Simulation and control of physical phenomena

Pierre-Luc Manteaux

April 26, 2016

Remerciements

Merci

Résumé

Dans ce document nous présentons des techniques pour la simulation efficace et le contrôle de phénomènes physiques.

Abstract

In this document we present techniques for the efficient simulation and control of physical phenomena.

Contents

Chapter 1

Introduction

An introduction.

1.1 Problems

Problems I was interested in during my PhD.

1.2 Contributions

Contributions I made during my PhD.

1.3 Structure of the document

Overview of the manuscript

1.4 Related publications

Some publications.

- [Man+13]
- [Lej+15]
- [Man+15]

Chapter 2

State of the art

A nice state of the art.

Chapter 3

ARPS

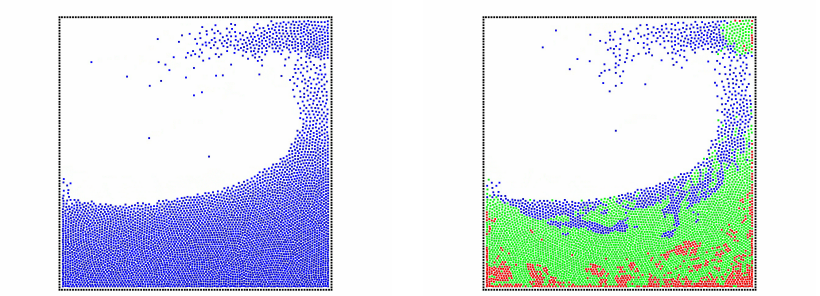


Figure 3.1: A dam break simulation with 5000 particles simulated with WCSPH (on the left) and with our adaptive method (on the right). On the right image, blue corresponds to full-dynamics particles, green to transition particles and red to restrained particles.

In this paper, we explore the use of Adaptively Restrained (AR) particles for graphics simulations. Contrary to previous methods, Adaptively Restrained Particle Simulations (ARPS) do not adapt time or space sampling, but rather switch the positional degrees of freedom of particles on and off, while letting their momenta evolve. 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.

We present the initial formulation of ARPS that was introduced for molecular dynamics simulations, and explore its potential for Computer Graphics applications: We first adapt ARPS to particle-based fluid simulations and propose an efficient incremental algorithm to update forces and scalar fields. We then introduce a new implicit integration scheme enabling to use ARPS for cloth simulation as well. Our experiments show that this new, simple strategy for adaptive simulations can provide significant speedups more easily than traditional adaptive models.

3.1 Introduction

Combining efficiency with visual realism had been one of the main goals of Computer Graphics research in the last decade. The general strategy for efficient graphical simulations is to concentrate the computational time on the most interesting parts of an animated scene (such as near the surface of a fluid), while simplifying the rest of the scene according to some visual quality criteria. A number of adaptive simulation methods, aimed at controlling the trade-off between performance and precision, have been developed. Most of them consist in changing time or space sampling, using adaptive time steps or multi-scale models. Although several of them give impressive results, they are often difficult to implement, may-be restricted to specific applications, sometimes generate discontinuity artifacts due to sudden simplifications.

A different approach for adaptive simulation [AR12] was recently proposed in the context of molecular dynamics (MD). 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. While freezing objects to gain computation time has been extensively used in video games, 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, introduces a physically sound approach with proven correctness, and has been successfully used in the context of predictive, energy- and momentum-conserving particle simulation.

We present the first applications of ARPS to physically-based animation, and we complement the approach with two novel extensions, to cope with the specificity of our domain. Damping forces, not present in the classical MD framework, create specific difficulties that we tackle using a novel freeze criterion. Additionally, we derive an implicit integration method for applying ARPS to stiff objects. The remainder of this paper is organized as follows. We first briefly review the previous work on adaptive mechanical simulations in computer graphics. We then summarize the ARPS method in Section 3.3. The question of damping is studied in Section 3.4 through viscosity forces in SPH simulations. An extension to implicit integration is presented in Section 3.5 using a cloth-like use case. Practical implementation and parameter tuning are then addressed in Section 3.6. We finally discuss results and perspectives in Section 3.7.

3.2 Previous work

There have been two main ways to address adaptivity in Computer Graphics: time adaptivity and space adaptivity. Time adaptivity has been used

to perform as large time steps as possible without compromising stability. [DC96] locally adapt the time step based on the Courant-Friedrichs-Lewy criterion [Pre+92] for early SPH simulation, and this was later extended to more recent SPH formulations [Ihm+10]. The same criterion was derived and used for deformable solids, using adaptive space sampling as well [Deb+99; Deb+01]. Time adaptivity has also been used to conservatively handle collisions [Har+09].

Space adaptivity has been first used in mass-spring systems [HPH96; GCS99] and then extended to continuous models such as FEM [Wu+01]. [Deb+01] use non-nested meshes, while [GKS02] propose to consider adaptivity from the shape functions viewpoint on a single mesh. [Sif+07] constrained T-nodes within other independent nodes. [Mar+08] solved multi-resolution junctions with polyhedral elements. [Ota+07] combine adaptivity and multigrid solution. Real-time remeshing has been applied to 1D elements such as rods and wires [Len+05; ST08; Ser+11] and to 2D surfaces like cloth [BD12], [NSO12] and paper [NPO13a]. In 3D, adaptive meshes have been used to simulate cutting [CDA00], plasticity [Bar+07; Wic+10] and thin fluid features [WT08], [ATW13]. Adaptive shape matching has been proposed using a mesh-less, octree-based approach [SOG08]. In addition, adaptive SPH fluid simulations were recently proposed [Ada+07],[SG11], [GP11], [OK12].

An interesting alternative to adaptive space sampling is adaptive deformation fields. [RGL05] dynamically create rigid clusters of articulated bodies, while [KJ09] decompose of the displacement field on a dynamically reduced set of deformation modes.

Despite decades of improvements, it seems that adaptive models are not yet mature or general enough to be used in mainstream software. Adaptivity is typically difficult to apply because it requires significant changes in the models or the equation solvers. In contrast, ARPS require comparatively small changes to the simulators and may become an interesting alternative.

3.3 Adaptively Restrained Particles

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

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

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

$$\mathcal{H}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T M^{-1}\mathbf{p} + V(\mathbf{q}) \tag{3.1}$$

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:

$$\mathcal{H}_{AR}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T \Phi(\mathbf{q},\mathbf{p})\mathbf{p} + V(\mathbf{q}) \tag{3.2}$$

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 equal to $\Phi_i(q_i,p_i) = m_i^{-1}[1-\rho_i(q_i,p_i)]\mathbf{I_{3x3}}$. 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 :

$$\rho_i(p_i) = \begin{cases} 1, & \text{if } 0 \le K_i(p_i) \le \epsilon_i^r \\ 0, & \text{if } K_i(p_i) \ge \epsilon_i^f \\ s(K_i(p_i)) \in [0,1], & \text{elsewhere} \end{cases} \tag{3.3}$$

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

Adaptive equations of motion: The adaptive equations of motions are derived from the AR Hamiltonian (3.2):

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}_{AR}}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}}$$
$$\frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}_{AR}}{\partial \mathbf{p}} = M^{-1}[I-\rho(\mathbf{p})]\mathbf{p} - \frac{1}{2}\mathbf{p}^T M^{-1}\frac{\partial \rho(\mathbf{p})}{\partial \mathbf{p}}\mathbf{p}$$

Applied to a particle, one can derive the rate of position change, which we call *effective velocity*, as:

$$\dot{q} = \frac{1}{m}\left((1-\rho(p))p - \frac{1}{2}\parallel p \parallel^2 \frac{\partial \rho(p)}{\partial p}\right) \tag{3.4}$$

While the momenta evolve as in classical Hamiltonian mechanics, position evolves 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.

A simple example: Consider a 1D harmonic oscillator : a particle attached to the origin with a perfect spring. Fig. 3.2 shows a phase portrait of the corresponding AR system. 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 big enough. These trajectories are connected by a transition corresponding to an energy between the two thersholds of eq. (3.3). 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, one can show that 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.

Computational performance: [AR12] obtained significant speedup exploiting immobility of particles. An incremental method was used to update the particles forces at each time step, while saving time on inactive particles :

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

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

The 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 speedup were 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 paper explores two applications of ARPS to graphical simulations:

1. Particle-based fluid simulation. In this case, damping forces are involved in contrast with the classic use of ARPS. We propose a method to handle them as well as an incremental algorithm to update the forces and the scalar fields.

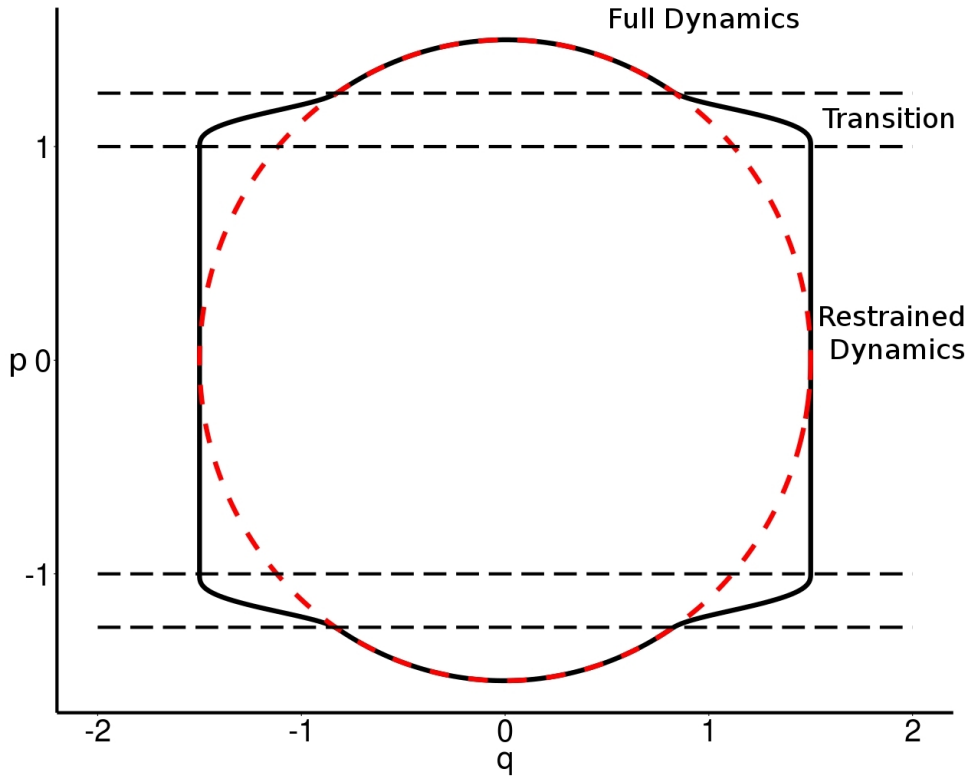


Figure 3.2: Phase portrait of a harmonic oscillator. The red dotted ellipse corresponds to standard Hamiltonian mechanics, while the solid black line corresponds to ARPS. During restrained dynamics momentum is accumulated. Then a transition deals with the accumulated energy before getting back to the full dynamics.

2. Stiff object simulation. We take the example of a cloth simulation. We will propose an implicit formulation of ARPS and a hybrid solver to exploit inactivity of particles.

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 user interaction, and the animation of large volumes of liquid, when most of it already came to rest.

3.4 Extension to SPH fluid simulation

SPH fluid simulation is widely used in computer graphics and many methods have been proposed [DC96], [MCG03], [SP09], [Ihm+13]. SPH approximates fluid dynamics with a set of particles. The particles are used to interpolate properties of the fluid anywhere in the space. Each particle samples 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. For a detailed and comprehensive introduction to SPH, you can refer to [Mon05]. To integrate ARPS, we chose WCSPH (Weakly Compressible Smoothed Particle Simulation) [BT07], a standard SPH formulation [DC96], [MCG03]. We limited our simulation to the main inter-particles forces: pressure and viscosity. Classically a SPH algorithm follows three steps:

1. Update mass density and pressure
2. Compute inter-particles forces : pressure, viscosity
3. Integrate velocities and positions

With ARPS, time can be saved on each computation step involving pairwise terms. In SPH, inter-particles forces and density field computation are the perfect candidates. As proposed in [AR12], we use an incremental algorithm to update only quantities involving active particles.

Viscosity

Viscosity forces involve particles velocities. The viscosity force of particle i with respect to particle j is :

$$f_{ij} = \begin{cases} -m_i m_j \Pi_{ij} \nabla W_{ij} & v_{ij}^T q_{ij} < 0 \\ 0 & v_{ij}^T q_{ij} \geq 0 \end{cases} \tag{3.5}$$

Π_{ij} is given as :

$$\Pi_{ij} = -\nu \left(\frac{v_{ij}^T q_{ij}}{\mid q_{ij} \mid^2 + \epsilon h^2} \right) \tag{3.6}$$

W_{ij} denotes a convolution kernel, v_{ij} the difference of velocities between the two particles, q_{ij} the difference of positions between the two particles, m_i is the mass, h is the particle smoothing radius and $\nu = \frac{2\alpha h c_s}{d_i + d_j}$ is a viscous term where α is a viscosity constant, d_i the particle i density. $\epsilon = 0.01$ is a constant to avoid singularities.

However, velocity is not explictly represented in ARPS, and can be seen in two different ways. We may define it based on the momentum and set $v_i = p_i / m_i$,

or based on the change of position $\dot{q}_i$. 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 eq.(3.4). Applied to a harmonic oscillator, this results in the behavior illustrated in Fig. 3.3. The more the particle is damped the longer it remains inactive, which is an intuitive behavior.

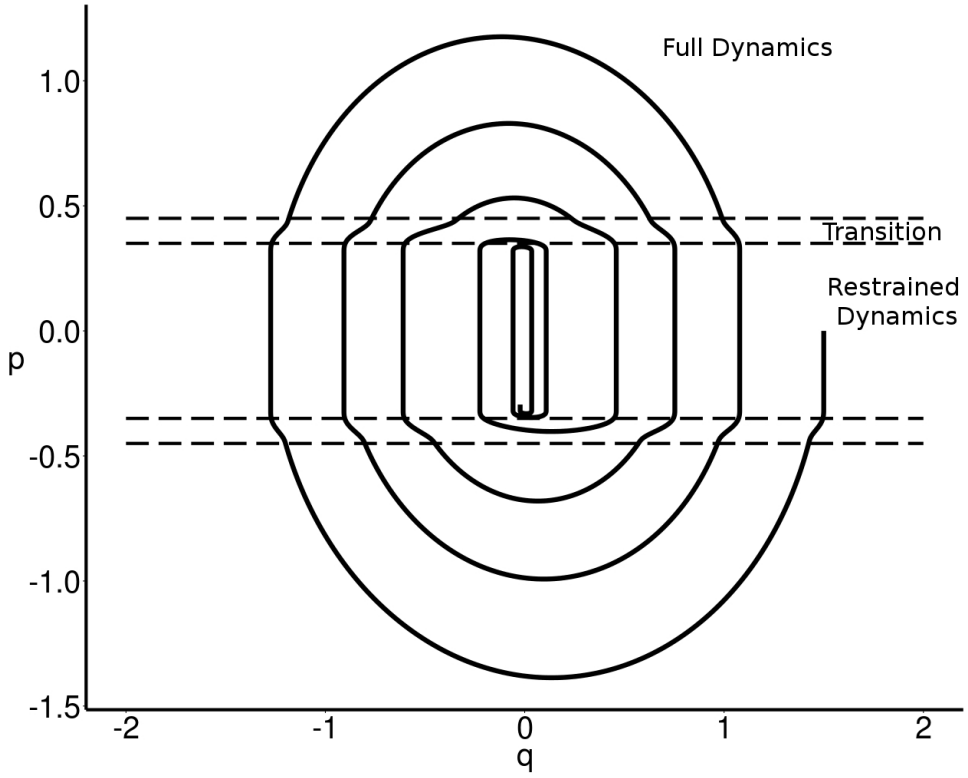


Figure 3.3: Phase portrait of our damping approach in ARPS. As with a classic damped oscillator we obtain a spiral phase portrait.

Modified inactivity criterion

Since our damping 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 damping 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.

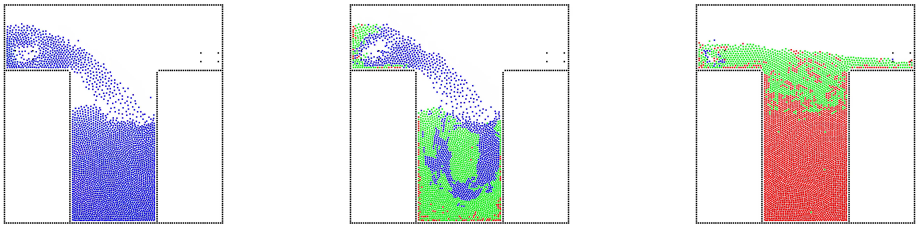
Performance

We performed two experiments to measure computation time. The first one (Table 3.1) is a fall of 5000 particles in a box. As soon as most particles come to rest, and become inactive the speedup can be significant. For 15s, the mean speedup is 3.8. The speedup can locally reach 25.7. We can see

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

Table 3.1: Fall of a block of water - Computation time and speedup {min, max, mean}

in Figure 4.1 that during speed movements most of the particles are active so that the adaptive simulation stay close to reference simulation. Therefore small scale details like splashes can be preserved.



(a) (b) (c)

Figure 3.4: A permanent flow simulation with 4240 particles. (a) is a classic WCSPH simulation. (b) is our adaptive method at the same time of (a) with restrained particles in red. (c) is our adaptive method once the permanent flow is installed.

The second experiment (see Table 3.2) is the creation of a permanent flow with 4240 particles. As we can see in Figure 3.4, once the permanent flow is installed a large amount of particles are restrained. We reach an interesting speedup while keeping a motion close to the reference.

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

Table 3.2: Fluid permanent flow - Computation time and speedup {min, max, mean}.

3.5 Extension to stiff objects: Implicit Integration

In this section we explore the application of ARPS to stiff object simulation and propose an implicit integration scheme which saves computation time for particles at rest. Implicit integration for cloth simulation was introduced in [BW98]. An introduction to implicit integration is proposed in [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 and after linearization with a linear system to solve to obtain the next momentum:

$$(I - h^2 K M^{-1})\Delta p = h(f + hKM^{-1}p) \ , \tag{3.7}$$

where $K = \dfrac{\partial f}{\partial q}$ 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.

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:

$$\begin{aligned} \Delta p &= h f(q_{n+1}, p_{n+1}) \\ \Delta q &= h \Big(M^{-1}(1 - \rho(p_{n+1}))p_{n+1} \\ &\quad -\frac{1}{2} p_{n+1}^T M^{-1} \frac{\partial \rho(p_{n+1})}{\partial p} p_{n+1} \Big) \end{aligned} \tag{3.8}$$

We perform a Taylor-Young expansion of $f(q_{n+1}, p_{n+1})$ and introduce Δq in the expended momenta equation. We then perform a Taylor-Young expansion of $\rho(p_{n+1})$ in the momentum equation, which gives us the following equation system:

$$(I - h^2 K R M^{-1})\Delta p = h(f + hKM^{-1}s) \tag{3.9}$$

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

$$\begin{aligned} R_{ii} &= I - \rho(p_n^i) - p_n^i \frac{\partial \rho(p_n^i)}{\partial p_i}^T \\ &\quad -\frac{1}{2} p_n^i p_n^{i^T} \frac{\partial^2 \rho(p_n^i)}{\partial p_i^2}^T - \frac{\partial \rho(p_n^i)}{\partial p_i} p_n^{i^T} \ , \end{aligned} \tag{3.10}$$

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

$$s_i = p_n^i - \rho(p_n^i) p_n^i - \frac{1}{2} p_n^{i^T} p_n^i \frac{\partial \rho(p_n^i)}{\partial p_i} \tag{3.11}$$

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

$$I\Delta p = hf \tag{3.12}$$

Conversely, if all particles are active then $R = I$ and $s = p$ and we get the classical implicit formulation of eq.(3.7). We loop over time using algorithm 1.

Algorithm 1 Implicit integration scheme

for each time step **do**
  compute ρ, R, s, f.
  compute $A = I - h^2 KRM^{-1}$
  compute $b = hf + h^2 KM^{-1}s$
  solve $A\Delta p = b$
  compute $p_{n+1} = p_n + \Delta p$
  compute $q_{n+1} = q_n + hM^{-1}\left(R\Delta p + s\right)$
end for

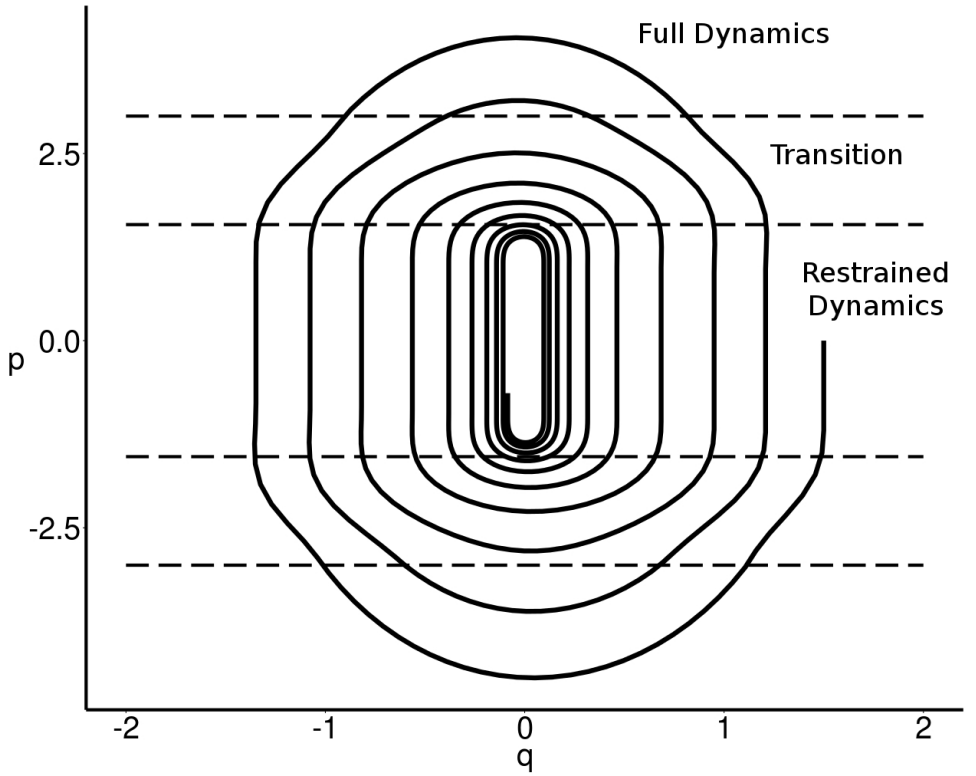


Figure 3.5: Phase portrait of a harmonic oscillator simulated using implicit ARPS.

Figure 3.5 shows the phase portrait of a harmonic oscillator simulated using our implicit formulation. As expected, the well-known numerical damping effect of implicit Euler provides us with the same behavior we could observe with a damped harmonic oscillator. To include a damping term in the physical model, we derived an implicit formulation which includes a damping term $f_d = -\gamma\dot{q}$:

$$(I + h\gamma M^{-1}R - h^2 KRM^{-1})\Delta p = h(f + hKM^{-1}s + f_d) \tag{3.13}$$

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 3.6 shows a hanging cloth with active and inactive

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

Table 3.3: Implicit solver vs Hybrid solver. Computation time and speedup {min, max, mean}.

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 3.3 shows

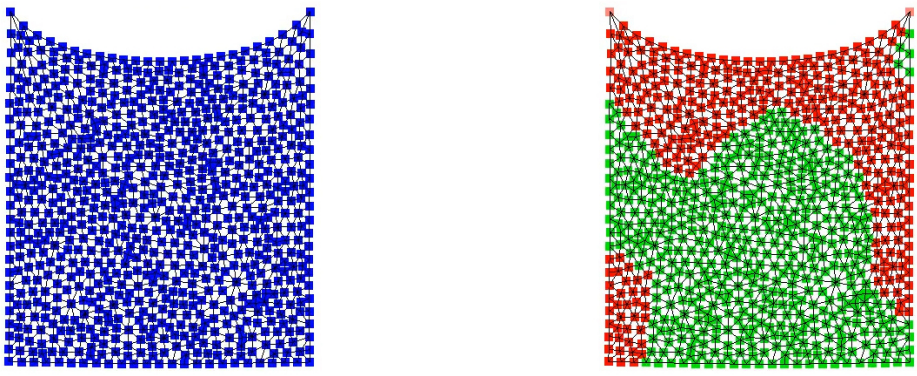


Figure 3.6: Hanging cloth. Left: traditional implicit simulation. Right: implicit ARPS simulation with a varying set of active and inactive particles.

performances we achieved with our hybrid solver. As soon as a large number of particles become inactive the simulation is explicitly integrated and interesting speedup can raise.
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.

3.6 Implementation

Parameters

ARPS use two parameters, ϵ^r and ϵ^f of Equation 3.3. 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 normal 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. Currently we manually set the parameters, and defer the automatic tuning to future work. In table 3.4 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 3.4: ARPS thresholds for SPH and Cloth simulation

Linear solver

A linear equation solver is necessary in implicit integration, as presented in Section 3.5. In contrast with most formulations, implicit ARPS generally results in an unsymmetrical equation matrix, due to the matrix products in eq.(3.9). We currently use a sparse LU solver from 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.

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. We plan to investigate this issue in future work.

3.7 Discussion and concluding remarks

We have shown that ARPS, a new, simple approach to adaptive simulation, can effectively be applied to Computer Graphics, and we have demonstrated two specific applications. The most successful one is the SPH simulation, for which we have obtained significant speedups with only minor changes to the original simulation method. In the case of stiff material, we have obtained promising results for implicit integration, and we will address stability issues in future work, starting with a careful study of the restraining function.

Another interesting avenue is to employ non-physically-based transition criteria. The current one, based on kinetic energy, is well adapted to molecular dynamics simulation. In Computer Graphics, however, we are more interested in visual results. In future work, we thus plan to investigate the tuning of the transition thresholds based on visibility or distance to the camera, to even more focus the computational power where it most contributes to the quality of the result.

Chapter 4

Frame-based cutting

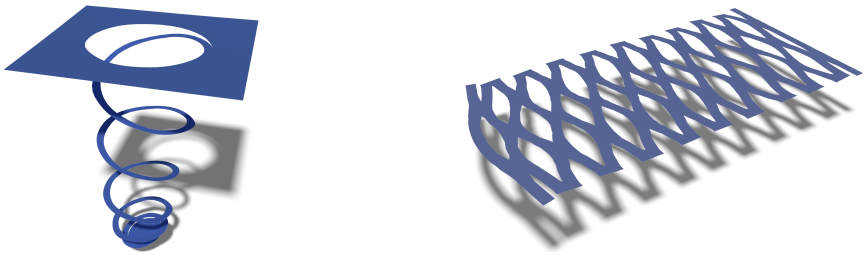
In this paper we propose a method for the interactive detailed cutting of deformable thin sheets. Our method builds on the ability of frame-based simulation to solve for dynamics using very few control frames while embedding highly detailed geometry - here an adaptive mesh that accurately represents the cut boundaries. Our solution relies on a non-manifold grid to compute shape functions that faithfully adapt to the topological changes occurring while cutting. New frames are dynamically inserted to describe new regions. We provide incremental mechanisms for updating simulation data, enabling us to achieve interactive rates. We illustrate our method with examples inspired by the traditional Kirigami artform.

4.1 Introduction

Over the last three decades, physics-based animation methods have been proposed to simulate a wide range of phenomena. Substantial progress has been achieved in terms of efficiency and realism. As a result, physics-based animation has found applications in film, games, craft, teaching, and training.

Combining interactive user actions and detailed convincing animations is crucial for user experience in simulation and games. Unfortunately, computational contraints limit the fidelity that can be achieved with physics-based animation in interactive simulations. Often, the simulated objects lack detail compared to the rest of the virtual environment. Furthermore, operations that modify the structure of the simulated objects, such as cutting, maybe 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 work, we address the issue of enabling detailed cuts at interactive



(a) (b)

Figure 4.1: Progressive cutting of a spiral using only five control frames (a). Simulating complex deformations resulting from Kirigami cutting (b).

rates in thin sheets of deformable materials. 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 samplings of the physical geometric models, is to use a mesh-less simulation method called the frame-based model [Gil+11]. 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 shaped region of the geometric mesh, only a few of them are typically required.

To achieve user-driven cuts in a frame-based simulation, we allow cuts to be performed on 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 frame node 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 (1) 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, (2) the dynamic re-sampling of new frames into disconnected parts and (3) the incremental update of the simulation data that were concerned by the cut.

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 complexity of the cuts.

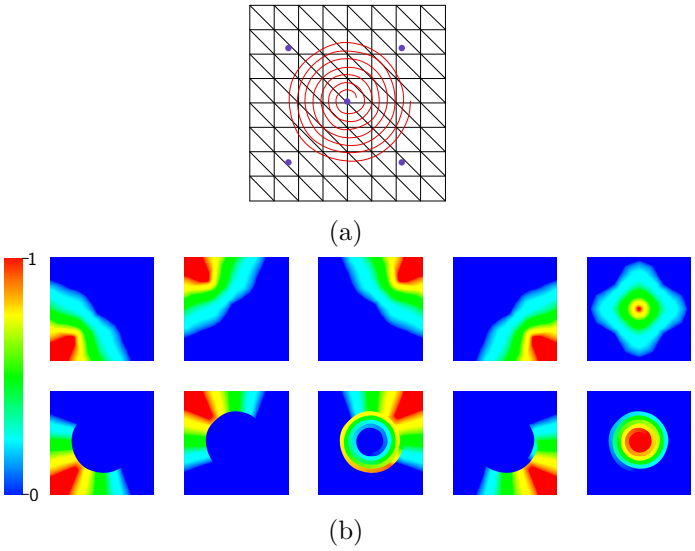


Figure 4.2: Comparison between shape functions computed on a uniform grid and on a non-manifold grid. (a) The underlying mesh (black lines) is cut by spiral (red line) and sampled with five control frames (blue circle). (b) The shape functions for each of the frame with a uniform grid (top row) and with 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.

4.2 Related Work

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 define 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.

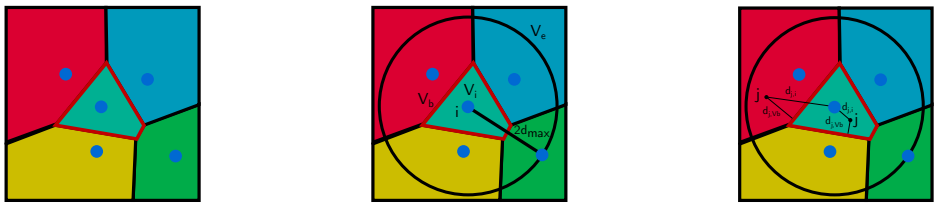
A first possibility consists in using the same model for physics simulation and visualization. Topological changes are then mostly modeled by remeshing operations. Simple and fast remeshing techniques such as element deletion or element splitting were proposed. The latter was used in the first simulation of brittle and ductile materials [OH99], [OBH02]. Methods that preserve element quality by local and global remeshing have also been developed. They recently lead 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 to robustly simulate 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. [Pau+05] proposed to use visibility criterion to perform fracture. [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 also have some 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 low basis of deformation modes in order to achieve real-time performance on detailed models. However, the low-basis is acquired after heavy precomputations. Interactive scenario could not handle the recomputation 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. For each frame, they can represent a large region of influence with complex shape. Other advantages are discussed in [Fau+11]. Unfortunately, the current frame-based method does not allow the shape functions to reflect the topological changes of the embedded mesh.



(a) (b) (c)

Figure 4.3: 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.

4.3 Overview

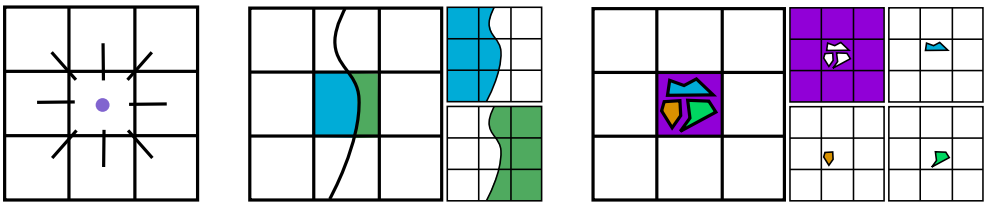
The goal of this work is to enable interactive detailed cutting of deformable thin sheets. The frame-based method exhibits some of the key features we are looking for: a very low number of nodes and a tunable separation between visual and physical models. We build on this framework and extend it to handle 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 4.2). The main idea is that cut cells are duplicated and store different connectivities. Therefore, grid resolution depends much less on the mesh topology while keeping all topological informations.

We detail the computation of the shape functions, motivate the use of a non-manifold grid and explain how to build it (Section 4.4). When several parts are disconnected, we need to make sure that they contain control frames. We present a simple method to detect those regions and re-sample them (Section 4.5). Even with such a low resolution physics model, it would be overkilling to update the whole system at each cut. Fortunately, cutting is often a local event. We leverage this fact and propose simple strategies to incrementally update the different components of the simulation (Section 4.6). We illustrate our method in different scenarios (Section 4.7) and discuss limitations and future work (Section 4.8). We summarize our simulation loop in Algorithm 2 and detail our remeshing algorithm in appendix 4.9.

Algorithm 2 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



(a) (b) (c)

Figure 4.4: 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.

4.4 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.

Voronoi shape function

Let $w_i(x) : \Omega \to \mathcal{R}$ 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 3 and in Figure 4.3.

In practice, 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 position.

Algorithm 3 Shapefunction computation

procedure COMPUTE_SHAPEFUNCTION
  for each frame i **do**
    $V_i \leftarrow$ Voronoi region of i
    $V_b \leftarrow$ boundary of V_i
    $d_{max} \leftarrow$ maximum distance to V_i boundary
    $V_e \leftarrow$ extend V_i to $2.0 \times d_{max}$
    ▷ dist(A,B) is the geodesic distance between A and B
    for each grid cell j in V_e **do**
      if j is inside V_i **then**
        $$w_i(j) = 0.5\left(1 + \frac{dist(j, V_b)}{dist(j, V_b) + dist(j, i)}\right)$$
      else if j is inside V_e **then**
        $$w_i(j) = 0.5\left(1 - \frac{dist(j, V_b)}{dist(j, i) - dist(j, V_b)}\right)$$
      end if
    end for
  end for
end procedure

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.

Non-manifold grid

As mentionned above, in [Fau+11], shape functions are computed on a uniform grid using Dijktra'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 grid 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 4.4).

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.

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 4 and illustrate the main steps in Figure 4.5.

4.5 Frame re-sampling

As long as no parts of the model are disconnected, our method allows to keep 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.

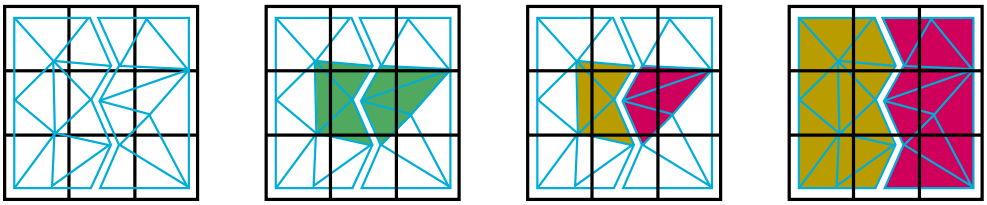
We start by detecting empty regions i.e lists of connected cells that are not influenced by any frame. This is done using a flood fill algorithm on the grid containing the shape functions values. These empty regions are then sampled using a farthest sampling algorithm. Finally, the samples are uniformly distributed by applying several Lloyd relaxation steps. For now,

Algorithm 4 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 ← 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 ← 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
```

the number of frames which are sampled is user-defined but we would like to investigate for setting it automatically (see Section 4.8).

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 behaviours. For rigid materials this is not a problem but for soft material this can quickly become unrealistic. We propose a simple strategy to



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

Figure 4.5: 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.

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 to detect 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 artefacts 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.

4.6 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 to separate

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.

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 4.5. We detect the disconnected parts using a flood fill algorithm and uniformly re-sample them.

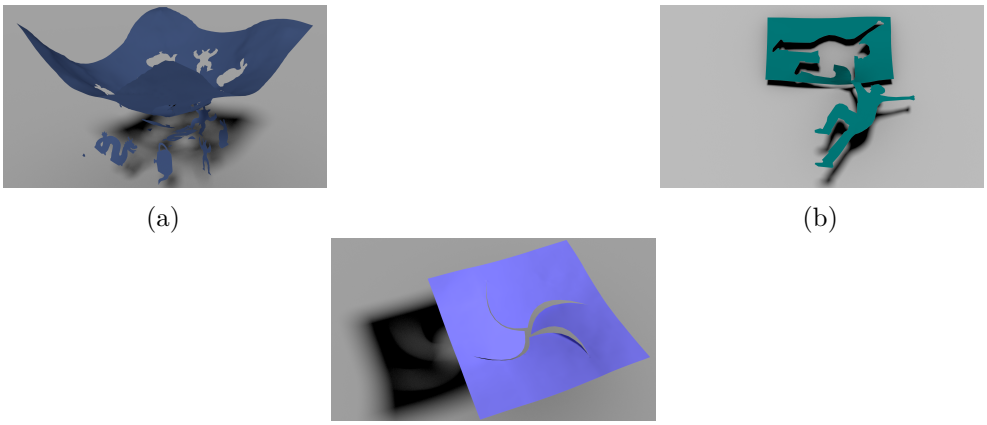
Integration point update

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.

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.



(a) (b)

(c)

Figure 4.6: (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.

Name	Grid Size	#frame		#vertices		#collision	#integration	Before Cuttin
		Initial	Final	Initial	Final			
Spiral (Fig. 4.1a)	40×40	5	5	81	2111	200	200	60
Kirigami (Fig. 4.1b)	68×68	47	47	4225	7453	600	800	11.3
Patchwork (Fig. 4.6a)	50×50	5	12	4225	8253	200	200	60
Vortex (Fig. 4.6c)	68×68	5	5	4225	4889	200	200	45
FallingGuy (Fig. 4.6b)	100×50	10	10	289	861	500	500	60

Table 4.1: Resolution of the different components of the simulation and timings.

4.7 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 (see Table 4.1). Frame

	Percentage of update for a cutting step				
Name	%grid cell	%shape function cell	%vertices	%collision nodes	%integra
Spiral (Fig. 4.2a)	0.07	28.1	61.5	27.2	
Kirigami (Fig. 4.1b)	1.06	15.9	17.8	15.8	
Patchwork (Fig. 4.6a)	0.02	2.78	4.33	2.55	
Vortex (Fig. 4.6c)	0.08	10.3	12.8	9.2	
FallingGuy (Fig. 4.6b)	0.09	5.84	11.4	5.77	

Table 4.2: 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.

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.

All our examples run at interactive frame rate during the whole simulation (see Table 4.1). Frame rates were collected on a twelve-core 3.20 GHz Intel Xeon CPU with 15.6 GB RAM. We highlight that our implementation is not optimized and there are still a lot of room for improvements that we did not have time to implement. Full animated results are shown in the accompanying video.

Figure 4.1a 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 4.2.

To illustrate that our method can handle multiple cuts and still simulate complex deformations, the creation of a Kirigami is shown in Figure 4.1b. 48 cuts are performed and it only required 47 frames and 400 integration points to produce a plausible behavior.

Detailed cuts can be performed and separated components can be handled as shown in Figure 4.6a. 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 4.6c.

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 to save numerous unnecessary computations as shown in Table 4.2.

4.8 Discussion

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 pre-computations. Of course, our work is not without limitations and and we see interesting directions for future work.

First, as very few frames are used, one cut may generate large changes in the weight distribution and produce popping artefacts that cannot be avoided using our interpolation strategy. This is particularly noticeable when simulating soft materials and can be seen in some of the examples of our accompanying video. Strategies proposed by [NPO13b] 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 produces well known artefacts 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.

Finally, 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.

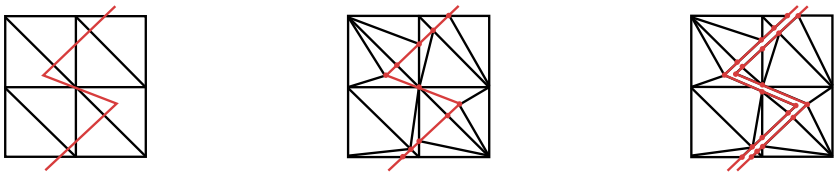
In future, we first plan to extend our work to 3D. The implementation of our current non-manifold grid would require a tetrahedron representation of the object. We would like to investigate the method of [RK13] to build this structure only from the object surface. We think that the frame-based framework 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]. Finally, 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.

4.9 Remeshing

As stated previously, 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 5 and illustrated in Figure 4.7. The remeshing part uses vertex insertion and edge split operations (see Figure 4.8). The splitting part only uses vertex split operation (see Figure 4.9).

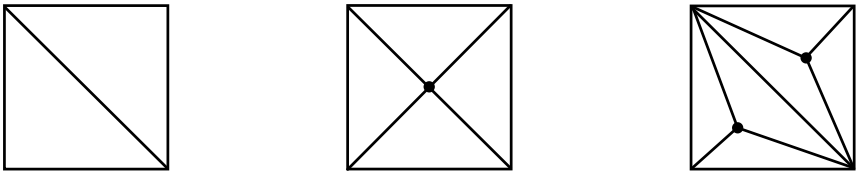
Algorithm 5 Remeshing Algorithm

```
1: procedure CUT_ALONG_SEGMENT(Segment S, mesh M)
2:     INSERT_SEGMENT(S, M)
3:     P̃ ← edges corresponding to S
4:     SPLIT_ALONG_POLYLINE(P̃,M)
5: end procedure
6:
7: procedure INSERT_SEGMENT(Segment S, Mesh M)
8:     S̃ ← subdivide S at intersection with M edges
9:     for each point i of S̃ do
10:        E ← closest edge to i
11:        V ← closest vertex to i
12:        F ← closest triangle to i
13:        if distance(E,i)< ε_edge then
14:            Split E at i
15:        else if distance(V,i)< ε_vertex then
16:            Snap i to V
17:        else
18:            Split F at i
19:        end if
20:    end for
21: end procedure
22:
23: procedure SPLIT_ALONG_POLYLINE(Polyline P, Mesh M)
24:    for each vertex V of P do
25:        Split triangles arround V according to P
26:    end for
27: end procedure
```



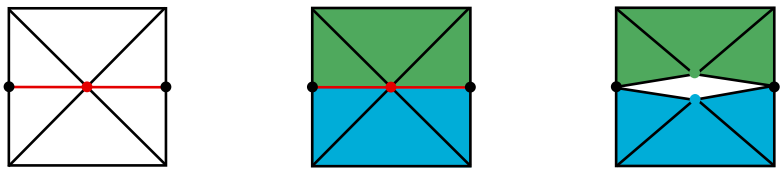
(a) (b) (c)

Figure 4.7: 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.



(a) (b) (c)

Figure 4.8: Illustrations for edge splitting and vertex insertions. (a) The input mesh. (b) After edge splitting. (c) After vertex insertions.



(a) (b) (c)

Figure 4.9: Illustrations for the vertex splitting operation. (a) A mesh which is conform with the polyline (in red). (b) We start by assigning each triangle arround 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.

Chapter 5

Fluid sculpting

Interactive fluid sculpting.

Chapter 6

Storyboarding physical animations

Semantic for physics-based animation.

Chapter 7

Conclusion

A conclusion

Bibliography

[Ada+07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas. "Adaptively sampled particle fluids". In: *ACM Trans. Graph.* 26.3 (July 2007).

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

[ATW13] Ryoichi Ando, Nils Thürey, and Chris Wojtan. "Highly Adaptive Liquid Simulations on Tetrahedral Meshes". In: *ACM Trans. Graph. (Proc. SIGGRAPH 2013)* (July 2013).

[Bar+07] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. "A finite element method for animating large viscoplastic flow". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Vol. 26. 2007.

[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.

[BJ05] Jernej Barbič and Doug L. James. "Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 982–990.

[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.

[CDA00] S. Cotin, H. Delingette, and N. Ayache. "A Hybrid Elastic Model allowing Real-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation". In: *The Visual Computer*. Vol. 16. 2000.

[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.

[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.

[Deb+01] G. Debunne, M. Desbrun, M-P. Cani, and A. H. Barr. "Dynamic Real-Time Deformations using Space and Time Adaptive Sampling". In: *Proc. ACM SIGGRAPH*. 2001.

[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. Milan, Italie: Springer, Sept. 1999, pp. 133–144.

[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.

[GCS99] F. Ganovelli, P. Cignoni, and R. Scopigno. "Introducing multiresolution representation in deformable object modeling". In: *ACM Spring Conference on Computer Graphics*. 1999.

[Gil+11] Benjamin Gilles, Guillaume Bousquet, François Faure, and Dinesh Pai. "Frame-based Elastic Models". In: *ACM Transactions on Graphics* 30.2 (Apr. 2011), Article No. 15.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. "CHARMS: a simple framework for adaptive simulation". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Vol. 21. 2002.

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

[Har+09] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. "Asynchronous Contact Mechanics". In: *SIGGRAPH (ACM Transactions on Graphics)* 28.3 (Aug. 2009).

[HPH96] D. Hutchinson, M. Preston, and T. Hewitt. "Adaptive refinement for mass/spring simulations". In: *Eurographics Workshop on Computer Animation and Simulation*. 1996, pp. 31–45.

[Ihm+10] Markus Ihmsen, Nadir Akinci, Marc Gissler, and Matthias Teschner. "Boundary handling and adaptive time-stepping for PCISPH". In: *Proceedings of Vriphys*. 2010.

[Ihm+13] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. "Implicit Incompressible SPH". In: *IEEE Transactions on Visualization and Computer Graphics* 99.PrePrints (2013), p. 1.

[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.

[KJ09] Theodore Kim and Doug L. James. "Skipping steps in deformable simulation with online model reduction". In: *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*. Vol. 28. 2009.

[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.

[Len+05] Julien Lenoir, Laurent Grisoni, Christophe Chaillou, and Philippe Meseure. "Adaptive resolution of 1D mechanical B-spline". In: *Proc. ACM GRAPHITE*. 2005.

[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.

[Mar+08] Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. "Polyhedral finite elements using harmonic basis functions". In: *Proc. Eurographics Symposium on Geometry Processing*. 2008.

[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.

[Mon05] J J Monaghan. "Smoothed particle hydrodynamics". In: *Reports on Progress in Physics* 68.8 (Aug. 2005), pp. 1703–1759.

[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.

[NPO13a] Rahul Narain, Tobias Pfaff, and James F. O'Brien. "Folding and Crumpling Adaptive Sheets". In: *ACM Transactions on Graphics* 32.4 (July 2013). Proceedings of ACM SIGGRAPH 2013, Anaheim, xxx:1–8.

[NPO13b] Rahul Narain, Tobias Pfaff, and James F. O'Brien. "Folding and Crumpling Adaptive Sheets". In: *ACM Transactions on Graphics* 32.4 (July 2013). Proceedings of ACM SIGGRAPH 2013, Anaheim, 51:1–8.

[NSO12] Rahul Narain, Armin Samii, and James F. O'Brien. "Adaptive Anisotropic Remeshing for Cloth Simulation". In: *ACM Transactions on Graphics* 31.6 (Nov. 2012). Proceedings of ACM SIGGRAPH Asia 2012, Singapore, 147:1–10.

[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.

[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: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer animation*. 2007.

[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.

[Pfa+14] Tobias Pfaff, Rahul Narain, Juan Miguel de Joya, and James F. O'Brien. "Adaptive Tearing and Cracking of Thin Sheets". In: *ACM Transactions on Graphics* 33.4 (July 2014). To be presented at SIGGRAPH 2014, Vancouver, xx:1–9.

[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.

[Pre+92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing*. New York, NY, USA: Cambridge University Press, 1992.

[RGL05] Stephane Redon, Nico Galoppo, and Ming C. Lin. "Adaptive dynamics of articulated bodies". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Vol. 24. 2005.

[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.

[Ser+11] M. Servin, C. Lacoursière, F. Nordfelth, and K. Bodin. "Hybrid, Multiresolution Wires with Massless Frictional Contacts". In: *IEEE Transactions on Visualization and Computer Graphics*. Vol. 17. 2011.

[SG11] Barbara Solenthaler and Markus Gross. "Two-scale particle simulation". In: *ACM Trans. Graph.* 30.4 (July 2011), 81:1–81:8.

[Sif+07] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. "Hybrid Simulation of Deformable Solids". In: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2007.

[SOG08] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. "Fast adaptive shape matching deformations". In: *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2008.

[SOG09] Denis Steinemann, Miguel A. Otaduy, and Markus Gross. “Splitting Meshless Deforming Objects with Explicit Surface Tracking”. In: *Graph. Models* 71.6 (Nov. 2009), pp. 209–220.

[SP09] B. Solenthaler and R. Pajarola. “Predictive-corrective incompressible SPH”. In: *ACM SIGGRAPH 2009 papers*. SIGGRAPH ’09. New Orleans, Louisiana: ACM, 2009, 40:1–40:6.

[ST08] Jonas Spillmann and Matthias Teschner. “An Adaptive Contact Model for the Robust Simulation of Knots”. In: *Computer Graphics Forum (Proc. Eurographics)*. Vol. 27. 2008.

[Tou+14] Maxime Tournier, Matthieu Nesme, François Faure, and Benjamin Gilles. “Velocity-based Adaptivity of Deformable Models”. In: *Computers and Graphics* 45 (Dec. 2014), pp. 75–85.

[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.

[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 Transactions on Graphics (Proc. SIGGRAPH)*. Vol. 29. 2010.

[WT08] Chris Wojtan and Greg Turk. “Fast viscoelastic behavior with thin features”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH)*. Vol. 27. 2008.

[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 (Proc. Eurographics)*. Vol. 20. 2001.

[WWD15] Jun Wu, Rüdiger Westermann, and Christian Dick. “A Survey of Physically Based Simulation of Cuts in Deformable Bodies”. In: *Computer Graphics Forum* (2015).

List of Figures

List of Algorithms

List of Tables