forces. We briefly summarize why the tuning of these parameters is particularly difficult. First, they only offer indirect control over the animation, which makes them quite non-intuitive. Second, it is usually not possible to have interactive visual feedback when modifying the parameters, due to the high computational cost of liquid simulation. Third, the inherently non-linear nature of fluid behavior makes it difficult to transfer parameter values from a low to a high resolution simulation. In consequence, achieving a desired effect requires a tedious trial-and-error loop, where computation is restarted multiple times from scratch with different parameters. In many cases, this process does not allow tight control over a sequence of waves and splashes with specific magnitudes or shapes and occurring in a specific order.

In this chapter, we attempt a significanly different approach. Instead of controlling a simulation, we propose an interactive sculpting system for seamlessly editing pre-computed animations of liquid, without the need for any re-simulation. Our system is based on a copy/edit/paste approach: The user can efficiently select consistent and visually important space-time parts of an animation, such as moving waves or droplets, that we call *space-time features*; Once selected, these space-time features can be copied and edited in both space and time in order to change their size, orientation, trajectory or speed; Finally, the edited space-time features can be pasted back into any destination animation at a specific position and time set by the user.

Using our tools, the user can edit and progressively refine any input simulation result, possibly using a library of pre-computed space-time features extracted from other animations. In contrast to the trial-and-error loop usually required to edit animation results through the tuning of indirect simulation parameters, our method gives the user full control over the edited space-time behaviors.

To enable the use of arbitrary animations of liquid computed using varying simulation techniques, we based our editing framework on generic inputs; our method allows input mesh sequences without point-wise correspondences between frames, and with arbitrary changes of topological genus between two consecutive time steps. Also, we focused on three requirements to make our method useful in realistic cases. First, the selection of the effect in the original simulation must be as simple and straightforward for the user as possible. Therefore, once space-time features have been computed, the user can select them using a simple click on the surface. Secondly, pasting the selected effect onto the final animation should be handled automatically, with seamless adaptation of the pasted fluid effect to the destination surface. Finally, the pipeline of selection, copy, edit and paste steps should be computed efficiently in order to enable interactive user feedback.

The key contributions of our work are as follows:

- A semi-automatic method to tag salient regions in an animation of liquid.
- An algorithm that extracts coherent *space-time features* from a mesh sequence with tagged vertices.
- A *space-time feature* representation independent from the original animation.
- A set of editing operations that allow the extraction, manipulation, and insertion of *space-time features* into an animation.

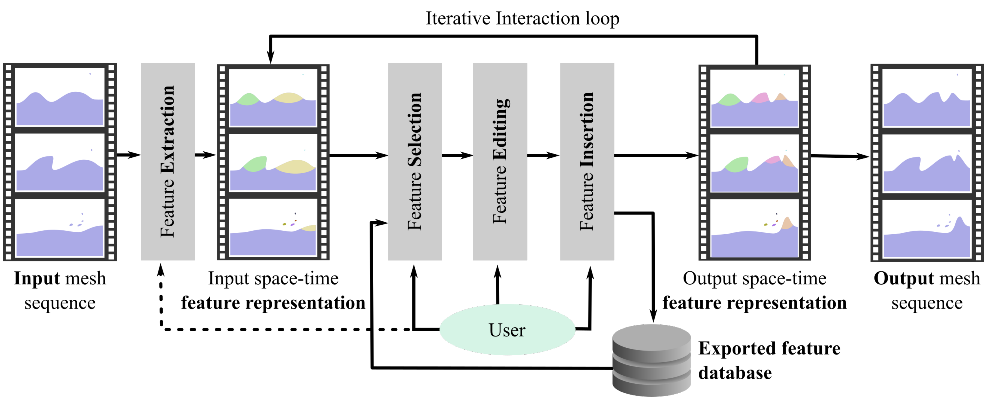
The remainder of this chapter is organized as follows: Section 6.1 presents our solution to this problem; Section 6.2 explains how *space-time features* are computed; Section 6.3 deals with the *space-time features* representation; Section 6.4 details the tools we offer for manipulating *space-time features*; Section 6.5 shows results obtained with our method; Section 6.6 draws the limits of our approach and gives some perspectives on future work.

The works described in this chapter have been submitted to the conference *MIG 2016*. We describe our method and results in the video here: https://www.dropbox.com/s/0cob2nuztdimjol/fluidSculpting_MIG2016.mp4?dl=0.

6.1 Overview of the method

As mentioned above, this chapter focuses on editing animations of liquid. To be independent from the simulation method, we take as input a sequence of meshes without any correspondences between the mesh vertices from one frame to another. Due to the arbitrary topology of the meshes and to the temporal coherence to be maintained for numerous geometric details, editing each frame with a shape modeling tool would represent a tremendous amount of work. Instead, we propose to manipulate a higher level representation of the animation of liquid that we call *space-time features.* A space time feature is a sub-part of the animation, i.e. a sequence of sub-parts of the liquid surface.

Our editing pipeline generalizes standard sculpting tools [FCG00] based on cut/copy/edit/paste operations. It is made of three steps which are illustrated in Figure 6.1.



Iterative Interaction loop
Feature Extraction
Feature Selection
Feature Editing
Feature Insertion
Input mesh sequence
Input space-time feature representation
Output space-time feature representation
Output mesh sequence
User
Exported feature database

Figure 6.1: Pipeline of our method: An input fluid animation is given as a mesh sequence. It is pre-processed into a higher-level space-time feature representation. This representation allows the user to iteratively select features from the animation and edit them before inserting them back to the animation. Alternatively, features can be saved and re-imported in this animation or a different one.

The first step extracts space-time features from the animation. As these features represent regions that deform over time, it would be too tedious for a user to define them by hand. We propose a semi-automatic method to detect salient regions in an animation of liquid from which space-time features will be automatically computed. The user can then easily select them using picking: a click at a specific location at a given frame in time results in the automatic selection of the associated feature with its full range in space and time. The second step computes representations of the selected space-time features that are independent from the input animation. They enable space-time features to be transferred from one animation to another. Finally, the last step consists of editing the space-time features and pasting them back into an animation.

6.2 Feature extraction

In the feature extraction step, our method defines the space-time features that the user would like to manipulate. This process is divided into three steps, as described in Figure 6.2: detection, segmentation and aggregation. While detection is semi-automatic (it is interleaved with user interaction to define customized regions of interest throughout the animation), segmentation and aggregation are fully automatic.

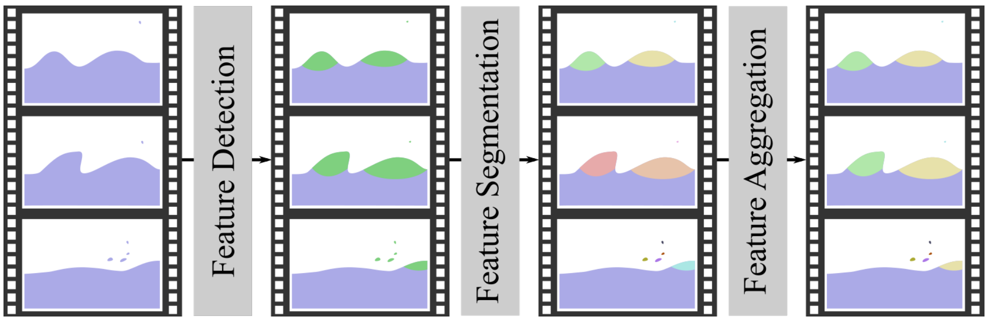


Figure 6.2: Feature extraction process, from left to right: an initial mesh sequence representing a fluid animation is subjected to a feature detection process, followed by a segmentation step, which results in a frame feature representation. A final aggregation step allows the building of a temporally coherent feature structure.

Notation The input of our method is a mesh sequence over the time steps t that we note $M = (M^t)$, where M^t is a manifold triangular mesh. We note $T(.)$ the temporal length (i.e. the number of frames) and $L(.)$ the characteristic spatial length of any space-time sequence (mesh sequence or feature). Given a triangular mesh X, we call N_X the set of its vertices and P_X the set of its faces. A vertex can carry attributes. We note $A(n, X)$ the value of the attribute A at the vertex n of the mesh X. In the following, we will note $pos(n, X)$, $norm(n, X)$, and $curv(n, X)$ for positions, normal and curvature respectively. $\Delta_A(n, X)$ designates the Laplace-Beltrami operator applied to the attribute A at vertex n of the mesh X.

6.2.1 Detection

The detection phase aims at defining a sequence of regions of interest $R = (R^t)$ on M. A region R^t is represented as a set of vertices of M^t; we call this structure a *mesh part*.

To let the user easily and intuitively define R, we propose a semi-automatic tool. This tool is based on two key components that we describe in detail below: curvature analysis and topological filtering. Combined together they let the user define R in a coarse-to-fine manner: First, curvature analysis is used to automatically detect salient features at each frame and initialize R. Then, topological filtering allows one to interactively adjust R. We also added a painting tool that allows the user to fine-tune each R^t if needed by locally removing or adding vertices from R by clicking.

Multi-resolution curvature analysis. We chose a curvature criteria to extract features as it is a natural asset for detecting waves and ripples in animations of liquid. Moreover, the intimate relationship between surface curvature and liquid surface dynamics had already led previous work to use curvature as a tool to enrich liquid simulations, for example with splashes [Tak+03], foam [Ihm+12] and textures [Nar+07].

Curvature is computed at each vertex n of the animation meshes M^t using the following formula,

$$curv(n, M^t) = norm(n, M^t) \cdot \Delta_{pos}(n, M^t). \tag{6.1}$$

Vertices are colored with respect to their curvature magnitude, enabling the user to interactively observe the curved regions and their deformations on the fluid surface while playing the animation (see Figure 6.3a). Then we provide two sliders that the user can interactively tune to filter the curvature and select meaningful regions. These sliders represent:

- A number of iterations β of Laplacian diffusion on the curvature values. We define the i-th iteration of the Laplacian curvature diffusion as

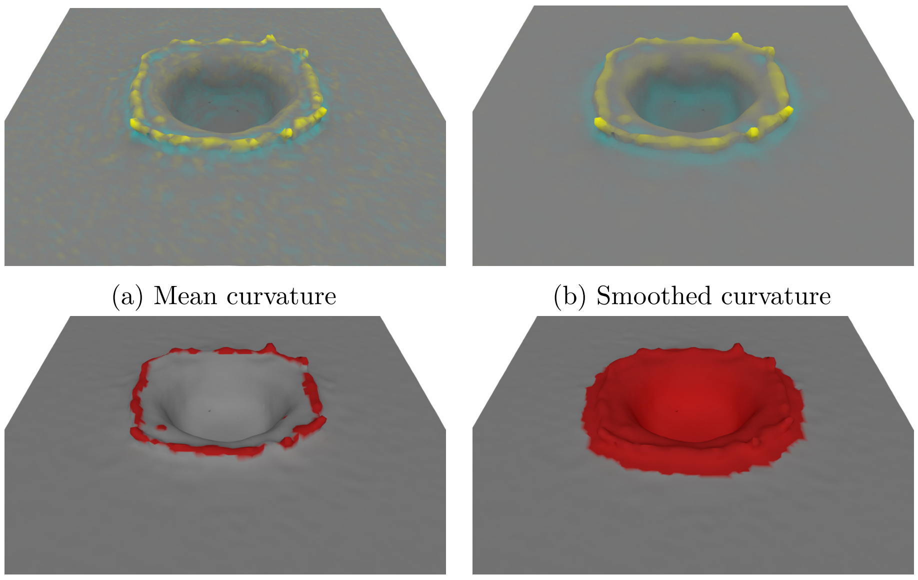
  $$curv^{i+1}(n, M^t) = curv^i(n, M^t) - \lambda.\Delta_{curv^i}(n, M^t) \tag{6.2}$$

  with $curv^0(n, M^t) = curv(n, M^t)$ and $i \in [0, \beta]$. In our experiment, we used $\lambda = 1$ as a diffusion factor. Laplacian diffusion of the computed curvature values is used to decrease the spatial frequency of the curvature function over the surface. This allows the user to select broader regions in an efficient way without actually smoothing the geometric details on the mesh (see Figure 6.3b).

- A threshold γ on the curvature of R. All the vertices whose curvature is above γ are added to R. This allows the user to control the extent of R (see Figure 6.3c).

  In the end, we can mathematically define a region of interest for a frame t as

  $$R^t = \{n \in N_{M^t} | curv^\beta(n, M^t) > \gamma\}. \tag{6.3}$$



(a) Mean curvature (b) Smoothed curvature

(c) Curvature thresholding (d) Topological closure

Figure 6.3: Curvature analysis-based feature detection.

Topological filtering. In many cases, curvature-based selection is not sufficient to extract meaningful animation features. For instance, in Figure 6.3c, the user might want to select the whole crown splash and not only its contour as it has been done with the curvature analysis tool. To remedy these issues, we extend mathematical morphological operators (MMOs) to polygonal meshes. They allow the user to interactively and easily refine the regions of interest detected by the curvature analysis. We propose two main tools:

- *Erosion* for disconnecting, reducing or removing parts of R.
- *Dilatation* for connecting and enlarging parts of R.

Both tools can be combined for performing *openings* and *closures* of R. In practice, these tools were particularly useful for selecting regions such as the interior part of the circular wave in Figure 6.3d, achieved with a closure. For a detail overview of MMOs, we refer the reader to the work of Serra [Ser86].

6.2.2 Segmentation

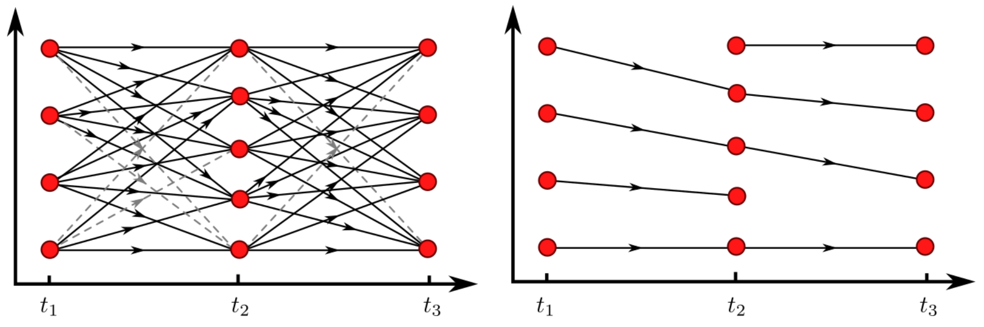
Once R has been computed, the segmentation step decomposes each R^t into connected components $(C_k^t)_{k,t}$, where k is the index of the component. Mathematically, a region of interest for a frame t can be defined as the disjoint union of its connected components,

$$R^t = \bigsqcup_k C_k^t \quad | \quad \forall t \in [0, T(M) - 1]. \tag{6.4}$$

The decomposition is computed using the straightforward breadth-first search on each frame in parallel. We call each C_k^t a *frame feature*.

6.2.3 Aggregation

Finally, the aggregation step extracts temporally coherent sequences of *frame features* that we call *space-time features*. The process is divided into two steps as illustrated in Figure 6.4. First, we build a graph of all possible *frame feature* connections, and then we compute a vertex-disjoint path cover of that graph. Temporal coherency of the resulting paths is enforced by minimizing a geometric matching cost described below. We call the resulting paths *space-time features*.



(a) Initial DAG (b) Vertex-disjoint path cover

Figure 6.4: Left: Frame features (red dots) are assembled into a directed acyclic graph as described in Section 6.2.3. Each edge of the graph carries a cost computed with Equation (6.5). Edges whose cost is over a user-defined threshold (gray dashed lines) are discarded. Right: a vertex-disjoint minimum-cost path cover has been computed based on Algorithm (7). The extracted paths represent space-time features.

Graph construction. We build a directed acyclic graph $G = (V_G, E_G)$ representing the possible connections between frame features (see figure 6.4a). The set of nodes V_G is made of the frame features $(C_k^t)_{k,t}$ while the set of edges E_G is made of oriented edges e_{ij} linking each pair of consecutive frame features C_i^t and C_j^{t+1}.

Edge cost computation For every edge $e_{ij} \in E_G$, we compute a cost measure ω_{ij}. This measure relates to the geometrical matching between its two endpoints v_i and v_j. We divided ω_{ij} into three terms:

- d_{ij}: The distance between the centers of mass of v_i and v_j.
- s_{ij}: The difference of the surface area between v_i and v_j.

- v_{ij}: The difference of volume between v_i and v_j. v_{ij} is computed only if both v_i and v_j are closed.

The edge cost ω_{ij} is a weighted sum of these terms, normalized by the appropriate power of $l = L(M)$, the characteristic size of the bounding box of M,

$$\omega_{ij} = \omega_d \left(\frac{d_{ij}}{l}\right)^2 + \omega_s \left(\frac{s_{ij}}{l^2}\right)^2 + \omega_v \left(\frac{v_{ij}}{l^3}\right)^2 . \tag{6.5}$$

For all the examples of this chapter we used $(\omega_d, \omega_s, \omega_v) = (0.6, 0.2, 0.2)$. We chose to favor the closeness between frame features and consider difference of surface and volume equally. After the cost computation, we discard edges whose cost is below a threshold ϵ that we set to $0.3 \times l$ in our examples. Higher thresholds lead to fewer edges in the graph and more disconnected paths.

Vertex-disjoint path cover computation. To the authors' knowledge, there is no standard algorithm for computing minimum weight vertex-disjoint path cover. We propose an algorithm based on Kruskal's algorithm for computing minimum spanning trees [Kru56]: All vertices are first copied from the input graph to the output one; edges of the input graph are considered in ascending order of cost and added to the output graph if they satisfy a given topological condition. In Kruskal's algorithm, the condition is that the edge does not form a cycle in the output graph. In ours, the condition is that both of its endpoint vertices have strictly fewer than two neighbors. This allows us to ensure that the resulting path cover will be vertex-disjoint.

We detail our vertex-disjoint path cover process in Algorithm 7 using the following notation:

- G, V and E represents respectively a graph, a set of vertices and a set of edges;
- *in* and *out* subscripts refer to input and output elements;
- v_0^e and v_1^e refer to the endpoints of edge e in both G_{in} and G_{out} (since $V_{in} = V_{out}$);
- $deg(v)$ is the degree of vertex v in G_{out};
- $sort(E)$ is the in-place sort of the edges of E in the ascending cost order.

Algorithm 7 Vertex-disjoint path cover computation

```
$G_{in} = (V_{in}, E_{in})$
$G_{out} = (V_{out}, E_{out})$
$V_{out} = V_{in}$
$E_{out} = \varnothing$
$sort(E_{in})$
for all $e \in E_{in}$ do
    if $deg(v_0^e) < 2$ and $deg(v_1^e) < 2$ then
        $E_{out} \leftarrow e$
    end if
end for
```

At the end of this algorithm, the graph G_{out} consists of all frame-features V_{out} connected by inter-frame links E_{out}. E_{out} represents independent paths, as illustrated in Figure 6.4b, which are optimal in the sense that the algorithm greedily minimizes our edge cost metric. These paths describe the *space-time features.*

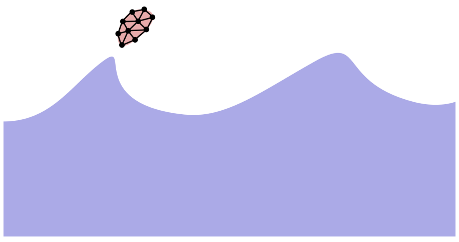
6.3 Feature representation

Space-time features can be seen as a simple set of vertices belonging to M. This representation is, however, inconvenient for direct manipulation as it strongly depends on the input animation and therefore cannot be transferred from one animation to another. To be able to copy, edit and paste space-time features in different animations, we propose to build a representation of a space-time feature which is independent from M.

In the remainder of this chapter, we will note a space-time feature representation $F_i = (F_i^t)_{t^s(F_i) \leq t \leq t^e(F_i)}$ where $t^{s/e}(F_i)$ are the starting/ending frame index of F_i and F_i^t is the frame feature representation of F_i at the frame t. Also, we denote by $S(F_i^t)$ the mesh part of M^t corresponding to F_i^t.

We distinguish two representations depending on whether the frame feature has boundaries or not (see Figure 6.5).

In the first case, we use a *mesh representation*, noted $M(F_i^t)$ and composed of a simple 3D mesh. It is used to represent a connected component of the liquid, such as droplet or a larger body of water. In the second case, we use a *differential representation*, noted $(\tau_d(F_i^t), \tau_n(F_i^t))$, and composed of a pair of textures representing a displacement map and a normal map. It is used for frame features representing a local sub-part of a larger body of water, such as a single wave on the surface of an ocean. A space-time feature can be composed of frame features from both categories. A typical case of mixed representation is an isolated drop falling into a larger body of water and becoming a detail of this larger surface.


(a) Mesh representation

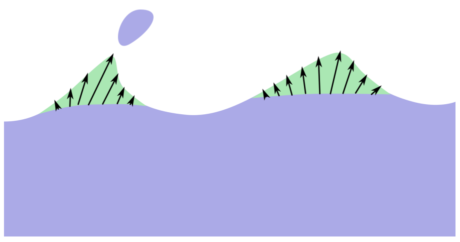
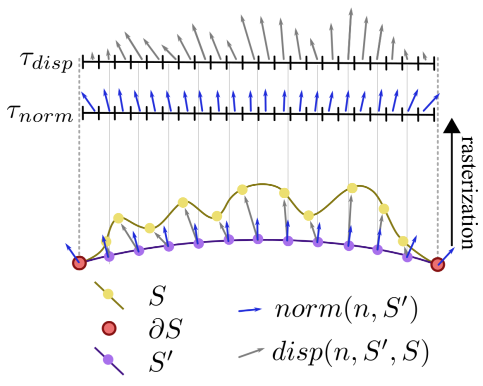
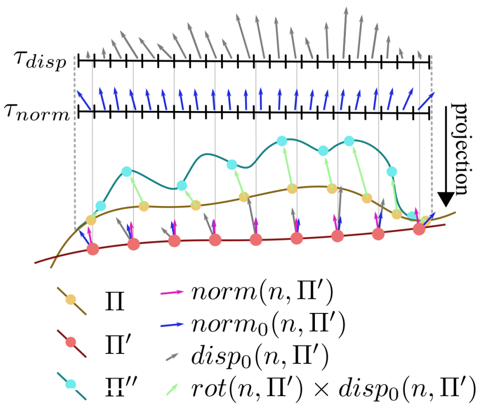

(b) Differential representation

Figure 6.5: Depending on whether the frame feature has closed boundaries or not, it is stored either as a mesh (left) or as a displacement field (right).

In the following of this section, we detail the computation of both representations and how they can be inserted back into a different animation. This will be useful later for copying and pasting features.



(a) Copying



(b) Pasting

Figure 6.6: Left: The displacement representation of a mesh part S is built from the sampling of the displacement field transporting S' toward S, and the normal field of S'. Right: This representation can be inserted back into a mesh part Π by projecting a displacement and a normal on vertices of Π'. The difference of normals between S' and Π' is used for orienting the displacements, which are in turn used for generating the deformed surface Π''.

6.3.1 Computation

Building the mesh representation of a frame feature F_i^t simply consists of transforming $S(F_i^t)$ into an independent mesh $M(F_i^t)$.

Building the differential representation of a frame feature is slightly more complex. The process is described in Figure 6.6a and consists of three steps: Starting from the initial frame feature surface $S(F_i^t)$ we compute a smooth version $S'(F_i^t)$ using Laplacian smoothing on the inner part of the surface $S(F_i^t) \setminus \partial S(F_i^t)$. We note $pos(n, S)$ the position of vertex n on surface S; note that $S(F_i^t)$ and $S'(F_i^t)$ describes the same vertices, but with different positions. Then we compute the displacement of each vertex $n \in N$ from $S'(F_i^t)$ to $S(F_i^t)$,

$$disp(n, S'(F_i^t), S(F_i^t)) = pos(n, S(F_i^t)) - pos(n, S'(F_i^t)). \tag{6.6}$$

Finally, we map for every vertex n, $disp(n, S'(F_i^t), S(F_i^t))$ onto $S'(F_i^t)$, and sample the linearly interpolated values into the texture $\tau_d(F_i^t)$. We similarly sample the normals of $S'(F_i^t)$ into $\tau_n(F_i^t)$. The samplings are performed on the GPU using the standard off-screen rasterization pipeline.

6.3.2 Insertion

Mesh representations are trivially inserted by copying $M(F_i^{t'})$ into M^t. To insert a feature representation $(\tau_d(F_i^{t'}), \tau_n(F_i^{t'}))$ into M^t at location p, we first need to identify the part $\Pi \subset M^t$ which will be deformed. Starting from $N_\Pi = \{n_0\}$ where

$$n_0 = \operatorname*{argmin}_{n \in N_{M^t}}(\|pos(n, M^t) - p\|), \tag{6.7}$$

we progressively dilate Π until it fills the bounding box of size $L(F_i^{t'})$ centered in p. Once Π has been computed, we compute its smooth version Π' on which we project $\tau_d(F_i^{t'})$ and $\tau_n(F_i^{t'})$, yielding two attributes for each vertex $n \in N_{\Pi'}$, a displacement $disp_0(n, \Pi')$ and a normal $norm_0(n, \Pi')$. We define a new attribute $rot(n, \Pi') = rot(norm_0(n, \Pi'), norm(n, \Pi'))$, a rotation matrix mapping $norm_0(n, \Pi')$ into $norm(n, \Pi')$. Each vertex $n \in N_\Pi$ is displaced of $rot(n, \Pi') \times disp_0(n, \Pi')$, yielding the deformed surface Π''. These operations allow one to counter the effects of low-resolution shapes of both $S(F_i^t)$ and Π. Figure 6.6b illustrates these steps.

6.4 Sculpting Tools

Once space-time features representations have been computed, they can either be manipulated by the user to modify the current animation, or they can be extracted and re-used in another animation to enrich it. This section described the set of tools we propose; they are essentially the space-time analogue of common tools used for sculpting static geometry [FCG00; SS10; Tak+11].

Selection The first thing one might want from an interaction system is to specify which of the multiple entities of the scene are to be interacted with. This is usually performed through object selection. In our case, objects are space-time features and they can be selected and grouped by clicking on their shape at a given frame.

Copy and cut The copy operation consists of creating the representation of the selected features, as explained in Section 6.3.1. The cut operation is similar to the copy operation, except that the representation of the feature is removed from the animation after it has been computed. Once a feature or a feature group has been copied or cut, its representation becomes the current input data of further tools. It is later designated as "the current feature."

Export and import The current feature can be exported into a dedicated binary file format which stores its representation at each frame. This allows it to be imported back later to the same animation, or into a different one. Once imported, a feature becomes the current feature.

Paste The pasting operation allows a user to insert the current feature into a target animation, as explained in Section 6.3.2.

Space-time Deformation The user might want to use the feature in a different spatial and temporal configuration from the one in which it was extracted, so we propose adapted deformation tools. The position, orientation and spatial scale of the current feature can be controlled with the mouse, and a real-time visual feed-back allows the user to set the feature in the configuration they require. By navigating in the animation, the user can also choose the initial frame of the current feature and set a time scale. This leads to a speed-up or slowdown of the feature animation.

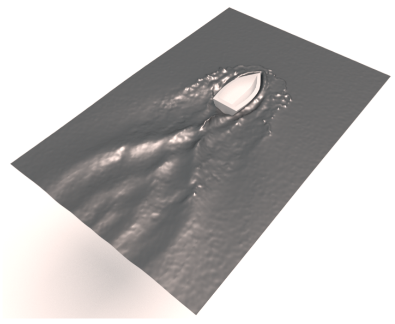
Fade in and out When pasting a wave, the user can specify a fade in and a fade out interval. This means that that the feature will not immediately appear, but instead it smoothly grows in the beginning of its lifetime and smoothly disappears before the last frame of its lifetime. We achieve this effect by linearly blending the pasted displacement field over time with weights varying between 0 and 1.

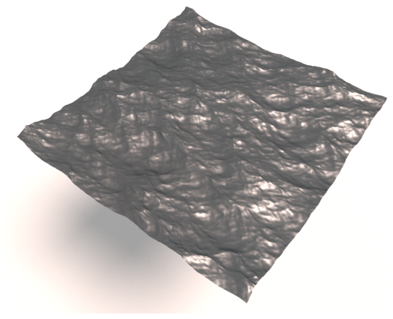
Trajectory editing Space-time deformations influence all frames at once, whereas the user might want to control each frame individually. Per-frame spatial feature manipulation is achieved through a dedicated feature trajectory edit tool. This tool allows a user to displace the representation of a feature at a given frame while visualizing the positions of the feature at all the frames.

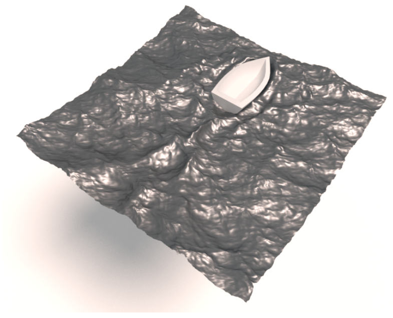
6.5 Results

In this section, we detail results achieved using our sculpting system. They illustrate the different tools described in the Section 6.4 and how we used them.

Boat wake In Figure 6.7, we illustrate the capability of our method to extract *space-time features* from arbitrary inputs (e.g. Eulerian or Lagrangian simulation, spectral methods, shallow water, real liquid surface acquisition) and combine them to create a plausible animation. We start from two animations: The first one (Figure 6.7a) was computed using the FLIP simulation method [ZB05] and represents a boat traversing a fluid tank and forming a wake. The second one (Figure 6.7b) is a procedural animation of ocean computed using the method of [Tes04b] and exhibits numerous small scale details. Then we extract the boat wake and paste it on the ocean animation at three different positions with different scales and orientations (Figures 6.7c and 6.7d).


(a) Boat simulation


(b) Procedural ocean


(c) Pasting of the wake

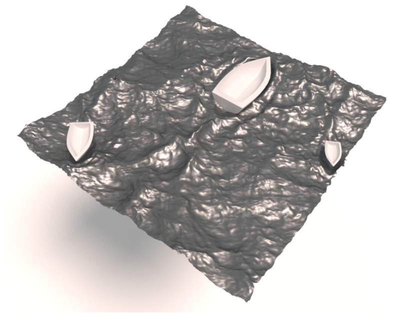
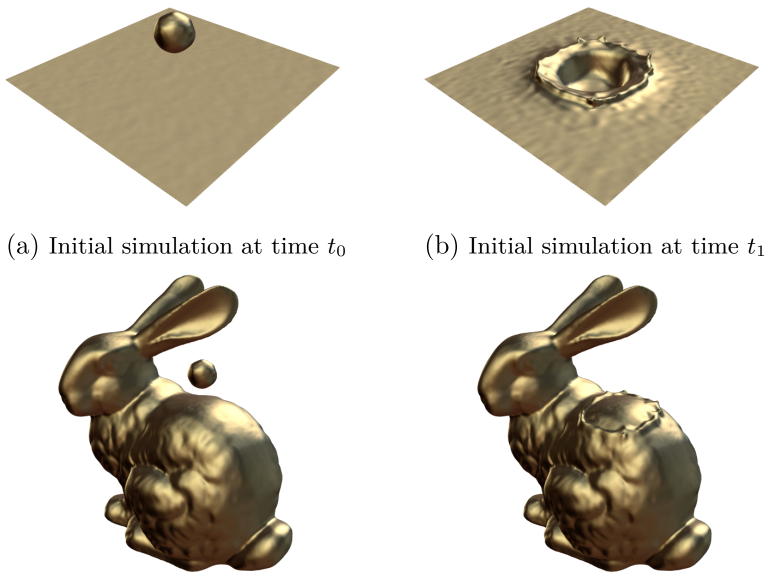

(d) Multiple pasting

Figure 6.7: From the animation of boat generated using a FLIP simulation (left), our sculpting system allows the extraction of the wake of the boat in a single *space-time feature*. Then we can manipulate this feature and paste it into an ocean animation generated procedurally (middle). Editing the feature and pasting it multiple times allows the interactively modeling a complex scene (right) without re-simulating.

Animation enrichment An interesting aspect of our method is that it can be used to enrich static objects or non-fluid objects with a fluid-like behavior. In Figure 6.8, we enriched a static object with a splash extracted from an animation of liquid. More generally, our method allows the combination of results obtained with very different methods such as procedural animation, Eulerian and Lagrangian simulations, shallow water simulation, or artist-created animations.

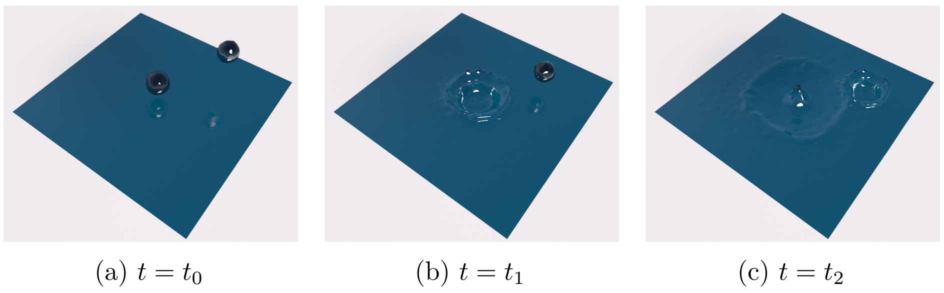


(a) Initial simulation at time t_0 (b) Initial simulation at time t_1

(c) Final animation at time t_0 (d) Final animation at time t_1

Figure 6.8: From an initial simulation of a falling drop (a, b) space-time features can be extracted and pasted back into an other mesh, generating a new animation (c, d).

Trajectory editing In Figure 6.9, we applied several edits to a *space-time feature* capturing a crown splash. First, we temporally remapped the feature to slow it down. Second, we pasted it twice on a static plane at different locations. Third, we edited the trajectory of the droplets to change the height of their fall. Finally, we used a fading out to obtain a smooth transition with the initial plane.



(a) $t = t_0$ (b) $t = t_1$ (c) $t = t_2$

Figure 6.9: From an existing animation of liquid we extracted a complete crown splash into a single *space-time feature*. The feature combines both the fall of a drop and the resulting splash. We edit and paste this feature twice at different locations and modify the height of the droplets. Here, we show different frames of the final animation.

6.6 Discussion and concluding remarks

In this chapter, we introduced a new method for interactively editing existing liquid animations. We based our approach on an intuitive sculpting metaphor where the user can select, copy, edit and paste coherent *space-time features*. This approach allows a user to quickly design new animations. Our method is not without limitations and we suggest several directions for future work.

Physical consistency Even though the space-time features selected by the user capture realistic behavior, the way they are edited and inserted may spoil the realism of the resulting animation. As we do not check for physical consistency, the plausibility of the result depends on the user's artistic skill. An extension of our method would be to adapt the destination surface so that it matches the input features under physical constraints such as volume preservation. To incorporate further physical constraints such as momentum conservation, using mesh sequences as input would not be sufficient anymore and additional information such as velocity would be required. Designing an interactive editing method given these constraints may be difficult to achieve.

Resolution issues Geometrical details may be lost when pasting a feature if the resolution of the target mesh sequence is too coarse. To remedy this limitation, we could add an automatic mesh refinement scheme such that the resolution of the target mesh always locally matches the resolution of details in the pasted feature.

Aggregation robustness The aggregation of regions of interest into space-time features is a key component of our approach. However, as it is based on geometrical similarities between two consecutive frames, it might fail if the time step between two frames is too large or if parts of the water body are moving too fast, such as in the case of dynamic splashes with lots of fast moving droplets. Even if it has not been an issue for our results, we would like to enforce the robustness of the aggregation step by adding a new metric which would measure the physical coherency between two regions of interest. This metric would take into account some inferred velocity for the region. It could also incorporate some cause and effect relationships; for example, a falling drop will cause waves.

Memory consumption For our results, we worked with short sequence of animations of liquid but when editing a large sequence of high resolution meshes and extracting potentially large space-time features, memory consumption may become a problem. A classical solution would be to use a multi-resolution approach. The user would manipulate a low resolution version of the animation which would ensure interactivity. Then, the user's edits choices would be transferred to the high resolution version of the animation as an off-line post-process.

Feature editing We proposed basic tools for the space-time edition of features and there are several avenues for future work. Firstly, our copy/paste method is only able to deal with simple deformations of a surface. By using the work of [Tak+03] to extract and insert displacement fields, we could handle much more complex cases. Secondly, we would like to propose a space-time sculpting tool close to space deformers such as constant volume tools [Ang+06; FTS06] and topology modifiers [SCC11]. The idea would be to let the user sculpt a specific frame and to interpolate the deformation over time. Finally, we think it would be useful to let any edited parameter (scale, rotation, etc.) to be key-framed in order to make time-varying effects more easily controllable.

Chapter 7

Conclusion

In this manuscript, we detailed several approaches to simulate and control mechanical models in computer graphics. Here, we summarize the different contributions, their limitations and perspectives of future work.

7.1 Summary of the contributions

We can divide this work into two main types of contributions.

The first focus was the study of new techniques towards the efficient simulation of mechanical models. Aside from our state of the art on adaptive models, we proposed two approaches tackling this goal. First we introduced a new adaptive method that allows the acceleration of simulations with very few modifications of an existing simulator. In contrast with the usual re-sampling schemes, the method only requires the adaptattion of the time integrator. We demonstrated the efficiency of this method on particle-based fluid and on cloth simulation. Second we proposed a method to handle detailed topological changes while keeping a very low number of degrees of freedom. This was made possible by a dynamic update of the shape functions associated to each degrees of freedom in order to take into account the new topology into the dynamics. Combined with an incremental update of the simulation data and an interpolation of the positions and velocities of the degrees of freedom, this method allows the simulation of detailed cutting of thin sheets at interactive rates.

Our second focus was on the control of physics-based animation. We proposed a new system to edit an animation of liquid. Instead of enforcing a simulation with user constraints, we proposed to build tools that would allow the user to edit existing animations. Starting from a simple sequence of meshes representing the surface of the liquid, the user can select coherent chunks of the animation and edit them with standard paradigms inspired from surface modeling such as copy, edit, paste and temporal tools from movie editing such as temporal remapping. Pushing forward this space-time deformation framework, the user can re-use parts of an animation in other animated scenes.

7.2 Limitations and future work

Adaptively Restrained Particles First, the method could be easily improved by exploring more visual criteria to adapt the simulation. In particular, the use of criteria varying in space and time was not explored and the robustness of the method to such variations is not clear. Second, the interest of the method is restrained to cases where parts of the simulation do not move. Closely related to this freezing technique, we would like to investigate the simplification of parts of a model that behave rigidly. In many cases, elastic deformations are concentrated near the surface of the object while the interior may have a rigid like behavior. We think it would be interesting to identify and simplify these parts.

Detailed cutting of thin sheets There are a large number of directions that could be investigated to improve this method. First, the popping artifacts that can occur when the number of degrees of freedom changes should be prevented. Pfaff et al. [Pfa+14] proposed an optimization-based approach to reduce popping as much as possible in fracture scenario. We could start from their work to see if it would fit the frame-based model. Second, fracture could be integrated by combining a sparse computation of the stress tensor and a procedural method to generate details along the crack. Third, the extension of our method to volumetric objects would require a robust method to build the non-manifold grid we use to update the shape functions. In related works the building of such a grid is made using the volumetric discretization of the object [MCS15; Mit+15]. In our case, it is essential to build the grid only from the surface of the object. Finally, we would like to incorporate plastic deformations while keeping a very low number of degrees of freedom and bring our model to real time performance.

Fluid sculpting The actual system cannot handle large simulations involving very turbulent flows. The first reason is the memory size of such animations. In order to alleviate this problem, we could interact with a low resolution version of the animation and then apply all the transformations of the user to the high resolution version. The second reason is that waves resulting from turbulent flows may present complex topology which could not be faithfully captured by our approach based on displacement field. We think that state of the art method in surface modeling are not sufficient to deal with such surfaces. Instead we would like to use the mesh of these waves as a target to the surface where the waves are being pasted and use a constant volume deformation tool. This would help solving another limitation of our work which is the physical consistency of the deformation. When pasting waves or droplets, there are no guarantee except the user's experience that the result would look realistic. With constant volume deformation we hope that physical consistency would be enforced. Finally we think that we only scratched the surface of the all the temporal edit operations. Building temporal tools that would enforce consistency from one frame to another is an exciting avenue for future research.

Appendix A

Remeshing

In Chapter 5, topological changes are applied on the visual model which is a triangular mesh. As the mesh is only used for visualization. Simulation robustness will not be determined by its elements' quality. Therefore we used an extremely simple remeshing algorithm. The input are the mesh and a polyline that represents the cut. We start by remeshing along the polyline so that the mesh conforms with it. Then we duplicate the mesh vertices along this polyline to create the crack. The whole procedure is summarized in algorithm 8 and illustrated in Figure A.1. The remeshing part uses vertex insertion and edge split operations (see Figure A.2). The splitting part only uses vertex split operation (see Figure A.3).

Algorithm 8 Remeshing Algorithm

```
procedure CUT_ALONG_SEGMENT(Segment $S$, mesh $M$)
    INSERT_SEGMENT($S$, $M$)
    $\tilde{P} \leftarrow$ edges corresponding to $S$
    SPLIT_ALONG_POLYLINE($\tilde{P}$,$M$)
end procedure

procedure INSERT_SEGMENT(Segment $S$, Mesh $M$)
    $\tilde{S} \leftarrow$ subdivide $S$ at intersection with $M$ edges
    for each point $i$ of $\tilde{S}$ do
        $E \leftarrow$ closest edge to $i$
        $V \leftarrow$ closest vertex to $i$
        $F \leftarrow$ closest triangle to $i$
        if distance($E$,$i$)$< \epsilon_{edge}$ then
            Split $E$ at $i$
        else if distance($V$,$i$)$< \epsilon_{vertex}$ then
            Snap $i$ to $V$
        else
            Split $F$ at $i$
        end if
    end for
end procedure

procedure SPLIT_ALONG_POLYLINE(Polyline $P$, Mesh $M$)
    for each vertex $V$ of $P$ do
        Split triangles around $V$ according to $P$
    end for
end procedure
```

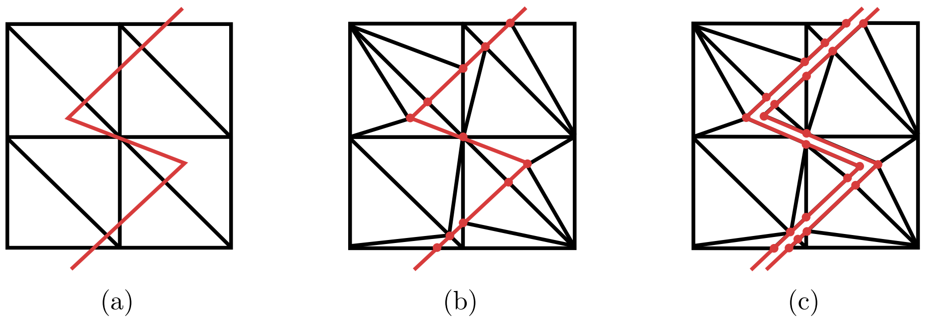


Figure A.1: Illustration of our remeshing algorithm. (a) For remeshing, we start from an input mesh and a polyline that represents the cut. (b) First we re-mesh along the polyline so that the mesh is conform with the cut. (c) Then we split the mesh vertices along the polyline.

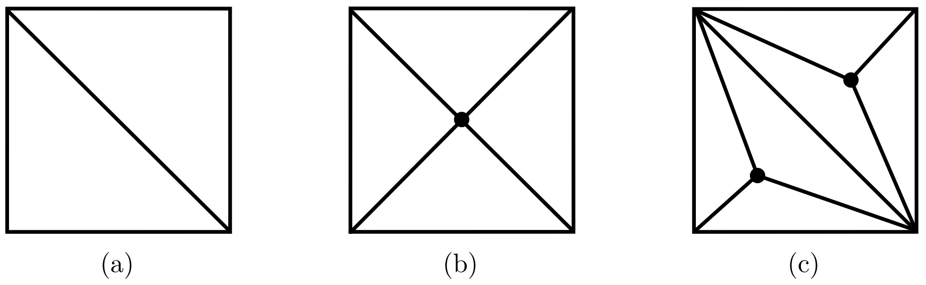


Figure A.2: Illustrations for edge splitting and vertex insertions. (a) The input mesh. (b) After edge splitting. (c) After vertex insertions.

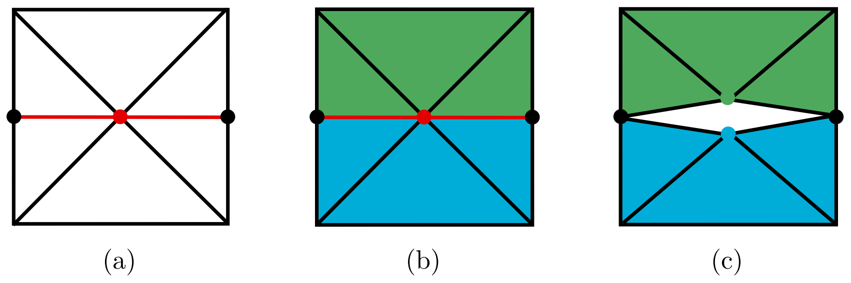


Figure A.3: Illustrations for the vertex splitting operation. (a) A mesh which is conform with the polyline (in red). (b) We start by assigning each triangle around the vertex to split to one side of the polyline. (c) We duplicate the vertex and modify each of the triangles accordingly to its side.

Bibliography

[AAT13] Nadir Akinci, Gizem Akinci, and Matthias Teschner. “Versatile Surface Tension and Adhesion for SPH Fluids”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 182:1–182:8.

[Ada+07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. “Adaptively Sampled Particle Fluids”. In: *ACM Trans. Graph.* 26.3 (July 2007).

[Ain+12] Samantha Ainsley, Etienne Vouga, Eitan Grinspun, and Rasmus Tamstorf. “Speculative Parallel Asynchronous Contact Mechanics”. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 151:1–151:8.

[Aki+12a] Gizem Akinci, Nadir Akinci, Markus Ihmsen, and Matthias Teschner. “An Efficient Surface Reconstruction Pipeline for Particle-Based Fluids”. In: *Workshop on Virtual Reality Interaction and Physical Simulation.* Ed. by Jan Bender, Arjan Kuijper, Dieter W. Fellner, and Eric Guerin. The Eurographics Association, 2012.

[Aki+12b] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. “Versatile Rigid-fluid Coupling for Incompressible SPH”. In: *ACM Trans. Graph.* 31.4 (July 2012), 62:1–62:8.

[All+08] Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. “Recent Advances in Remeshing of Surfaces”. English. In: *Shape Analysis and Structuring.* Ed. by Leila De Floriani and Michela Spagnuolo. Mathematics and Visualization. Springer Berlin Heidelberg, 2008, pp. 53–82.

[Ang+06] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, and Scott King. “Swirling-sweepers: Constant-volume Modeling”. In: *Graph. Models* 68.4 (July 2006), pp. 324–332.

[AR12] Svetlana Artemova and Stephane Redon. “Adaptively Restrained Particle Simulations”. In: *Phys. Rev. Lett.* 109 (19 Nov. 2012), p. 190201.

[ATT12] Ryoichi Ando, Nils Thurey, and Reiji Tsuruno. “Preserving Fluid Sheets with Adaptively Sampled Anisotropic Particles”. In: *IEEE Transactions on Visualization and Computer Graphics* 18.8 (Aug. 2012), pp. 1202–1214.

[ATW13] Ryoichi Ando, Nils Thürey, and Chris Wojtan. “Highly Adaptive Liquid Simulations on Tetrahedral Meshes”. In: *ACM Trans. Graph.* 32.4 (July 2013), 103:1–103:10.

[Bar+06] Adam W. Bargteil, Tolga G. Goktekin, James F. O’Brien, and John A. Strain. “A semi-Lagrangian Contouring Method for Fluid Simulation”. In: *ACM Trans. Graph.* 25.1 (Jan. 2006), pp. 19–38.

[Bar+07] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. “A Finite Element Method for Animating Large Viscoplastic Flow”. In: *ACM Trans. Graph.* 26.3 (July 2007).

[BB09] Tyson Brochu and Robert Bridson. “Robust Topological Operations for Dynamic Explicit Surfaces”. In: *SIAM J. Sci. Comput.* 31.4 (June 2009), pp. 2472–2493.

[BBB10] Tyson Brochu, Christopher Batty, and Robert Bridson. “Matching Fluid Simulation Elements to Surface Geometry and Topology”. In: *ACM Trans. Graph.* 29.4 (July 2010), 47:1–47:9.

[BC14] Adam W Bargteil and Elaine Cohen. “Animation of Deformable Bodies with Quadratic Bézier Finite Elements”. In: *ACM Transactions on Graphics (TOG)* 33.3 (2014), p. 27.

[BD12] Jan Bender and Crispin Deul. “Efficient cloth simulation using an adaptive finite element method”. In: *Virtual Reality Interactions and Physical Simulations (VRIPhys)*. Ed. by Jan Bender, Arjan Kuijper, Dieter Fellner, and Éric Guérin. 2012.

[BDW13] Oleksiy Busaryev, Tamal K. Dey, and Huamin Wang. “Adaptive Fracture Simulation of Multi-layered Thin Plates”. In: *ACM Trans. Graph.* 32.4 (July 2013), 52:1–52:6.

[Ben+12] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. “Interactive Simulation of Rigid Body Dynamics in Computer Graphics”. In: *EUROGRAPHICS 2012 State of the Art Reports*. Cagliari, Sardinia, Italy: Eurographics Association, 2012.

[Ber+03] F. Bertails, T-Y. Kim, M-P. Cani, and U. Neumann. “Adaptive Wisp Tree: A Multiresolution Control Structure for Simulating Dynamic Clustering in Hair Motion”. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’03. San Diego, California: Eurographics Association, 2003, pp. 207–213.

[Ber+07] Miklós Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. "TRACKS: Toward Directable Thin Shells". In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: ACM, 2007.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. "Robust Treatment of Collisions, Contact and Friction for Cloth Animation". In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 594–603.

[BGV09] Ted Belytschko, Robert Gracie, and Giulio Ventura. "A review of extended/generalized finite element methods for material modeling". In: *Modelling and Simulation in Materials Science and Engineering* 17.4 (2009), p. 043001.

[BH11] Christopher Batty and Ben Houston. "A Simple Finite Volume Method for Adaptive Viscous Liquids". In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 111–118.

[BIT09] Markus Becker, Markus Ihmsen, and Matthias Teschner. "Corotated SPH for Deformable Solids". In: *Proceedings of the Fifth Eurographics Conference on Natural Phenomena*. NPH'09. Munich, Germany: Eurographics Association, 2009, pp. 27–34.

[BJ05] Jernej Barbič and Doug L. James. "Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 24.3 (Aug. 2005), pp. 982–990.

[BK15] Jan Bender and Dan Koschier. "Divergence-Free Smoothed Particle Hydrodynamics". In: *Proceedings of the 2015 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2015.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. "Simulation of Clothing with Folds and Wrinkles". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '03. San Diego, California: Eurographics Association, 2003, pp. 28–36.

[Bri08] Robert Bridson. *Fluid Simulation for Computer Graphics*. Ak Peters Series. Taylor & Francis, 2008.

[BSG12] Jernej Barbič, Funshing Sin, and Eitan Grinspun. "Interactive Editing of Deformable Simulations". In: *ACM Trans. Graph.* 31.4 (July 2012), 70:1–70:8.

[BT07] Markus Becker and Matthias Teschner. “Weakly compressible SPH for free surface flows”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation.* SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 209–217.

[BW98] David Baraff and Andrew Witkin. “Large Steps in Cloth Simulation”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’98. New York, NY, USA: ACM, 1998, pp. 43–54.

[BXH10] Christopher Batty, Stefan Xenos, and Ben Houston. “Tetrahedral Embedded Boundary Methods for Accurate and Flexible Adaptive Fluids”. In: *Proceedings of Eurographics.* 2010.

[Cap+02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. “A Multiresolution Framework for Dynamic Deformations”. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’02. San Antonio, Texas: ACM, 2002, pp. 41–47.

[CDS12] Siu-Wing Cheng, Tamal Krishna Dey, and Jonathan Richard Shewchuk. *Delaunay Mesh Generation.* CRC Press, Dec. 2012.

[CF00] Stephen Chenney and D. A. Forsyth. “Sampling Plausible Solutions to Multi-body Constraint Problems”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 219–228.

[CFL28] R. Courant, K. Friedrichs, and H. Lewy. “Über die partiellen Differenzengleichungen der mathematischen Physik”. German. In: *Mathematische Annalen* 100.1 (1928). English translation: Courant et al., “On the partial difference equations of mathematical physics”, IBM J. Res. Dev. (1967)., pp. 32–74.

[Che+07] Nuttapong Chentanez, Bryan E. Feldman, François Labelle, James F. O’Brien, and Jonathan R. Shewchuk. “Liquid Simulation on Lattice-based Tetrahedral Meshes”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 219–228.

[Che+14] Zhili Chen, Miaojun Yao, Renguo Feng, and Huamin Wang. “Physics-inspired Adaptive Fracture Refinement”. In: *ACM Trans. Graph.* 33.4 (July 2014), 113:1–113:7.

[Cla+13] Pascal Clausen, Martin Wicke, Jonathan R. Shewchuk, and James F. O'Brien. "Simulating Liquids and Solid-liquid Interactions with Lagrangian Meshes". In: *ACM Trans. Graph.* 32.2 (Apr. 2013), 17:1–17:15.

[CM03] E. Cerda and L. Mahadevan. "Geometry and Physics of Wrinkling". In: *Phys. Rev. Lett.* 90 (7 Feb. 2003), p. 074302.

[CM10] Nuttapong Chentanez and Matthias Müller. "Real-time Simulation of Large Bodies of Water with Small Scale Details". In: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA '10. Madrid, Spain: Eurographics Association, 2010, pp. 197–206.

[CM11] Nuttapong Chentanez and Matthias Müller. "Real-time Eulerian Water Simulation Using a Restricted Tall Cell Grid". In: *ACM Trans. Graph.* 30.4 (July 2011), 82:1–82:10.

[CMK14] Nuttapong Chentanez, Matthias Müller, and Tae-Yong Kim. "Coupling 3D Eulerian, Heightfield and Particle Methods for Interactive Simulation of Large Scale Liquid Phenomena". In: *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation.* Ed. by Vladlen Koltun and Eftychios Sifakis. Eurographics Association, 2014, pp. 1–10.

[CMT04] Mark Carlson, Peter J Mucha, and Greg Turk. "Rigid fluid: animating the interplay between rigid bodies and fluid". In: *ACM Transactions on Graphics (TOG).* Vol. 23. 3. ACM. 2004, pp. 377–384.

[Coh92] Michael F. Cohen. "Interactive Spacetime Control for Animation". In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '92. New York, NY, USA: ACM, 1992, pp. 293–302.

[Cor+12] Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. "Deformable Objects Alive!" In: *ACM Trans. Graph.* 31.4 (July 2012), 69:1–69:9.

[Cra+13] Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. "Digital Geometry Processing with Discrete Exterior Calculus". In: *ACM SIGGRAPH 2013 courses.* SIGGRAPH '13. Anaheim, California: ACM, 2013.

[CTG10] Jonathan M. Cohen, Sarah Tariq, and Simon Green. "Interactive Fluid-particle Simulation Using Translating Eulerian Grids". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games.* I3D '10. Washington, D.C.: ACM, 2010, pp. 15–22.

[DC96] Mathieu Desbrun and Marie-Paule Cani. “Smoothed Particles: A New Paradigm for Animating Highly Deformable Bodies”. In: *Proceedings of the Eurographics Workshop on Computer Animation and Simulation ’96.* Poitiers, France: Springer-Verlag New York, Inc., 1996, pp. 61–76.

[DC99] Mathieu Desbrun and Marie-Paule Cani. *Space-Time Adaptive Simulation of Highly Deformable Substances.* Anglais. Rapport de recherche RR-3829. INRIA, 1999.

[Deb+00] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. “Adaptive Simulation of Soft Bodies in Real-Time”. In: *Computer Animation 2000, April, 2000.* Philadelphia, Etats-Unis, June 2000, pp. 133–144.

[Deb+01] Gilles Debunne, Mathieu Desbrun, Marie-Paule Cani, and Alan H. Barr. “Dynamic Real-time Deformations Using Space & Time Adaptive Sampling”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’01. ACM, 2001, pp. 31–36.

[Deb+99] Gilles Debunne, Mathieu Desbrun, Alan H. Barr, and Marie-Paule Cani. “Interactive multiresolution animation of deformable models”. In: *Eurographics Workshop on Computer Animation and Simulation’99, September, 1999.* Ed. by Nadia Magnenat-Thalmann and Daniel Thalmann. Computer Science. Springer, Sept. 1999, pp. 133–144.

[DGW11] C. Dick, J. Georgii, and R. Westermann. “A Hexahedral Multigrid Approach for Simulating Cuts in Deformable Objects”. In: *Visualization and Computer Graphics, IEEE Transactions on* 17.11 (Nov. 2011), pp. 1663–1675.

[Dob+08] Yoshinori Dobashi, Yasuhiro Matsuda, Tsuyoshi Yamamoto, and Tomoyuki Nishita. “A Fast Simulation Method Using Overlapping Grids for Interactions between Smoke and Rigid Objects”. In: *Computer Graphics Forum* 27.2 (2008), pp. 477–486.

[EB12] Essex Edwards and Robert Bridson. “A high-order accurate particle-in-cell method”. In: *International Journal for Numerical Methods in Engineering* 90.9 (2012), pp. 1073–1088.

[EB14] Essex Edwards and Robert Bridson. “Detailed Water with Coarse Grids: Combining Surface Meshes and Adaptive Discontinuous Galerkin”. In: *ACM Trans. Graph.* 33.4 (July 2014), 136:1–136:9.

[Eng+13] R. Elliot English, Linhai Qiu, Yue Yu, and Ronald Fedkiw. “An Adaptive Discretization of Incompressible Flow Using a Multitude of Moving Cartesian Grids”. In: *J. Comput. Phys.* 254 (Dec. 2013), pp. 107–154.

[Fau+11] François Faure, Benjamin Gilles, Guillaume Bousquet, and Dinesh K. Pai. “Sparse Meshless Models of Complex Deformable Solids”. In: *ACM Transactions on Graphics.* Proceedings of SIGGRAPH’2011 30.4 (July 2011), Article No. 73.

[FCG00] Eric Ferley, Marie-Paule Cani, and Jean-Dominique Gascuel. “Practical Volumetric Sculpting”. In: *Visual Computer* 16.8 (2000), pp. 469–480.

[FDL07] William Fong, Eric Darve, and Adrian Lew. “Stability of Asynchronous Variational Integrators”. In: *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation.* PADS ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 38–44.

[FF01] Nick Foster and Ronald Fedkiw. “Practical Animation of Liquids”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’01. ACM, 2001, pp. 23–30.

[FL04] Raanan Fattal and Dani Lischinski. “Target-driven Smoke Animation”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 441–448.

[FSH11] B. Fierz, J. Spillmann, and M. Harders. “Element-wise Mixed Implicit-explicit Integration for Stable Dynamic Simulation of Deformable Objects”. In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 257–266.

[FTS06] Wolfram von Funck, Holger Theisel, and Hans-Peter Seidel. “Vector field based shape deformations”. In: *ACM TOG, proc. of SIGGRAPH.* 2006.

[FWD14] F. Ferstl, R. Westermann, and C. Dick. “Large-Scale Liquid Simulation on Adaptive Hexahedral Grids”. In: *Visualization and Computer Graphics, IEEE Transactions on* PP.99 (2014), pp. 1–1.

[Gay+07] Russell Gayle, Stephane Redon, Avneesh Sud, Ming C Lin, and Dinesh Manocha. “Efficient motion planning of highly articulated chains using physics-based sampling”. In: *Robotics and Automation, 2007 IEEE International Conference on.* IEEE. 2007, pp. 3319–3326.

[GB14] Prashant Goswami and Christopher Batty. “Regional Time Stepping for SPH”. Anglais. In: *Eurographics 2014.* Ed. by Eric Galin and Michael Wand. Eurographics Association, Apr. 2014, pp. 45–48.

[GBF03] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. “Nonconvex Rigid Bodies with Stacking”. In: *ACM Trans. Graph.* 22.3 (July 2003), pp. 871–878.

[GCS99] Fabio Ganovelli, Paolo Cignoni, and Roberto Scopigno. “Introducing Multiresolution Representation in Deformable Object Modeling”. In: *Proceedings of SCCG99.* 1999, pp. 149–158.

[GH04] S. T. Greenwood and D. H. House. “Better with Bubbles: Enhancing the Visual Realism of Simulated Fluid”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’04. Grenoble, France: Eurographics Association, 2004, pp. 287–296.

[Gil+11] Benjamin Gilles, Guillaume Bousquet, Francois Faure, and Dinesh K. Pai. “Frame-based Elastic Models”. In: *ACM Trans. Graph.* 30.2 (Apr. 2011), 15:1–15:12.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. “CHARMS: A Simple Framework for Adaptive Simulation”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 281–290.

[GLM06] Russell Gayle, Ming C. Lin, and Dinesh Manocha. “Adaptive Dynamics with Efficient Contact Handling for Articulated Robots.” In: *Robotics: Science and Systems.* The MIT Press, 2006, pp. 231–238.

[Goe+15] Fernando de Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. “Power Particles: An Incompressible Fluid Solver Based on Power Diagrams”. In: *ACM Trans. Graph.* 34.4 (July 2015), 50:1–50:11.

[GP11] Prashant Goswami and Renato Pajarola. “Time Adaptive Approximate SPH”. In: *Proceedings Eurographics Workshop on Virtual Reality Interaction and Physical Simulation.* 2011.

[Hag+05] Trond Runar Hagen, Jon M Hjelmervik, K-A Lie, Jostein R Natvig, and M Ofstad Henriksen. “Visual simulation of shallow-water waves”. In: *Simulation Modelling Practice and Theory* 13.8 (2005), pp. 716–726.

[Hah+12] Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. “Rig-space Physics”. In: *ACM Trans. Graph.* 31.4 (July 2012), 72:1–72:8.

[Hah+14] Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, Forrester Cole, Mark Meyer, Tony DeRose, and Markus Gross. “Subspace Clothing Simulation Using Adaptive Bases”. In: *ACM Trans. Graph.* 33.4 (July 2014), 105:1–105:9.

[Har+09] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. “Asynchronous Contact Mechanics”. In: *ACM Trans. Graph.* 28.3 (July 2009), 87:1–87:12.

[Heg+13] Jan Hegemann, Chenfanfu Jiang, Craig Schroeder, and Joseph M Teran. “A level set method for ductile fracture”. In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* ACM. 2013, pp. 193–201.

[HHK08] Woosuck Hong, Donald H. House, and John Keyser. “Adaptive Particles for Incompressible Fluid Simulation”. In: *Vis. Comput.* 24.7 (July 2008), pp. 535–543.

[Hil+11] Klaus Hildebrandt, Christian Schulz, Christoph Von Tycowicz, and Konrad Polthier. “Interactive Surface Modeling Using Modal Analysis”. In: *ACM Trans. Graph.* 30.5 (Oct. 2011), 119:1–119:11.

[Hil+12] Klaus Hildebrandt, Christian Schulz, Christoph von Tycowicz, and Konrad Polthier. “Interactive Spacetime Control of Deformable Objects”. In: *ACM Trans. Graph.* 31.4 (July 2012), 71:1–71:8.

[HK04] Jeong-Mo Hong and Chang-Hun Kim. “Controlling Fluid Animation with Geometric Potential: Research Articles”. In: *Comput. Animat. Virtual Worlds* 15.3-4 (July 2004), pp. 147–157.

[HK05] Jeong-Mo Hong and Chang-Hun Kim. “Discontinuous Fluids”. In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 915–920.

[HK13] Ruoguan Huang and John Keyser. “Automated sampling and control of gaseous simulations”. English. In: *The Visual Computer* 29.6-8 (2013), pp. 751–760.

[HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. “Interactive Animation of Ocean Waves”. In: *ACM-SIGGRAPH - EG Symposium on Computer Animation (SCA’02).* San Antonio, United States, July 2002.

[Hon+08] Jeong-Mo Hong, Ho-Young Lee, Jong-Chul Yoon, and Chang-Hun Kim. “Bubbles Alive”. In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 48:1–48:4.

[Hop96] Hugues Hoppe. “Progressive Meshes”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 99–108.

[Hor15] Christopher J Horvath. “Empirical directional wave spectra for computer graphics”. In: *Proceedings of the 2015 Symposium on Digital Production.* ACM. 2015, pp. 29–39.

[HPH96] Dave Hutchinson, Martin Preston, and Terry Hewitt. “Adaptive Refinement for Mass/Spring Simulations”. In: *Proceedings of the Eurographics Workshop on Computer Animation and Simulation ’96*. Poitiers, France: Springer-Verlag New York, Inc., 1996, pp. 31–45.

[HS13] C Horvath and Barbara Solenthaler. *Mass Preserving Multi-Scale SPH*. Pixar Technical Memo 13-04. Pixar Animation Studios, 2013.

[HW04] Nathan Holmberg and Burkhard C. Wünsche. “Efficient Modeling and Rendering of Turbulent Water over Natural Terrain”. In: *Proceedings of the 2Nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*. GRAPHITE ’04. Singapore: ACM, 2004, pp. 15–22.

[HZ13] David Harmon and Denis Zorin. “Subspace Integration with Local Deformations”. In: *ACM Trans. Graph.* 32.4 (July 2013), 107:1–107:10.

[Ihm+10] Markus Ihmsen, Nadir Akinci, Marc Gissler, and Matthias Teschner. “Boundary Handling and Adaptive Time-stepping for PCISPH”. In: *Proceedings of the Seventh Workshop on Virtual Reality Interactions and Physical Simulations, VRIPHYS 2010, Copenhagen, Denmark, 2010*. Ed. by Kenny Erleben, Jan Bender, and Matthias Teschner. Eurographics Association, 2010, pp. 79–88.

[Ihm+11] Markus Ihmsen, Nadir Akinci, Markus Becker, and Matthias Teschner. “A Parallel SPH Implementation on Multi-Core CPUs”. In: *Computer Graphics Forum* (2011).

[Ihm+12] Markus Ihmsen, Nadir Akinci, Gizem Akinci, and Matthias Teschner. “Unified Spray, Foam and Air Bubbles for Particle-based Fluids”. In: *Vis. Comput.* 28.6-8 (June 2012), pp. 669–677.

[Ihm+14a] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. “Implicit Incompressible SPH”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.3 (2014), pp. 426–435.

[Ihm+14b] Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, and Matthias Teschner. “SPH Fluids in Computer Graphics”. In: *Eurographics 2014 - State of the Art Reports*. Ed. by Sylvain Lefebvre and Michela Spagnuolo. The Eurographics Association, 2014.

[Irv+06] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. “Efficient Simulation of Large Bodies of Water by Coupling Two and Three Dimensional Techniques”. In: *ACM Trans. Graph.* 25.3 (July 2006), pp. 805–811.

[Jef85] David R. Jefferson. “Virtual Time”. In: *ACM Trans. Program. Lang. Syst.* 7.3 (July 1985), pp. 404–425.

[Jia+16] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. “The Material Point Method for Simulating Continuum Materials”. In: *ACM SIGGRAPH 2016 Courses.* SIGGRAPH ’16. Anaheim, California: ACM, 2016, 24:1–24:52.

[Jia07] Xiangmin Jiao. “Face Offsetting: A Unified Approach for Explicit Moving Interfaces”. In: *J. Comput. Phys.* 220.2 (Jan. 2007), pp. 612–625.

[Jon+14] Ben Jones, Stephen Ward, Ashok Jallepalli, Joseph Perenia, and Adam Bargteil. “Deformation Embedding for Point-Based Elastoplastic Simulation”. In: *ACM Trans. Graph.* 33.2 (Mar. 2014).

[Jon+16] Ben Jones, Nils Thuerey, Tamar Shinar, and Adam W. Bargteil. “Example-based Plastic Deformation of Rigid Bodies”. In: *ACM Trans. Graph.* 35.4 (July 2016).

[JW15] Stefan Jeschke and Chris Wojtan. “Water Wave Animation via Wavefront Parameter Interpolation”. In: *ACM Transactions on Graphics (TOG)* 34.3 (2015), p. 27.

[Kau+09] Peter Kaufmann, Sebastian Martin, Mario Botsch, Eitan Grinspun, and Markus Gross. “Enrichment Textures for Detailed Cutting of Shells”. In: *ACM Trans. Graph.* 28.3 (July 2009), 50:1–50:10.

[Kau+14] Danny M. Kaufman, Rasmus Tamstorf, Breannan Smith, Jean-Marie Aubry, and Eitan Grinspun. “Adaptive Nonlinearity for Collisions in Complex Rod Assemblies”. In: *ACM Trans. Graph.* 33.4 (July 2014), 123:1–123:12.

[KB14] Todd Keeler and Robert Bridson. “Ocean waves animation using boundary integral equations and explicit mesh tracking”. In: *ACM SIGGRAPH 2014 Posters.* ACM. 2014, p. 11.

[KGL07] Ilknur Kabul, Russell Gayle, and Ming C. Lin. “Cable Route Planning in Complex Environments Using Constrained Sampling”. In: *Proceedings of the 2007 ACM Symposium on Solid and Physical Modeling.* SPM ’07. Beijing, China: ACM, 2007, pp. 395–402.

[Kim+06] Janghee Kim, Deukhyun Cha, Byungjoon Chang, Bonki Koo, and Insung Ihm. “Practical Animation of Turbulent Splashing Water”. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’06. Vienna, Austria: Eurographics Association, 2006, pp. 335–344.

[Kim+07] Byungmoon Kim, Yingjie Liu, Ignacio Llamas, Xiangmin Jiao, and Jarek Rossignac. "Simulation of Bubbles in Foam with the Volume Control Method". In: *ACM Trans. Graph.* 26.3 (July 2007).

[Kim10] Byungmoon Kim. "Multi-phase Fluid Simulations Using Regional Level Sets". In: *ACM Trans. Graph.* 29.6 (Dec. 2010), 175:1–175:8.

[KJ09] Theodore Kim and Doug L. James. "Skipping Steps in Deformable Simulation with Online Model Reduction". In: *ACM Trans. Graph.* 28.5 (Dec. 2009), 123:1–123:9.

[KLB14] Dan Koschier, Sebastian Lipponer, and Jan Bender. "Adaptive Tetrahedral Meshes for Brittle Fracture Simulation". In: *Proceedings of the 2014 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Copenhagen, Denmark: Eurographics Association, 2014.

[Kli+06] Bryan M. Klingner, Bryan E. Feldman, Nuttapong Chentanez, and James F. O'Brien. "Fluid Animation with Dynamic Meshes". In: *ACM Trans. Graph.* 25.3 (July 2006), pp. 820–825.

[KM90] Michael Kass and Gavin Miller. "Rapid, stable fluid dynamics for computer graphics". In: *ACM SIGGRAPH Computer Graphics.* Vol. 24. 4. ACM. 1990, pp. 49–57.

[KMD06] Yootai Kim, Raghu Machiraju, and Thompson David. "Path-based Control of Smoke Simulations". In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA '06. Vienna, Austria: Eurographics Association, 2006, pp. 33–42.

[KNO14] Woojong Koh, Rahul Narain, and James F. O'Brien. "View-Dependent Adaptive Cloth Simulation". In: *The Eurographics / ACM SIGGRAPH Symposium on Computer Animation, SCA '14, Copenhagen, Denmark, 2014.* Eurographics Association, 2014, pp. 159–166.

[KRK08a] Sujeong Kim, Stephane Redon, and J. Kim Young. "Continuous Collision Detection for Adaptive Simulation of Articulated Bodies". English. In: *Visual Computer* 24.4 (Apr. 2008), pp. 261–269.

[KRK08b] Sujeong Kim, Stephane Redon, and Young J. Kim. "View-dependent dynamics of articulated bodies". In: *Computer Animation and Virtual Worlds* 19.3-4 (2008), pp. 223–233.

[Kru56] Joseph B Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical society* 7.1 (1956), pp. 48–50.

[KS07] Bryan Matthew Klingner and Jonathan Richard Shewchuk. “Aggressive Tetrahedral Mesh Improvement”. In: *Proceedings of the 16th International Meshing Roundtable*. Oct. 2007, pp. 3–23.

[LC96] Alexis Lamouret and Marie-Paule Cani. “Scripting Interactive Physically-Based Motions with Relative Paths and Synchronization.” In: *Computer Graphics Forum*. Computer Graphics Forum 15.1 (1996). Preliminary version in Graphics Interface 95, published under the name Marie-Paule Gascuel, pp. 25–34.

[Lej+15] Thibault Lejemble, Amélie Fondevilla, Nicolas Durin, Thibault Blanc-Beyne, Camille Schreck, Pierre-Luc Manteaux, Paul G. Kry, and Marie-Paule Cani. “Interactive Procedural Simulation of Paper Tearing with Sound”. In: *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*. MIG ’15. Paris, France: ACM, 2015, pp. 143–149.

[Lew+04] A. Lew, J. E. Marsden, M. Ortiz, and M. West. “Variational time integrators”. In: *Int. J. Numer. Methods Eng* 60 (2004), pp. 153–212.

[LFO05] Frank Losasso, Ronald Fedkiw, and Stanley Osher. “Spatially adaptive techniques for level set methods and incompressible flow”. In: *Computers and Fluids* 35 (2005).

[LGF04] Frank Losasso, Frédéric Gibou, and Ron Fedkiw. “Simulating Water and Smoke with an Octree Data Structure”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 457–462.

[Li+14] Siwang Li, Jin Huang, Fernando de Goes, Xiaogang Jin, Hujun Bao, and Mathieu Desbrun. “Space-time Editing of Elastic Motion Through Material Optimization and Reduction”. In: *ACM Trans. Graph.* 33.4 (July 2014), 108:1–108:10.

[Los+08] Frank Losasso, Jerry Talton, Nipun Kwatra, and Ronald Fedkiw. “Two-Way Coupled SPH and Particle Level Set Fluid Simulation”. In: *IEEE Transactions on Visualization and Computer Graphics* 14.4 (July 2008), pp. 797–804.

[LP02] Anita T Layton and Michiel van de Panne. “A numerically efficient and stable algorithm for animating water waves”. In: *The Visual Computer* 18.1 (2002), pp. 41–53.

[LR56] P. D. Lax and R. D. Richtmyer. “Survey of the stability of linear finite difference equations”. In: *Communications on Pure and Applied Mathematics* 9.2 (1956), pp. 267–293.

[LS07] François Labelle and Jonathan Richard Shewchuk. “Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles”. In: *ACM Trans. Graph.* 26.3 (July 2007).

[LV05] Ling Li and Vasily Volkov. “Cloth Animation with Adaptively Refined Meshes”. In: *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38.* ACSC ’05. Newcastle, Australia: Australian Computer Society, Inc., 2005, pp. 107–113.

[Man+13] Pierre-Luc Manteaux, François Faure, Stephane Redon, and Marie-Paule Cani. “Exploring the Use of Adaptively Restrained Particles for Graphics Simulations”. In: *VRIPHYS 2013 - 10th Workshop on Virtual Reality Interaction and Physical Simulation.* VRIPHYS 2013 - 10th Workshop on Virtual Reality Interaction and Physical Simulation. Lille, France: Eurographics Association, 2013, pp. 17–24.

[Man+15] Pierre-Luc Manteaux, Wei-Lun Sun, François Faure, Marie-Paule Cani, and James F. O’Brien. “Interactive Detailed Cutting of Thin Sheets”. In: *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games.* MIG ’15. Paris, France: ACM, 2015, pp. 125–132.

[Man+16] P.-L. Manteaux, C. Wojtan, R. Narain, S. Redon, F. Faure, and M.-P. Cani. “Adaptive Physically Based Models in Computer Graphics”. In: *Computer Graphics Forum* (2016), n/a–n/a.

[Mar+11] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. “Example-based Elastic Materials”. In: *ACM SIGGRAPH 2011 Papers.* SIGGRAPH ’11. Vancouver, British Columbia, Canada: ACM, 2011, 72:1–72:8.

[MB12] Marek Krzysztof Misztal and Jakob Andreas Bærentzen. “Topology-adaptive Interface Tracking Using the Deformable Simplicial Complex”. In: *ACM Trans. Graph.* 31.3 (June 2012), 24:1–24:12.

[MBF04] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. “A Virtual Node Algorithm for Changing Mesh Topology During Simulation”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 385–392.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. “Particle-based Fluid Simulation for Interactive Applications”. In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’03. San Diego, California: Eurographics Association, 2003, pp. 154–159.

[McN+04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. “Fluid Control Using the Adjoint Method”. In: *ACM Trans. Graph.* 23.3 (Aug. 2004), pp. 449–456.

[MCS15] Nathan Mitchell, Court Cutting, and Eftychios Sifakis. "GRIDiron: An Interactive Authoring and Cognitive Training Foundation for Reconstructive Plastic Surgery Procedures". In: *ACM Trans. Graph.* 34.4 (July 2015), 43:1–43:12.

[Mer+15] Olivier Mercier, Cynthia Beauchemin, Nils Thuerey, Theodore Kim, and Derek Nowrouzezahrai. "Surface Turbulence for Particle-based Liquid Simulations". In: *ACM Trans. Graph.* 34.6 (Oct. 2015), 202:1–202:10.

[Mir00] Brian Mirtich. "Timewarp Rigid Body Simulation". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 193–200.

[Mis+12] M. K. Misztal, K. Erleben, A. Bargteil, J. Fursund, B. Bunch Christensen, J. A. Bærentzen, and R. Bridson. "Multiphase Flow of Immiscible Fluids on Unstructured Moving Meshes". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA '12. Lausanne, Switzerland: Eurographics Association, 2012, pp. 97–106.

[Mit+15] Nathan Mitchell, Mridul Aanjaneya, Rajsekhar Setaluri, and Eftychios Sifakis. "Non-manifold Level Sets: A Multivalued Implicit Surface Representation with Applications to Self-collision Processing". In: *ACM Trans. Graph.* 34.6 (Oct. 2015), 247:1–247:9.

[MM13] Jamie Madill and David Mould. "Target Particle Control of Smoke Simulation". In: *Proceedings of Graphics Interface 2013.* GI '13. Regina, Sascatchewan, Canada: Canadian Information Processing Society, 2013, pp. 125–132.

[Mol+03] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. "A Crystalline, Red Green Strategy for Meshing Highly Deformable Objects with Tetrahedra". In: *IMR.* 2003, pp. 103–114.

[Mon05] J. J. Monaghan. "Smoothed particle hydrodynamics". In: *Reports on Progress in Physics* 68.8 (Aug. 2005), pp. 1703–1759.

[Mon92] J. J. Monaghan. "Smoothed particle hydrodynamics". In: *Annual Review of Astronomy and Astrophysics* 30 (1992), pp. 543–574.

[MR07] S. Morin and S. Redon. "A Force-Feedback Algorithm for Adaptive Articulated-Body Dynamics Simulation". In: *Robotics and Automation, 2007 IEEE International Conference on.* Apr. 2007, pp. 3245–3250.

[Mül+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. “Point Based Animation of Elastic, Plastic and Melting Objects”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’04. Grenoble, France: Eurographics Association, 2004, pp. 141–151.

[Mül+08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. “Real Time Physics: Class Notes”. In: *ACM SIGGRAPH 2008 Classes*. SIGGRAPH ’08. Los Angeles, California: ACM, 2008, 88:1–88:90.

[MY97] David Mould and Yee-Hong Yang. “Modeling water for computer graphics”. In: *Computers & Graphics* 21.6 (1997), pp. 801–814.

[Nar+07] Rahul Narain, Vivek Kwatra, Huai-Ping Lee, Theodore Kim, Mark Carlson, and Ming C. Lin. “Feature-guided Dynamic Texture Synthesis on Continuous Flows”. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR’07. Grenoble, France: Eurographics Association, 2007, pp. 361–370.

[NB11] Michael B. Nielsen and Robert Bridson. “Guide Shapes for High Resolution Naturalistic Liquid Simulation”. In: *ACM Trans. Graph.* 30.4 (July 2011), 83:1–83:8.

[NC10] Michael B. Nielsen and Brian B. Christensen. “Improved Variational Guiding of Smoke Animations”. In: *Computer Graphics Forum* 29.2 (2010), pp. 705–712.

[Nea+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. “Physically Based Deformable Models in Computer Graphics”. In: *Computer Graphics Forum* 25.4 (2006), pp. 809–836.

[Nes+09] Matthieu Nesme, Paul Kry, Lenka Jerabkova, and François Faure. “Preserving Topology and Elasticity for Embedded Deformable Models”. In: *ACM Transactions on Graphics*. Proceedings of ACM SIGGRAPH 2009 28.3 (Aug. 2009), Article No. 52.

[Nie+09] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. “Guiding of Smoke Animations Through Variational Coupling of Simulations at Different Resolutions”. In: *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’09. New Orleans, Louisiana: ACM, 2009, pp. 217–226.

[NPO13] Rahul Narain, Tobias Pfaff, and James F. O’Brien. “Folding and Crumpling Adaptive Sheets”. In: *ACM Trans. Graph.* 32.4 (July 2013), 51:1–51:8.

[NSO12] Rahul Narain, Armin Samii, and James F. O'Brien. "Adaptive Anisotropic Remeshing for Cloth Simulation". In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 152:1–152:10.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. "Graphical Modeling and Animation of Ductile Fracture". In: *Proceedings of ACM SIGGRAPH 2002*. San Antonio, Texas: ACM Press, Aug. 2002, pp. 291–294.

[OH95] J. F. O'Brien and J. K. Hodgins. "Dynamic Simulation of Splashing Fluids". In: *Proceedings of the Computer Animation*. CA '95. IEEE Computer Society, 1995, pp. 198–.

[OH99] James F. O'Brien and Jessica K. Hodgins. "Graphical Modeling and Animation of Brittle Fracture". In: *Proceedings of ACM SIGGRAPH 1999*. ACM Press/Addison-Wesley Publishing Co., Aug. 1999, pp. 137–146.

[OK12] Jens Orthmann and Andreas Kolb. "Temporal Blending for Adaptive SPH". In: *Comp. Graph. Forum* 31.8 (Dec. 2012), pp. 2436–2449.

[Ota+07] Miguel A. Otaduy, Daniel Germann, Stephane Redon, and Markus Gross. "Adaptive Deformations with Fast Tight Bounds". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '07. San Diego, California: Eurographics Association, 2007, pp. 181–190.

[Pan+13] Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. "Interactive Localized Liquid Motion Editing". In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 184:1–184:10.

[Pat+13] Saket Patkar, Mridul Aanjaneya, Dmitriy Karpman, and Ronald Fedkiw. "A Hybrid Lagrangian-Eulerian Formulation for Bubble Generation and Dynamics". In: *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '13. Anaheim, California: ACM, 2013, pp. 105–114.

[Pau+05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré, Markus Gross, and Leonidas J. Guibas. "Meshless Animation of Fracturing Solids". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 957–964.

[PCS04] Frédéric Pighin, Jonathan M. Cohen, and Maurya Shah. "Modeling and Editing Flows Using Advected Radial Basis Functions". In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '04. Grenoble, France: Eurographics Association, 2004, pp. 223–232.

[Pee+15] Andreas Peer, Markus Ihmsen, Jens Cornelis, and Matthias Teschner. "An Implicit Viscosity Formulation for SPH Fluids". In: *ACM Trans. Graph.* 34.4 (July 2015), 114:1–114:10.

[Pfa+14] Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien. "Adaptive Tearing and Cracking of Thin Sheets". In: *ACM Trans. Graph.* 33.4 (July 2014), 110:1–110:9.

[PO09] Eric G. Parker and James F. O'Brien. "Real-Time Deformation and Fracture in a Game Environment". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Aug. 2009, pp. 156–166.

[Pop+00] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. "Interactive Manipulation of Rigid Body Simulations". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 209–217.

[Pro97] Xavier Provot. "Collision and self-collision handling in cloth model dedicated to design garments". English. In: *Computer Animation and Simulation '97.* Ed. by Daniel Thalmann and Michiel van de Panne. Eurographics. Springer Vienna, 1997, pp. 177–189.

[PSE03] Jovan Popović, Steven M. Seitz, and Michael Erdmann. "Motion Sketching for Control of Rigid-body Simulations". In: *ACM Trans. Graph.* 22.4 (Oct. 2003), pp. 1034–1054.

[QLF16] Linhai Qiu, Wenlong Lu, and Ronald Fedkiw. "An Adaptive Discretization of Compressible Flow Using a Multitude of Moving Cartesian Grids". In: *J. Comput. Phys.* 305.C (Jan. 2016), pp. 75–110.

[Rav+12] Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. "Controlling Liquids Using Meshes". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA '12. Lausanne, Switzerland: Eurographics Association, 2012, pp. 255–264.

[Rav+14] Karthik Raveendran, Chris Wojtan, Nils Thuerey, and Greg Turk. "Blending Liquids". In: *ACM Trans. Graph.* 33.4 (July 2014), 137:1–137:10.

[RGL05] Stephane Redon, Nico Galoppo, and Ming C. Lin. "Adaptive Dynamics of Articulated Bodies". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 936–945.

[RK13] Olivier Rémillard and Paul G. Kry. “Embedded Thin Shells for Wrinkle Simulation”. In: *ACM Trans. Graph.* 32.4 (July 2013), 50:1–50:8.

[RL06] Stephane Redon and C. Lin Ming. “An Efficient, Error-Bounded Approximation Algorithm for Simulating Quasi-Statics of Complex Linkages”. Anglais. In: *Computer-Aided Design* (2006).

[Rob+08] Avi Robinson-Mosher, Tamar Shinar, Jon Gretarsson, Jonathan Su, and Ronald Fedkiw. “Two-way coupling of fluids to rigid and deformable solids and shells”. In: *ACM Transactions on Graphics (TOG).* Vol. 27. 3. ACM. 2008, p. 46.

[SA07] Olga Sorkine and Marc Alexa. “As-rigid-as-possible Surface Modeling”. In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing.* SGP ’07. Barcelona, Spain: Eurographics Association, 2007, pp. 109–116.

[SCC11] Lucian Stanculescu, Raphaëlle Chaine, and Marie-Paule Cani. “Freestyle: Sculpting meshes with self-adaptive topology”. In: *Computers & Graphics.* 2011.

[Sch+14] Christian Schulz, Christoph von Tycowicz, Hans-Peter Seidel, and Klaus Hildebrandt. “Animating Deformable Objects Using Sparse Spacetime Constraints”. In: *ACM Trans. Graph.* 33.4 (July 2014), 109:1–109:10.

[Sch02] H. Schmidl. “Optimization-based animation”. PhD thesis. The University of Miami, 2002.

[SDE05] Joshua Schpok, William Dwyer, and David S. Ebert. “Modeling and Animating Gases with Simulation Features”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’05. Los Angeles, California: ACM, 2005, pp. 97–105.

[SDF07] Eftychios Sifakis, Kevin G. Der, and Ronald Fedkiw. “Arbitrary Cutting of Deformable Tetrahedralized Objects”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 73–80.

[Sei+11] Martin Seiler, Denis Steinemann, Jonas Spillmann, and Matthias Harders. “Robust Interactive Cutting Based on an Adaptive Octree Simulation Mesh”. In: *Vis. Comput.* 27.6-8 (June 2011), pp. 519–529.

[Ser+11] M. Servin, C. Lacoursière, F. Nordfelth, and K. Bodin. “Hybrid, Multiresolution Wires with Massless Frictional Contacts”. In: *Visualization and Computer Graphics, IEEE Transactions on* 17.7 (July 2011), pp. 970–982.

[Ser86] Jean Serra. “Introduction to mathematical morphology”. In: *Computer vision, graphics, and image processing* 35.3 (1986), pp. 283–305.

[SG11] Barbara Solenthaler and Markus Gross. “Two-scale Particle Simulation”. In: *ACM Trans. Graph.* 30.4 (July 2011), 81:1–81:8.

[Sha+04] Maurya Shah, Jonathan M. Cohen, Sanjit Patel, Penne Lee, and Frédéric Pighin. “Extended Galilean Invariance for Adaptive Fluid Simulation”. In: *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’04. Grenoble, France: Eurographics Association, 2004, pp. 213–221.

[She02] J Shewchuk. “What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint)”. In: *University of California at Berkeley* 73 (2002).

[Sif+07] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. “Hybrid Simulation of Deformable Solids”. In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’07. San Diego, California: Eurographics Association, 2007, pp. 81–90.

[SLD09] Timothy J. R. Simnett, Stephen D. Laycock, and Andy M. Day. “An Edge-based Approach to Adaptively Refining a Mesh for Cloth Deformation”. In: *TPCG.* 2009, pp. 77–84.

[SLN08] M Servin, C Lacoursiere, and F Nordfelth. “Adaptive resolution in physics based virtual environments”. In: *SIGRAD 2008* (2008).

[SOG09] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. “Splitting meshless deforming objects with explicit surface tracking”. In: *Graphical Models* 71.6 (2009). 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006), pp. 209–220.

[SP09] B. Solenthaler and R. Pajarola. “Predictive-corrective Incompressible SPH”. In: *ACM Trans. Graph.* 28.3 (July 2009), 40:1–40:6.

[SS10] Ryan Schmidt and Karan Singh. “Meshmixer: an interface for rapid mesh composition”. In: *ACM SIGGRAPH 2010 Talks.* ACM. 2010.

[SSF08] Tamar Shinar, Craig Schroeder, and Ronald Fedkiw. “Two-way coupling of rigid and deformable bodies”. In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Eurographics Association. 2008, pp. 95–103.

[ST08] Jonas Spillmann and Matthias Teschner. “An Adaptive Contact Model for the Robust Simulation of Knots”. In: *Comput. Graph. Forum* 27.2 (2008), pp. 497–506.

[Sta99] Jos Stam. “Stable Fluids”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[Str04] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations, Second Edition.* Society for Industrial and Applied Mathematics, 2004. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9780898717938.

[Sue+11] Shinjiro Sueda, Garrett L. Jones, David I. W. Levin, and Dinesh K. Pai. “Large-scale Dynamic Simulation of Highly Constrained Strands”. In: *ACM Trans. Graph.* 30.4 (July 2011), 39:1–39:10.

[SY04] Lin Shi and Yizhou Yu. “Visual smoke simulation with adaptive octree refinement”. In: *Proceedings of the Seventh IASTED International Conference on Computer Graphics and Imaging.* 2004, pp. 13–19.

[SY05a] Lin Shi and Yizhou Yu. “Controllable Smoke Animation with Guiding Objects”. In: *ACM Trans. Graph.* 24.1 (Jan. 2005), pp. 140–164.

[SY05b] Lin Shi and Yizhou Yu. “Taming Liquids for Rapidly Changing Targets”. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* SCA ’05. Los Angeles, California: ACM, 2005, pp. 229–236.

[Tak+03] Tsunemi Takahashi, Hiroko Fujii, Atsushi Kunimatsu, Kazuhiro Hiwada, Takahiro Saito, Ken Tanaka, and Heihachi Ueki. “Realistic Animation of Fluid with Splash and Foam”. In: *Computer Graphics Forum* 22.3 (2003), pp. 391–400.

[Tak+11] Kenshi Takayama, Ryan Schmidt, Karan Singh, Takeo Igarashi, Tamy Boubekeur, and Olga Sorkine. “GeoBrush: Interactive Mesh Geometry Cloning”. In: *Computer Graphics Forum (proceedings of EUROGRAPHICS)* 30.2 (2011), pp. 613–622.

[Tak+15] Tetsuya Takahashi, Yoshinori Dobashi, Issei Fujishiro, Tomoyuki Nishita, and Ming C. Lin. “Implicit Formulation for SPH-based Viscous Fluids”. In: *Computer Graphics Forum* 34.2 (2015), pp. 493–502.

[Ten+15] Yun Teng, Mark Meyer, Tony DeRose, and Theodore Kim. “Subspace Condensation: Full Space Adaptivity for Subspace Deformations”. In: *ACM Trans. Graph.* 34.4 (July 2015), 76:1–76:9.

[Tes+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. “Collision Detection for Deformable Objects”. In: *Computer Graphics Forum* 24.1 (2005), pp. 61–81.

[Tes04a] Jerry Tessendorf. “Interactive water surfaces”. In: *Game Programming Gems* 4 (2004), pp. 265–274.

[Tes04b] Jerry Tessendorf. “Simulating Ocean Surface”. In: *Siggraph course notes*. ACM, 2004.

[Thü+06] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. “Detail-preserving Fluid Control”. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’06. Vienna, Austria: Eurographics Association, 2006, pp. 7–12.

[TJ07] Christopher D. Twigg and Doug L. James. “Many-worlds Browsing for Control of Multibody Dynamics”. In: *ACM Trans. Graph.* 26.3 (July 2007).

[Tou+14] Maxime Tournier, Matthieu Nesme, François Faure, and Benjamin Gilles. “Seamless Adaptivity of Elastic Models”. English. In: *Graphics Interface*. Ed. by Paul G. Kry and Andrea Bunt. Canadian Information Processing Society Toronto, May 2014, pp. 17–24.

[TPS08] Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. “Asynchronous cloth simulation”. In: *Computer Graphics International* (2008).

[TRS06] Nils Thürey, Ulrich Rüde, and Marc Stamminger. “Animation of Open Water Phenomena with Coupled Shallow Water and Free Surface Simulations”. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA ’06. Vienna, Austria: Eurographics Association, 2006, pp. 157–164.

[VB05] J. Villard and H. Borouchaki. “Adaptive Meshing for Cloth Animation”. In: *Eng. with Comput.* 20.4 (Aug. 2005), pp. 333–341.

[Wan+13] Chang-Bo Wang, Qiang Zhang, Fan-Long Kong, and Hong Qin. “Hybrid Particle-grid Fluid Animation with Enhanced Details”. In: *Vis. Comput.* 29.9 (Sept. 2013), pp. 937–947.

[Wan+14] Yuting Wang, Chenfanfu Jiang, Craig Schroeder, and Joseph Teran. “An Adaptive Virtual Node Algorithm with Robust Mesh Cutting”. In: *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*. Ed. by Vladlen Koltun and Eftychios Sifakis. The Eurographics Association, 2014.

[WBK01] Andrew Witkin, David Baraff, and Michael Kass. “Physically Based Modeling”. In: *Online SIGGRAPH Course Notes.* 2001.

[WDW11] Jun Wu, Christian Dick, and Rüdiger Westermann. “Interactive High-Resolution Boundary Surfaces for Deformable Bodies with Changing Topology”. In: *Proceedings of 8th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS) 2011.* 2011, pp. 29–38.

[WDW13] Jun Wu, Christian Dick, and Rüdiger Westermann. “Efficient Collision Detection for Composite Finite Element Simulation of Cuts in Deformable Bodies”. In: *The Visual Computer (Proc. CGI 2013)* 29.6-8 (2013), pp. 739–749.

[Wic+10] Martin Wicke, Daniel Ritchie, Bryan M. Klingner, Sebastian Burke, Jonathan R. Shewchuk, and James F. O’Brien. “Dynamic Local Remeshing for Elastoplastic Simulation”. In: *ACM Trans. Graph.* 29.4 (July 2010), 49:1–49:11.

[WK88] Andrew Witkin and Michael Kass. “Spacetime Constraints”. In: *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’88. New York, NY, USA: ACM, 1988, pp. 159–168.

[WMT06] Chris Wojtan, Peter J Mucha, and Greg Turk. “Keyframe control of complex particle systems using the adjoint method”. In: *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation.* Eurographics Association. 2006, pp. 15–23.

[WT08] Chris Wojtan and Greg Turk. “Fast Viscoelastic Behavior with Thin Features”. In: *ACM Trans. Graph.* 27.3 (Aug. 2008), 47:1–47:8.

[Wu+01] 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* 20.3 (2001), pp. 349–358.

[WWD14] Jun Wu, Rüdiger Westermann, and Christian Dick. “Real-Time Haptic Cutting of High Resolution Soft Tissues”. In: *Studies in Health Technology and Informatics (Proc. Medicine Meets Virtual Reality 2014)* 196 (2014). Published by IOS Press, pp. 469–475.

[WWD15] Jun Wu, Rüdiger Westermann, and Christian Dick. “A Survey of Physically Based Simulation of Cuts in Deformable Bodies”. In: *Computer Graphics Forum* (2015).

[Yan+09] He Yan, Zhangye Wang, Jian He, Xi Chen, Changbo Wang, and Qunsheng Peng. “Real-time Fluid Simulation with Adaptive SPH”. In: *Comput. Animat. Virtual Worlds* 20.2-3 (June 2009), pp. 417–426.

[Yan+13] Ben Yang, Youquan Liu, Lihua You, and Xiaogang Jin. “Technical Section: A Unified Smoke Control Method Based on Signed Distance Field”. In: *Comput. Graph.* 37.7 (Nov. 2013), pp. 775–786.

[YCZ11] Zhi Yuan, Fan Chen, and Ye Zhao. “Pattern-guided Smoke Animation with Lagrangian Coherent Structure”. In: *ACM Trans. Graph.* 30.6 (Dec. 2011), 136:1–136:8.

[ZB05] Yongning Zhu and Robert Bridson. “Animating Sand As a Fluid”. In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 965–972.

[Zhu+13] Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. “A New Grid Structure for Domain Extension”. In: *ACM Trans. Graph.* 32.4 (July 2013), 63:1–63:12.

[Zhu+14] Bo Zhu, Ed Quigley, Matthew Cong, Justin Solomon, and Ronald Fedkiw. “Codimensional Surface Tension Flow on Simplicial Complexes”. In: *ACM Trans. Graph.* 33.4 (July 2014), 111:1–111:11.

[ZSP08] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola. “Adaptive Sampling and Rendering of Fluids on the GPU”. In: *Proceedings of the Fifth Eurographics / IEEE VGTC Conference on Point-Based Graphics*. SPBG’08. Los Angeles, CA: Eurographics Association, 2008, pp. 137–146.

List of Figures

List of Algorithms

List of Tables