Adaptive Methods for Elastic Deformation

Alexandre Mercier-Aubin, School of Computer Science
McGill University, Montreal
December, 2024

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Doctor of Philosophy (Ph.D.) in Computer Science

Table of Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ARPS Adaptively restrained particle system.

BFS Breadth first search.

CG Conjugate Gradient.

CPU Central processing unit.

CS Curtis Scaling.

CUDA Compute unified device architecture.

DOF Degree of freedom.

FEM Finite element method.

GPU Graphics processing unit.

JS Jacobi Scaling.

KKT Karush–Kuhn–Tucker.

PBD Position-based dynamics.

PCG Preconditioned conjugate gradient.

PGS Projected Gauss-Seidel.

SCA ACM SIGGRAPH / Eurographics Symposium on Computer Animation.

SIGGRAPH Special Interest Group on Computer Graphics and Interactive Techniques.

StVK Saint-Venant Kirchoff.

XPBD Extended position-based dynamics.

Abstract

We present methods for the efficient simulation of various types of deformable bodies by using non-deformation as a measure for model reduction. Our algorithm identifies non-deforming elements as those with low strain rates over multiple frames. We use adaptive rigidification as a tool to create approximation methods for the simulation of deformable bodies. We first use the method in the context of soft bodies, yielding simulations often orders of magnitude faster than the elastic simulations. Moreover, we present an oracle that allows rigidified elements to become elastic again as needed. Then, we adapt our method to thin shell models and tackle their respective challenges. Namely, we present how to handle rigidifying bending elements and an edge filter to improve the elastification oracle on contacts that cause bending deformation. Additionally, we measure the impact of numerical scaling on the conditioning of our system. We also present a fundamentally different view of rigidification, where we generate a sequence of resolutions with rigid patterns for an iterative multi-layer method in constraint-based approaches. In this last contribution, we aim to find the ground truth solution through a fast solver, rather than only generating visually consistent simulations. We demonstrate our results on various examples in 2D or 3D. We also dive into the implementation of the algorithms.

Abrégé

Nous présentons des méthodes pour la simulation efficace des corps déformables, en utilisant la non-déformation comme critère de réduction. Les éléments non déformables sont identifiés comme ceux ayant des taux de tension faibles sur plusieurs pas dans le temps. Nous utilisons la rigidification adaptative comme un outil pour créer des méthodes d'approximation pour la simulation d'objets déformables, ce qui produit des simulations souvent plusieurs ordres de magnitude plus rapides que les simulations élastiques. Nous présentons aussi un oracle qui permet aux éléments rigidifiés de redevenir élastiques lorsque nécessaire. Ensuite, nous adaptons notre méthode aux modèles de maillages triangulaires minces et relevons leurs défis respectifs. Plus précisément, nous présentons comment gérer la rigidification des éléments lors de la flexion et un filtre d'arc pour améliorer l'oracle d'élastification sur les contacts qui provoquent une déformation par flexion. De plus, nous mesurons l'impact de l'échelle numérique sur le conditionnement de notre système. Nous présentons également une approche fondamentalement différente de la rigidification, où nous générons une séquence de résolutions avec des motifs rigides pour une méthode itérative multigrille dans des approches de résolution de systèmes sous contraintes. Dans cette dernière contribution, nous visons à trouver la solution de référence grâce à un solveur rapide, plutôt que de simplement générer des simulations visuellement cohérentes. Nous démontrons nos résultats sur une variété d'exemples en 2D ou 3D et explorons la mise en oeuvre des différentes parties de la thèse.

Acknowledgements

This research was funded by the FRQNT Doctoral scholarship 332127. We are grateful to the Fonds de recherche du Québec for their resources. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) via the Discovery grant program and an Alliance grant ALLRP-570702-21 with Symgery. We also thank Hydro-Québec for their admission scholarships that supported the early stage funding of this thesis. We likewise show gratitude towards the reviewers and collaborators for their valuable insights throughout these research projects.

Aside from funding opportunities, I personally want to thank my thesis supervisor Paul G. Kry who acted as an exceptional mentor throughout this whole journey. I also thank David I.W. Levin for his counselling for the original adaptive rigidification project and Alexander Winter for his involvement in the early prototypes. I am also grateful for the welcoming students in the rest of the lab that I now regard not only as coworkers, but as friends. I thank everyone who indirectly contributed to the thesis through insightful comments and recommendations, such as the reviewers for the many papers contained in this work, the thesis committee, and the defence committee. Looking forward to the future, I hope to generate more opportunities to work and collaborate with everyone that had a critical role in my success.

Chapter 1

Introduction

In any standard elastodynamic simulation, interactions between models are governed by physical laws for realistic simulations. Time integration often requires expensive iterative solvers that terminate before convergence or introduce significant numerical damping. Scaling large simulations with many degrees of freedom (DOFs) is typically a challenging task. Many approximations are set a priori, such as discretizing time according to a fixed step size and discretizing models into elements. The process of creating well-shaped elements is often time consuming. However, precomputing the initial discretization of models can help alleviate some of the computational stress and adaptive techniques can then refine or coarsen the objects as needed. As the number of vertices in a scene grows, reducing the size of the system becomes increasingly important for achieving efficient simulations.

A well-known approach for quasi-static objects is to simulate them as rigid bodies [54], effectively reducing the numbers of DOFs from three times the number of vertices to exactly six DOFs: three for linear velocity and three for angular velocity of the entire rigid body. This drastic reduction allows for faster computations and more efficient simulations, making rigid body dynamics particularly suitable for applications where deformation is negligible or unnecessary. We see an opportunity to extend the use of rigid bodies to other types of bodies that allow more deformation. Instead of treating the entire body as fully deformable or fully rigid, our method introduces an adaptive approach by monitoring deformation and letting collections of elements become rigid as needed. In regions where the deformation is minimal or falls below a threshold, soft elements can be simulated as rigid, reducing the computational overhead. The challenge lies in identifying these regions in real-time, determining appropriate thresholds for rigidification,

and ensuring smooth transitions between rigid and elastic elements to avoid numerical artifacts or stability issues.

The goal of this thesis is to use non-deformation as an indicator for model reduction. With our methods for adaptive reduction, we aim to achieve fast and accurate simulation of deformables. This includes not only elastic solids, but also thin shells, and potentially other deformables such as dirt or snow in future contributions. With our methods for on demand elastification, physics simulation of solid geometry becomes appropriately input-sensitive; the rigid and deformable modelling decision becomes a function of physical parameters and environmental interactions rather than a guess made prior to runtime. Through our methods for on-demand elastification, the decision to model parts of the geometry as rigid or deformable is no longer a static choice made at the beginning of a simulation. By transitioning between rigid and elastic states based on active deformation metrics, our method optimizes computational resources and ensures accurate physical representation, significantly improving performance and scalability while preserving visual consistency. This is the main goal of this thesis, we strive to reach optimal efficiency for the simulation, while preserving faithfulness to the ground truth simulation.

This thesis brings together novel methods for the simulation of deformable bodies. Each chapter features a contribution peer-reviewed and published in either conference proceedings or scientific journals. All methods show speedups with respect to the ground truth simulation by approximating motions or using approximations to improve convergence. In the published work, we provide algorithms for visually consistent and fast simulations for deformable bodies.

The contributions of this thesis are threefold. First, we investigate the adaptive rigidification of elastic solids, focusing on how selective rigidification can enhance the performance of physics-based simulations while maintaining accuracy. Second, we extend adaptive rigidification to discrete shell structures, addressing the unique challenges posed by materials such as cloth and membranes. Using a contact filter, we improve the detection of elastic deformation for rigidified bodies on first contact. Finally, we introduce a multi-layer solver for XPBD that uses hierarchies of rigid patterns to improve convergence of the solvers instead of generating fast approximations of the scenes.

The research presented in this thesis was done by Alexandre Mercier-Aubin for the purpose of this thesis under the supervision of Professor Paul G. Kry. Additionally, Professor David I.W. Levin provided valuable feedback and guidance for the original adaptive rigidification work. Likewise, Alexander Winter, an undergraduate student, contributed to the early adaptive rigidification prototype's implementation for the Chapter 3 based on the first paper: A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124. Chapter 4 is based on a second paper: A. Mercier-Aubin and P. G. Kry. Adaptive Rigidification of Discrete Shells. *Proc. ACM Comput. Graph. Interact. Tech.*, 6(3):1–17, Aug. 2023. DOI: 10.1145/3606932. In this contribution, Paul G. Kry acted as an astounding mentor throughout the whole process of design and publication of the work. He helped to improve the ideas and provided coaching on how to better communicate them in a comprehensive way. The same roles and distribution of contributions were taken by the authors for the Chapter 5 based on the third paper: A. Mercier-Aubin and P. G. Kry. A Multi-layer Solver for XPBD. *Computer Graphics Forum*, 43(8), 2024. DOI: 10.1111/cgf.15186.

The thesis is structured as follows. In Chapter 2, we provide a detailed review of the literature on physics-based simulation methods. In Chapter 3, we introduce our method for adaptive rigidification of elastic solids and evaluate its performance in various simulation contexts. Chapter 4 extends this method to discrete shells, highlighting the unique challenges and solutions involved in simulating thin, flexible structures. Chapter 5 presents our multi-layer solver for XPBD, detailing its implementation and comparing its performance to traditional XPBD methods. Finally, in Chapter 6, we summarize the key findings of this work and discuss future directions for research in physics-based simulation.

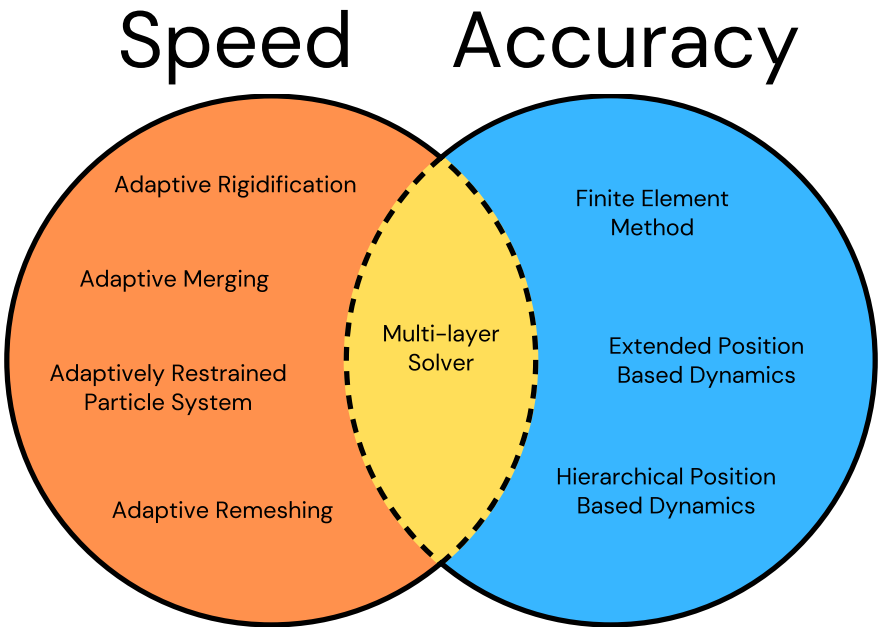


Figure 1.1: Our first two contribution focus on improving speed at the cost of some accuracy much like remeshing or freezing methods. Our latest contribution lives at the edge of both speed and accuracy, not allowing performance as fast as rigidification, but preserving the accuracy of the underlying simulation method.

To illustrate the scope of each of our contributions in relation to other research in the field, Figure 1.1 features a Venn diagram comparing the focus of each contribution to the work of others. In this thesis, we present evidence to support that adaptive rigidification methods hold great promise, particularly for simulations involving localized deformations. Our assertion is substantiated by our own published results. Furthermore, our methods can be used to encompass a broader range of models, such as the ones featured in a surgery simulator where deformation primarily occurs in the regions near the operation.

Chapter 2

Background

In the realm of physics-based animation, objects follow the fundamental principles of physics to achieve lifelike movements and deformations. To create believable animations, we must begin by formulating the equations of motion for our materials, which are derived from their unique properties such as infinitesimal energy formulations.

These formulations can encompass a wide variety of objects. For instance, both a compressed spring and a squashed cube exhibit valid deformed shapes, though the specific calculations for their internal stresses and strains will naturally differ. Similarly, a rigid body does not need to include elastic deformation to produce valid motions, saving computation time by only using a center of mass position and body rotation. What unifies all these formulations is the underlying relationship between energy, deformation, and forces, as described by Lagrangian mechanics [40].

This thesis introduces a series of advancements in adaptive methods, focusing on improving computational efficiency while maintaining accuracy of physical systems. The first major contribution is the development of adaptive rigidification for elastic solids, which reduces computational costs by treating low-strain regions as rigid while dynamically preserving accuracy in actively deforming areas. The method is extended to discrete shells, addressing the unique challenges of triangular meshes by introducing a discrete-curvature-based oracle for transitioning between rigid and elastic states. Finally, the thesis proposes a novel multigrid-like solver that eliminates the need for an external oracle, instead using hierarchical layers to progressively refine simulations from coarse, rigid approximations to fully elastic solutions.

2.1 Elastic Solids

In this context, we present a particular representation of elastic solids, recognizing that alternative models exist. While Lagrangian mechanics are just one among several physical models available, it is widely adopted for its capacity to generate highly accurate simulations that faithfully replicate a broad spectrum of real-world interactions.

2.1.1 Finite Element Method

The finite element method (FEM) is a numerical method to solve partial differential equations, typically found in complex engineering and mathematical problems [21]. FEM finds applications in various fields such as structural analysis, heat transfer, fluid dynamics, electromagnetism, and physics-based animation.

When faced with a challenging problem, employing divide-and-conquer strategies is often wise. To apply the FEM, the first step involves dividing the complex problem domain into smaller interconnected subdomains or elements. Within each element, an approximation function, often called a shape function or basis function, is used to represent the solution of the problem. These approximation functions are chosen to be simple and well-behaved mathematical functions based on the element type and desired accuracy level. In the case of triangles and tetrahedra, these shape functions take on a piecewise linear form, while quadrilaterals are described by bilinear functions, and hexahedra exhibit trilinear functions [106].

The governing equations are transformed into a set of algebraic equations for the entire domain by combining contributions from individual elements. The resulting system of algebraic equations is typically solved numerically, using techniques like direct solvers, iterative methods, or matrix factorization methods. It is often the case in physics-based animation that problems become nonlinear. Nonlinear problems may require iterative techniques and convergence criteria to stop in a reasonable time.

Once the system of equations is solved, the approximate solution is obtained. Post-processing involves extracting useful information from the solution, like displacements,

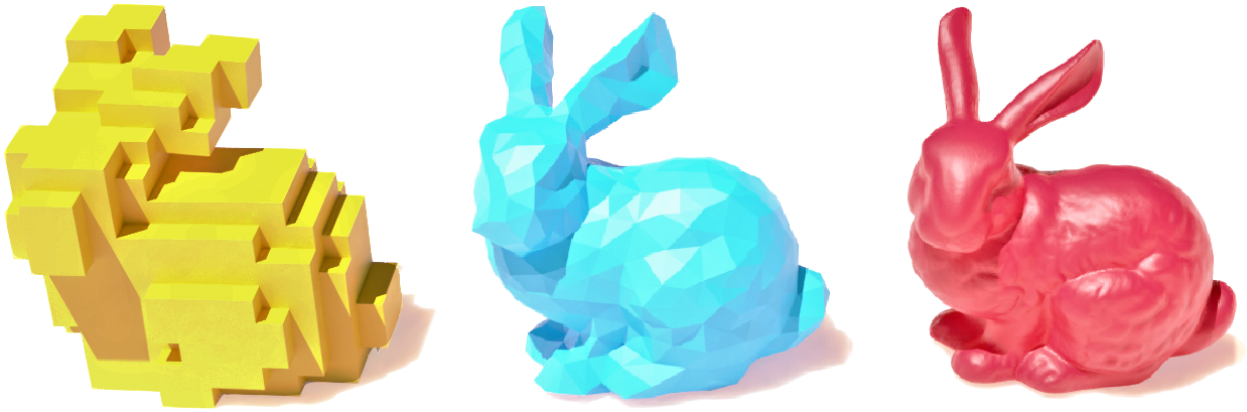


Figure 2.1: Voxelized (yellow), tetrahedralized (blue), and surface (red) meshes.

stresses, temperatures, or other relevant quantities, often represented graphically using visualization tools such as Blender [27].

The finite element method offers versatility, adaptability, and tunable accuracy, making it applicable to a wide range of engineering problems. The accuracy and efficiency of FEM solutions depend on factors like the choice of elements, mesh refinement, and the numerical methods.

2.1.2 Discretizing and Embedding

To enhance the performance of a simulator, a reasonable approach is to limit the size of the inputs. Embedding consists of displaying a high resolution (fine) triangular mesh by interpolating positions from low-resolution (coarse) tetrahedral or hexahedral elements. As stable time integration is expensive, we use the coarse cage to compute the motions of the models. We encode the mapping $\boldsymbol{x}' = W\boldsymbol{x}$ from the reduced coordinates $\boldsymbol{x}$ (coarse mesh vertices) to the surface vertices $\boldsymbol{x}'$ in a weight matrix W which is computed at rest pose like Jacobson et al. [51]. Subsequently, we carry out collision detection on the fine display mesh, and interpolate the forces back into the cage.

There are two main approaches to discretize and embed a model: tetrahedralization and voxelization, both of which will be briefly elucidated. In Figure 2.1, we illustrate the various types of meshes that we present in this section.

Tetrahedralization This method entails the creation of a tetrahedral mesh (cage) that surrounds the surface's triangular mesh. Artists create the cage manually or through a tetrahedralization software. The software generates a volumetric tetrahedral cage resembling the surface mesh by inserting new vertices within the surface mesh [49]. These points serve as the vertices of tetrahedra in the 3D mesh. The exact placement of these points depends on the specific tetrahedralization algorithm used. In conforming meshes, the first layer of tetrahedra will have faces exactly matching the surface mesh, but for typical application the surface mesh need not perfectly match the cage.

Tetrahedral elements are connected to their neighbouring tetrahedra through shared faces and edges. Establishing these connectivity relationships is crucial for performing computations in 3D simulations, such as finite element analysis. We use such connectivity for our adaptive algorithms later in the thesis. Likewise, we use tetrahedralization to generate the models for our various examples.

Depending on the application, additional steps may be taken to improve the quality of the tetrahedral mesh. Quality improvement aims to ensure that the tetrahedra are well shaped and do not exhibit extreme aspect ratios or volume distortions. Techniques like mesh smoothing or optimization algorithms may be applied for this purpose. For every vertex of the display mesh, we compute the barycentric coordinates with respect to the nearest tetrahedron in the surface mesh. Time integration deforms the cage, and vertices on the surface mesh move to preserve barycentric coordinates. It is worth noting that this approach can lead to visual artefacts, especially along the boundaries of each element in the cage, particularly when employing coarse resolutions.

Voxelization The process of voxelization on a mesh is to create a grid-shaped cage made out of hexahedra called voxels, which stands for volume elements. The hexahedra are usually of equal sizes and cover the full mesh. We create the cage from the rest pose of the display mesh to fit it as tightly as possible [88]. Compared to tetrahedralization, it only requires an argument to select the number of subdivisions. For every point of the display mesh, we compute the barycentric coordinates of its vertices within the nearest

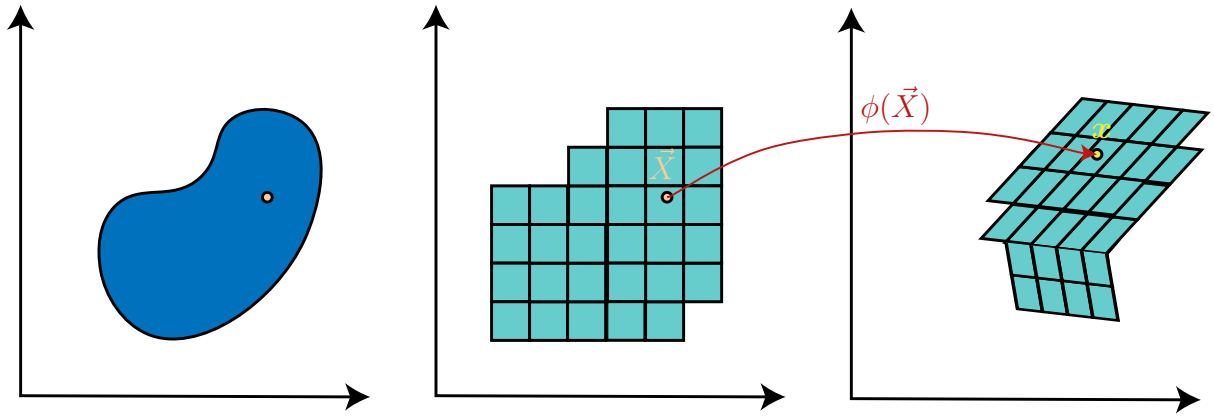


Figure 2.2: A 2D continuous model is discretized and embedded in squares, then deformed, and interpolated within the elements.

hexahedron in the cage. We illustrate this process of discretization and deformation in Figure 2.2.

Tetrahedralization can create a more accurate and potentially smoother representation of the object's surface that allows conforming meshes unlike voxelization. However voxelization can be faster and more efficient than tetrahedralization, especially for large objects. Tetrahedralization is commonly used in finite element analysis and animations softwares, while voxelization is commonly used in medical imaging, computer-aided design, and ray marching algorithms.

2.1.3 Deformation Gradient

From a discretized mesh, we compute individual element energies with the assumption of continuum. Using the energies, we can derive the per element equations of motions. The energy density we utilize for formulating the equations of motion in elastic simulations only requires local information about the deformation. This is captured by the deformation gradient, which provides a description of how material points are displaced within a small region, i.e., an element.

The elastic potential energy depends on the mesh deformation [99]. Each element has deformation functions $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ that map the rest-state positions $\vec{X} \in \mathbb{R}^3$ in material frame to the deformed-state positions $\vec{x} \in \mathbb{R}^3$ in the world coordinate frame as shown in

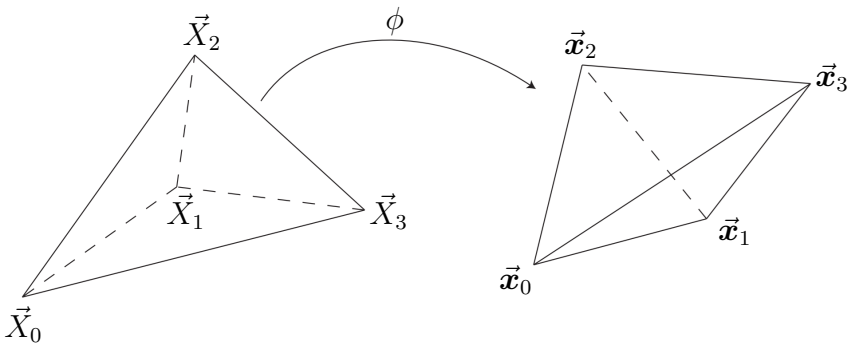


Figure 2.3: Deformation function for a 3D tetrahedron element from Sifakis et al. [99]

Figure 2.3. The vector X contains all of the rest-state vertex positions $\vec{X}$ of an elements. Likewise, the vector $\boldsymbol{x}$ contains all of the deformed-state vertex positions $\vec{\boldsymbol{x}}$ for the same element. Inside the elements, the deformation function ϕ interpolates the position from vertex displacements, i.e., barycentric coordinates for triangles and tetrahedra. For each element, we compute a deformation gradient

$$F = \frac{\partial \phi(\vec{X})}{\partial \vec{X}}, \tag{2.1}$$

which is useful to define strain measures. In our papers, we describe the deformation gradient for volumetric elements as $F = B\boldsymbol{x}$ where B is a constant kinematic mapping and reuse that throughout our research as a step to cheaply compute deformation. As a reminder, here $\boldsymbol{x}$ is the vector containing all of the world space positions for the vertices of an element.

To precompute the B matrix, we first determine the per-element material frame [99]

$$D_m = \left[\vec{X}_1 - \vec{X}_0, \vec{X}_2 - \vec{X}_0, \vec{X}_3 - \vec{X}_0\right], \tag{2.2}$$

also called the reference shape matrix. We then find the constant component of our mapping

$$D = \left[-\mathbf{1}D_m^{-1}, D_m^{-1}\right]^T, \tag{2.3}$$

where bold **1** is a row vector of ones. This extra row allow us to have equal and opposite contributions

$$\phi_0(\vec{X}) = 1 - \phi_1(\vec{X}) - \phi_2(\vec{X}) - \phi_3(\vec{X}), \tag{2.4}$$

for the particle at the center of the frame when mapping the forces or stiffness later. For convenience, we assemble

$$B = \begin{bmatrix} D_{00} & 0 & 0 & D_{10} & 0 & 0 & D_{20} & 0 & 0 & D_{30} & 0 & 0 \\ D_{01} & 0 & 0 & D_{11} & 0 & 0 & D_{21} & 0 & 0 & D_{31} & 0 & 0 \\ D_{02} & 0 & 0 & D_{12} & 0 & 0 & D_{22} & 0 & 0 & D_{32} & 0 & 0 \\ 0 & D_{00} & 0 & 0 & D_{10} & 0 & 0 & D_{20} & 0 & 0 & D_{30} & 0 \\ 0 & D_{01} & 0 & 0 & D_{11} & 0 & 0 & D_{21} & 0 & 0 & D_{31} & 0 \\ 0 & D_{02} & 0 & 0 & D_{12} & 0 & 0 & D_{22} & 0 & 0 & D_{32} & 0 \\ 0 & 0 & D_{00} & 0 & 0 & D_{10} & 0 & 0 & D_{20} & 0 & 0 & D_{30} \\ 0 & 0 & D_{01} & 0 & 0 & D_{11} & 0 & 0 & D_{21} & 0 & 0 & D_{31} \\ 0 & 0 & D_{02} & 0 & 0 & D_{12} & 0 & 0 & D_{22} & 0 & 0 & D_{32} \end{bmatrix}, \tag{2.5}$$

such that $F = B\boldsymbol{x}$ gives us a flattened vector representation of the deformation gradient. Notice that while we use the flattened representation in the manuscript chapters, the following section will consider F to be the matrix form of the deformation gradient $F = \frac{\partial \phi}{\partial X}$.

2.1.3.1 Strain

We need to define the strain to formulate various types of energies. Strain is a dimensionless measure of displacement, i.e., how the shape changes. There exists two variants of the strain commonly used in physics-based animation [9], the Green-Lagrange and the Cauchy strains.

Using the right Cauchy-Green deformation tensor $F^T F$, we obtain a rotation invariant measure of deformation. To keep the strain constant on a mesh at rest, the Green-Lagrange strain subtracts the identity to the right Cauchy-Green deformation tensor. As such the Green-Lagrange strain formula

$$E = \frac{1}{2}(F^T F - I), \tag{2.6}$$

is a nonlinear strain formulation that is invariant under translations, rotation and reflections. In a virtual world, a good strain formulation that allows larger time steps must be invariant to rigid body motions, i.e., movement without deformations. This is why we prefer to use the Green-Lagrange strain throughout the manuscripts of this thesis.

When speed is more important than accuracy, linearizing the Green-Lagrange strain leads to a faster computation of a strain measure. The Cauchy strain

$$\epsilon = \frac{1}{2}(F + F^T) - I, \tag{2.7}$$

also called small strain tensor, remains invariant under translations and reflections. This in turn ignores nonlinearities such as rotational motions which can generate significant artefacts on large rotations.

2.1.4 Lagrangian Mechanics

To formulate the equations of motions, we use the Lagrangian mechanics which require kinetic and potential energies. By balancing these two energy components, the Lagrangian method allows us to derive the equations that govern the motion and deformation of elastic materials. This formulation is useful to handle complex interactions by enabling realistic simulations of dynamic systems under various physical constraints.

2.1.4.1 Kinetic Energy

The kinetic energy which accounts for the motion of the system follows the conventional formula

$$T = \frac{1}{2}\int_{\Omega} \rho \|\boldsymbol{v}\|_2^2 d\Omega, \tag{2.8}$$

integrated over the domain Ω, with density ρ times the velocity $\boldsymbol{v}$ magnitude squared. Note that we are working with a discrete environment with volumetric elements and as

such, we discretize this equation

$$T = \frac{1}{2}\dot{\boldsymbol{x}}^T M \dot{\boldsymbol{x}}, \tag{2.9}$$

that now multiplies the discrete velocities $\dot{\boldsymbol{x}}$ squared by the mass matrix M for the discrete element. The mass matrix

$$M = \int_\Omega \rho N^T N d\Omega, \tag{2.10}$$

integrates the shape functions N over the domain of the element [117]. In the case of a linear tetrahedral element with 4 nodes, the shape function matrix takes the form

$$N = [\phi_0 I, \phi_1 I, \phi_2 I, \phi_3 I,], \tag{2.11}$$

with shape functions ϕ evaluated at the quadrature points.

To save on computation time, the density is often approximated as a lumped mass matrix containing only per particle masses on the diagonal. The mass of a particle becomes the sum of the masses of neighbouring elements divided by the number of particles per element. This misses some of the inertial components, but trivializes the inversion of the matrix, requires less storage, and eliminates spurious oscillations in the acceleration [14]. This is what we use in all of our contributions.

2.1.4.2 Potential Energy

The potential energy represents the stored energy within the system due to deformation or external forces. We find the potential energy

$$V = \int_\Omega \Psi(\boldsymbol{x}) d\Omega, \tag{2.12}$$

by integrating the infinitesimal energies Ψ over the domain Ω. The infinitesimal energy measures the strain energy per unit of the volume at rest. We note that here Ψ takes in as

argument the nodal positions, which we assume are converted to the relevant measures for the chosen energy formulation like the deformation gradient, strain, or stretches.

We explore three common infinitesimal energy Ψ formulations available in most elastic solid simulators. Different models will generate different physical behaviours. We note that there exists many more formulations for the strain energy density, but we limit ourselves to some variations that are overly popular in physics-based animation.

The Saint-Venant Kirchoff (StVK) material [58] energy

$$\Psi_{\text{StVK}} = \mu \|E\|_{\text{fro}}^2 + \frac{\lambda}{2} \text{Tr}^2(E), \tag{2.13}$$

has two parts. It first computes the Frobenius norm of the strain $\|E\|_F^2$ multiplied by the shear modulus μ to model how much the element will resist shearing. The second part $\text{Tr}^2(E)$ represents compression of the mesh. By adding both terms, we can model a wide range of deformations, all applicable to elastic solids.

The neo-Hookean material [89] energy

$$\Psi_{\text{Neo}} = \frac{\mu}{2}(I_C - 3 - \log(I_C + 1)) + \frac{\lambda}{2}\left(\det(F) - 1 - \frac{\mu}{\lambda} + \frac{\mu}{4\lambda}\right)^2, \tag{2.14}$$

uses the same principle of splitting the strain's shearing and compression components. The shearing component is instead the sum of diagonal entries $I_C = \text{Tr}(F^T F)$ of the right Cauchy strain deformation tensor. This sets the strain value of zero back to the rest state by subtracting the sum of the identity diagonal entries. Then the second part takes into account the volume in comparison to the rest state and its behaviour according to the Lamé parameters. This technique has the nice property of being resistant to element inversion because it introduces a barrier function $\log(I_C + 1)$ that peaks at $F = 0$, where there is no deformation [100]. This also means that this formulation is not rest stable.

The corotational [37] approach separates the deformation of a material into a rotational component and a purely deformational component. This is done by rotating the deformed configuration back into a reference frame, where the rotational effects are removed, and only the planar strain is considered. It takes the principal stretches $\boldsymbol{s}$ as argument instead of the deformation gradient F directly, which we can cheaply obtain from a singular value

decomposition [95] of the 3 by 3 deformation gradient. The corotational energy

$$\Psi_{\text{CR}} = \mu \sum_{i=1}^{3} (s_i - 1)^2 + \frac{\lambda}{2} \left(\sum_{i=1}^{3} s_i - 3 \right)^2, \tag{2.15}$$

considers only the deformations due to stretching. This negates the effect of rotational motions on the energy.

2.1.4.3 Lagrangian

This system leads us to the core idea of Lagrangian mechanics; the relationship between forces and energies. We define the Lagrangian

$$L = T - V, \tag{2.16}$$

as the difference between the kinetic energy and potential energy. We wish to respect the principle of least action, which states that a particle follows the most efficient path in space-time. This also leads to conservation of the energy. By applying the Euler-Lagrange [41] equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{x}}} = -\frac{\partial L}{\partial \boldsymbol{x}}, \tag{2.17}$$

$$\frac{d}{dt} M\dot{\boldsymbol{x}} = -\frac{\partial V}{\partial \boldsymbol{x}}, \tag{2.18}$$

$$M\ddot{\boldsymbol{x}} = -\frac{\partial V}{\partial \boldsymbol{x}}, \tag{2.19}$$

we obtain an equation that we must satisfy to get valid physical motions by minimizing the action through this coupled system. That final equation is essentially Newton's second law: the mass times the acceleration equals the forces. We note that we instead use the letter V as the volume of a tetrahedral element in the later chapters, but temporarily consider V to be the potential energy for the background section to match the standard literature.

2.1.5 Time Integration

The choice of a good time integrator is important when setting up a simulator. In this process, we discretize space and time; we simulate only at specific intervals with a time step h. This approximation can cause non-physical phenomena. There is also a compromise between stability and efficiency. Explicit methods such as forward Euler

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_t, \tag{2.20}$$

$$\dot{\boldsymbol{x}}_{t+1} = \dot{\boldsymbol{x}}_t + hM^{-1}f(\boldsymbol{x}_t), \tag{2.21}$$

are quick, but unstable. The energy of the mesh will often increase until it explodes. In this equation, $\boldsymbol{x}$ are the nodal positions of a mesh, f are the forces from the potential energy derivatives or external interactions, and $\dot{\boldsymbol{x}}$ are the velocities. The subscript t indicates values at the current time step while $t+1$ are values at the next time step.

Symplectic Euler time integration

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_{t+1}, \tag{2.22}$$

$$\dot{\boldsymbol{x}}_{t+1} = \dot{\boldsymbol{x}}_t + hM^{-1}f(\boldsymbol{x}_t), \tag{2.23}$$

has a similar cost as forward Euler. It uses the velocities at the next time step to update the positions, which slightly hinders parallelization. It is more stable as it tends to conserve energy, yet still requires small time steps to properly simulate without explosions. We consider a simulation to be stable when the energy is not subject to unbounded growth.

We want our simulations to remain stable, even at the cost of higher computational time per step. The typical approach is to use the backward Euler implicit time integration

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + h\dot{\boldsymbol{x}}_{t+1}, \tag{2.24}$$

$$\dot{\boldsymbol{x}}_{t+1} = \dot{\boldsymbol{x}}_t + hM^{-1}f(\boldsymbol{x}_{t+1}), \tag{2.25}$$

which is more stable, but features numerical damping. We use nonlinear solvers to find the positions and velocities at the next time step.

2.1.6 Nonlinear Solvers for Backward Euler

We describe two types of solvers that we can use for the implicit time integration in physics-based animation simulators. We aim to find the unknown variables like the velocities at the next step

$$\dot{\boldsymbol{x}}_{t+1} = \underset{\dot{\boldsymbol{x}}_{t+1} \in \mathbb{R}^n}{\arg\min}\, Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}), \tag{2.26}$$

by minimizing an objective function Q. For the purpose of this work, we use the optimization function [3]

$$Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}) = \frac{1}{2} \left(\dot{\boldsymbol{x}}_{t+1} - \dot{\boldsymbol{x}}_t\right)^T M \left(\dot{\boldsymbol{x}}_{t+1} - \dot{\boldsymbol{x}}_t\right) + V, \tag{2.27}$$

where M is the mass matrix and V is the potential energy. The closer the initial iterate is to the final solution, the faster the algorithms will converge to the optimal value. Hence, we start solvers start with an initial iterate $\dot{\boldsymbol{x}}_{t+1} = \dot{\boldsymbol{x}}_t$, assuming that velocities will remain similar between steps.

Gradient descent While solving nonlinear systems, the gradient descent [16] approach iteratively steps

$$\dot{\boldsymbol{x}}_{t+1} \leftarrow \dot{\boldsymbol{x}}_{t+1} - \alpha \nabla Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}), \tag{2.28}$$

in the opposite direction of the gradient of the objective function with step size α. Using a good step size can greatly impact convergence, and one way to find it is to use line search. The gradient descent direction points towards a locally minimal solution. Eventually, this technique will converge to one of the local minima. There is no guarantee of ever reaching the global minimum for the problem. This is typically acceptable because one is mainly interested in a physically plausible result.

Newton's method We can use more sophisticated approaches. Newton's method [56] steers the solution

$$\dot{\boldsymbol{x}}_{t+1} \leftarrow \dot{\boldsymbol{x}}_{t+1} - \alpha H^{-1} \nabla Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}), \tag{2.29}$$

towards the optimal value by creating a local quadratic approximation $H = \nabla^2 Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1})$ of the function and finds the next point in its path by minimizing this approximation. Hence, this method does not only take into account the local steepness of the function, but also its curvature. Likewise, a step size of one is usually good in this context, but line search can still be useful to reduce the step size as needed. This often results in better convergence towards a local optimum. This is the method that we use in the first two papers constituting this thesis.

We can improve the convergence of these methods by using line searches, which is a technique commonly used to find an optimal step size that maximizes the progress of an iterative method. Essentially, line search transforms the multidimensional problem into a 1D optimization problem, where we optimize for a step size α that improves the objective the most along the search direction. Instead of solving for the best possible step, it is reasonable to employ a stop criterion.

A popular stopping criterion is Armijo's rule [4]. , which ensures sufficient decrease in the objective function relative to the step size and search direction. By checking whether the objective improves sufficiently at each candidate step size, Armijo's rule helps to prevent overstepping, which could lead to divergence or inefficient progress. In practice, when the solution does not improve beyond a specified threshold, we iteratively reduce the step size, typically by halving it, until the solution satisfies the required improvement criterion. This ensures a more controlled descent towards the solution.

2.1.7 Semi-Implicit Backward Euler

In physics-based animation, some motions cause nonlinearities, i.e., rotational motions, and large deformation of some material types like the neo-Hookean energy. Solving for backward Euler using Newton's method can be slow, we often want to simplify it. With a small enough time step and relatively small deformation, the nonlinearities become negligible. With small time steps, a single iteration of the Newton solve can lead to stable simulations. We linearize the backward Euler formula using a Taylor expansion. By developing the equation using the definition of $\boldsymbol{x}_1$ and $\dot{\boldsymbol{x}}_1$ successively, we obtain the

common semi-implicit backward Euler formula

$$\underbrace{(M - hD - h^2K)}_{A} \Delta\dot{\boldsymbol{x}} = h(D\dot{\boldsymbol{x}}_t + f(\boldsymbol{x}_t) + hK\dot{\boldsymbol{x}}_t + \boldsymbol{f}_{\text{ext}}), \tag{2.30}$$

which we use in a single fast and efficient Newton iteration [6]. Consider stiffness as a material property that measures how resistant is an element to deformations. Here, the stiffness matrix

$$K = \frac{\partial^2 V}{\partial \boldsymbol{x}^2}, \tag{2.31}$$

is the Hessian of the potential energy. The Rayleigh damping matrix

$$D = \alpha_0 M + \alpha_1 K, \tag{2.32}$$

comes from predetermined terms where α_0 is the mass damping coefficient, and α_1 is the stiffness damping coefficient.

The size of A directly correlates to the number of degrees of freedom for our system. Notice that Equation 2.30 is of the form $A\boldsymbol{x} = \boldsymbol{b}$. There exists many linear solvers for this type of problem. As the matrix A is sparse and usually symmetric positive definite (SPD), we can make use of methods such as the preconditioned conjugate gradient or a direct solve with permutations that minimizes non-zero fill.

Typically, a direct solve for a linear system involves performing Cholesky decomposition, where the matrix A matrix is factored as $A = LL^T$, such that L is a lower triangular matrix. This decomposition allows us to solve the linear system $A\boldsymbol{x} = \boldsymbol{b}$ in two stages through forward-backward substitution. First, we solve the intermediate system $L\boldsymbol{y} = \boldsymbol{b}$ by forward substitution. Because L is lower triangular, each row introduces only one new unknown variable. We then proceed with backward substitution to solve $L^T\boldsymbol{x} = \boldsymbol{y}$, which yields the solution vector $\boldsymbol{x}$.

For poorly conditioned systems, the method's numerical accuracy can greatly improve by using various decompositions useful for direct solves. For instance, LDL decomposition factorizes $A = LDL^T$, where L is a lower triangular matrix and D is a diagonal matrix. With this factorization, we can sequentially solve three easier systems $L\boldsymbol{y} = \boldsymbol{b}$, $D\boldsymbol{z} = \boldsymbol{y}$,

$L^T\boldsymbol{x} = \boldsymbol{z}$ instead of the full system $A\boldsymbol{x} = \boldsymbol{b}$. Note that the A matrix of the system is still sparse, because it represents the topology of the mesh. There is an opportunity to reorder the entries which may speed up convergence. We can apply a permutations matrix P on the full system $(P^TAP)(P^T\boldsymbol{x}) = P^T\boldsymbol{b}$ to further reduce the number of non-zero entries in the decomposed matrices. With fewer values in the system, using this technique in sparse matrices is storage and operation efficient.

Simulations of poorly shaped or sliver elements, may generate ill-conditioned A matrices [111], which are problematic for iterative solvers. Preconditioning can greatly help convergence by creating a better conditioned system. We can apply a left precondition matrix P_1

$$P_1^{-1}A\boldsymbol{x} = P_1^{-1}\boldsymbol{b}, \tag{2.33}$$

a right precondition matrix P_2

$$AP_2^{-1}(P_2\boldsymbol{x}) = \boldsymbol{b}, \tag{2.34}$$

or both

$$P_1^{-1}AP_2^{-1}(P_2\boldsymbol{x}) = P_1^{-1}\boldsymbol{b}, \tag{2.35}$$

to the linear system in order to reduce the condition number, i.e., how large of a change in output will a small change in input cause. In other terms the condition number is a ratio of the largest and smallest eigenvalues of the system, which is a good indicator of the quality of the system.

Having a good approximation of the inverse matrix A^{-1} for the precondition matrices P_1 and P_2 leads to faster convergence in iterative solvers like conjugate gradient. There is a trade-off in the choice of a preconditioner. On one end, preconditioners can be as simple and inexpensive as the inverse of the diagonal elements of A, which is named the Jacobi preconditioner. While computationally efficient, this approach provides only a rough approximation. On the other end, preconditioners that closely resemble the inverse

matrix A^{-1} can significantly improve convergence but are nearly as difficult to compute as solving the original system itself. The choice of the preconditioner is hence dependent on the application and often picked based on the properties of the problem to solve or according to resource limitations.

One popular preconditioner is the incomplete Cholesky that relaxes the requirement of exact factorization for a Cholesky decomposition by partially factoring the matrix. It only computes the non-zero elements in the positions corresponding to the non-zero elements of the original matrix A, while ignoring the new non-zero elements that would be introduced in the factorization process. In our work, we use incomplete Cholesky as our preconditioner because it strikes a good balance between speed of computation and the quality of the approximation.

2.1.7.1 Multigrid

An efficient and scalable approach to solve linear systems of equations is to use multigrid algorithms. These methods iterate through a hierarchy of resolutions. The method requires the definition of restriction and prolongation operations, which respectively reduce the size of the system or interpolates a reduced system back to a finer resolution. The user must determine how the solver will move through the resolutions a priori.

A standard v-cycle pattern starts from the original system representation, e.g., $A\boldsymbol{x} = \boldsymbol{b}$, does a series of restrictions followed by prolongations back to the original resolution. Between each of the prolongation and restriction operations, the residual error is smoothed using Jacobi or Gauss-Seidel iterations. While we just described the multigrid solver's steps using a standard v-cycle pattern, it is often the case that users will select different patterns like a w-cycle. In this type of pattern, another v-cycle follows the first, sometimes before completely prolonging the system back to the original resolution.

Multigrid methods, while approximate at coarse levels, scales much better than other techniques on larger systems as coarser resolutions greatly reduce the system to solve, yet efficiently propagate information on prolongation. With a good approximate solution interpolated to the original resolution, a few full resolution smoothing iterations can often

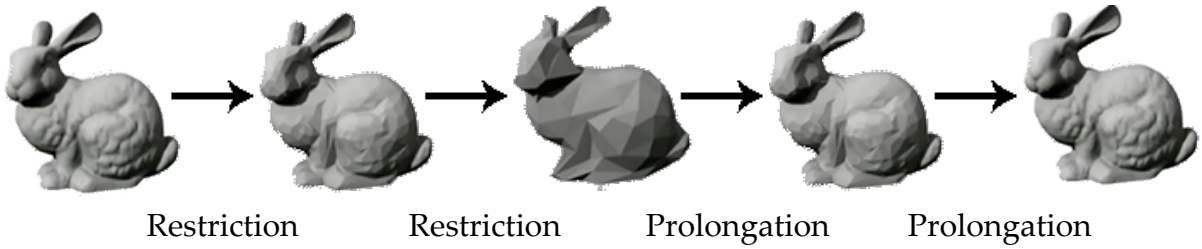


Figure 2.4: Visual representation of the different resolutions used in a v-cycle for a bunny model in a geometric multigrid solver [65].

correct the leftover residual error. Multigrid methods can define prolongation and restriction on an algebraic level, changing a linear system to solve [42]. Likewise, the operations can be at a geometric level [116, 113], discretizing the problem for each grid resolution with varying coarseness and generating a mapping between resolutions for restriction and prolongation. Later in this thesis, we will present a novel algorithm inspired by multigrid methods that is not geometric nor algebraic. In Figure 2.4 we show a visual representation of restriction and interpolation on a 3D model.

2.1.8 Position Based Dynamics

Real-time simulators often use iterative methods for efficient time integration. A popular model for both video games and training simulators is the position-based dynamics (PBD) Müller et al. [83]. This type of simulation is easy to parallelize, making it a fast alternative to methods such as the matrix factorization from Section 2.1.1. Position-based dynamics employs a Gauss-Seidel-like solver to compute the motions of constrained particles within the simulation. Elasticity in PBD is represented as a set of per-element constraints, which allows for straightforward integration of new constraints. It first computes positions using symplectic Euler time integration while ignoring internal forces. Then, it updates positions by projecting the explicit integration results onto each constraint. Finally, it updates velocities using finite differences relative to the previous time step. It accommodates various types of geometries, including cloth, rods, shells, and solids, all within a single simulator. While PBD aims to produce fast, stable, and robust simulations,

it does not have a rigorous foundation in continuum mechanics, which can lead to unsatisfiable constraints, or completely stiff models. The constraint solve has a dissipative effect on the energies, damping the elastic system. This can lead to nearly rigid motions with enough iterations. As such, PBD does not use any physical quantities or units to define deformables, but instead deformation is iteration and time-step dependent. Likewise, the constraint solver struggles to converge compared to other appropriate methods such as the Newton solve mentioned previously. Aside, the constraint solve is order dependent, changing the solution based on which constraint is solved first.

Later, extended position based dynamics (XPBD) [70] enhances PBD to alleviate its major issue; excessive stiffness upon convergence. A new compliance term lets the models stay elastic independently of the number of Gauss-Seidel iterations. One of the major improvements in XPBD is that it is time-step independent, meaning the constraints and compliance are better integrated with the time step. This allows this method to produce stable results even when using larger time steps. These modifications enable XPBD to simulate soft, elastic materials with much greater accuracy while still benefiting from the efficiency of PBD's iterative approach. While XPBD iterations can be parallelized, communication of information is only local. Instead of assembling the Hessian of the energy, the stiffness matrix $K \approx M$ is approximated as the mass matrix M to save on computation time per iteration at the cost of convergence. Even if the convergence is slow, only a few iterations lead to stable simulations, making such method attractive for applications where speed is more important than accuracy. Some previous work has made efforts to speed up convergence by providing constraints for exchanging information between distant locations with long-range attachments [59] or long-range constraints [82]. Our final contribution will improve convergence similar to long-range constraints and attachments by efficiently creating coupling to distant elements in order to improve propagation.

2.2 Adaptivity

Removing DOFs from our system at runtime can drastically enhance performance. Likewise, algorithms also benefit from increasing the DOFs to accurately capture the intricate

details of deformation, particularly in cases demanding accuracy, albeit at the expense of increased computational costs. Adaptive algorithms strive to infuse details selectively, responding to the simulation's specific requirements. In this section, we review some state-of-the-art algorithms in physics-based animation, some of which are originally designed for specific object types such as clothes or rigid bodies.

2.2.1 Subspace Condensation

In physics-based animation, model reduction involves simplifying the underlying mathematical and computational models while still preserving the essential dynamics and characteristics of the simulated systems. By reducing the complexity of these models, simulations become more efficient, making it possible to achieve real-time or interactive animations, albeit losing some level of accuracy.

Model reduction techniques in physics-based animation simplify the geometrical representation of large models. These approaches strike a delicate balance between accuracy and computational efficiency, allowing animators, engineers, and researchers to create captivating simulations without overwhelming computational demands.

Simulating a full mesh is costly, perhaps we only need to simulate part of the mesh to produce plausible motions. Condensation reduces the degrees of freedom of the A matrix to a subspace of the mesh [104]. Static condensation has a subspace that involves only the surface nodes [18]. Assuming a quasistatic model, i.e., a model with small displacements and no rotations, we can formulate $\boldsymbol{f} = K\boldsymbol{x}$, a simple version of the equations of motions. For linear material formulations [7], internal forces are modelled by cubic polynomials. The coefficients of these polynomials can be precomputed, and as such, the stiffness matrix K and its inverse can be cached for the simulations. We can generate a blocked matrix

$$\begin{bmatrix} K_{ss} & K_{si} \\ K_{is} & K_{ii} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_s \\ \boldsymbol{x}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_s \\ \boldsymbol{f}_i \end{bmatrix}, \tag{2.36}$$

by reordering the entries to separate the surface vertices $\boldsymbol{x}_s$ and the internal vertices $\boldsymbol{x}_i$. Using the Schur complement of K_{ss}, we obtain

$$K_{ss}^* = K_{ss} - K_{si}K_{ii}^{-1}K_{is}, \tag{2.37}$$

$$\boldsymbol{f}_s^* = \boldsymbol{f}_s - K_{si}K_{ii}^{-1}\boldsymbol{f}_i, \tag{2.38}$$

a decomposition of the domain. The Schur complement allows us to isolate part of the system, namely the surface vertices. As the inverse stiffness is precomputed, this whole process is efficient, allowing us to account for the static deformation of internal vertices without computing their positions. We now have a condensed form

$$K_{ss}^*\boldsymbol{x}_s = \boldsymbol{f}_s^*, \tag{2.39}$$

where internal vertices are no longer needed. This process requires the inversion of the K_{ii} block of the stiffness matrix, which is precomputed. The displacement of the internal nodes

$$\boldsymbol{x}_i = K_{ii}^{-1}(\boldsymbol{f}_i - K_{is}\boldsymbol{x}_s), \tag{2.40}$$

can still be recovered. However, using a fixed stiffness matrix can lead to poor handling of rotational motions. Another drawback of static condensation is that it cannot handle contacts without leaving major artefacts. Adaptive subspace condensation [104] uses static condensation and improves it by adaptively adding and removing internal vertices near contact points to the set of subspace vertices [104]. In a way, this is similar to adaptive rigidification, where degrees of freedoms of the full space are reintroduced where needed.

2.2.2 Multi-Resolution

In video games, object resolutions change depending of the distance between the camera and the props in order to accelerate the renders. Objects closer to the camera are rendered with more detail, while distant ones are simplified. This concept of adaptive resolution extends beyond rendering and can also be applied to physics-based simula-

tions. In simulators, it is feasible to switch between different levels of detail depending on the requirements of the scene, conserving computational resources while maintaining necessary accuracy.

From a geometric perspective, this involves computing mappings between meshes to enable force interpolation between resolutions as needed. This is seen as a hierarchy of resolutions or mesh discretizations [88]. The generation of a hierarchy can make use of Voronoï regions to locally control resolutions [30, 31]. These regions are a partition of space with respect to the control points. Each region is delimited by the midsection, between the control points. In this case, the control points are vertices of the parent (coarse) resolution. The vertices of the finer resolution mesh within the Voronoï region are child vertices of a parent vertex. Using these regions to define a bottom up hierarchy allows local coarsening and refinement of different regions of a mesh. While this technique is efficient in terms of computation time, it requires the user to generate different version of the same mesh. This process is ultimately time consuming. Careful manual tuning of regions is required to avoid scenarios where the simulator would benefit from a resolution in between those initially designed by the artist, or to avoid poorly shaped elements.

Instead of relying on elements like FEM, some methods use different control coordinate frames spread through the mesh in a hierarchy with interpolation of nearby vertices [19, 44, 109]. The surface vertices have weights corresponding to the frames, enabling smooth interpolation of vertex positions. The interpolation is linearly weighted based on proximity with the closest control points. This is called linear blend skinning. If the number of control point is too small, this can lead to significant artefacts near the bent regions, and it generally does not account for self-collision without extra basis functions. The frames activate and deactivate according to a deformation rate threshold. The hierarchical structure allows frames to depend on their parent frame's position and state, facilitating independent behavior for components like the branches of a tree while maintaining a shared dependency on the trunk.

Closer examples to our work can be found in the special cases of soft bodies with rigid transforms [22, 105]. In this approach, motions are solved rigidly first, while a secondary hexahedral mesh simulation handles the internal deformations. This approach, like ours,

uses rigid and elastic representations for efficient simulation while balancing flexibility and computational performance. This can be seen like a two-layer version of our multi-layer solver, but always starting from a fully rigid model.

2.2.3 Remeshing

Adaptive remeshing [87, 90, 97] addresses the challenges of efficiently simulating deformable objects by dynamically adjusting the mesh geometry of the object. For instance, a piece of cloth is represented using triangles within a 3D environment. When simulating the deformation of a tablecloth resting on a table, only a couple of triangles may be required to produce reasonable behavior. When the tablecloth is pulled off the table, more triangles become necessary to accurately depict the changing curvature and to ensure visual consistency.

Large triangles are no longer ideal in such cases, because they can lead to noticeable artefacts, conversely small triangles require additional work with their added DOFs. To address these challenges, adaptive remeshing techniques dynamically update the models using operations such as splitting, collapsing, and flipping triangles at runtime to optimize the simulation as discussed in Narain et al. [87]. This approach captures finer details in the high curvature regions, while omitting some details in other regions to gain performance. In turn, these methods greatly improve the performance of shells, especially in flatter regions [86].

For cloth, the technique must anticipate buckling and wrinkling. To do so, measures like relative edge size or careful monitoring of dihedral angles can be used to compare the current discretization with the expected curvature of the cloth model. By comparing these measurements, the simulation can dynamically adjust the mesh to ensure that it accurately reflects the material's deformations and preserves realistic behavior. Global bounds are set on the change in vertex normals and material compression. This affects the resmeshing operators like these three types of manipulations shown in Figure 2.5. The manipulations to the mesh triangulation go as follows:

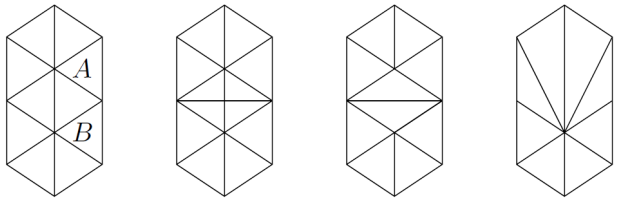


Figure 2.5: Three remeshing applied on triangle A and B. From left to right: split, flip, collapse [87].

- On **split**, add a new vertex that minimizes the quadric error with the surface curvature and the strain of adjacent elements.
- On **flip**, the edge between two triangles is disconnected and reconnected to the two other vertices in the square that contains the initial triangles.
- On **collapse**, vertices are removed arbitrarily. The vertices must satisfy three conditions before collapsing an edge: it does not change the boundaries of the pieces of cloth in material space, it does not produce any inverted face, and it does not create invalid sized edges.

While this adaptive approach works well for cloth, adaptively remeshing an elastic solid can prove to be challenging [72, 112] as poorly shaped elements can render the problem ill-conditioned. While recent development led to improved implementations [35] that better handles remeshing of tetrahedral elements in contact scenarios, we consider such work as orthogonal to our own efforts. Therefore, we have opted not to incorporate these methods into our framework, but look forward to future work merging the different approaches.

2.2.4 Homogenization

We can use homogenization [57, 88] to preserve a certain quality of the material for a heterogeneous mesh during coarsening. For instance, a thin slice of soft material within a stiff mesh could be ignored completely if not accounted for in the coarsened elements. Homogenization approximates multiple heterogeneous elements inside a coarse homogeneous element (similar to a cage) using a weighted average. It leaves out fine details that we might want to simulate in some circumstances.

In the methods that we developed for this thesis, we will not require such weighted average as we will fully preserve the original elements and their materials, unlike methods like remeshing.

2.2.5 Freezing

In physics-based animation, freezing methods are techniques used to mitigate the computational complexity of simulations by temporarily removing certain parts of the simulation that exhibit minimal or no motion. At any point in time, scenes typically encompass both static and dynamic elements. This enables a substantial reduction in computational overhead. To identify these motion-insignificant regions or objects within the simulation, freezing techniques monitor attributes like kinetic energies or velocities across multiple time steps. This monitoring allows for the selective exclusion or simplification of identified regions.

Static elements, such as environmental features, rigid structures, or immobile objects, make ideal candidates for freezing. This is because they remain stationary and do not necessitate continuous simulation updates. By sparing computational resources from the integration of static vertices, more focus and capacity can be directed towards other dynamic aspects of the simulation.

Frozen regions are reintegrated into the dynamic simulation when their motion surpasses the predefined threshold or when they become involved in interactions. For rigid bodies, freezing techniques involve treating certain objects as fixed or immovable during parts of the simulation [96, 33]. This is achieved by temporarily disabling certain physics calculations, such as collision detection and response, with the assumption that these frozen objects maintain their positions and orientations.

In the context of elastic bodies, freezing typically targets individual vertices or elements within a volumetric mesh. The adaptively restrained particle system (ARPS), represents a form of freezing [5, 73]. The particles with kinetic energy below a threshold E_r are static, those with energies above a threshold E_f are fully simulated with thresholds set such that $E_r < E_f$. Each degree of freedom can be independently restrained, effectively

removing these degrees of freedom from the time integration process. Trajectories of the particles between both thresholds smoothly transition between the restrained state and fully simulated state. Over time, particles may experience multiple cycles of restraint and release. Adjusting the values of E_r and E_f can influence simulation speed at the expense of fine-grained details.

While freezing can significantly reduce computational costs, it comes with trade-offs. Overly aggressive freezing leads to loss of detail and realism in the simulation. Therefore, careful tuning of freezing parameters and thresholds is essential to strike a balance between performance and fidelity. This is a common thread amongst freezing methods, including the following approach.

2.2.5.1 Merging

Adaptive merging, as introduced by Coevoet et al. [26], is a type of freezing which enhances time integration efficiency by reducing system size. However, what sets this technique apart is the ability to dynamically gather moving meshes based on environmental inputs. The process begins by establishing a threshold. When two rigid bodies come into contact and exhibit low relative motions below this threshold, they merge to form a single rigid body. This consolidation leads to a substantial reduction of the system's degrees of freedom, six for each merge, encompassing translational and rotational motions.

The technique also incorporates unmerging capabilities. Meshes that have merged will unmerge when contact is broken or when sliding occurs. To achieve this, a single iteration of projected Gauss-Seidel (PGS) is performed to compute approximate velocities. In turn, this oracle yields approximate relative motions. An additional threshold, applied to the approximate relative motion, triggers the unmerging of bodies.

To further optimize system efficiency, adaptive merging can be combined with sleeping mechanisms, diminishing the degrees of freedom within the system and those subjected to the single iteration of the contact solver. Although merging has primarily been explored for rigid bodies, there is a promising opportunity to extend this approach to elastic solids, which we explore in this thesis. While this work aims at speeding up rigid bodies moving in similar directions, it brought us an important adjacent question: can we

simplify elastic elements that behave similar to one another? This is the question that we explore in the next chapter.

Chapter 3

Adaptive Rigidification of Elastic Solids

This chapter is a from a published paper: A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124. It is provided as is with only minimal editorial modifications. Section 3.7 is added at the end to transition to the next work. We also provide videos in the supplemental materials.

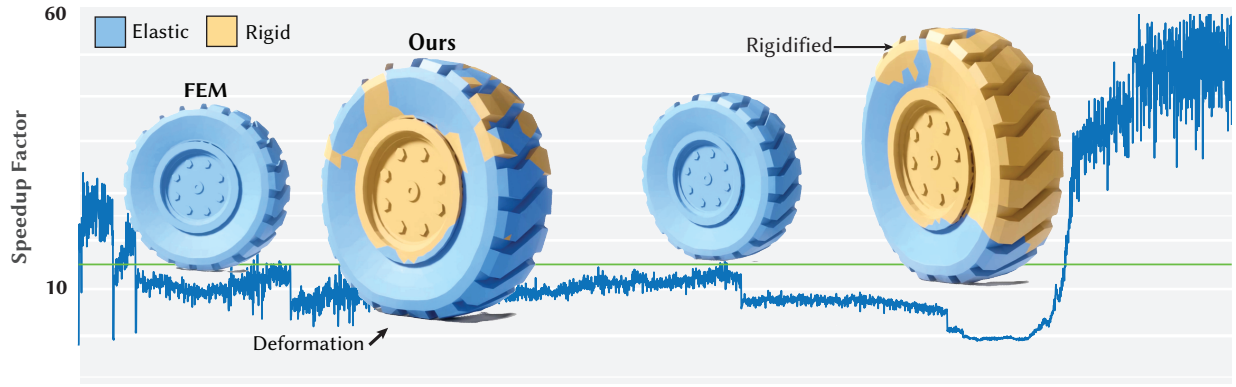


Figure 3.1: Our algorithm can identify and adaptively rigidify undeforming portions of simulated elastic objects in order to improve performance without sacrificing visual fidelity. Per-step computation time for this tire simulation, with rubber tread and steel hub, shows a mean performance improvement of 10×, resulting in a 5× reduction in total simulation time.

3.1 Introduction

Physics-based animation produces complex emergent behavior and motion from relatively compact physical laws. Prescribing a handful of physical properties to an object can generate animations of phenomena as wide ranging as realistic dripping honey, to an immense castle collapsing, to the delightful bounciness of a cartoon character.

In solid mechanics, parameters such as the Young's modulus and Poisson's ratio determine the effective compliance of an object relative to its environment. For any finite, realistic choice of these parameters, there are scenarios during which an object will drastically deform, and others during which it will move rigidly through the world. Imagine squishing a sponge in your hand. During compression the sponge changes shape drastically, but if you swing your clenched fist, the sponge moves rigidly in its new deformed state. Complicating things further is the spatially varying nature of this effect. For large objects, local external forces cause deformation that quickly decays to rigid motion as distance from the loading point increases. Despite these effects being well-known, for the past 40 years, physics-based animation algorithms have forced practitioners to decide a-priori if an object should be modeled as a rigid body or a deformable one – regardless of the applied loads it may experience. The goal of this work is to free users from this forced choice.

Simulating large deformations requires a suitably expressive kinematic map with sufficient degrees-of-freedom to model shape change. This comes with an unavoidable performance overhead. In contrast, purely rigid deformation can be represented compactly by an isometric transformation. This enables fast simulation, meaning large complicated scenes are often simulated using rigid bodies due to performance considerations. In doing so, they give up on potentially interesting behavior due to deformations. Ideally, simulation algorithms should apply the appropriate mapping based on the physical properties of the system and its underlying interactions, not based on user intuition. Rigid sections of objects should be simulated using the compact, performant, rigid body mapping while parts of the scene undergoing deformation should be upgraded to a deformable model. We should be able to have our simulated cake and eat it too.

In this chapter we present the first hierarchy-free algorithm for the elastodynamic simulation of deformable objects that can dynamically transition between rigid and deformable kinematic models at runtime. Our method maintains two dynamically evolving, spatially varying kinematic maps, one rigid and one deformable. We introduce a deformation velocity metric that predicts which parts of an object can be represented as rigid bodies, and therefore integrated efficiently using geometric methods, leading to fast

aggregate simulation times for complex scenes. Additionally, we devise an inexpensive method for estimating which parts of an object should transition between rigid and deformable states in the presence of transient forces such as contact and friction. Crucially we accomplish this without requiring predefined hierarchies or introducing additional constraints on the geometry or physical parameters of the simulation.

With our method for on-demand elastification, physics simulation of solid geometry becomes properly input-sensitive. The rigid/deformable modeling decision becomes a function of physical parameters and environmental interactions rather than a guess made prior to runtime. By putting the physics first, our algorithm reduces user burden, avoids filtering away salient emergent behavior and expedites computation of results. In what follows we detail our adaptive approach to the modeling and simulation of elastic, deformable objects and show that under a number of commonly encountered scenarios our method yields significant speed-ups over purely deformable finite element simulations.

3.2 Related Work

Degree-of-freedom reduction is a common technique for accelerating physics-based animation. Modal analysis [7] projects the equations of motion into a reduced linear space while frame-based approaches [38] replace dense volumetric discretizations with sparse skinning handles. Finally, numerical coarsening [88, 57, 22] allows simulations to produce results, using a low resolution volumetric discretization to produce results commensurate with a more expensive, high resolution one. While these methods can produce significant speed-ups, they have two fundamental limitations. First, they are applied during the modeling phase, meaning they do not and cannot take into account environmental stimulus, only the geometry and material properties of the model, in isolation. Second they do not naturally progress to the completely rigid case, rather they approach it, but remain deformable, always including a few additional, potentially unnecessary degrees of freedom.

Runtime approaches attempt to modify the number of degrees-of-freedom as the simulation progresses. The simplest of such approaches are freezing methods, so named

because they deactivate, or freeze, degrees-of-freedom when they are deemed unnecessary to evolve the system in time [5, 73]. In contrast to these methods, which freeze in the inertial frame, our approach permits rigid motion of components with degrees-of-freedom freezing only relative to one another. Freezing is also popular in rigid body simulations [96, 33], and more robust sleeping approaches have been developed for contacting rigid bodies [26], but these do not directly apply to the deformable bodies we study here.

Rather than deactivate degrees-of-freedom entirely, they can instead be adaptively down sampled. Early variants of this approach actually worked in reverse – they assume a coarse simulation mesh which was refined as a pre-process to create a fixed hierarchy [44]. At runtime this hierarchy could be traversed to locally enhance detail [31]. However the availability of a coarse mesh should not be assumed, and it is not always possible to refine back to the input high-resolution geometry. To address this problem, fixed hierarchy approaches have evolved to act on embedded mesh [88] and frame-based [109] simulations. These approaches use relatively coarse discretizations for even the finest levels of their hierarchies, meaning that the full motion of an object can never be resolved. Additionally, fixed hierarchies limit the location and amount of refinement that can take place.

D. Chen et al. [22] presents a special case of the hierarchical approach, using a two-level hierarchy where the root is a rigid transform [105] and the second level is a hexahedral simulation mesh. They transition to using the rigid transformation only when elastic potential energy goes to zero, which misses the opportunity for rigidification in other equilibrium states (e.g., resting contact). Modal hierarchies have also been explored [60, 104]. These are close in spirit to our approach but require the precomputation of a modal basis and can have trouble using the reduced basis in the presence of large rotational motions. Adaptive remeshing [87, 97] combats this issue by using geometric operations to add and remove degrees-of-freedom from the simulation mesh. However mesh operations are complex, difficult to implement and are not typically capable of reducing to pure rigid motion as our method does.

Our approach is not based on a hierarchy, but on connected components. Lack of a fixed hierarchy gives us maximum flexibility in terms of when to treat parts of an object

Algorithm 1: Main loop

Jc, $\Phi \leftarrow$ Find contacts	// §3.3.3
$\lambda \leftarrow$ WarmStart, approximate for new contacts	// §3.3.3
$\Delta\dot{\boldsymbol{x}}_{\text{approx}}$, $\dot{E}_{\text{approx}} \leftarrow$ QuickSolve	// §3.3.5
$\dot{E} \leftarrow$ Compute strain rates	// §3.3.4
BFS to identify rigid components	// §3.3.4
$M_{\mathsf{R}} \leftarrow$ Compute rigid properties	// §3.3.4
$\Delta\dot{\boldsymbol{x}}_{\mathsf{A}} \leftarrow$ LDLT Solve	// Eq. 3.11
$\Delta\dot{\boldsymbol{x}}_{c} \leftarrow$ Contact Solve	// Eqs. 3.14-3.15
Update velocities and positions	

as rigid or deformable. This means our algorithm can collapse an arbitrarily complex deformable object to a rigid body if permitted. While methods for mixing simulations of rigid and elastic parts have been previously proposed [52, 61, 114], we introduce a relative deformation metric which allows rigidification of regions of an object that are deformed but moving rigidly, as well as an efficient way to elastify local parts of rigid sections in response to deformation caused by external loads including contact and friction. These contributions, taken together result in an adaptive scheme for on-demand input-sensitive elasticity that is both more flexible, and more performant than prior art.

3.3 Elastic and Rigid Simulation

We will start with a brief review, describing how we set up our elastic finite element model simulation. We use tetrahedral meshes with linear shape functions and the standard semi-implicit backward Euler approach for numerical integration [6]. Following this, we will introduce how the formulation changes when portions of the elastic solid are rigid, and how we handle contact. This will lead us to the problem of identifying what parts of the mesh should be rigid, and when rigid parts should become elastic again, which we will describe in Sections 3.3.4 and 3.3.5. An overview of the main loop of our method is shown in Algorithm 1.

3.3.1 Simulating Finite Elements

We define matrix B such that the deformation gradient may be computed as $F = B\,\boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^{3n_v}$ contains the positions of the n_v vertices of the finite element model. While the deformation gradient of each element is a matrix, we pack column vector $F \in \mathbb{R}^{9n_e}$ by stacking vertically the deformation gradients of all n_e elements in column order. Using tetrahedral elements and barycentric interpolation as a shape function, the deformation gradient is constant across each element (otherwise, B can be seen as computing F at quadrature points). We compute the infinitesimal elastic energy ψ as a function of the deformation gradient F. For a given element i with rest volume V_i and deformation gradient F_i we compute element-integrated PK1 stress and stiffness

$$P_i = -V_i \frac{\partial \psi}{\partial F_i}^T, \tag{3.1}$$

$$C_i = -V_i \frac{\partial^2 \psi}{\partial F_i^2}^T. \tag{3.2}$$

We assemble the vector $\boldsymbol{P} \in \mathbb{R}^{9n_e}$ containing the element-integrated stress of all n_e elements in column order, and the sparse block matrix C with the 9-by-9 blocks C_i, which permits us to write the nodal forces and sparse stiffness matrix,

$$\boldsymbol{f} = B^T \boldsymbol{P}, \tag{3.3}$$

$$K = B^T C B, \tag{3.4}$$

where $\boldsymbol{f} \in \mathbb{R}^{3n_v}$ and $K \in \mathbb{R}^{3n_v \times 3n_v}$.

Thus, the system we solve for semi-implicit backward Euler integration is

$$\underbrace{(M - hD - h^2K)}_{A}\,\Delta\dot{\boldsymbol{x}} = h(D\dot{\boldsymbol{x}} + \boldsymbol{f} + hK\dot{\boldsymbol{x}} + \boldsymbol{f}_{\text{ext}}), \tag{3.5}$$

where we use a lumped mass matrix M, Rayleigh damping $D = \alpha_0 M + \alpha_1 K$, and $\boldsymbol{f}_{\text{ext}}$ are external forces such as gravity and user interaction forces (contact forces are discussed in Section 3.3.3). We use an LDLT factorization of matrix A to solve for the change in

velocities $\Delta\dot{\boldsymbol{x}}$ and then update the velocities and subsequently the positions with the updated velocities.

For heterogeneous materials, we build the damping matrix D as a sum of a mass damping matrix M_d and stiffness damping matrix K_d. We build the stiffness damping matrix with α_1 weighted diagonal blocks, that is, $K_d = B^T C_d B$ with each block of C_d being $\alpha_{1i}\, C_i$. For the Rayleigh mass damping matrix we lump the mass damping property in the same way that we lump the mass. That is, for element i with density ρ_i we distribute $1/4\, \alpha_{0i}\, V_i\, \rho_i$ to each vertex making up the tetrahedral element.

3.3.2 Mixing Rigid and Elastic DOFs

During the simulation of an elastic solid, there may be extended periods of time where portions of the model are moving rigidly. This occurs trivially when an object is at rest in static equilibrium, but can also happen with non-zero linear and rotational velocity during flight or sliding contact. We can treat as rigid the regions of a model that have a zero strain rate for a period of time. While we discuss the rigidification process and related issues in Section 3.3.4, we will first present how we set up the equations of motion for a mixed elastic and rigid simulation.

For simplicity, let us consider the case with a single rigid body. Let $\mathcal{R}$ be the set of vertex indices that make up the rigid body. The simulation state of the body will consist of a position p and an orientation $R \in SO(3)$ along with a linear and angular velocity $\phi = (v^T \omega^T)^T$. Following the form of Equation 3.5 we can write the rigid body equation of motion as

$$M_{\mathsf{R}} \Delta\phi = h(c(\phi) + w_{\text{ext}}), \tag{3.6}$$

where M_{R} is the 6-by-6 mass matrix with rotational inertia sub-matrix rotated into the world aligned frame given the body's current orientation, $c(\phi)$ are the velocity-depended rigid body torques, and w_{ext} are external forces such as gravity and user interaction.

When a portion of the elastic mesh is made rigid, we store the positions of vertices making up the rigid body in the rigid body frame. Letting r_i for $i \in \mathcal{R}$ be the rigid

vertex positions in coordinates of the rigid body frame, we can compute the positions and velocities of these vertices in the world frame as $x_i = R\,r_i + p$ and $\dot{x}_i = -(Rr_i) \times \omega + v$. This second expression can be written instead as a matrix product,

$$\dot{x}_i = \underbrace{\begin{bmatrix} I & -(R\,r_i)^\times \end{bmatrix}}_{\Gamma_i} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{3.7}$$

where $(\cdot)^\times$ denotes construction of the 3-by-3 cross product operator. Furthermore, we can now write the velocity of all vertices in the finite element model as a product of a matrix G with velocities of our active (elastic and rigid) degrees of freedom, $\dot{\boldsymbol{x}}_\mathsf{A}$,

$$\dot{\boldsymbol{x}} = \underbrace{\begin{bmatrix} I & 0 \\ 0 & \Gamma \end{bmatrix}}_{G} \underbrace{\begin{bmatrix} \dot{\boldsymbol{x}}_\mathcal{A} \\ \phi \end{bmatrix}}_{\dot{x}_\mathsf{A}}. \tag{3.8}$$

Here, $x_\mathcal{A}$ are the active elastic vertices (i.e., those which are not part of a rigid body), and for simplicity we assume that these vertices have lower indices and are copied by the identity block of G, while the Γ block is a stack of all Γ_i for $i \in \mathcal{R}$. When there are many rigid bodies, the lower part of matrix G and vector $\dot{\boldsymbol{x}}_\mathsf{A}$ grow accordingly.

With some of the elements being simulated as a rigid body, we only need to do the elastic solve for the set of elements which are still elastic. Let $\mathcal{E}$ be the set of elastic element degrees of freedom indices, and notice that we can compute the deformation gradient of only these elements as the product $F_\mathcal{E} = B_\mathcal{E} G \boldsymbol{x}_\mathsf{A}$, where matrix $B_\mathcal{E}$ consists of only those rows of B that correspond to the elastic elements. The elastic force and stiffness for active degrees of freedom can therefore be written

$$\boldsymbol{f}_\mathsf{A} = G^T B_\mathcal{E}^T \boldsymbol{P}_\mathcal{E}, \tag{3.9}$$

$$K_\mathsf{A} = G^T B_\mathcal{E}^T C_\mathcal{E} B_\mathcal{E} G, \tag{3.10}$$

where we only need to use the elastic element subset $\mathcal{E}$ of element-integrated stress and stiffness. Notice that the force $\boldsymbol{f}_\mathsf{A}$ and stiffness K_A include the rigid degrees of freedom be-

cause G provides the kinematic mapping for velocities of vertices that are on the boundary between a rigid body and the elastic elements.

With careful bookkeeping, the Γ block in G need only deal with those rigid nodes that are on the boundary of an elastic element. In practice, we find it simpler to compute the product $B_{\mathcal{E}}\, G$ without removing those rigid vertices that are not on the boundary with an elastic element.

Before we assemble the mixed elastic and rigid equations of motion, observe that the mass term in the Rayleigh damping must be included as a damping force on the rigid degrees of freedom. The rigid body damping force is computed as $\Gamma^T M_{d\mathcal{R}} \Gamma\, \phi$, and we note that this will damp both linear and rotational motion of the rigid body. The Rayleigh damping on both elastic and rigid degrees of freedom will therefore have a mass damping component $M_{d\mathsf{A}} = G^T M_d G$. Rigid motions are in the null space of the stiffness matrix, thus the stiffness term in Rayleigh damping does not contribute to damping of the rigid degrees of freedom. The active elastic degrees of freedom will be damped with the stiffness damping matrix $K_{d\mathsf{A}} = G^T K_d G$. This allows the application of stiffness damping specifically to the deformable elements.

Thus, the system we solve for semi-implicit backward Euler integration of the mixed elastic rigid system is

$$A_{\mathsf{A}} \Delta \dot{\boldsymbol{x}}_{\mathsf{A}} = h \left(D_{\mathsf{A}} \dot{\boldsymbol{x}}_{\mathsf{A}} + \boldsymbol{f}_{\mathsf{A}} + h K_{\mathsf{A}} \dot{\boldsymbol{x}}_{\mathsf{A}} + \begin{pmatrix} \boldsymbol{f}_{\mathcal{A}\text{ext}} \\ c(\phi) + w_{\text{ext}} \end{pmatrix} \right), \tag{3.11}$$

where $A_{\mathsf{A}} = M_{\mathsf{A}} - hD_{\mathsf{A}} - h^2 K_{\mathsf{A}}$, with M_{A} being block diagonal containing $M_{\mathcal{A}}$ and M_{R}. Just as before, we use LDLT factorization of A_{A} to solve $\Delta \dot{\boldsymbol{x}}_{\mathsf{A}}$. However, this system can be *much* smaller than the original fully elastic system and consequently much faster to solve. Furthermore, in the case of a system in elastic static equilibrium (whether the solid is stationary or moving rigidly), it all reduces to solving Equation 3.6 for the rigid body motion alone.

Once we have a solution for $\Delta \dot{\boldsymbol{x}}_{\mathsf{A}}$, we update the elastic and rigid position level variables with the updated velocities, where we use Rodrigues's formula [85] to turn angular

velocities over a time step into an incremental rotation matrix R to update rigid body positions.

3.3.3 Contact Handling

Contacts are regions on surfaces where objects collide. We detect the contacts through collision detection, finding discrete contact pairs where there is interpenetration between the models. We wish to correct this interpenetration so the simulation remains physical.

We use signed distance computations for collision detection and generate a contact at every boundary vertex of one mesh that ends up inside another mesh (we do not process self-contact). The contact is defined by the interpenetrating vertex and the location of the closest point on the surface of the other mesh. We consider contacts to be discrete point-triangle collisions with position, contact normal, and interpenetration properties. Using the barycentric coordinates of the closest point, we create three rows in the contact Jacobian matrix J_c for each contact (one for the contact normal and two tangent directions).

We use projected Gauss-Seidel (PGS) to solve for contact impulses λ, and in turn, a velocity update due to contact forces to compute the velocity at the next time step. Let us consider the update for a fully elastic system:

$$\dot{\boldsymbol{x}}^{+} = \dot{\boldsymbol{x}} + \Delta\dot{\boldsymbol{x}} + \underbrace{A^{-1}J_c^T\lambda}_{\Delta\dot{\boldsymbol{x}}_c}. \tag{3.12}$$

We multiply by J_c to compute the slip and separation of contacts at the next time step, and rearrange to form the system

$$\underbrace{J_c A^{-1} J_c^T}_{H}\lambda = -\underbrace{J_c(\dot{\boldsymbol{x}} + \Delta\dot{\boldsymbol{x}})}_{b}. \tag{3.13}$$

Solving this system with PGS, clamping the normal and tangential components of λ to their respective bounds at each iteration, provides a solution to the frictional contact prob-

lem:

$$\lambda_i^+ \leftarrow \lambda_i - (b + H_i\,\lambda)/H_{ii}, \tag{3.14}$$

$$\lambda_i^+ \leftarrow \max\left(\min\left(\lambda_i^+, \lambda_{i\text{MAX}}\right), \lambda_{i\text{MIN}}\right), \tag{3.15}$$

Bounds $\lambda_{i\text{MIN}}$ and $\lambda_{i\text{MAX}}$ are zero and positive infinity for normal direction constraints, while for tangent directions the bounds are set based on the current normal force and Coulomb coefficient of friction. While the PGS inner loop is fast, and depends only on the number of contacts, there is a cost to assembling H. We use the LDLT factorization of A to assemble H, and that cost is greatly reduced as larger portions of the elastic solid get mapped to rigid bodies. In a mixed elastic and rigid system, the velocity update $\Delta\dot{\boldsymbol{x}}_{\text{A}c}$ for active degrees of freedom due to frictional contact uses the smaller system matrix A_{A} and a smaller Jacobian $J_{\text{A}c} = J_c G$ relating contact velocities with only the active degrees of freedom. As such, the contact solve greatly benefits from the smaller matrices that arise when large portions of the elastic solid are rigidified.

We use Baumgarte stabilization [10] to deal with interpenetration that arises on collision and at resting contacts. We do this by setting $b = J_c(\dot{\boldsymbol{x}} + \Delta\dot{\boldsymbol{x}}) + k_b\Phi$ in Equation 3.13 where k_b is the Baumgarte feedback coefficient and Φ contains the constraint violations (penetration at each contact). We include a small amount of compliance to maintain an invisible amount of interpenetration, which is helpful for warm startsing the PGS solve. We do this by modifying the Lagrange multiplier solve in Equation 3.14 to be $\lambda_i^+ \leftarrow (\lambda_i H_{ii} - b + J_{ci}\,\Delta\dot{\boldsymbol{x}}_c)/(H_{ii} + \gamma)$, for compliance γ. We typically set $\gamma = 10^{-3}$ and $k_b = 0.2/h$ for rapid convergence of interpenetration values (see also [101]).

We always warm-start the PGS solve with λ values of contacts that existed at the previous time step. This is important for more than PGS convergence because it plays an important role in the elastification process. When an elastic object is in static resting contact, we require the contact solve on the rigid system to produce the same forces as the contact solve with elastic elements. While many different contact solutions are feasible for a rigid body, most of these, if applied to the elastic system, would lead to a dynamic deformation and confuse our elastification oracle (see Section 3.3.5). At rest, the compliant contact

will have normal interpenetration proportional to the normal contact force, $\lambda_i = \frac{k_b}{\gamma}\Phi_i$ for constraint i in the normal direction, regardless if solving for a fully elastic or fully rigid object.

Other contact solvers can also be used, for instance, penalty based methods would also respect our requirement of matching rigid and elastic contact forces. We also note that other constraint stabilization approaches can be used, for instance the post-step of Cline et al. [25], provided compliance is still included as a regularization to ensure that the interpenetration records the force distribution desired by the full elastic solve.

3.3.4 Rigidification

Identifying the portions of an elastic solid that can be simulated as a rigid body is relatively easy. If the rotationally invariant Green strain tensor $E = \frac{1}{2}(F^T F - I)$ remains constant for a period of time, then we allow the element to become rigid. In our method, we set a threshold τ_R based on the squared Frobenius norm of the strain rate, which is much like setting a speed limit on the slowest elastic deformation that we will simulate. The threshold is meaningful, easy to set, and works well for lively elastic systems, but may require low values for highly overdamped systems that only move slowly to static equilibrium. We flag elements as ready to become rigid if the norm is less than the set threshold for a given number of simulation steps (3 to 5 in our examples). Requiring the strain rate to stay below for a number of simulation steps prevents premature rigidification (for example, at the moment of maximum deformation in a vibrating cantilever).

While we could use $F = B\,\boldsymbol{x}$ and $\dot{F} = B\,\dot{\boldsymbol{x}}$ to compute

$$\dot{E} = \frac{1}{2}\left(\dot{F}^T F + F^T \dot{F}\right), \tag{3.16}$$

this does not produce zero strain rate for rotational motion. The velocity used to step elastic node positions to a rotated position does not correspond to an instantaneous rigid rotational velocity, and $\dot{E}$ will contain non-zero eigenvalues indicating instantaneous com-

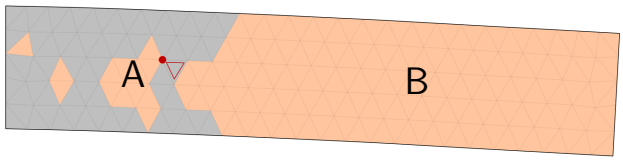


Figure 3.2: Orange regions show rigid bodies. If the red triangle is a rigid candidate, it must not become part of body B as the red adjacent vertex already belongs to body A. Otherwise, bodies A and B would share the red vertex and require a hinge constraint for correct motion.

pression. Instead, for time step k, we compute the finite difference

$$\dot{E}_k = \frac{E_k - E_{k-1}}{h}, \tag{3.17}$$

ignoring rotation by comparing strain in material space for the purpose of evaluating rigidification and elastification heuristics only.

Each element of the mesh has a Boolean *rigid* property that we set when it is flagged for possible rigidification, that is, based on having $\|\dot{E}_k\|^2_{\text{fro}}$ below threshold for several time steps. Likewise, elements that are already rigid will also have the rigid property set (unless it is flagged for elastification, see Section 3.3.5). If the rigid properties are unchanged from the previous time step (and none have been identified for elastification), then the current set of rigid bodies is left unchanged. Otherwise, we must recompute the set of rigid bodies from connected components of elements flagged rigid.

Computing connected components and rigid bodies has linear time complexity. Each element and each vertex has a *rigidID* property to identify which rigid body it is part of (if any), and all are initially assigned -1 (i.e., none). We then use a breadth first search (BFS) to construct a connected rigid component for every element which is flagged for rigidification. We use an adjacency graph for tetrahedral elements with shared faces (or with shared edges for triangle elements in 2D simulations). While each connected component forms a rigid body in our simulation, we must be careful that different components do not share vertices. A shared vertex between two rigid components require a spherical joint constraint (similarly a shared edge would require a hinge constraint). Figure 3.2 shows an example of this in a 2D simulation, where a shared vertex would require a hinge constraint

Algorithm 2: Identify a new rigid body by greedy connected component BFS. Called for all elements flagged as rigid.

Data: element e flagged rigid, rigid body index j to assign.
Result: number of elements in the new rigid component connected to e, or zero if no rigid body can be formed.

```
if e.visited then
    return 0                                          // already in a rigid body
end
Q.enqueue(e)
c ← 0                                                 // initialize count
while |Q| > 0 do
    e ← Q.dequeue
    e.visited ← true
    if any vertex of e already assigned to a body then
        continue                                      // avoid hinges
    end
    e.rigidID ← j                                     // assign element to rigid body
    c ← c + 1
    for vertex v in element e do
        v.rigidID ← j                                 // assign vertex to rigid body
    end
    S ← unvisited & flagged-rigid face-neighbors of e
    Q.enqueue(S)
end
return c
```

to keep the shared vertex at the same location. Thus, the BFS in Algorithm 2 includes a greedy vertex assignment and a vertex check to avoid hinge creation. As such the traversal order has an impact on the rigid pattern at hinges.

Once we have identified the connected components and the total number of rigid bodies, we then do a final linear pass over all vertices to compute the properties and state of each rigid body: center of mass p, linear and rotational mass matrix M_{R}, linear and angular velocity ϕ. The orientation R of the bodies are set to be the identity, and the linear and angular velocity are computed such that the linear and angular momentum about the center of mass, $M_{\mathsf{R}}\phi$, matches that of elastic degrees of freedom. If an object is moving rigidly, rigidification exactly preserves momentum. We lose momentum associated with non-rigid motion below the rigidification threshold, but this is small due to our thresholds being low to preserve visual consistency with the elastic simulation.

3.3.5 Elastification

The rigid parts of an elastic solid must be able to become elastic again when necessary. Tractions on the surfaces of the rigid regions may hold the elastic material within a rigid region in static equilibrium, or may not change enough (be large enough) to produce a noticeable deformation in the case of a very stiff elastic region. However, tractions can change, for instance, with the arrival of a traveling elastic wave, and this should cause rigid elements to become elastic again. Similarly, changing contact forces or new collisions should also cause elastification.

Ideally we would like to have an oracle that has low computational cost and can exactly identify only those elements that need to become elastic. However, the true solution requires a full solve to compute the strain rate of the full elastic system given the current state (position, velocity, contacts). Instead we propose a *quick solve* to inexpensively compute an approximate change in the velocities of all vertices, $\Delta\dot{\boldsymbol{x}}_{\text{approx}}$, from which we can identify elements to make elastic *before* solving the system at a given time step. The key observation here is that the quick solve does not need to provide an accurate solution; it only needs to identify when rigid elements should become elastic.

We choose conjugate gradient for the quick solve. While the residual does not decrease monotonically, every iteration of conjugate gradient does reduce the error, and we can avoid the costly step of assembling the matrix A for the full elastic system. Preconditioning is essential, otherwise each iteration only propagates information between neighbours following the sparsity structure of A (for instance, an impulse at one vertex will only be able to influence the $\Delta\dot{\boldsymbol{x}}_{\text{approx}}$ of that vertex and adjacent vertices with only one multiplication by A). Diagonal Jacobi conditioning, while meeting our requirement of low computational cost, does not alleviate this problem. In contrast, an incomplete Cholesky factorization is a good choice because the forward and backward substitution provides an excellent opportunity for an impulse at one vertex to influence $\Delta\dot{\boldsymbol{x}}_{\text{approx}}$ at distant vertices, even with only one iteration of conjugate gradient (see Figure 3.7 in Section 3.4).

The system matrix in Equation 3.5 changes on each time step because the stiffness is dependent on the current state. Recomputing the preconditioner, even periodically [28],

is costly. We neither want to incur the cost of the incomplete factorization on each quick solve, nor the cost to assemble the system matrix for just one multiplication. Instead we precompute the incomplete factorization of the constant Hessian approximation based on the mesh Laplacian as proposed by Liu et al. [66]. With a suitable drop tolerance, we find that the incomplete factorization is both inexpensive, having a number of non-zeros similar to A, and performs very well in our quick solve (i.e., predicting elastification), even with only one iteration of conjugate gradient. Furthermore, we do not observe any noticeable difference in the predictive power when using the full Cholesky decomposition of the fixed preconditioner. In contrast, while similar in cost to using a drop tolerance, we do not see the same performance with no-fill incomplete Cholesky or no-fill modified incomplete Cholesky, regardless the permutation of variables (alternative minimum degree, reverse Cuthill-McKee, or nested dissection).

The quick solve in all our examples uses a drop tolerance of 10^{-6}, a nested dissection permutation, and a single iteration of conjugate gradient without assembly of A to solve the full system in Equation 3.5. The quick solve is done independently on each distinct elastic objects, irrespective of contacts. For gravity forces, We note that splitting is generally beneficial, especially in scenarios involving contact. That is, we update velocities based on gravity prior to the quick solve, rather than asking our single step preconditioned conjugate gradient to deal with these forces on the right hand side of the equation.

We have considered alternatives for the quick solve oracle. For instance, Gauss-Seidel iteration is effective for merging and splitting rigid bodies [26], and a careful ordering provides good propagation of information for correct treatment of impacts. But Gauss-Seidel is a poor choice for elastic material without additional mechanisms for long range information exchange [59]. While other alternatives may be an interesting avenue for future work, we believe we have found a good balance of simplicity and speed with our current solution.

3.3.5.1 Contacts

We take contact forces into account in the quick solve by using the contact forces from the previous time step. Recall that we use a warm start for the contact solve in Section 3.3.3.

We include these warm start contact forces as explicit external forces in the quick solve by adding $J_c^T \lambda$ as a known quantity to the right hand side of Equation 3.5.

New contacts pose a slightly harder problem. We handle new contacts by conservatively approximating the forces necessary to resolve these contacts. Just like warm started contacts, we include approximate contact forces for these new contacts on the right hand side of Equation 3.5. We treat new contacts as bilateral constraints, thus, with the new-contact Jacobian J_{cn}, we can write the KKT system

$$\begin{pmatrix} A & J_{cn}^T \\ J_{cn} & 0 \end{pmatrix} \begin{pmatrix} \Delta\dot{\boldsymbol{x}}_{\text{approx}} \\ \lambda_n \end{pmatrix} = \begin{pmatrix} z \\ -J_{cn}\dot{\boldsymbol{x}} \end{pmatrix}, \tag{3.18}$$

where z is the right hand size of Equation 3.5 along with explicit warm started contacts. Forming the Schur complement gives

$$J_{cn}A^{-1}J_{cn}^T\lambda_n = J_{cn}A^{-1}z + J_{cn}\dot{\boldsymbol{x}}, \tag{3.19}$$

which will be a small system for a small number of contacts. The complication here is A^{-1}, and recall that the main simulation loop will only compute the LDLT factorization of the elastic-rigid system matrix A_{A}, as opposed to the full elastic A. In the interest of having the fastest possible conservative solution, we assume the new vertices are isolated and uncoupled, and solve for their contact forces independently using an approximation of A^{-1} consisting of precomputed diagonal blocks (i.e., we disregard the implicit stiffness coupling), computed for the rest configuration. When many new contacts form simultaneously, for instance, a large contact patch forming on impact, we will overestimate the contact force due to missing coupling terms. But this will simply leads to a larger quick solve $\Delta\dot{\boldsymbol{x}}_{\text{approx}}$ solution, and in turn a larger region of elements will be conservatively converted to elastic for solving the next time step. In contrast, when new contacts form in isolation, for instance when a contact patch grows to include a new vertex, we will obtain an accurate estimate of the contact force.

Finally, with old warm-started and new approximate contact forces in account, a single iteration preconditioned conjugate gradient solve provides $\Delta\dot{\boldsymbol{x}}_{\text{approx}}$, which we combine

with the current state $\boldsymbol{x}$ to compute $F = B\,x$ and $\dot{F} = B\,(\dot{x} + \Delta\dot{\boldsymbol{x}}_{\text{approx}})$, and in turn, the approximate strain rate $\dot{E}_{\text{approx}}$ using Equation 3.17. Every element that has $\|\dot{E}_{\text{approx}}\|^2_{\text{fro}}$ exceeding a elastification threshold τ_E is flagged for elastification. During the BFS described in Section 3.3.4, this can cause the outer layers of a rigid body to become elastic again, or can even break a rigid body into multiple components. It is the difficulty of tracking fragmentation of rigid bodies during elastification that motivates our strategy of recomputing the rigid bodies when there is a change. We briefly discuss incremental approaches in future work (see Section 3.5).

3.4 Results

Table 3.1 shows parameters and performance measurements for all examples in this chapter. Simulations were carried out on a Windows 10 PC using an Intel Core I7-6700K processor, with 64 GB of DDR3 RAM. Both the adaptive and non-adaptive simulations are primarily implemented in MATLAB, with performance critical components implemented in C++. We use GPToolbox [50] for some geometry processing and simulation specific functionality.

Our adaptive method is faster than its non-adaptive counterpart in all cases, both in terms of total simulation time and mean improvement in per-timestep computation time. The maximum performance improvement is approximately one order of magnitude in both cases (Table 3.1). Figure 3.8 shows detailed per-timestep speedups for many of our simulations which demonstrates that even in the worst-case, the adaptive code offers equivalent performance to the non-adaptive setup, and is often many times faster.

Our method is compatible with standard hyperelastic material models. We use Saint-Venant Kirchoff, neo-Hookean, and corotational energies in our simulator. The examples we show in this chapter use StVK for 2D scenes and neo-Hookean for 3D scenes. While all of our examples use semi-implicit backward Euler integration (i.e., backward Euler on the linearized system), it is straightforward to use a full Newton solve with line search.

Table 3.1: Speedup for different examples, and the parameters used: rigidification and elastification thresholds τ_R and τ_E, time step h, Rayleigh parameters α_0 and α_1, Young's modulus E, Poisson's ratio ν, and number of tetrahedra $\#T$. All material energies in these specific scenes are neo-Hookean. Comparing the mean of per-frame speedup to the total speedup, we can see if the improvement in computation time is localized to specific frames or distributed over the full scene. For instance, the blob has big speedups at the start and end of the simulation, while the forest has a constant speedup. No-contact (NC) total and mean speedups show values computed without counting the contact solve time and demonstrate that our choice of contact solver does not exaggerate the benefit of adaptive rigidification. In all examples, we see a significant increase in performances, even in highly deforming scenes with non-localized deformations.

Sim	Time (s) Adaptive	Time (s) Default	Total Speedup	Mean Speedup	NC Total Speedup	NC Mean Speedup	τR	τE	h	Objects	α_0	α_1	ν	E	$\#T$
octopus	175.57	754.03	4.30	5.81	4.29	7.00	1e-5	1e-4	1e-2	octopus	1e-4	1e-1	0.30	5e3	6170
blob	1876.40	3915.30	2.09	9.10	1.81	5.74	1e-5	1e-4	1e-2	blob	1e-3	1e-2	0.22	5e3	6386
wheel	258.21	2966.80	11.49	16.43	9.74	12.39	5e-4	5e-3	1e-3	tire	1e-5	5e-2	0.30	3e3	15797
										rim	1e-5	2e-1	0.40	1e6	
forest	96.56	800.82	8.29	9.24	8.27	9.21	5e-3	5e-2	5e-3	pine tree	1e-5	1e-2	0.37	2e5	56818
										other tree	1e-5	7e-2	0.37	7e5	
										baseball bat	1e-4	1e-1	0.37	5e4	
										leaves	1e-4	1e-4	0.37	7e5	
pachinko	347.18	3803.20	10.96	13.24	5.23	5.63	7e-3	7e-2	1e-2	red pills	1e-5	5e-2	0.35	5e4	39808
										white pills	1e-5	1e-1	0.35	5e3	

3.4.1 Threshold Selection

Selecting a threshold for rigidification is not difficult, and is largely a question of choosing a trade-off between error and speed. However, a large threshold will lead to large errors; a good choice is crucial so as not to generate visual artifacts.

The square of the Frobenius norm provides an upper bound on the sum of squared eigenvalues of the strain rate. The eigenvalues correspond to stretch rates, which provides intuition. For example, a rigidification threshold of τ_R =1e-4, can be thought of as letting material become rigid if it is deforming at slower than 1% per second. This can be a good threshold for many scenes involving damped oscillations, and recall that making a portion of the mesh rigid does not prevent it from quickly become elastic again.

Figure 3.3 shows a clear relationship between between error and speed in the simulation of a 2D cantilever for different choices of τ_E with $\tau_R = 10^{-1}\tau_E$. The test scenario involves the under damped cantilever falling under gravity, visually coming to rest after a period of damped oscillation, and then reacting to a scripted force applied to its free

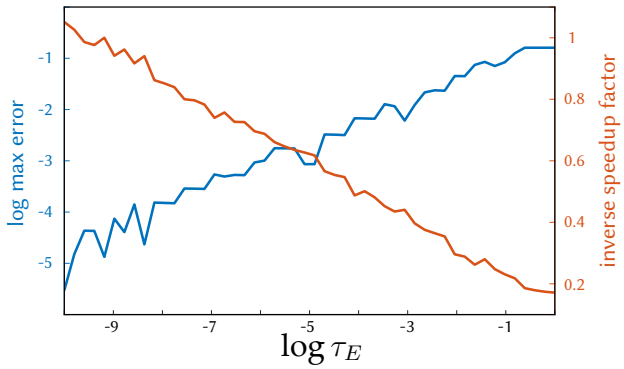


Figure 3.3: Cantilever max vertex error (measured relative to cantilever length), and inverse speedup factor (lower is better), for a varying range of elastification thresholds and $\tau_R = 10^{-1}\tau_E$.

end (see video). The error is computed as the magnitude of the maximum displacement of any vertex at any time from the fully elastic simulated trajectory, and is normalized by the length of the cantilever. We consider the elastic trajectory to be the ground-truth simulation while neglecting the possibility that our simulation is potentially more accurate numerically in some cases, e.g., on purely rigid motions. While this underestimates the strengths of our method, it provides a fair evaluation of the accuracy. We observe a maximum speedup with 10% max error above τ_E equal to 1, but this threshold setting is a poor choice because the cantilever simply remains rigid for the full simulation. In contrast, setting $\tau_E = 10^{-5}$ leads to a maximum error of approximately 0.1% in the simulation trajectory with a speedup of 1.7 times. This modest speedup comes from the fact that this is a simple 2D example with only 2739 triangular elements, while fine meshes in 3D lead to the more impressive speedups seen with other examples in this chapter. A relationship between error and speedup is also observable in richer scenes. Such a relationship is likely due to both the complexity of the solver and the distribution of the strain rates in the given scene. Figure 3.4 shows that the wheels using more conservative thresholds fall on the same side at approximately the same time while featuring good rigidification behavior.

While the rigidification threshold needs to be high enough to let elastic elements become rigid, it is ultimately the elastification threshold that determines the behavior. If the elastification threshold τ_E is set too low then it will prevent the formation of any rigid bodies, while if it is set too high then the mesh can lock in a state far from a static

Figure 3.4: Rolling wheels with different elastification thresholds τ_E fall at approximately the same time matching the fully elastic simulation in the provided video. Lower thresholds lead to more accurate simulations while higher thresholds lead to faster simulations. Here, all of the adaptive simulations share a common rigidification threshold, $\tau_R = 10^{-1}\tau_E$.

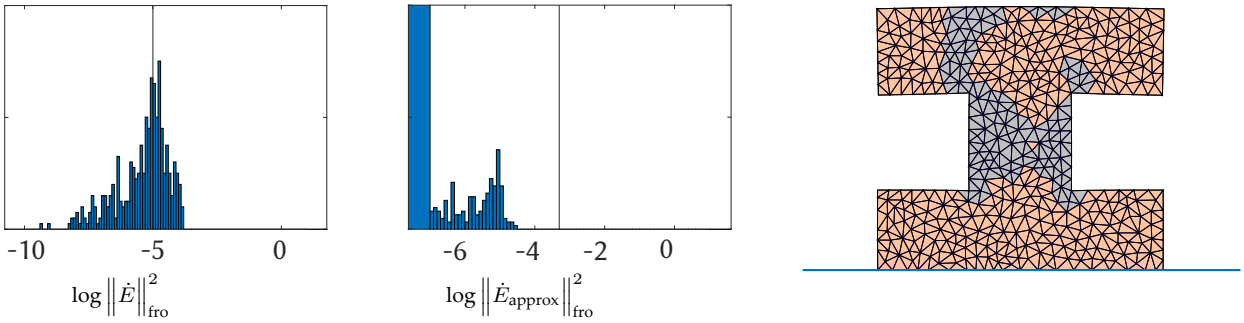
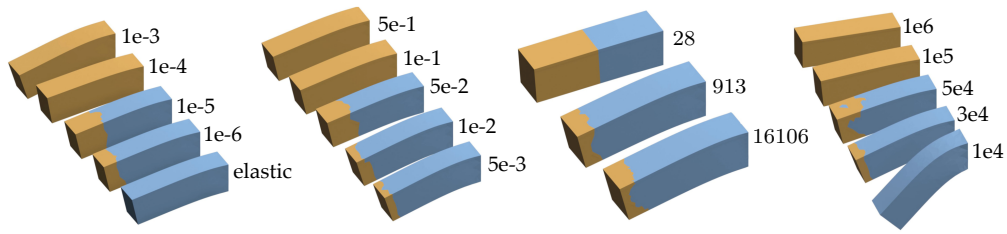


Figure 3.5: Animated histograms (see video) of elastic element strain rate (left) and rigid element approximate strain rate from the quick solve (middle) provide intuition about simulation behavior for given thresholds.

equilibrium. This can be seen at top left of Figure 3.6 for the cantilever with τ_R =1e-3 which stops oscillating too early (see also the supplementary video).

Recall that the quick solve provides only an approximate prediction of the strain rate, and in our experience the solutions are always an overestimate. Intuition can come from observing the evolution of $\dot{E}$ and $\dot{E}_{\text{approx}}$ histograms during a simulation (see Figure 3.5). To account for the quick solve error, we typically set the elastification threshold to be one or two orders of magnitude higher than the rigidification threshold.

Figure 3.6a shows effect of different elastification thresholds with $\tau_R = 10^{-2}\tau_E$. High thresholds give premature rigidification and a large error, while $\tau_E = 10^{-5}$ and lower match the fully elastic simulation. Our method accounts for varying material parameters because rigidification only depends on the strain rate (it is agnostic to the selection of parameters, as well as the choice of elastic energy and damping model). Increasing the



(a) Elastification threshold (b) Damping α_1 (c) Resolution vertices (d) Young's modulus

Figure 3.6: (a) The rigidification threshold decides when rigid bodies are created, but it is the elastification threshold which is critical for deciding what regions will stay rigid. Lower values provide simulations that more faithfully reproduce the fully elastic behaviour, but at greater cost. (b) Rigidification takes place faster when there is higher damping in the simulation scenario, leading to greater speedups. (c) We observe very similar rigidification patters independent of the resolution of the mesh, while very coarse meshes rigidify more quickly because of resolution dependent stiffness and numerical damping. (d) Lower stiffness leads to larger and longer lived oscillations, while the damping in stiffer examples has them come to rest and rigidify earlier.

damping (Figure 3.6b) or stiffness (Figure 3.6d), causes the simulated beam to rigidify more quickly. Finally, notice that similar rigidification regions form in meshes at different resolutions (Figure 3.6c), while an extremely low resolution mesh undergo rapid rigidification due to discretization-based numerical stiffening.

We observe that geometry can influence our threshold selection. For instance, geometry with long thin parts may benefit from a lower threshold to better capture global behavior (e.g., octopus, τ_R = 5e-7). Likewise, we may choose a lower threshold when small local details are important (e.g., forest, τ_R = 5e-5).

3.4.2 Example Simulations and Features

Our method supports local elastification and rigidification in response to external forces such as those arising from contact. Figure 3.7 shows two bowling balls dropped onto a mattress from different heights. Each impact elastifies a different local patch of the mattress mesh, with size proportional to the total force at impact. This conservative estimate of which elements need elastification is thanks to the approximation of a contact response in combination with the single iteration preconditioned conjugate gradients solve.

Taken together, the expressivity of the elastification threshold parameter, along with the spatially varying behavior of the adaptive scheme allow us to find suitable settings

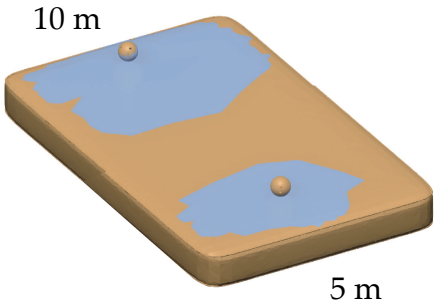


Figure 3.7: The higher ball drop elastifies a larger region.

which yield both a performance improvement and good visual agreement with non-adaptive simulations. For example, the elastic blobs (seen at left in Figure 3.8) are 2.6 times faster in terms of wall clock time, but visually indistinguishable from the standard finite element result.

This performance advantage persists even in more complicated scenes involving objects undergoing frictional contact. For instance, in the pachinko example in Figure 3.8, objects both become partially elastic and rigid as they navigate their way down multiple platforms. This ability for sliding and rotating objects to be rigid while in a deformed state is an important feature that improves runtime performance, in contrast to previous approaches which require the object to be in an undeformed state to be simulated as rigid [22].

In this scene, the high number of contacts make it harder to compute. The meshes are very coarse, but due to the optimization of the time integration on contact and the high number of meshes, our technique still outperforms the default mesh. Due to how coarse the meshes are, the overhead of our technique is more visible. The complexity of rigidification is much lower than the time integration, making it very scalable.

Local elastification and rigidication also extend to more involved geometries and interactions. This can be seen in several of our examples, but the octopus scene provides a good example. As various parts of the octopus come to rest they rigidify while allowing other pieces to keep moving. Pulling on a tentacle causes only a small local portion of the mesh to become elastic, while the rest is simulated as a single rigid body.

A key contribution of our method is that adaptivity is material-independent and so objects with heterogeneous material properties behave accordingly. Equivalent forces

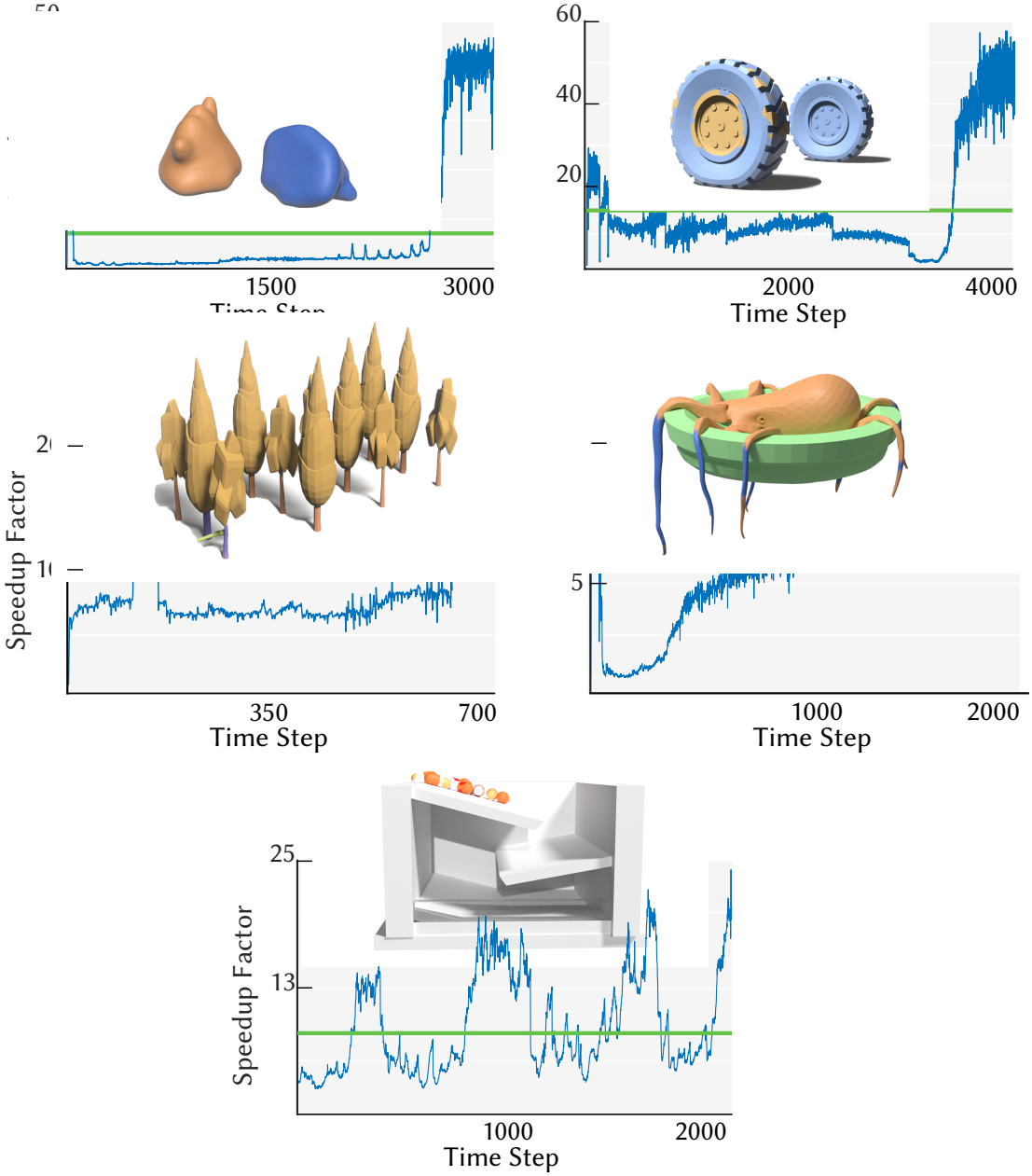


Figure 3.8: Per-frame speedup factors for blob, wheel, forest, octopus, and pachinko. Using conservative elastification and rigidification thresholds, we can obtain very accurate simulations in reduced computation time. Adaptive rigidification works for collision, rotation, frictional contact, and proves a large benefit in big scenes with local deformations, such as forest, where it is possible to elastify individual trees as needed. The green line in each plot shows mean speedup. Each rendered image shows one frame of the corresponding simulation to visually identify the scene.

cause less strain on stiffer materials, which correctly leads to more aggressive rigidification. Our method is likewise compatible with arbitrary hyperelastic material models. This is a significant advantage over requiring a user to apriori intuit how an object will interact with a scene. Figure 3.1 demonstrates this with a rolling wheel that has a rubber tire and steel hub. The hub remains rigid for much of the simulation, simulated as a single rigid body, while the tire can elastify and rigidify on demand. The adaptive simulation retains excellent agreement with the non-adaptive simulation and is 5 times faster to simulate overall.

Finally, the forest example shows a particularly compelling use case of our method. In this scene, every tree is a finite element object and we move an axe through the scene, chopping at the trees. Our method correctly activates only parts of the trees required to capture the deformation, the rest are quickly simulated as rigid bodies. This leads to a 10 times performance improvement over a standard FEM simulation. Such scenes, in which interaction in a large world is focused on a single hero character, are common in video games and movies.

All our examples feature objects which partially rigidify while in motion, and while deformed, and unfreeze (elastify) correctly in response to contact, avoiding common artifacts such as erroneous floating bodies. To our knowledge none of these examples are compatible with previous freezing approaches. Our method yields significant, often order of magnitude, speedups and is also compatible with other approaches to accelerate deformable object simulation. For instance, we could leverage updated sparse Cholesky factors [48] to reduce factorization costs. Our method is complementary to subspace condensation [104], which lists as its limitations a difficulty in handling scenarios where contacts can cause large changes in global motions, something our method excels at. We avoid the stated limitation of requiring careful construction of a reduced basis.

3.5 Discussion and Limitations

We currently recompute rigid body properties on each step only when there is a change in the elements making up the rigid bodies. However, if there is minimal change, such as just

a small number of elements added or removed from a rigid body, there is an opportunity to do inexpensive incremental updates to the mass properties and rigid state. Currently the main challenge is when a rigid collection of elements splits apart into multiple rigid bodies, and designing efficient algorithms for this case is an interesting direction for future work. Likewise, it would be interesting to explore efficient ways to adapt the graph to topological changes like cutting or fracture. While the current linear algorithm to generate connected component is not a bottleneck, an incremental approach could further improve the efficiency of adaptive rigidification.

It would be interesting to consider how to make rigidification work for hexahedra with trilinear shape functions, or likewise any model with higher order shape functions. We currently monitor a single strain rate $\dot{E}$ for each element to identify if it should be rigid or not, but it could be possible to identify collections of vertices (control points) that could move rigidly based on monitoring their motion rather than the strain rate of quadrature points.

We currently only merge adjacent elements to form rigid bodies, while it could be interesting to also merge elements that are in contact. Our example simulations include cases where multiple elastic bodies stack and become rigid, and these examples could be further speed by following an approach similar to that of Coevoet et al. [26], which merges rigid bodies and thus reduces the cost of collision detection and contact force computation.

3.6 Conclusions

Our method limits the size of the instances by reducing and increasing the degrees-of-freedom as needed. Our hierarchy free algorithm outperforms the standard FEM in every scene tested due to its cost efficiency. With this new approach we show that the simulation of elastic solids can be not only easy and fast, but accurate in comparison to the ground truth elastic simulation. Without the need for a coarse grid or down sampled meshes, we can achieve increasingly big speedup factors as elements settle within the scene. We can finally, simulate the cake and eat it. The code and data will remain available on the first author's website.

We have presented a new adaptive method that uses on-the-fly rigidification to accelerate deformable object simulation. Our method is faster than standard, non-adaptive methods in all the examples presented and in many cases dramatically so (up to an order of magnitude wall clock time improvement). This is accomplished without sacrificing visual agreement with fully deformable methods. We believe our method to be a first and significant step in the grand unification of rigid body and deformable simulations. Up until now, whether to simulate rigidly or with deformation was a high-level modelling choice, made prior to ever running the simulation – a decision made for the purposes of improving performance. We imagine a world where users are free from this choice. Rather, algorithms will correctly adapt their chosen model based on the emergent physics of the simulated system. Our work shows that, not only is this possible, but such a strategy could be of immense practical value. In order to spur future work in this important direction, code and data will be made available. We look forward to the more flexible future that awaits.

3.7 From 3D to 2D

In the previous sections, we explored the concept of adaptive rigidification within the context of elastic solids. By selectively rigidifying regions of low deformation, we achieved significant computational efficiencies without sacrificing the physical accuracy of the simulations. This technique proved particularly effective in scenarios where the material exhibited varying stiffness, allowing for the efficient resolution of large-scale deformations while maintaining the ability to capture intricate details in regions of higher elasticity.

However, extending this approach to other model types like shells, snow, or dirt is not necessarily trivial. Not all physical systems can be adequately modeled as volumetric solids. In many real-world applications, the objects of interest are better represented as thin structures, such as plates, membranes, or shells. These systems, characterized by their two-dimensional surface geometry and reduced thickness, pose unique challenges for simulation. Unlike volumetric solids, where deformation can occur in all three spatial dimensions, shells primarily deform in-plane (stretching and shearing) and out-of-plane

(bending) directions. The thin nature of these structures means that even small out-of-plane deformations can result in significant changes to the overall shape, necessitating a different approach to modeling and simulation.

This is where the concept of adaptive rigidification must be revisited and tailored to the specific needs of discrete shell models. The principles underlying adaptive rigidification remain applicable, but the implementation must account for the unique geometric and physical properties of shells. For example, in shell simulations, the balance between in-plane stiffness and out-of-plane flexibility is critical. Rigidifying a region of a shell too aggressively can lead to unrealistic constraints on bending, while insufficient rigidification may fail to capture the global structural stability.

To address these challenges, we must adapt the methodology of adaptive rigidification to the discrete shells formulation. This involves developing new criteria for selecting regions to be rigidified, which take into account both the geometric curvature and the local strain rates. Additionally, the interaction between the shell's thickness and its mechanical response requires careful consideration, as the thin-walled nature of shells can lead to different failure modes and stability issues compared to volumetric solids.

By transitioning from elastic solids to discrete shells, we expand the applicability of adaptive rigidification, demonstrating its versatility across a broader range of physical systems. This not only enhances the efficiency of the simulation models but also opens up new avenues for visually accurate simulations of complex structures in various engineering and scientific domains. This is why we address such a model in the next chapter. We will explore the adaptive rigidification of discrete shells, present how to handle the tricky bending cases, and properly handle contact elastification by using an edge filter for fast diffusion.

References for Chapter 3

[5] S. Artemova and S. Redon. Adaptively Restrained Particle Simulations. *Physical Review Letters*, 109:1–5, 19, Nov. 2012. DOI: 10.1103/PhysRevLett.109.190201.

[6] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1998, pages 43–54, New York, NY, USA. Association for Computing Machinery, 1998. DOI: 10.1145/280814.280821.

[7] J. Barbič and D. L. James. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Transactions on Graphics*, 24(3):982–990, July 2005. DOI: 10.1145/1073204.1073300.

[10] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[22] D. Chen, D. I. W. Levin, W. Matusik, and D. M. Kaufman. Dynamics-Aware Numerical Coarsening for Fabrication Design. *ACM Transactions on Graphics*, 36(4):1–15, July 2017. DOI: 10.1145/3072959.3073669.

[25] M. Cline and D. Pai. Post-stabilization for Rigid Body Simulation with Contact and Constraints. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3744–3751, 2003. DOI: 10.1109/ROBOT.2003.1242171.

[26] E. Coevoet, O. Benchekroun, and P. G. Kry. Adaptive Merging for Rigid Body Simulation. *ACM Transactions on Graphics*, 39(4):1–13, Aug. 2020. DOI: 10.1145/3386569.3392417.

[28] H. Courtecuisse, J. Allard, C. Duriez, and S. Cotin. Asynchronous Preconditioners for Efficient Solving of Non-linear Deformations. In K. Erleben, J. Bender, and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation*. The Eurographics Association, 2010. DOI: 10.2312/PE/vriphys/vriphys10/059-068.

[31] G. Debunne, M. Desbrun, M.-P. Cani, and A. 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, pages 31–36, New York, NY, USA. Association for Computing Machinery, Aug. 2001. DOI: 10.1145/383259.383262.

[33] K. Erleben. *Stable, robust, and versatile multibody dynamics animation*. PhD thesis, University of Copenhagen, 2004.

[38] B. Gilles, G. Bousquet, F. Faure, and D. K. Pai. Frame-Based Elastic Models. *ACM Transactions on Graphics*, 30(2):1–12, Apr. 2011. DOI: 10.1145/1944846.1944855.

[44] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002. DOI: 10.1145/566654.566578.

[48] F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien. Updated Sparse Cholesky Factors for Corotational Elastodynamics. *ACM Transactions on Graphics*, 31(5):1–13, Sept. 2012. DOI: 10.1145/2231816.2231821.

[50] A. Jacobson. gptoolbox: Geometry Processing Toolbox, 2021.

[52] J. Jansson and J. S. M. Vergeest. Combining Deformable and Rigid-Body Mechanics Simulation. *The Visual Computer*, 19(5):280–290, Aug. 2003. DOI: 10.1007/s00371-002-0187-6.

[57] L. Kharevych, P. Mullen, H. Owhadi, and M. Desbrun. Numerical Coarsening of Inhomogeneous Elastic Materials. *ACM Transactions on Graphics*, 28(3):1–8, July 2009. DOI: 10.1145/1531326.1531357.

[59] T.-Y. Kim, N. Chentanez, and M. Müller-Fischer. Long Range Attachments a Method to Simulate Inextensible Clothing in Computer Games. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '12, pages 305–310, Lausanne, Switzerland. Eurographics Association, 2012.

[60] T. Kim and D. L. James. Skipping steps in deformable simulation with online model reduction. *ACM Transactions on Graphics*, 28(5):1–9, Dec. 2009. DOI: 10.1145/1618452.1618469.

[61] J. Lenoir and S. Fonteneau. Mixing Deformable and Rigid-body Mechanics Simulation. In *Proceedings Computer Graphics International*, pages 327–334, 2004. DOI: 10.1109/CGI.2004.1309229.

[66] T. Liu, S. Bouaziz, and L. Kavan. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Transactions on Graphics*, 36(4):1–16, May 2017. DOI: 10.1145/3072959.2990496.

[73] P.-L. Manteaux, F. Faure, S. Redon, and M.-P. Cani. Exploring the Use of Adaptively Restrained Particles for Graphics Simulations. In *Proceedings of the 10th Workshop on Virtual Reality Interaction and Physical Simulation*, pages 17–24, Lille, France. Eurographics Association, Nov. 2013. DOI: 10.2312/PE.vriphys.vriphys13.017-024.

[78] A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124.

[85] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., USA, 1st edition, Aug. 1994. DOI: 10.1201/9781315136370.

[87] R. Narain, A. Samii, and J. F. O'Brien. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Transactions on Graphics*, 31(6):1–10, Nov. 2012. DOI: 10.1145/2366145.2366171.

[88] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure. Preserving Topology and Elasticity for Embedded Deformable Models. *ACM Transactions on Graphics*, 28(3):1–9, July 2009. DOI: 10.1145/1531326.1531358.

[96] H. Schmidl and V. J. Milenkovic. A Fast Impulsive Contact Suite for Rigid Body Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):189–197, Mar. 2004. DOI: 10.1109/TVCG.2004.1260770.

[97] C. Schreck, D. Rohmer, S. Hahmann, M.-P. Cani, S. Jin, C. C. L. Wang, and J.-F. Bloch. Nonsmooth Developable Geometry for Interactively Animating Paper Crumpling. *ACM Transactions on Graphics*, 35(1):1–18, Dec. 2016. DOI: 10.1145/2829948.

[101] R. Smith et al. *Open dynamics engine user manual*. 2005.

[104] Y. Teng, M. Meyer, T. DeRose, and T. Kim. Subspace Condensation: Full Space Adaptivity for Subspace Deformations. *ACM Transactions on Graphics*, 34(4):1–9, July 2015. DOI: 10.1145/2766904.

[105] D. Terzopoulos and A. Witkin. Physically Based Models with Rigid and Deformable Components. *IEEE Computer Graphics and Applications*, 8(6):41–51, 1988. DOI: 10.1109/38.20317.

[109] M. Tournier, M. Nesme, F. Faure, and B. Gilles. Seamless Adaptivity of Elastic Models. In *Proceedings of Graphics Interface*, GI 2014, pages 17–24, Montréal, Québec, Canada. Canadian Human-Computer Communications Society, 2014.

[114] Y. Yang, G. Rong, L. Torres, and X. Guo. Real-time Hybrid Solid Simulation: Spectral Unification of Deformable and Rigid Materials. *Computer Animation and Virtual Worlds*, 21(3-4):151–159, 2010. DOI: 10.1002/cav.373.

Chapter 4

Adaptive Rigidification of Discrete Shells

This chapter is a from a published paper: A. Mercier-Aubin and P. G. Kry. Adaptive Rigidification of Discrete Shells. *Proc. ACM Comput. Graph. Interact. Tech.*, 6(3):1–17, Aug. 2023. DOI: 10.1145/3606932. It is provided as is with only minimal editorial modifications. Section 4.7 is added at the end to transition to the next work. We also provide videos in the supplemental materials.

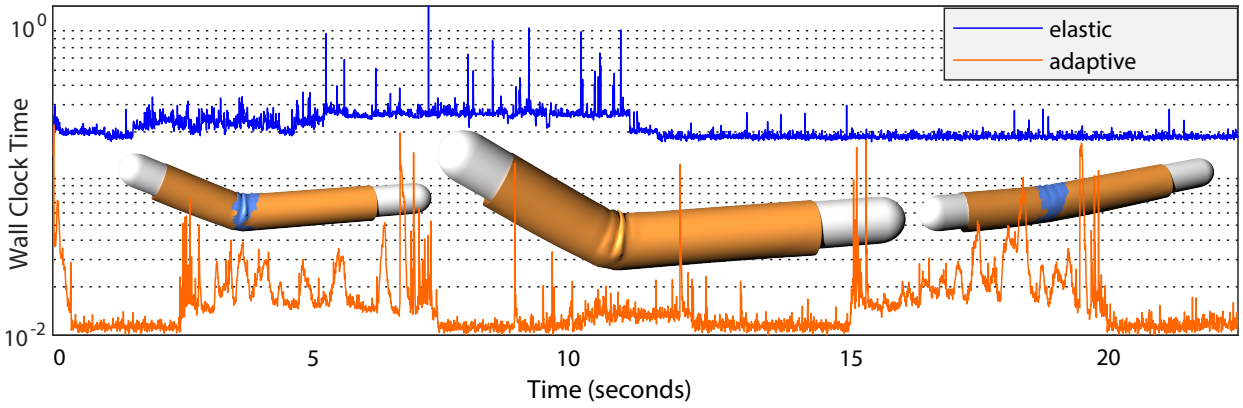


Figure 4.1: Local deformations at the elbow require elastic deformation during bending, while the rest of the sleeve simulates as a rigid body, providing an order of magnitude faster computation in comparison to a fully elastic simulation. A bending arm tends to only generate local deformation on a sleeve near the elbow with otherwise rigidly moving wrinkles. This allows the coarsening of fine detailed wrinkles through adaptive rigidification. In this example, our implementation achieves a nearly constant improvement with performances more than an order of magnitude faster than the elastic simulation.

4.1 Introduction

In any standard elastodynamic simulation, interactions between models are governed by physical laws for realistic simulations. Integrating in time the degrees of freedom

often requires expensive iterative solvers that terminate before convergence or introduce significant numerical damping. Scaling these simulations to large scenes without approximations can be challenging, as many approximations are set a priori, such as discretizing time according to a fixed step size and discretizing models into elements. However, precomputing the initial discretization of models can help alleviate some of the computational stress, and adaptive techniques can refine or coarsen the objects as needed. As the number of elements in a scene grows, reducing the size of the system becomes increasingly important for achieving efficient and accurate simulations.

The simulation of thin tetrahedral meshes is subject to even more problems, as it often leads to poorly conditioned system due to sliver shaped elements, or extremely high resolution models necessary for the deformation of very thin models. Instead of volumetric meshes, 2D models with a thickness parameter and assumption of non-deformation in the normal directions can be used to create plausible simulations with a potentially coarser mesh while achieving similar results. However, thin shells using 2D elements poses its own set of challenges. The bending energies introduce non-linearity, necessitating smaller step sizes or an higher number of iterations for time integration. Nevertheless, bending energies are necessary to create dynamic, cloth-like wrinkles with rich wavy motions.

In this chapter, we propose a method to adaptively coarsen and refine thin shells at runtime. This builds upon the rigidification method introduced by Mercier-Aubin et al. [78]. In all large scale simulations, some elements will inevitably be static or moving rigidly, and therefore wasting computational resources on the deformation of non-deforming degrees of freedom. In Figure 4.1, a horizontally oriented sleeve has large undeforming regions when it bends, but also features dynamic deformation near the elbow. Adaptive rigidification is efficient even on highly wrinkled models. On-demand elastification makes dynamic scenes properly dependent on their environment, allowing elements to alternate between a fast rigid representation and the accurate elastic counterpart.

Direct application of the previous adaptive rigidification method in the case of shells introduces new challenges. The bending deformation occurs over edges, making it impossible to evaluate using only the per triangle strain rate from membrane deformation. Additionally, bending along the discretized edges is unrelated to stretching. For example,

a flat hanging piece of cloth experiences only stretching in the planar directions due to gravity and does not express any bending deformation. Likewise, making each triangle an independent rigid body capable of bending would require more degrees of freedom per element, which defeats the purpose of adaptive rigidification. Because bending and membrane deformation are two independent types of deformation, we must consider both in our rigidification process as opposed to the previous work. Shells also require a new threshold for the bending motions to prevent premature rigidification, which would prevent elements from bending.

Collisions are a common mechanism for elastification, and for good performance we need an oracle that can identify a small local region of the mesh to elastify based on a threshold. The previous work uses a single iteration of preconditioned conjugate gradient, which can be problematic for shells, in particular, those with high membrane stiffness. If only a small number of contacts are active, the single iteration of preconditioned conjugate gradient is not enough to properly propagate the elastification and cause the bending component to wake up. We present different approaches, including a fast contact filter that approximate the behaviour of the impact on the nodes adjacent to the contact location.

In the following sections, we present a new adaptive simulation method for thin shells, which can produce results that closely resemble a fully elastic simulation while often having computation times more than an order of magnitude faster. We test our improvement to the oracle's contact handling by comparing our filter with the previous method for handling contacts as well as slower, but more accurate approaches to solve for the approximate contact velocities. We also discuss the difference between the curvature rate and the dihedral angle rate as well as some of their potential applications in the rigidification process.

4.2 Related Work

Remeshing [2] is often used to adaptively coarsen shells on the fly. This generates new positions using either subdivision rules [12, 63, 67, 24] or splitting coupled with edge flipping to improve the conditioning elements [53]. This type of technique generally

must handle the buckling or instabilities that can arise when remeshing curved surfaces with coarser triangles. One way to handle this is to use a measure the quality of the new elements to determine when to remesh them [87, 112]. These techniques are good for surfaces that are developable or that have large nearly flat regions, because these regions will remain indistinguishable from fine to coarse. Another approach refines basis functions [44, 47] to allow deformation in the coarse models by adding new degrees of freedom. In contrast, our method is orthogonal to these approaches and reduces the size of the system by simulating those regions of a mesh without dynamic deformations as rigid bodies, regardless if the region is flat or consisting of a dense collection of triangulated folds.

In contrast to mesh refinement, via remeshing, with the goal of simulating a mesh with the appropriate resolution, other approaches separate the problem into two distinct parts, i.e., a coarse simulation and a second mechanism for adding details. For instance, the approach of Rohmer et al. [93] first simulates a coarse models, and then refines the coarse mesh in a post-process refinement to add wrinkle details. A different strategy is taken by Müller et al. [81], where fine mesh vertices permit wrinkles via approximate constraints to a coarse simulation. There likewise exists other strategies and heuristics for upscaling simulations. The formation of wrinkles can be data-driven [79, 91], and machine learning techniques can predict physically plausible coarse-to-fine mappings of vertices [55]. Our approach instead starts with a high-resolution model and lets the simulation choose which parts need the degrees of freedom, and which regions can be evolved with only rigid motion.

While not directly related to adaptive simulation, we note that the simulation of stiff shells can be challenging because large stiffness ratios can lead to poorly conditioned systems. Physical fabrics typically have a non-linear stress-strain relationship that increase the stiffness rapidly as it deforms. One way to reduce the conditioning issue is to add strain limits. This can be done with correction strain constraints or projections after each integration step [92, 39]. In the case of inextensible cloth [32], constraints provide this strain limit in a quadrilateral mesh which avoids locking artifacts due to discretization. In contrast to these other techniques, our method makes strain limited regions rigid when

they maintain the limit (and have zero deformation rate in the orthogonal direction). This addresses the conditioning problems, i.e., when these strain limited regions are simulated as rigid.

Freezing and sleeping techniques are well known approaches to save computation during static periods of dynamic simulations. For instance, using an adaptive Hamiltonian can reduce the size of the simulations at run-time by disabling positional degrees of freedom of a system in regions where there is low momentum [5, 73]. Freezing can likewise be implemented in a hierarchy to dynamically tune the complexity of a viscoelastic body [109]. Different metrics can be used to initiate freezing, for example, by analyzing kinetic energies or velocities in simulations of rigid body contact [96, 33]. In the context of rigid bodies, another approach is to merge [26] collections of contacting bodies when they have zero relative velocity. This is similar to our work in that it reduces the degrees of freedom of the system, simplifies collision detection cost, while still permitting the collection to move as a rigid body.

Merging degrees of freedom into rigid bodies is likewise a key idea in adaptive rigidification of elastic solids [78], i.e., that it is valuable to permit regions of an elastic simulation to continue moving rigidly, as opposed to freezing degrees of freedom in the inertial frame.

4.3 Methods

We first extend adaptive rigidification to support triangular elements in a 3D setting. Thin shells feature two major hindrances that require a different approach for rigidification; the bending component cannot be analyzed from the deformation gradient, and the constant thickness of shells requires the definition of the deformation gradient to be changed for rigid rotations so as to yield a zero strain.

4.3.1 Simulation of Shells

For standard volumetric meshes, we define the deformation gradient $F = B\boldsymbol{x}$ from a kinematic mapping B and the generalized coordinates $\boldsymbol{x}$ of our element's vertices [18]. However, shells are not inherently volumetric. We use an approximation of the thickness by projecting out any deformation in the normal directions $\boldsymbol{n}$ of the faces [62]. This is akin to simulating a fake prismatic element. We do this projection by adding a new term η to the definition of the deformation gradient

$$F = B\boldsymbol{x} + \eta\boldsymbol{n}, \tag{4.1}$$

which then gives us the deformation gradient for shells. This $\eta \in \mathbb{R}^{9\times 3}$ term

$$\eta = \begin{bmatrix} \boldsymbol{n^0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{n^0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{n^0} \end{bmatrix} \tag{4.2}$$

is a block matrix where each zero $\mathbf{0}$ is a 3 by 1 column vector of zeros and $\boldsymbol{n^0}$ is the reference space normal of the element. This gives us contributions of the change in face normal from reference to world space for the deformation gradient. The B and η matrices are pre-computed at the start of the simulation with the fully elastic mesh and cached.

Baraff et al. [6] separate the potential energy into three energies. Both the shear and stretch energies use a function that maps a 2D projection of thin shells to their 3D world positions, effectively computing the energies as 2D problems, these are named membrane energies and only affect the planar deformations. Grinspun et al. [43] introduces a bending energy using the angle between two face normals, which is coupled with a stiffness parameter and creates resistance to bending. We use a similar approach by separating the membrane energies from the bending energies while they still remain coupled and interact with each other during the solve.

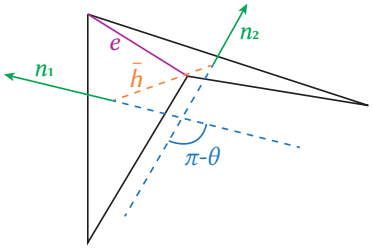


Figure 4.2: dihedral angle

4.3.1.1 Membrane Energy

To help ensure stability we use implicit time integration [34], so for the membrane energy density Ψ_e we must compute the gradient $\frac{\partial \Psi_e}{\partial F}$ and Hessian $\frac{\partial^2 \Psi_e}{\partial F^2}$. To get the internal forces $f_e = -V\frac{\partial \Psi_e}{\partial F}$ we multiply the gradient by the negative volume. In the case of a shell, the volume is the area of the triangle multiplied by a thickness parameter. In our examples, we tried various membrane energy formulations including neoHookean, Saint-Venant Kirchoff, and more. We note that we use the Grinspun et al. [43] membrane energy in our time comparisons because in our experience it allows a bigger step size with fewer Newton iterations.

4.3.1.2 Bending Energy

We compute the dihedral angle

$$\theta = \pi - \tan^{-1} \frac{\boldsymbol{e} \cdot (\boldsymbol{n}_1 \times \boldsymbol{n}_2)}{\boldsymbol{n}_1 \cdot \boldsymbol{n}_2}, \tag{4.3}$$

$$\Psi_b = \sum_e (\theta_e - \bar{\theta}_e)\|\bar{\boldsymbol{e}}\|/\bar{h}_e, \tag{4.4}$$

for each edge, where $\boldsymbol{n}_1$, and $\boldsymbol{n}_2$ are the adjacent face normals, and $\boldsymbol{e}$ is the edge vector (see Figure 4.2). We then use the resulting angles to compute the bending energies from Tamstorf et al. [102]. Here $\bar{\theta}_e$ is the dihedral angle at rest $\bar{h}_e$ is the length of the dual edge connecting the barycenters of two triangles at rest, and $\bar{\boldsymbol{e}}$ is the edge at rest.

We sum the bending energy to the membrane energy $\Psi = \Psi_b + V\Psi_e$ locally for each element and edge. Then, we find their gradient and Hessian to allow implicit time integration.

4.3.2 Simulating Coupled Rigid Bodies

In our mixed simulations, the elastic portions of the shell can have dynamic deformations while other portions replaced with rigid bodies do not allow for bending, with each connected component acting as a single thin rigid body. The adaptive model reduction uses the matrix

$$G = \begin{bmatrix} I & 0 \\ 0 & \Gamma \end{bmatrix}, \tag{4.5}$$

to map degrees of freedom from the mixed rigid-elastic system to the fully elastic model. Each vertex inside the rigidified body has a vector r that points to it from the center of mass in rigid body frame. From this we can build a matrix

$$\Gamma_i = \begin{bmatrix} I & -(R\,\boldsymbol{r}_i)^\times \end{bmatrix} \tag{4.6}$$

that maps the rigid body velocities to each individual vertex of the rigid body.

For every mesh, we group rigid elements using an adjacency graph of elements with shared edges for shells. We compute this graph prior to the simulation. Using a breadth first search algorithm, we navigate the graph to find the rigid connected component. Each connected component is a new rigid body in our simulation. When building the bodies, we compute properties like the center of mass p, the rotation $R \in SO(3)$, the angular velocity ω, the linear velocity v. These are set according to the state of the degrees of freedom rigidified to exactly preserve momentum under rigid motions.

On rigidification, we replace the elastic degrees of freedom in our system with the appropriate rigid degrees of freedom using the current step's G mapping from Equation 4.5.

Lumping elements together has an impact on the mass ratio of the system which can lead to a higher condition number. We scale the per Newton-step linear system to get the condition number down to a similar or better range as the fully elastic system. We use the sparse matrix scaling of Curtis et al. [29], which iteratively solves a least-square problem for the row and columns scaling factors. The inclusion of a scaling matrix therefore makes adaptive rigidification easier to integrate with iterative solvers.

4.3.3 Time Integration

Shells are subject to tricky scenarios like buckling that require more than one Newton iteration. Instead of a single linearized Newton step, we simulate shells by using the common optimization function

$$Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}) = \dot{\boldsymbol{x}}_{t+1}^T M \left(\frac{1}{2} \dot{\boldsymbol{x}}_{t+1} - \dot{\boldsymbol{x}}_t \right) + \Psi - h \, \dot{\boldsymbol{x}}_{t+1}^T \boldsymbol{f}_{\text{ext}}, \tag{4.7}$$

where M is the mass matrix, $\boldsymbol{f}_{\text{ext}}$ are the external forces such as gravity, and h is the step size. Here $\dot{\boldsymbol{x}}_t$ is the velocity vector at time t. For each time step we use a Newton-Raphson algorithm to solve for the next step velocities

$$\dot{\boldsymbol{x}}_{t+1} = \underset{\dot{\boldsymbol{x}}_{t+1} \in \mathbb{R}^n}{\arg\min} \, Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}). \tag{4.8}$$

We use a line search based on Armijo's rule where we simply reduce the Newton iteration's step length until we get an improved solution. We modify the optimization function to instead take as input the reduced system using the G matrix from Equation 4.5, as shown in Algorithm 3.

In the previous adaptive rigidification work [78], speedups were significant for a semi-implicit backward Euler integration method. Increasing the number of Newton iterations further increases the speedup as adaptive rigidification enhances the speed of each iteration, but has a linear overhead only prior to the solve.

4.3.4 Rigidification

If the membrane strain of an element and the bending deformation with its adjacent elements are constant over a period of time, we allow the element to become rigid. We monitor the change in deformations over several time steps to prevent momentary rigidification such as when a pendulum hits its highest point of swing, or when a cantilever plate hits its potential energy peak. Groups of adjacent elements are concatenated into a

Algorithm 3: Adaptively rigidifying Newton's method

input : Velocities $\dot{\boldsymbol{x}}_t$ at the start of the step
Function Q to minimize
Kinematic elastic-adaptive mapping G
Predicted reduction σ stop parameter
Scaling matrix S

output: Velocities $\dot{\boldsymbol{x}}_{t+1}$ after stepping

$\dot{x}_{t+1} \leftarrow \dot{x}_t$
for $i <$ *Newton iterations* **do**
  $A \leftarrow SG^T(\nabla^2 Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}))G$
  $\boldsymbol{b} \leftarrow -SG^T(\nabla Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}))$
  $\Delta\dot{\boldsymbol{x}} \leftarrow A^{-1}\boldsymbol{b}$
  $\alpha \leftarrow 1$
  while $Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1}) - Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1} + \alpha\Delta\dot{\boldsymbol{x}}) < \sigma\alpha Q(\dot{\boldsymbol{x}}_t, \dot{\boldsymbol{x}}_{t+1})$ **do**
    $\alpha \leftarrow \frac{1}{2}\alpha$
  end
  $\Delta\dot{\boldsymbol{x}}_c \leftarrow$ Contact Solve
  $\dot{\boldsymbol{x}}_{t+1} \leftarrow \dot{\boldsymbol{x}}_{t+1} + G^T(\alpha(\Delta\dot{\boldsymbol{x}}) + \Delta\dot{\boldsymbol{x}}_c)$
end

single rigid body using a connected component detection algorithm which is equivalent to the adaptive rigidification of 2D meshes.

We must monitor all deformation changes. For the membrane deformations, we compute the membrane strain rate

$$\dot{E}_{t+1} = \frac{E_{t+1} - E_t}{h}, \tag{4.9}$$

using finite differences of the Green strain $E = \frac{1}{2}(F^T F - I)$. At the beginning of a simulation, we initialize the cached strain of the previous step to be the identity matrix. We monitor $\dot{E}$, and rigidify the elements of a mesh when they satisfy a threshold over a set of frames. This is enough for two dimensional simulations or for tetrahedral meshes.

For the bending deformations, we could monitor the dihedral angle for every edge with two adjacent elements. But to make the thresholds discretization independent, we instead monitor the discrete curvature

$$\kappa_e = 2\frac{\theta_e}{\bar{h}_e}. \tag{4.10}$$

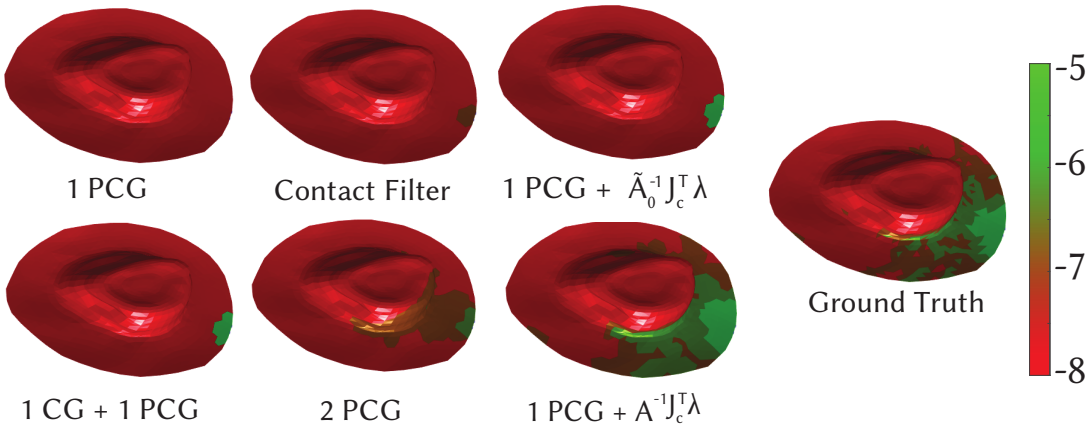


Figure 4.3: A tiny upward impulse applied to the tip of the hat elastifies regions according to the oracle's contact handler and different thresholds on a log scale.

Because we already need the dihedral angles for the bending energies, computing the discrete curvature is simply an element-wise multiplication by a constant value for each edge. We then approximate the discrete curvature rates with finite differences

$$\dot{\kappa}_{e,t} = \frac{\kappa_{e,t+1} - \kappa_{e,t}}{h}. \tag{4.11}$$

For an element to rigidify, it must satisfy both the threshold on the maximum of the per edge discrete curvature rates $\dot{\kappa}_{e,t}$, and the threshold on the membrane strain rate $\dot{E}$.

4.3.5 Elastification

Rigidified parts of the models must become elastic again on visually significant deformation due to elastic waves, changing contact forces or new contact forces. Within a rigid body, because the elastic degrees of freedom are replaced, we do not have information about the deformation rate of the elastic model when it is solved as rigid. We must predict the parts of a rigidified model that will have a deformation velocity in the next time step.

We can approximate the change in velocities due to elastic motions with a single preconditioned conjugate gradient iteration on the first Newton step with a fully elastic system

$$\Delta \dot{\boldsymbol{x}} = A^{-1}(\boldsymbol{b} - J_c^T \boldsymbol{\lambda}) + h\ddot{\boldsymbol{x}}_g, \tag{4.12}$$

where $J_c^T \boldsymbol{\lambda}$ are contact forces, A and $\boldsymbol{b}$ are the derivatives of the reduced system as shown in Algorithm 3, and where f_{ext} in Q does not include forces due to gravity. Instead, the gravity velocities $\ddot{\boldsymbol{x}}_g$ are injected into $\boldsymbol{b}$ (i.e., $\dot{\boldsymbol{x}}_t$ modified to be $\dot{\boldsymbol{x}}_t + h\ddot{\boldsymbol{x}}_g$) and also added to the change in velocities after the solve, because this helps the approximate solve produce a better solution.

New or moving contacts can also create elastification. In the oracle, we handle previously existing contacts, and new contacts differently. We concatenate the two types of constraints in $J_c^T \boldsymbol{\lambda}$, and add them to the system to solve. For existing contacts, we reuse the previous adaptive step impulses as contacts for the oracle. Like the original adaptive rigidification, we approximate only the new contacts for the oracle using a cheap bilateral constraint solve

$$J_{cn} \tilde{A}_0^{-1} J_{cn}^T \boldsymbol{\lambda}_n = J_{cn}(\tilde{A}_0^{-1} \boldsymbol{b} + \dot{\boldsymbol{x}}), \tag{4.13}$$

where the constraint Jacobian J_{cn} contains only the new contact constraints, and $\tilde{A}_0^{-1}$ is the precomputed 3-by-3 block diagonal inverse matrix at rest (i.e., an approximation of A^{-1} with zero coupling between vertices).

We use a preconditioner following that of Liu et al. [66] and used by Mercier-Aubin et al. [78], which is computed as $A_p = M + h^2 L$ were $L = B^T W B$ is the Laplacian of the mesh with a block-diagonal weight matrix W, where each block entry contains the per-element Young's modulus. We use a single iteration of a preconditioned conjugate gradient using an incomplete Cholesky factorization of A_p to approximate the velocities due to elastic motions. With the approximated velocities we can use the same formula for the membrane strain rate $\dot{E}$ from Equation 4.9 and pick a threshold for elastification. Compared to the original adaptive rigidification we also use the approximate velocities to compute the approximate discrete curvature rate $\dot{\kappa_e}$.

The original adaptive rigidification oracle, however, does not always produce a change in velocities significant enough to allow elastification for small localized impulses. This appears to be due to a lack of local propagation, and we speculate that this may stem from

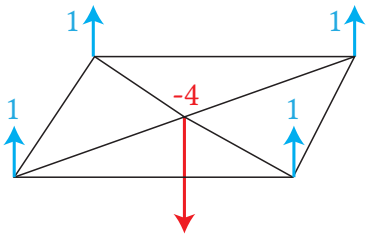


Figure 4.4: Contact filter at a degree 4 vertex.

how the Laplacian-based preconditioner propagates information globally and changes convergence behaviour during the first iterations of PCG.

We suggest a few cheap heuristics to warm-start the oracle with approximate contact velocities from the impulses of the bilateral solve in Equation 4.13. We evaluate these approaches only for new contacts and assume the old contacts were solved adequately during the time integration.

An intuitive solution for local propagation is to do a single conjugate gradient iteration without preconditioning, and then use the resulting approximate change in velocities due to contacts as an initial iterate $\Delta\dot{x}_0$ for the PCG solve. This only costs an extra linear pass over all the vertices. In a similar train of thought, we can use the precomputed $\tilde{A}_0^{-1}$ block diagonal matrix used for the bilateral solve and obtain approximate contact change in velocities,

$$\Delta\dot{\boldsymbol{x}}_0 = \Delta\dot{\boldsymbol{x}}_c + h\,\boldsymbol{f}_{\text{ext}}, \tag{4.14}$$

$$\Delta\dot{\boldsymbol{x}}_c = \tilde{A}_0^{-1} J_c^T \boldsymbol{\lambda}, \tag{4.15}$$

where $\boldsymbol{f}_{\text{ext}}$ is the precomputed external force vector. This option comes at the cost of sparse matrix multiplications with only the new contact constraints, and Lagrange multipliers.

This leads us to yet another strategy. While a single iteration of unpreconditioned conjugate gradient provides information about a local response, we can design an approach with a similar local propagation effect, without the need to iterate through all the vertices. We propose to warm-start the preconditioned conjugate gradient solve using approximate new contact velocities

$$\Delta\dot{x}_c = J_{cl}^T\,\boldsymbol{\lambda}_l, \tag{4.16}$$

with impulses $\boldsymbol{\lambda}_l$ obtained with a discrete Laplacian operator for local diffusion of the impulse over edges of the mesh with neighbouring non-colliding vertices. Figure 4.4 shows an example with a contact force at a center vertex, and the filter weights applied to the patch. We avoid applying the filter on neighbouring elements that are already in contact to avoid cancellation of the impulses for the warm-start when multiple neighbours are in contact. We add rows to the Jacobian J_c that contain the corresponding impulse normals aligned with the degrees of freedom of neighbouring vertices to obtain J_{cl}. This is the cheapest approach, with the cost being a simple sparse matrix multiplication of the Lagrange multipliers with the constraint Jacobian of new contacts.

While none of these approaches fundamentally alter the system to solve, they can significantly improve the accuracy of the approximate one iteration solve. Our warm-start techniques create initial solutions that mimic the mesh's behaviour under compression caused by a new contact with respect to the impact magnitude; a new contact will compress the mesh proportionally to how hard it hits a surface.

For existing contacts, we reuse the previous adaptive step impulses as contacts for the oracle, ensuring good continuity of the oracle's prediction with respect to the solve for the time integration.

4.3.6 Threshold Selection

Selecting the appropriate threshold for a simulation is akin to determining the acceptable level of error. In general, smaller thresholds result in a more conservative simulation. To optimize this process, we propose an approach that selects both thresholds simultaneously, using the concept of speed limits to establish a relationship between bending deformations and membrane stretches. However, for inextensible shells and other non-typical materials, decoupling the thresholds may be necessary and advantageous.

To form a heuristic relationship between discrete curvature rate and membrane strain rate thresholds we consider stretching and bending an initially straight vertical line segment as shown in Figure 4.5. A small membrane stretch increases the length by d_s as measured in the vertical direction from the top and bottom of the original line segment.

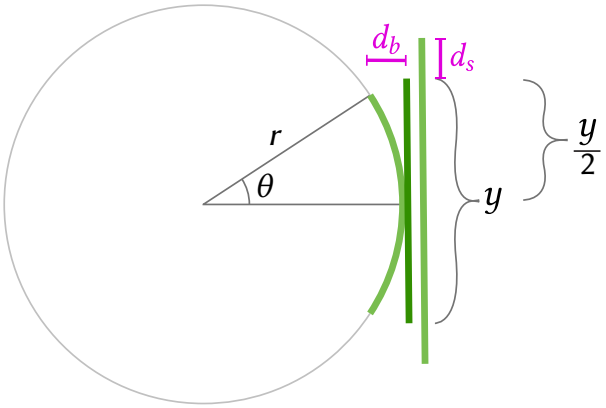


Figure 4.5: A green line segment of length y is shown wrapped around a circle with radius r, and stretched in the vertical direction. Equating displacements d_b and d_s leads us to equivalent thresholds for discrete curvature rate and membrane strain rate. While bending and membrane deformations can be independent, this makes the threshold selection easier.

A small bend from flat to a curvature $\kappa = 1/r$ can be seen as creating a horizontal displacement of d_b at the top and bottom of the line segment when it wraps around a circle of radius r. This horizontal distance $d_b = r - r\cos(\theta)$.

Because the line wraps around the circle, we have

$$r\theta = \frac{y}{2}, \tag{4.17}$$

or $\theta = \kappa y/2$. If we approximate d_b with a Maclaurin expansion for the cos function up to and including the quadratic term, we have

$$d_b = r - r\left(1 - \frac{\theta^2}{2} + O(\theta^4)\right). \tag{4.18}$$

Ignoring higher order terms and substituting the angle in Equation 4.18 using Equation 4.17 we obtain

$$d_b \approx \frac{y^2}{8r}, \tag{4.19}$$

or $d_b \approx \kappa y^2/8$. Thus, we know how displacement d_b grows as the discrete curvature is increased from zero. We can likewise see this as the displacement observed for a discrete curvature rate change of $\dot{\kappa}$ over a small time step h, i.e., $d_b \approx h\dot{\kappa}y^2/8$.

Now, considering a stretch rate of $\dot{s}$ in the vertical direction over a small time step h we have $d_s = h\dot{s}y/2$. Equating d_b and d_s and cancelling terms gives $\dot{\kappa} = \frac{4}{y}\dot{s}$. Here, for simulating arbitrary models, we interpret y as the diameter of the mesh at rest. Finally, because the membrane thresholds are for the squared Frobenius norm of the Green strain rate (i.e., see $\tau_m = \dot{s}^2$ if we choose the stretch rate above to be at the threshold for a mesh at rest), we can compute the discrete curvature rate threshold τ_b in terms of the membrane strain rate threshold τ_m using the formula

$$\tau_b = \frac{4}{y}\sqrt{\tau_m}. \tag{4.20}$$

The concept of discrete curvature, while resolution independent, can lead to large rates for highly bent elements. This is because the 2D curvature is derived from the inverse radius of the osculating circle, which becomes small for a large bend, producing large curvatures. When dealing with large curvatures, even a slight change in angle on a highly bent edge can generate significant curvature rates. While the resolution-independence is a benefit for measuring curvature rates in meshes with different discretizations, the high rates in areas of high curvature can make the threshold selection challenging. Discrete curvature rates are better suited for fine models where per-edge angles are flatter. In contrast, for a coarse model with elements of roughly consistent size, monitoring dihedral angles is a reasonable alternative, but it is much harder to choose a dihedral angle rate threshold for meshes with different element sizes. An interesting possibility to explore in the future would be not to measure bending at the edges, but instead as a property of the surface.

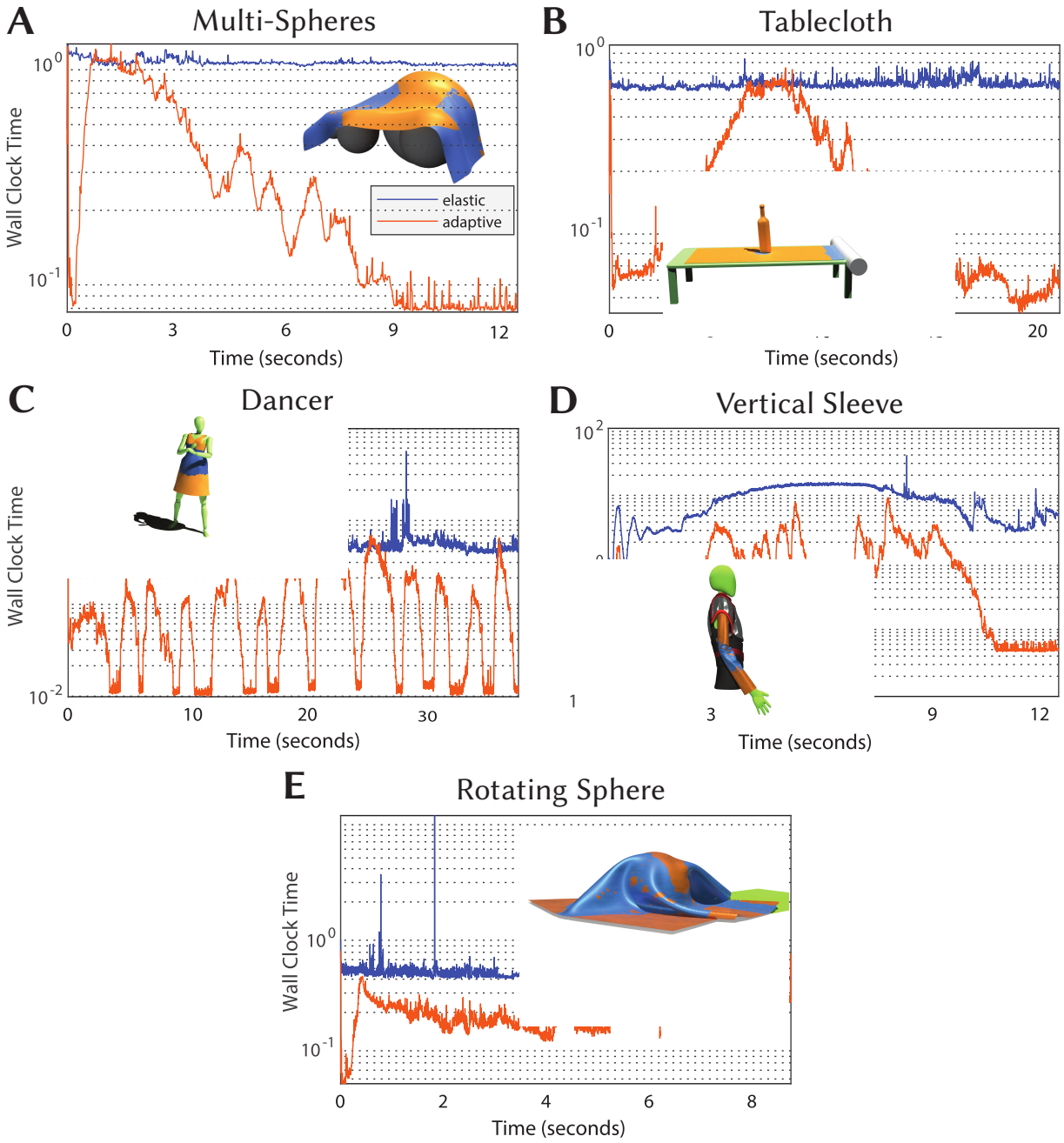


Figure 4.6: Set of comparisons all using seconds as the unit for wall clock times.

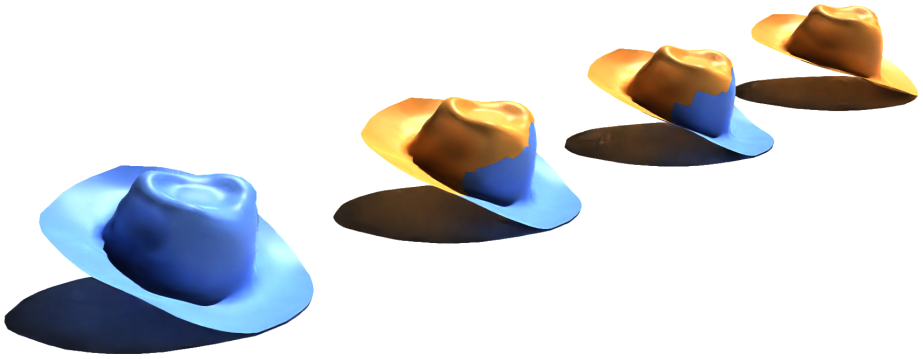


Figure 4.7: From left to right: elastic, adaptive with contact filters, adaptive with a CG iteration before the PCG iteration, and the original adaptive rigidification algorithm.

4.4 Results

Our oracle will produce elastification due to a new contact at a single vertex, and efficiently propagate the impact without expensive extra computation. In Figure 4.7, for instance, once the brim of the hat hits the floor, we obtain appropriate elastification of the deforming elements near the initial contact. The elements adequately elastify as the elastic wave propagates through the hat, while preserving rigidly moving parts at the opposite side of the model.

Our method allows locally bending or deforming regions to remain elastic while resting contacts and rigidly moving chunks of elements rigidify. For instance, a cloth hanging on a sphere in Figure 4.6-A has elastic regions where its edges are dangling and has a large central rigidified region where the cloth is in static equilibrium.

We believe our method is the first adaptive technique to completely coarsen densely wrinkled regions while preserving the fine definition of the cloth's curvature. Figure 4.1 and Figure 4.6-D show excellent examples of this. These examples feature sleeves with different cloth stiffness parameters, and we observe rigid motion of rigidified wrinkles during arm motions. We note that care is necessary when setting aggressive (higher) thresholds because this can lead to visual artifacts. For example, there appears to be a small bump on the sleeve caused by an increase in velocity near the end of the vertical sleeve simulation partially due to our threshold selection, but also due to the choice of energies pushing the wrinkles downwards back to rest as the arm unbends.

Because our shell implementation natively works with tetrahedra (it uses the same 3 by 3 deformation gradient thanks to the normal projection), we can easily mix elastic solids and shells. In Figure 4.6-B we present a tablecloth modelled as a shell and wine bottle modelled as a tetrahedral mesh. The tablecloth has a large moving region that remains rigidified while pulled, and likewise demonstrates local deformation near the region stretched due to frictional contacts from both the table and the bottle.

Figure 4.6-E shows a cloth on a high friction spinning sphere which is inspired by an example of Bridson et al. [17]. The floor and an adjacent obstacle are frictionless. While large portions of the mesh rigidify as the cloth spins on the sphere, the cloth also continues to exhibit significant dynamics because there is no self contact in the simulation. We believe that adaptive rigidification would work well with both penalty [17] and barrier [64] based contacts.

4.4.1 Speed

Table 4.1 shows performance measurements for the examples from the figures in this chapter. The simulations were timed including and excluding contacts to give a fair assessment of the wall clock times. We note some improvements between $2x$ and $13x$ on scenes that were especially designed to generate elastification and dynamic motions. We ran the simulations on a Windows 10 PC with an Intel Core i7-6700K processor, and 64 GB of DDR3 RAM. The simulator is a fork of the publicly available github from the adaptive rigidification project, with the core system in Matlab, and critical snippets implemented in Mex C++.

Adaptive rigidification allows performance improvements more than an order of magnitude faster than the non-adaptive meshes at various steps of the simulations. We also note that in any scene, the overhead of the single PCG iteration and rigid body building remains negligible, even when fully elastic. We present our time comparisons using log scales for fairness. In Figure 4.1 we see that rigidification can accurately detect local deformation and maintain steady improvements in computation time that is more than an order of magnitude faster than a fully elastic model. The region of deformation is local

near the elbow, where wrinkles form and rigidify, hence creating a coarser model of the mesh under rigid motions while preserving the fine details of the wrinkles.

4.4.2 Conditioning

Mixing rigid bodies with elastic elements increases the mass ratio, leading to poorly conditioned systems during time implicit integration. However, we show that a simple scaling technique [29] can effectively reduce the condition number of our system, resulting in faster simulations. This scaling $SA\,x = S\,b$ makes the condition number more competitive with the fully elastic system, while also benefiting from the reduced number of degrees of freedom inherent to adaptive rigidification.

While conditioning is generally not a problem for direct solvers, it is beneficial for iterative solvers. Likewise, while Jacobi preconditioning can easily resolve mass ratios, it is not as beneficial for stiffness ratios. Nonlinear elastic materials undergoing large deformations can also exhibit poor conditioning, as demonstrated by the streched cross model in Figure 4.8-A. Scaling will benefit many iterative solvers, and such solvers may be preferred or needed for bigger scenes. Figure 4.8-A shows that the Curtis et al. [29] scaling (CS) outperforms Jacobi scaling (JS) when it comes to reducing the condition number of the adaptive system. See that the condition number also sharply drops as the rigid chunks increase in size. This suggests that there is significant potential for optimization when using adaptive rigidification in conjunction with larger rigid chunks, and poorly shaped elements.

In Figure 4.8-B we compare the time for each step of the stretched cross simulation example using a direct solve via LDL decomposition and scaling to that of a PCG solve with scaling and tolerance of $1e-4$. For efficiency, our PCG solve reuses the oracle's preconditioner with kinematic mapping to the coupled rigid-elastic system, i.e., SG^TA_pG. As expected, stopping the iterative solver before full convergence provides big savings in the computation time. Nevertheless, while large systems can benefit from an iterative solver, it can be beneficial to switch to a direct solver for the small systems that arise when there is extensive rigidification.

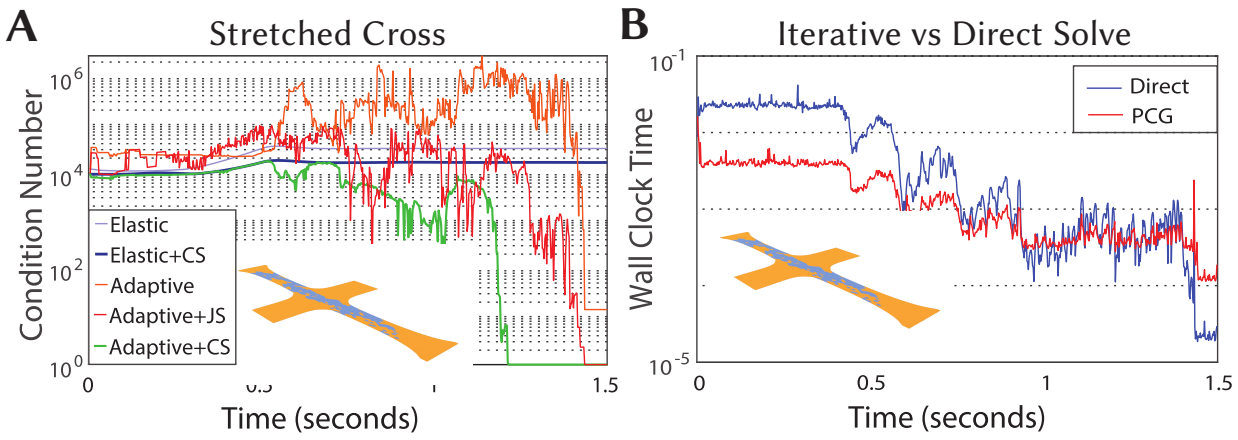


Figure 4.8: A: comparing condition numbers with or without scaling. B: comparing a direct solver to an iterative solver.

4.4.3 Strain Limiting

Strain limiting prevents deformation above or below a strain limit. This models physical behaviour of deformable objects acting almost rigidly when under heavy loads. This has the added benefit of reducing the change in deformation, further increasing the rate of rigidification, and the speed of the simulation. We implement strain limiting with the singular value decomposition of $F = USV^T$ and clamp the principal stretches $s \in S$ like Wang et al. [111]. This formulation of strain limiting is particularly elegant as it allows us to reuse the singular value decomposition when computing materials like the corotational energies. Moreover, we can save on the SVD computation for the strain limiting of rigidified elements as they are non-deforming.

By limiting the strain, we also limit the rates of deformations, hence increasing the rate of rigidification. Figure 4.6-A shows a piece of cloth with strain limiting where singular values are clamped between 0.90 and 1.1. A large patch of stretched cloth quickly rigidifies in the middle of the spheres, while motion is allowed where needed. The final result is visually indistinguishable from the fully elastic simulation as we present it in the supplemental video.

In Figure 4.6-C, we chose a crinolette type of dress for the dancer to show that rigidification natively handles stiff cloth/plastic even when the strain limits are reached. The adaptive approach is also significantly faster than the elastic simulation at almost any point in time. The mannequin model and dress are from Narain et al. [87].

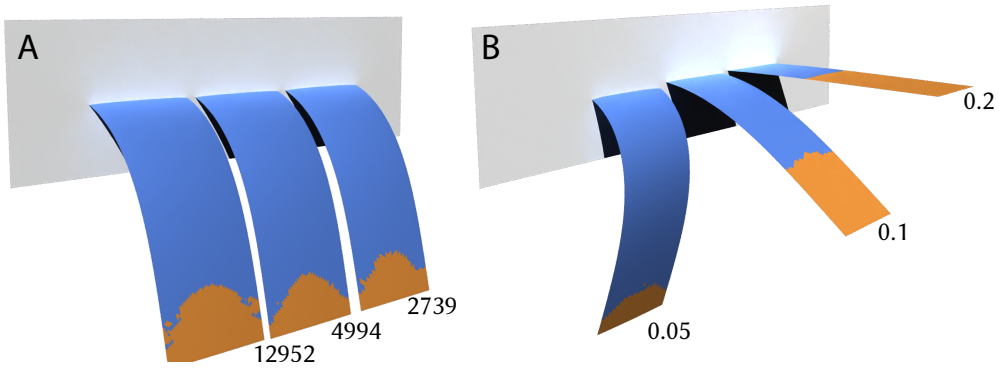


Figure 4.9: A: The rigidification patterns are consistent across resolutions. We show the total number of elements in the shell cantilevers next to them. B: The simulation retains its quality regardless of the material properties like the thickness. We list the respective thickness of each shell cantilever next to them.

4.4.4 Oracle

In Figure 4.9-A, we see that using our bending threshold correspondence yields similar rigidification patterns independent of the resolutions of our meshes. This holds across all of our scenes. We can also pick custom bending thresholds for incompressible materials or if fine tuning is needed. The rigidification process is material independent as it relies on the speed of deformation. Therefore, choosing the thresholds for the membrane stretches, and then using our equivalence for the bending thresholds yield a consistent quality of simulation regardless of the material used. In Figure 4.9-B, we compare different material thicknesses and note that the regions of elastification intuitively match their regions of deformation.

We compare various approaches to improve the new contact elastification regions in Figure 4.3. Using the inverse matrix A^{-1} to find the contact velocities from the bilateral solve's impulses yields the closest elastification region to the ground truth of elastic propagation. However such approach is overly costly. The contacts from the bilateral solve are already approximations, and the oracle's contact velocities need not be accurate. We only require a solution that generates enough elastification to preserve momentum under contact. A reasonable alternative is to instead use the precomputed block diagonal inverse matrix $\tilde{A}_0^{-1}$ which has a notion of the geometry of our system at rest. Using this method yields an elastification region equivalent to doing a single CG iteration prior to

Table 4.1: We generated and simulated scenes with varying numbers of elements, and provide the full simulation times, as well as the simulation times without contacts (NC).

Scene	Vertices	Elements	NC Time Adaptive(s)	NC Time Elastic(s)	Time Adaptive(s)	Time Elastic(s)	Speedup Factor
Dancer	2000	3861	289.32	977.24	815.51	3318.06	3.37x
Multi-Spheres	8437	16493	222.13	682.05	3653.78	14230.15	3.07x
Tablecloth	9896	20440	483.23	2433.54	10629.65	34496.75	5.04x
Horizontal Sleeve	2450	4830	79.66	1060.95	10322.95	43116.48	13.31x
Hat	1412	2760	12.98	18.49	28.20	42.04	1.42x
Vertical Sleeve	7777	15488	916.96	5524.75	5722.66	41937.27	6.03x
Rotating Sphere	4602	8932	707.48	1546.10	21398.00	47070.00	2.19x

the PCG iteration. Using two PCG iterations gives slightly worse local propagation, but better overall distant elastification.

In practice, most of the momentum loss on contact comes from simulating a contact as rigid when it should be deforming. The fast contact filter generates just enough elastification to handle typical simulations well, while the precomputed diagonal matrix multiplication is more appealing for fine contact handling like the soft touch of a feather. In Figure 4.7 the contact filter allows visually consistent simulation of the hat even with a small impact, and only requires the diffusion of an impulse over a few vertices as opposed to a CG iteration over all degrees of freedom or a sparse matrix multiplication with A. We see that using only a single PCG iteration without any of our contact handling approaches does not allow elastification of the hat on first contact, which leads to a loss in momentum.

Our approach is designed to work with any standard collision detection and handling technique. We have implemented bounding volumes, signed distance functions, and use projected Gauss-Seidel to solve for contact impulses. We can see from Table 4.1 that the rigidification has an impact on collision handling times as it speeds up the assembly of the Delassus operator for a PGS solver. Other approaches such as that of Verschoor et al. [110] would likely retain similar collision handling speeds regardless of rigidification.

4.5 Discussion and Limitations

Although our approach is currently efficient for wrinkled, non-deforming parts, our approach lacks the ability to remesh actively deforming regions. Nonetheless, we believe that adaptive rigidification of shells could complement remeshing, resulting in an even more efficient oracle by reducing the size of our linear pass over elements, as well as the coarsening of actively deforming elements in wrinkle-free areas. Speedups are dependent on the thresholds (level of accuracy), and the dynamics of the scenes, as opposed to remeshing where the speedups are dependant on the quality of the coarsening, and the continuous shape of the model. While adaptive rigidification cannot coarsen a flat actively stretching piece of cloth, remeshing would be able to simulate this deformation with a low number of degrees of freedom. Likewise, remeshing cannot coarsen dense triangle folds to a constant number of degrees of freedom as we do when the folds are moving rigidly.

Our approach to handle new contact reduces the likelihood of missing elastification, but some motions can still be problematic. A slow constant creep type of motion that accumulates over time while remaining below the elastification threshold would cause deformation on a fully elastic model, but could not deform a rigidified body. However, such cases are perhaps rare and can otherwise be addressed by adjusting thresholds or designing new custom solutions.

While not directly related to rigidification, our contact handling is slow for large number of contacts because the assembly of the Delassus operator creates a bottleneck. Using a contact handling method like that of Verschoor et al. [110] does not require this assembly and would greatly speed up the simulations. There is also an opportunity to prune contacts on rigid bodies by considering only 3 contacts per body, and diffusing the impulses on the rigid body for the oracle's elastic contacts to reduce the number of constraints during the time integration. Likewise, the internal forces of rigidified elements could be cached on rigidification, and rotated with the rigid body properties to save on the computation of the energy derivatives.

The BFS algorithm to build rigid bodies is the same as the original adaptive rigidification with a minor modification to sequentially iterate through mixed geometry. Using

parallel algorithms like the ones described by Zhang et al. [115] to find connected components would further reduce the overhead of adaptive rigidification.

Finally, we note that poorly conditioned elements will likely create large approximate changes in velocity. Thus, an adaptive threshold that considers conditioning could be created to uniformly set the tolerance across the meshes.

4.6 Conclusions

In this chapter, we present an extension of adaptive rigidification to target thin shells. By adding a second rigidification criterion based on the curvature rate, we achieve the benefits of rigidification with minimal overhead, similar to the tetrahedral version. Our approach leverages computations from different parts of the simulation, such as the dihedral angles of the bending energy, so as to reduce redundant work and improve efficiency. To further enhance the performance of the elastification oracle in the presence of small contact patches, we incorporate a fast contact filter, and explore other valuable approaches for diffusing contact information during the oracle solve. Finally, we demonstrate that scaling the per-Newton step system can significantly reduce the condition number, making it competitive with that of the fully elastic scaled system while also benefiting from its reduced size.

4.7 From Approximations to Ground Truth

The preceding two chapters were centered on generating reliable approximations of physical motions. Both methods relied on an oracle to identify regions of deformation within the reduced parts of deformable bodies. However, this oracle necessitates careful threshold tuning, often requiring a significant degree of intuition. This led us to a fundamental and pivotal question: What if we could eliminate the need for this oracle altogether?

Achieving this goal requires solutions that are not mere approximations but are instead accurate representations of physical behavior. Without such accuracy, regions with low strain rates could potentially remain undeformed indefinitely. This challenge has driven

us to develop a fundamentally different approach to solving motion, inspired by both geometric and algebraic multigrid methods.

To align our work with practical applications, we considered how to inject ideas of rigidification into extended position-based dynamics (XPBD), which is favored by our industrial partner for their simulations. XPBD is widely used in real-time applications due to its ease of parallelization, making it a fast alternative to methods like the matrix factorization discussed in subsection 2.1.1. This is especially important for our industrial partner working on surgery simulators connected to a haptic feedback device. For these applications, the physical simulation must be not only realistic but also computationally efficient, capable of delivering hundreds of frames per second. Likewise, it is unclear how the oracle could be implemented in XPBD as the hessian matrix needed for the oracle is not computed during the simulations. We speculate that using a few initial Gauss-Seidel iterations could lead to a reasonable oracle, but such a system would require even more parameter tuning.

We developed a hierarchical solver that leverages a sequence of progressively refined resolutions to accelerate convergence. In this context, the resolutions correspond to different rigidification patterns. Initially, we experimented with v-cycle methods, but the results were not ideal. We found that starting with rigid components and gradually introducing elasticity proved to be highly effective. Because of this, our solver deviates from traditional multigrid methods, as the prolongation (or refinement) operator is seldom used. The goal with this hierarchical solver is to iteratively solve the full system, enabling a more efficient and accurate simulation process. Our approach solves some of the parameter tuning issues from adaptive rigidification while also providing accurate solutions. In the following chapter, we present how to automatically and efficiently generate the layers for this new type of solver and we introduce the novel concept of residual velocities to preserve vibrations within rigidified patterns.

References for Chapter 4

[2] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. *Recent Advances in Remeshing of Surfaces*. In *Shape Analysis and Structuring*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pages 53–82. DOI: 10.1007/978-3-540-33265-7_2.

[5] S. Artemova and S. Redon. Adaptively Restrained Particle Simulations. *Physical Review Letters*, 109:1–5, 19, Nov. 2012. DOI: 10.1103/PhysRevLett.109.190201.

[6] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1998, pages 43–54, New York, NY, USA. Association for Computing Machinery, 1998. DOI: 10.1145/280814.280821.

[12] J. Bender and C. Deul. Efficient Cloth Simulation using an Adaptive Finite Element Method. In *Virtual Reality Interactions and Physical Simulations*, pages 21–30, Darmstadt, Germany. Eurographics Association, Dec. 2012. DOI: 10.2312/PE/vriphys/vriphys12/021-030.

[17] R. Bridson, R. Fedkiw, and J. Anderson. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Transactions on Graphics*, 21(3):594–603, July 2002. DOI: 10.1145/566654.566623.

[18] M. Bro-Nielsen and S. Cotin. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Computer Graphics Forum*, 15(3):57–66, 1996. DOI: 10.1111/1467-8659.1530057.

[24] F. Cirak, M. Ortiz, and P. Schröder. Subdivision Surfaces: A New Paradigm for Thin-shell Finite-element Analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–2072, 2000. DOI: 10.1002/(SICI)1097-0207(20000430)47:12<2039::AID-NME872>3.0.CO;2-1.

[26] E. Coevoet, O. Benchekroun, and P. G. Kry. Adaptive Merging for Rigid Body Simulation. *ACM Transactions on Graphics*, 39(4):1–13, Aug. 2020. DOI: 10.1145/3386569.3392417.

[29] A. R. Curtis and J. K. Reid. On the Automatic Scaling of Matrices for Gaussian Elimination. *IMA Journal of Applied Mathematics*, 10(1):118–124, Aug. 1972. DOI: 10.1093/imamat/10.1.118.

[32] E. English and R. Bridson. Animating Developable Surfaces using Nonconforming Elements. *ACM Transactions on Graphics*, 27(3):1–5, Aug. 2008. DOI: 10.1145/1360612.1360665.

[33] K. Erleben. *Stable, robust, and versatile multibody dynamics animation*. PhD thesis, University of Copenhagen, 2004.

[34] O. Etzmub, M. Keckeisen, and W. Straber. A Fast Finite Element Solution for Cloth Modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 244–251, USA. IEEE Computer Society, 2003. DOI: 10.1109/PCCGA.2003.1238266.

[39] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient Simulation of Inextensible Cloth. *ACM Transactions on Graphics*, 26(3):49–57, July 2007. DOI: 10.1145/1276377.1276438.

[43] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder. Discrete Shells. In D. Breen and M. Lin, editors, *Proceedings of the 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '03, pages 62–67, San Diego, California. The Eurographics Association, 2003. DOI: 10.2312/SCA03/062-067.

[44] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002. DOI: 10.1145/566654.566578.

[47] F. Hahn, B. Thomaszewski, S. Coros, R. W. Sumner, F. Cole, M. Meyer, T. DeRose, and M. Gross. Subspace Clothing Simulation Using Adaptive Bases. *ACM Transactions on Graphics*, 33(4):1–9, July 2014. DOI: 10.1145/2601097.2601160.

[53] X. Jiao, A. Colombi, X. Ni, and J. C. Hart. Anisotropic Mesh Adaptation for Evolving Triangulated Surfaces. In P. P. Pébay, editor, *Proceedings of the 15th International Meshing Roundtable*, pages 173–190, Berlin, Heidelberg. Springer Berlin Heidelberg, 2006. DOI: 10.1007/978-3-540-34958-7_11.

[55] L. Kavan, D. Gerszewski, A. W. Bargteil, and P.-P. Sloan. Physics-Inspired Upsampling for Cloth Simulation in Games. In *ACM Transactions on Graphics*, volume 30 of number 4, pages 1–10, New York, NY, USA. Association for Computing Machinery, Jan. 2011. DOI: 10.1145/1964921.1964988.

[62] D. I. Levin. Physics-based animation lecture 6: cloth simulation, Oct. 2020.

[63] L. Li and V. Volkov. Cloth Animation with Adaptively Refined Meshes. In *Proceedings of the Twenty-Eighth Australasian Conference on Computer Science*, ACSC '05, pages 107–113, Newcastle, Australia. Australian Computer Society, Inc., 2005.

[64] M. Li, D. M. Kaufman, and C. Jiang. Codimensional Incremental Potential Contact. *ACM Transactions on Graphics*, 40(4):1–24, Jan. 2021. DOI: 10.1145/3450626.3459767.

[66] T. Liu, S. Bouaziz, and L. Kavan. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Transactions on Graphics*, 36(4):1–16, May 2017. DOI: 10.1145/3072959.2990496.

[67] C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, The University of Utah, Jan. 1987.

[73] P.-L. Manteaux, F. Faure, S. Redon, and M.-P. Cani. Exploring the Use of Adaptively Restrained Particles for Graphics Simulations. In *Proceedings of the 10th Workshop on Virtual Reality Interaction and Physical Simulation*, pages 17–24, Lille, France. Eurographics Association, Nov. 2013. DOI: 10.2312/PE.vriphys.vriphys13.017-024.

[77] A. Mercier-Aubin and P. G. Kry. Adaptive Rigidification of Discrete Shells. *Proc. ACM Comput. Graph. Interact. Tech.*, 6(3):1–17, Aug. 2023. DOI: 10.1145/3606932.

[78] A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124.

[79] E. Miguel, D. Bradley, B. Thomaszewski, B. Bickel, W. Matusik, M. A. Otaduy, and S. Marschner. Data-Driven Estimation of Cloth Simulation Models. *Computer Graphics Forum*, 31(2pt2):519–528, May 2012. DOI: 10.1111/j.1467-8659.2012.03031.x.

[81] M. Müller and N. Chentanez. Wrinkle Meshes. In M. Popovic and M. Otaduy, editors, *Proceedings of the 2010 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '10, pages 85–92, Madrid, Spain. The Eurographics Association, 2010. DOI: 10.2312/SCA/SCA10/085-091.

[87] R. Narain, A. Samii, and J. F. O'Brien. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Transactions on Graphics*, 31(6):1–10, Nov. 2012. DOI: 10.1145/2366145.2366171.

[91] T. Popa, Q. Zhou, D. Bradley, V. Kraevoy, H. Fu, A. Sheffer, and W. Heidrich. Wrinkling Captured Garments Using Space-Time Data-Driven Deformation. *Computer Graphics Forum*, 28(2):427–435, 2009. DOI: 10.1111/j.1467-8659.2009.01382.x.

[92] X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour. In *Proceedings of Graphics Interface*, GI '95, pages 147–154, Quebec, Quebec, Canada. Canadian Human-Computer Communications Society, 1995. DOI: 10.20380/GI1995.17.

[93] D. Rohmer, T. Popa, M.-P. Cani, S. Hahmann, and A. Sheffer. Animation Wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. *ACM Transactions on Graphics*, 29(6):1–8, Dec. 2010. DOI: 10.1145/1882261.1866183.

[96] H. Schmidl and V. J. Milenkovic. A Fast Impulsive Contact Suite for Rigid Body Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):189–197, Mar. 2004. DOI: 10.1109/TVCG.2004.1260770.

[102] R. Tamstorf and E. Grinspun. Discrete bending forces and their Jacobians. *Graphical Models*, 75(6):362–370, 2013. DOI: 10.1016/j.gmod.2013.07.001.

[109] M. Tournier, M. Nesme, F. Faure, and B. Gilles. Seamless Adaptivity of Elastic Models. In *Proceedings of Graphics Interface*, GI 2014, pages 17–24, Montréal, Québec, Canada. Canadian Human-Computer Communications Society, 2014.

[110] M. Verschoor and A. C. Jalba. Efficient and Accurate Collision Response for Elastically Deformable Models. *ACM Transactions on Graphics*, 38(2):1–20, Mar. 2019. DOI: 10.1145/3209887.

[111] H. Wang, J. O'Brien, and R. Ramamoorthi. Multi-Resolution Isotropic Strain Limiting. *ACM Transactions on Graphics*, 29(6):1–10, Dec. 2010. DOI: 10.1145/1882261.1866182.

[112] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, and J. F. O'Brien. Dynamic Local Remeshing for Elastoplastic Simulation. *ACM Transactions on Graphics*, 29(4):1–11, July 2010. DOI: 10.1145/1778765.1778786.

[115] Y. Zhang, A. Azad, and A. Buluç. Parallel Algorithms for Finding Connected Components using Linear Algebra. *Journal of Parallel and Distributed Computing*, 144:14–27, 2020. DOI: 10.1016/j.jpdc.2020.04.009.

Chapter 5

A Multi-layer Solver for XPBD

This chapter is a from a published paper: A. Mercier-Aubin and P. G. Kry. A Multi-layer Solver for XPBD. *Computer Graphics Forum*, 43(8), 2024. DOI: 10.1111/cgf.15186. It is provided as is with only minimal editorial modifications. We also provide videos in the supplemental materials.

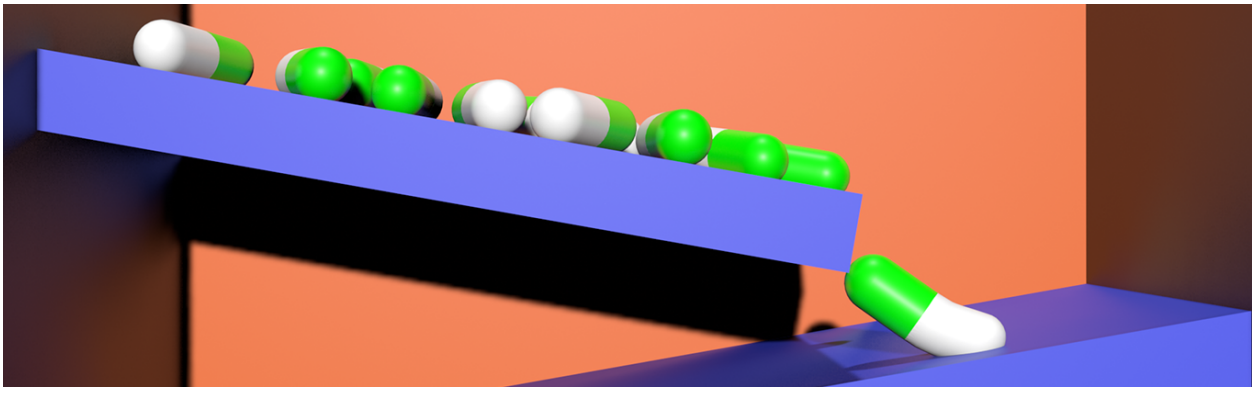


Figure 5.1: The pills have green parts are stiff and white parts that are soft. Due to independent rigid motions of different sliding pills, and mixed rigid-elastic contacts, we obtain an efficient and stable simulation of this contact heavy scene.

5.1 Introduction

Real-time simulations of soft bodies is an important component in many interactive applications, such as training simulators and videos games. In these applications, the compute resources are often limited, and must be shared between rendering, application logic, and the computation of physics. Real-time applications typically requires steady frame rates for the rendering of intricate geometry often leaving tight budgets for the computation

of physics. In this setting, deformation computations must be interactive, stable, and accurate for optimal immersion.

Step and project methods like XPBD are popular for the simulation of soft or rigid bodies as they allow fast stable simulations. However, the convergence of the solver is slow due to local propagation of information through Jacobi or Gauss-Seidel iterations. Hence, while these methods offer fast computation for time stepping, the slow convergence can lead to results that poorly approximate the expected physical behaviors of a system. To improve propagation, long-range attachments create additional constraints to couple distant elements. For instance, coupling the ends of a chain with a distance constraint can locally propagate information between the two ends and therefore improve convergence. Identifying where extra constraints are needed can be challenging or specific to a given scenario, and overall, the extra constraints increase the complexity of the system and the cost of the solve.

We introduce a multi-layer method for soft body simulation. Drawing inspiration from multigrid methods, the concepts of long-range constraints, and adaptive rigidification, we automatically generate various resolutions as mixes of rigid and elastic components with varying degrees of freedom. Our work derives from an opportunity to reduce models based on the current simulation state rather than the geometry or representation of our models. Through a coarse-to-fine sequence of solver iterations using these reduced models, the different couplings of elastic and rigid parts propagate distant information, improving the convergence of the XPBD solve. We handle contacts as both rigid and elastic throughout a time step, allowing efficient propagation of impulses. Our method has been tested across a range of scenarios, including contact-rich scenarios, purely rigid motions, and global deformation challenges. The results demonstrate improved convergence in all of our examples. We further explore the performance of different input parameters such as rigid patterns, number of iterations per layer, and the number of layers to provide deeper insights into the behavior of our method.

5.2 Related Work

One of the most important contributions to physics-based animation with respect to efficient time integration is unequivocally position-based dynamics (PBD) [83]. The method proposes a Gauss-Seidel-like solve for the motions of constrained particles by projecting linearized constraints one at a time. Bender et al. [13] demonstrate the applicability of PBD continuum elasticity simulation by modeling constraints as per-element elastic potentials. The extended position based dynamics (XPBD) [70] framework addresses one of the early, yet major shortcomings of traditional PBD. Without the compliance term, elastic models are infinitely stiff and too many iterations causes them to behave as rigid bodies. In contrast, when rigid bodies are desired, XPBD can still be applied, noting that most of the displacement occurs in the fast symplectic stepping prior to the constraint solve, which instead deals with environmental interactions, such as integration of joints, mixed bodies and more [84].

Multigrid solvers find solutions to systems of equations by first eliminating, i.e., smoothing, high-frequency errors in the solution iterates. The remaining low-frequency error is then eliminated by solving a reduced version of the original problem, where such low frequencies become high frequencies, revealing its recursive structure. Applied to linear systems, a multigrid solver typically features multiple resolutions, i.e., levels, of the full space problem (e.g., constructed by simplification [65]), with the intention of doing a few Jacobi or Gauss-Seidel iterations at each level as error smoothing agents. The exponential reduction in problem size between the hierarchy's levels yields fast convergence to the solution.

Xian et al. [113] present a linear multigrid solver for physics-based animation based on projective dynamics [15] using Newton-Raphson iterations, where resolutions are created from furthest point samplings of the high-resolution simulated mesh. Coarser level point samples act as linear blend skinning (LBS) handles for the immediate finer level points, with simulation mesh vertices as the finest level. By clamping the LBS weights to discrete binary values, they obtain restriction and prolongation operators which encourage linear system sparsity at coarser levels, yielding impressive computational efficiency. Unlike

PBD and XPBD [68], the projected dynamics approach is tailored to the primal formulation.

In contrast, Müller [80] proposes a non-linear multigrid solver for PBD where each layer is similarly a coarse point sampling of the original particle system. Here, the prolongation operator consists of weighted averaging of coarse grid solutions, with weights inversely proportional to approximate geodesic distance between coarse *parent* particles and their fine *children* particles. This multigrid solver goes solely from coarse to fine, approximately solving a reduced set of constraints at each level. The proposed scheme lends itself well to mass spring PBD deformable models, where coarsening occurs via edge collapses. Unfortunately, it remains unclear how to generalize the approach to the preferred continuum constraints [107].

While XPBD is a fast method for real time simulation, the local nature of the constraint solve prevents efficient global propagation of deformation computations, which hinders convergence. To address this issue, long range attachements [59] and long-range constraints [82] have been proposed. Such approaches successfully create constraints between distant elements to explicitly enforce the desired propagation of local elastic effects, at the cost of added constraints. Nonetheless, the improved convergence dominates the additional per-iteration overhead. We take inspiration from this approach by using rigid bodies to improve propagation of information across long distances. The new constraints, while helping propagation, change the original optimization problem, impacting the final solution. Hence, these constraints need to be removed during the solve to obtain a ground truth simulation and ensure correct convergence. Unlike the previous work, the rigid bodies in our approach serve as approximate long range constraints that accelerate the convergence across solver iterations on a sequence of approximate systems, and these rigid bodies are absent in the final fully elastic solve. In contrast, long range attachments and constraints [59, 82] create additional constraints that must be solved in addition to those of the fully elastic system.

We are also inspired by the concept of adaptive rigidification [78], which proposes that non-deforming parts of the simulation can be automatically discovered and adaptively transformed into rigid bodies on the fly. This technique is specifically tailored to Newton-

Raphson solvers for standard finite element simulation, where the involved global linear systems of equations are significantly reduced by substituting elastic degrees of freedom with low-dimensional rigid motions. Unfortunately, the original adaptive rigidification algorithm is unsuitable to the XPBD framework, which explicitly ignores the problem's Hessian matrix needed for the oracle. As the assembly of such a matrix is costly, we instead propose an assembly-less method benefiting from many advantages of adaptive rigidification, without the need for an oracle.

Instead of approximating a simulation using coarsening methods like Delaunay remeshing [1], our proposed method is more similar to the work of Müller [80]. Our resolutions are based on the current state of the simulation and its interactions with the environment, which is likewise similar to adaptive rigidification [78] but without the need of an oracle. This allows us to use a single unified model for all of our resolutions without resorting to the more involved geometric, algebraic, and functional hierarchical constructions of existing multigrid methodologies [75, 45, 94]. Because we propose a fully-fledged iterative solver instead of approximating motions as rigid, we avoid contact handling problems like those described by Mercier-Aubin et al. [77] that would otherwise surface from the use of an oracle. Our approach instead speeds up the XPBD method without the need for domain knowledge, and by efficiently creating long-range propagation through a variety of temporary rigid patterns efficiently coupling distant elements. Layers of rigid patterns, much like the resolutions of a multigrid method, are built from increasing percentages of rigidification. Similar to the work of Barbié et al. [8], our systems of different resolutions will be generated automatically. Rather than using refinement, our layers are simplifications of a fine model without any change to the geometry. As such, our method is not subject to the problems that would otherwise surface in subdivision methods.

5.3 Standard XPBD

We briefly review the original XPBD method for the relevant information and refer to the original paper for the details. Following Macklin et al. [70], we start from the discretized

formulation of Newton's equations of motion

$$M\left(\frac{\boldsymbol{x}^{t+1}-2\boldsymbol{x}^{t}+\boldsymbol{x}^{t-1}}{h^2}\right)=-\nabla U^T(\boldsymbol{x}^{t+1}), \tag{5.1}$$

where $U(\boldsymbol{x})$ is an energy potential, $\boldsymbol{x}$ is a vector of positions with superscript denoting the time step, M is the lumped mass matrix, and h a time step size. From a vector of constraints C and compliance block diagonal matrix α, we obtain the forces

$$-\nabla_{\boldsymbol{x}}\, U^T(\boldsymbol{x})=-\nabla C^T(\boldsymbol{x})\alpha^{-1}C(\boldsymbol{x}), \tag{5.2}$$

of our system. In typical XPBD fashion, this is solved using the approximate linearized constraint formulation of the system.

In XPBD, we first step the vertices in time using the symplectic Euler method. We start by updating the velocities, and then the positions

$$\dot{\boldsymbol{x}} \leftarrow \dot{\boldsymbol{x}}+hM^{-1}\boldsymbol{f}, \tag{5.3}$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x}+h\dot{\boldsymbol{x}}, \tag{5.4}$$

of each elastic particle due to force $\boldsymbol{f}$. The solver uses the standard XPBD Gauss-Seidel-like updates, which includes a compliance term $\overline{\alpha}=\frac{\alpha}{h^2}$. With the Lagrange multipliers λ first initialized to zero, we iteratively solve for incremental updates

$$\Delta\lambda_j=\frac{-C_j(\boldsymbol{x})-\overline{\alpha}_j\lambda_j}{\nabla C_j(\boldsymbol{x})M^{-1}\nabla C_j^T(\boldsymbol{x})+\overline{\alpha}_j}, \tag{5.5}$$

for constraint j. We then convert the $\Delta\lambda_j$ impulse updates into position updates

$$\Delta\boldsymbol{x}=M^{-1}\nabla C(\boldsymbol{x})^T\Delta\lambda_j. \tag{5.6}$$

We use the Saint Venant-Kirchoff constitutive model expressed with Voigt notation [20, 98] as our elastic potential. As such, our constraints are the lower triangular entries of the strain tensor $E=\frac{1}{2}(F^TF-I)$, thus, a vector of 6 constraints in 3D (or a vector of 3

constraints in 2D). We define the per-element blocks of the compliance matrix as

$$\alpha_e = \begin{bmatrix} \zeta + 2\mu & \zeta & \zeta & 0 & 0 & 0 \\ \zeta & \zeta + 2\mu & \zeta & 0 & 0 & 0 \\ \zeta & \zeta & \zeta + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}^{-1}, \tag{5.7}$$

where ζ and μ are the first and second Lamé parameters. We must preserve the coupling of constraints due to the off diagonals of the upper-left of α_e. This requires a solve such that the denominator of Equation 5.5 forms the left-hand side matrix and the numerator becomes the right-hand side vector. For each element, we solve the set s of coupled constraints as a 3-by-3 system (or 2-by-2 for 2D elements),

$$\left(\nabla C_s(\boldsymbol{x}) M^{-1} \nabla C_s^T(\boldsymbol{x}) + \overline{\alpha}_{ss}\right) \Delta\lambda_s = -C_s(\boldsymbol{x}) - \overline{\alpha}_{ss}\lambda_s. \tag{5.8}$$

For the uncoupled constraints, i.e., corresponding to the lower right block of coefficients of Equation 5.7, we simply use the standard XPBD update from Equation 5.5.

For rigid bodies, the steps are similar [84]. We use symplectic Euler to step each rigid body's linear velocities and center of mass with Equation 5.3 and Equation 5.4. We also need to update each rigid body's torques τ, angular velocities $\omega \in \mathbb{R}^3$, and rigid body rotation $R \in SO(3)$ as

$$\tau \leftarrow r \times f, \tag{5.9}$$

$$\omega \leftarrow \omega + hI^{-1}(\tau - \omega \times I\omega), \tag{5.10}$$

$$R \leftarrow e^{h\widehat{\omega}} R, \tag{5.11}$$

where $I \in \mathbb{R}^{3\times3}$ is the inertia tensor and $\widehat{\omega}$ is the skew-symmetric cross product matrix. The matrix exponential provides the rotation update from the angular velocity vector and time step size, and is computed using the Rodrigues' formula [85].

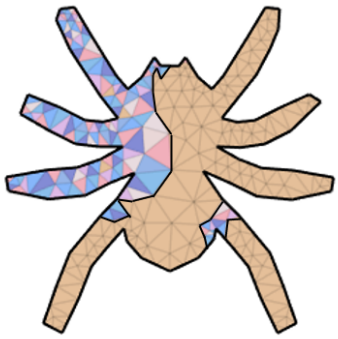


Figure 5.2: Graph coloring example.

Thus, the position of each vertex $\boldsymbol{x}_i$ making up rigid body r has position

$$\boldsymbol{x}_i = R_r \boldsymbol{r}_i + \boldsymbol{x}_r, \tag{5.12}$$

that is, they can be computed from the properties of rigid body r, with rotation R_r, center of mass $\boldsymbol{x}_r$, where $\boldsymbol{r}_i$ specifies the location of vertex $\boldsymbol{x}_i$ in the local frame of the rigid body.

5.3.1 Graph Coloring

We find independent constraints using graph coloring. While minimal graph coloring is a difficult problem, it is often reasonable to precompute a greedy graph coloring [36] by simply assigning each element a color unassigned to shared degrees of freedom from a set of colors. In Figure 5.2, we show an example of mesh graph coloring. We run constraints of the same color in parallel as constraints of the same color are independent. This allows for a fast parallel Gauss-Seidel-like solve of constraints.

We model the resolutions of our multigrid-like solver in terms of rigidification patterns instead of discretizations. When using partially rigid layers, the rigid bodies create distant coupling, hindering parallelization. However we note that each rigid body is independent from the others, otherwise they would be merged together. Therefore, it is reasonable to solve the rigid body constraints in parallel for the different rigid bodies. Because the constraints on a single rigid body are coupled, we solve them in a Jacobi style [71], while the rest of the constraints are solved in a Gauss-Seidel-like fashion similar to standard XPBD.

5.3.2 XPBD Elastic and Rigid Coupling

In our rigid-elastic XPBD implementation, the coupling is not implicit. The elastic element update does not take into account the rigid bodies within the mesh. This creates a discrepancy between the rigid body and elastic body views of the position of vertices that are on the boundaries between rigid and elastic regions. We couple both views via an equality constraint similar to that of Müller et al. [84], that is, for vertex $\boldsymbol{x}_i$ on the boundary with rigid body r we have

$$C_i(\boldsymbol{x}) = \|\boldsymbol{x}_i - (R_r\,\boldsymbol{r}_i + \boldsymbol{x}_r)\|^2, \tag{5.13}$$

$$\Delta\lambda_i = \frac{-C_i(\boldsymbol{x}) - \overline{\alpha}_i\lambda_i}{w_i + \frac{1}{m_i} + \overline{\alpha}_i}, \tag{5.14}$$

where m_i is the mass of the elastic vertex, w_i is the generalized inverse mass of the vertex in the rigid body, and $\bar{\alpha}_i = 0$ to specify a hard constraint and preserve the rigid body boundary. We compute the generalized inverse mass as

$$w_i = \frac{n_C}{m_r} + (\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{d})^T I^{-1} (\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{d}), \tag{5.15}$$

where the numerator of the first term accounts for mass splitting [108] with n_C being the number of constraints affecting the rigid body. Here, the vector $\boldsymbol{r}_i^{\mathrm{g}} = R_r\,\boldsymbol{r}_i$ points from the rigid body center of mass to the constrained vertex in the global frame, and $\boldsymbol{d}$ is the normalized direction of vector $\boldsymbol{x}_i - (R_r\,\boldsymbol{r}_i + \boldsymbol{x}_r)$, i.e., the correction direction for the equality constraint.

We get a valid coupling when the boundary vertices match the rigid body surface. We use a modified position update similar to that of Müller et al. [84] except that instead of

working with two rigid bodies we update the elastic particle and a rigid body,

$$\boldsymbol{p} = \Delta\lambda\, \boldsymbol{d}, \tag{5.16}$$

$$\Delta\boldsymbol{x}_i = \frac{\boldsymbol{p}}{m_i}, \tag{5.17}$$

$$\Delta\boldsymbol{x}_r = -\frac{\boldsymbol{p}}{m_r}, \tag{5.18}$$

$$\Delta\omega_r = -\frac{1}{h} I^{-1}(\boldsymbol{r}_i^{\mathrm{g}} \times \boldsymbol{p}). \tag{5.19}$$

Some iterations feature strongly coupled constraints, but quaternion multiplications are not commutative, hindering constraint parallelization. So typical XPBD simulations approximate quaternion multiplications as commutative sums with the assumption of infinitesimal time steps (e.g., as done in Kalman filters [74]). Larger time steps can lead to wrong orientations causing non-physical behaviour. In Equation 5.19, we instead use a Jacobi style parallel accumulation of angular velocities for dependent constraints on rigid bodies [11], which offers the same benefits. Commutativity simplifies our constraint solves by allowing the computation of all rigid body boundary constraints simultaneously.

After solving all the boundary constraints of a rigid body in parallel, we update the rotation using Equation 5.11 from the accumulated change of angular velocity. We then update the relevant particle positions of a rigid body with respect to the new rotation and center of mass. We solve the rigid body constraints last, therefore we only update their vertex positions once per iteration.

For instance, if the face of an elastic element is part of a rigid body, we add three equality constraints, each solved sequentially. After correcting the second vertex's equality constraint, the rigid body is potentially no longer satisfying the first vertex's constraint. Having a notion of the full constrained coupling would potentially reduce the number of constraints and improve convergence by allowing a more accurate correction of the iterates.

5.3.3 Multi-Layer Method For XPBD

Our method has the nice property of allowing natural generation of multiple resolutions on the fly without remeshing by using adaptive rigidification concepts. Because the adaptive layers preserve the mesh vertices, the coarsening and refinement operations are intuitive. In the context of this multi-layer solver, we define coarse as a geometric model with more elements simulated as rigid, and fine as a model with more elastic elements. We first need a sorting process to determine the priority of element rigidification, e.g., sorting elements by strain rate. Then we gather elements into rigid bodies incrementally by inserting elements in the given order.

5.3.3.1 Rigidification

In the work of Mercier-Aubin et al. [78], the rigidification process consists of monitoring the strain rate computed as a finite difference

$$\dot{E} = \frac{E^t - E^{t-1}}{h}, \tag{5.20}$$

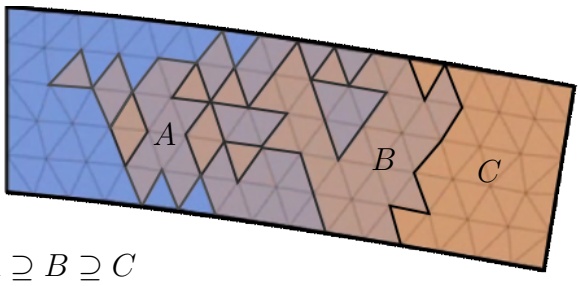
to detect non-deforming elements at each time step. This is compatible with any per-element strain measure E like the Green strain. While the adaptive rigidification method uses the strain rate to determine non-deformation, we use it to generate problems of different sizes to approximate the solution of the fully elastic model.

5.3.3.2 Hierarchy generation

We create the resolutions of a multi-layer solver using strain-rate dependent rigidification patterns. We build the hierarchies prior to the solve, from fine to coarse, by inserting the elements to be treated as rigid, increasing in strain-rate order, into a disjoint set. This is efficient due to path compression and the use of contiguous memory [103]. Construction of each coarser layer includes the previously rigid elements from the past layer (continuing with the same disjoint set) as shown in Figure 5.3.

Algorithm 4: IncrementalMerging

input : Sorted elements grouped by layer, fine to coarse
output: Connectivity of layers
Initialize union-find structure $u[e] = -1$ for all elements e
Initialize claimed vertices $v.\mathit{claimed} = -1$ for all vertices v
foreach layer l **do**
  foreach uninserted element e of layer l **do**
    $u[e] \leftarrow e$
    foreach vertex v of element e **do**
      // find returns -1 when the input has no root
      $s \leftarrow u.\mathit{find}(v.\mathit{claimed})$ // with path compression
      if $e \neq s \wedge s \neq -1$ **then**
        $u[e] \leftarrow s$
      end
      $v.\mathit{claimed} \leftarrow e$
    end
  end
  $L[l] \leftarrow u$ // snapshot of u provides connectivity of layer l
end



$A \supseteq B \supseteq C$

Figure 5.3: Different incremental rigid patterns created from strain-rate insertion.

Initially the vector containing our disjoint set is initialized with all entries set to -1. This vector has a size equivalent to the number of elements. On element insertion, a new set is created at the index of the inserted element, pointing to itself as its root. We then verify if the vertices of the element are already part of a set. When a rigidifying element is vertex-adjacent to existing rigid sets, we merge the sets. We do path compression when the disjoint-set function find is called. Because we are only ever inserting one new element at a time, the computation time to find the connectivity remains linear. Therefore the algorithm has a computation upper bound tied to the sorting algorithm rather than the creation of components. We do a single pass over the elements to generate the hierarchies as shown in Algorithm 4. Unlike adaptive rigidification, we do not need to handle hinges

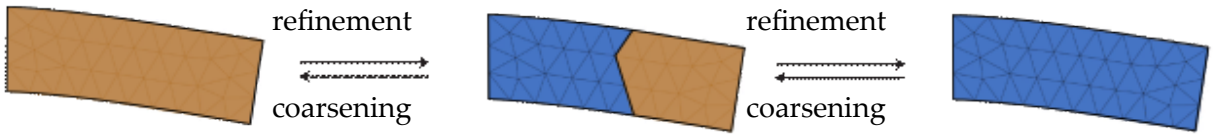


Figure 5.4: Example of a sequence of layers with precomputed rigidification patterns. The solver iterates through the different layered resolutions using refinement operations to add elasticity during the solve.

as the partially rigid layers only serve as useful intermediate models. As such, at a given layer, if two rigid bodies share a single vertex we merge them into a single rigid body.

This type of insertion allows us to incrementally update the connected components that will form rigid bodies in each resolution. While inserting elements, we keep a copy of the connected components for each slice of a predetermined percentage of all elements. After building these models of the system at different resolutions, we can switch layers freely, using refinement and coarsening operations (see Figure 5.4). Here, *coarsening* consists of removing the elastic constraints of the elements that are rigid in the coarse layer, and adding extra coupling constraints on the new rigid boundary. In contrast, *refinement* queries all the current vertex positions from rigid body properties, updates boundary constraints, handles residual velocities, and reintroduces elasticity constraints to newly elastic elements. We introduce residual velocities in Subsection 5.3.4.

5.3.4 Iterating Through Layers

Doing a symplectic step before projecting the constraints would lead to inflated elements because the particles of an element move on straight lines before the first rigid layer on rotational motions. This would lead to inaccurate shapes for rigid bodies and hinder convergence. Instead we progressively step the velocities by depleting them over many layers using residual velocities.

Residual velocities are leftover internal elastic velocities inside of rigid bodies. Initially the residual velocities replace the velocity update for the entire system. Rigid body angular and linear velocities are computed from the per-particle residual velocities $\boldsymbol{v}_i$. On layer switch, we must compute the new rigid body properties from the vertices of rigid

body r,

$$m_r = \sum_{i \in r} m_i, \tag{5.21}$$

$$\boldsymbol{x}_r = \frac{1}{m_r} \sum_{i \in r} m_i \boldsymbol{x}_i, \tag{5.22}$$

$$\dot{\boldsymbol{x}}_r = \frac{1}{m_r} \sum_{i \in r} m_i \boldsymbol{v}_i, \tag{5.23}$$

$$\boldsymbol{r}_i = \boldsymbol{x}_i - \boldsymbol{x}_r, \tag{5.24}$$

$$I_r = \sum_{i \in r} m_i \hat{\boldsymbol{r}}_i^T \hat{\boldsymbol{r}}_i, \tag{5.25}$$

$$\omega_r = I_r^{-1} \sum_{i \in r} m_i \boldsymbol{r}_i \times (\boldsymbol{v}_i - \dot{\boldsymbol{x}}_r). \tag{5.26}$$

These values remain fixed throughout the solver iterations of the layer. Hence, we compute the relative positions $\boldsymbol{r}_i$ once per layer. Throughout the solve, We let $\boldsymbol{x}_i^{t+1}$ be the current approximation of the position vertex i at the next time step (it is initialized to $\boldsymbol{x}_i^t$), and $\boldsymbol{x}_i$ we use to denote the position of the vertex at the beginning of the solve of the current layer. For vertices that are part of rigid bodies, the layer solve steps their positions with rigid motion. That is, we obtain a rotation update R_r by stepping the rigid bodies, and vertices that are part of the body have their positions computed based on the rigid body state (Equation 5.12). At this point, the vertex velocities for this rigid motion can be computed as $\frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i)$, but there can still be non rigid velocities in the residual velocity $\boldsymbol{v}_i$. Thus, we rotate the current residual and subtract the current velocity to update the residuals, i.e.,

$$\boldsymbol{v}_i \leftarrow R_r \boldsymbol{v}_i - \frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i). \tag{5.27}$$

The process of updating residuals throughout the layers continues until the elements are solved as elastic in the final iterations. Thus each layer has less residual velocities to rigidly step as they are depleted through rigid body motions. To be clear, it is only vertices that are part of rigid bodies that have residual velocities, while residual velocities are zero (depleted) for vertices that are stepped elastically.

Note that the residual velocities must be rotated in Equation 5.27 because they are unstepped velocities at time t while the finite difference of particle positions are rigidly stepped velocities at time $t + 1$. In the case of a purely rigid motion of a spinning elastic body in equilibrium, these velocities exactly cancel out. This is because the initial residual velocities are exactly a rigid motion (the previous step velocities, which are computed using finite differences at the end of each simulation step in XPBD).

Because rigid bodies move during the symplectic steps, we rotate the residual velocities to match the rigid body frames and preserve the inner elastic velocities. The constraint solves also change the rigid body positions and orientations, which requires a rotation update to the residual velocities before changing layers where $\Delta\omega_r$ is the change in angular velocities due to constraint solves.

This modification to XPBD requires only minor changes around the solver, making it compatible with most existing frameworks. This can be seen in Algorithm 5.

5.3.5 Contact Handling

To handle contacts, we use penalty constraints

$$C_c(\boldsymbol{x}) = \min(0, \boldsymbol{d}_c \cdot (\boldsymbol{p}_c - \boldsymbol{x}_c)), \tag{5.28}$$

for each vertex $\boldsymbol{x}_c$ in contact, and correct the interpenetration at contact $\boldsymbol{p}_c$ with normal $\boldsymbol{d}_c$. We set the compliance parameter α to 10^{-4}, which is the value recommended in the original XPBD work. The constraint is treated in a consistent way, regardless if the vertex $\boldsymbol{x}_c$ is part of a rigid body or an elastic element.

For a given layer, all contacts on elastic vertices are independent and solved in parallel. We find the constraint Lagrangian update using Equation 5.5. All contacts on rigid vertices are solved using the same procedure as Subection 5.3.2, but with the elastic particle in this case being replaced with an infinite mass contact position $\boldsymbol{p}_c$, and the constraint direction being the contact normal. This means that contacts are often solved as both rigid and elastic, during a single time step.

Algorithm 5: MULTI-LAYERXPBD

input : Position vector $\boldsymbol{x}_t$
Velocity vector $\dot{\boldsymbol{x}}_t$
Step size h
External forces vector $\boldsymbol{f}_{ext}$
output: Positions $\boldsymbol{x}^{t+1}$ and velocities $\dot{\boldsymbol{x}}^{t+1}$ after stepping

$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t$
$\boldsymbol{v} \leftarrow \dot{\boldsymbol{x}}^t + hM^{-1}\boldsymbol{f}_{\text{ext}}$ // residual velocity initialization
$L \leftarrow$ INCREMENTALMERGING // Algorithm 4
foreach layer $l \in L$ **do**
  foreach newly elastic vertex i of layer l **do**
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i^{t+1}$ // vertex at the start of the layer
    $\boldsymbol{x}_i^{t+1} \leftarrow \boldsymbol{x}_i^{t+1} + h\boldsymbol{v}_i$ // step with residual velocity
  end
  foreach rigid body r in layer l **do**
    $\omega_r, \dot{\boldsymbol{x}}_r, I_r, M_r \leftarrow$RIGIDPROPERTIES$(l, \boldsymbol{v})$ // Subsection 5.3.4
    $R_r, \boldsymbol{x}_r \leftarrow$STEPRIGIDBODIES$(\omega_r, \dot{\boldsymbol{x}}_r)$ // Section 5.3
    foreach vertex i of each rigid body r **do**
      $\boldsymbol{x}_i^{t+1} \leftarrow R_r\,\boldsymbol{r}_i + \boldsymbol{x}_r$ // Equation 5.12
      $\boldsymbol{v}_i \leftarrow R_r\boldsymbol{v}_i - \frac{1}{h}(\boldsymbol{x}_i^{t+1} - \boldsymbol{x}_i)$ // Equation 5.27
    end
  end
  foreach iteration for layer l **do**
    $\Delta\omega_r, \boldsymbol{x}^{t+1} \leftarrow$SOLVECONSTRAINTS
  end
  foreach rigid vertex i of layer l **do**
    $\boldsymbol{v}_i \leftarrow e^{h\Delta\widehat{\omega}_r}\,\boldsymbol{v}_i$ // rotate residual velocities
  end
end
$\dot{\boldsymbol{x}}^{t+1} \leftarrow \frac{1}{h}(\boldsymbol{x}^{t+1} - \boldsymbol{x}_t)$
$\dot{\boldsymbol{x}}^{t+1} \leftarrow$RESTITUTIONUPDATE$(\dot{\boldsymbol{x}}^{t+1})$ // Equation 5.29

As proposed by Müller et al. [84], we handle restitution after the solve with a velocity update

$$\Delta\dot{\boldsymbol{x}}_c = \boldsymbol{n}_c(\min(-\epsilon\,\boldsymbol{n}_c \cdot \dot{\boldsymbol{x}}_c, 0) - \boldsymbol{n}_c \cdot \dot{\boldsymbol{x}}_c), \tag{5.29}$$

where ϵ is a restitution parameter, and $\boldsymbol{n}_c$ is the contact normal. As this is a post-solve operation, we consider the soft body contacts as fully elastic and run this operation in parallel as independent contacts.

5.3.6 Layer-Stop Criterion

Solving constraints in a partially rigidified mesh can lead to stagnation if the remaining error is located inside of rigidified elements. Implementing a stop criterion based on error improvement can enhance speed, at the cost of the constant-time property of our solver. Because the first few steps of XPBD solvers often lead to an increase in residual error, we activate this feature only after observing the first decrease in constraint residual error

$$\|C + \overline{\alpha}\lambda\|. \tag{5.30}$$

When the change in constraint residual is near zero (below 1e-8) for an iteration, we switch directly to the next layer in the sequence. This allows the solver to quickly switch to the fully elastic layer.

5.4 Results

We evaluate our method on different fronts to develop the intuition on how to tune the parameters of our multi-layer solver and to validate our various hypotheses on the behaviour of the solver.

5.4.1 Choice of Pattern

The choice of an adequate rigid pattern is critical to efficiently propagate motions. In Figure 5.5 we compare orders of insertion into the disjoint set, leading to different rigid patterns. Our tests include insertions in random order, strain-rate based, stretch-based, and vertical or horizontal stripes. We note that the stretch-based ordering using the eigenvalues of $M^{-1}K$ is simply here for comparison purposes as it is an expensive measure due to the assembly of the stiffness matrix and eigenvalue decomposition. The strain-based approach provides similar convergence rates to the stretch-based approach, albeit at a much cheaper cost. The random patterns sometimes lead to good convergence, but are unreliable and just as often lead to worse performance. The stripe patterns correspond to

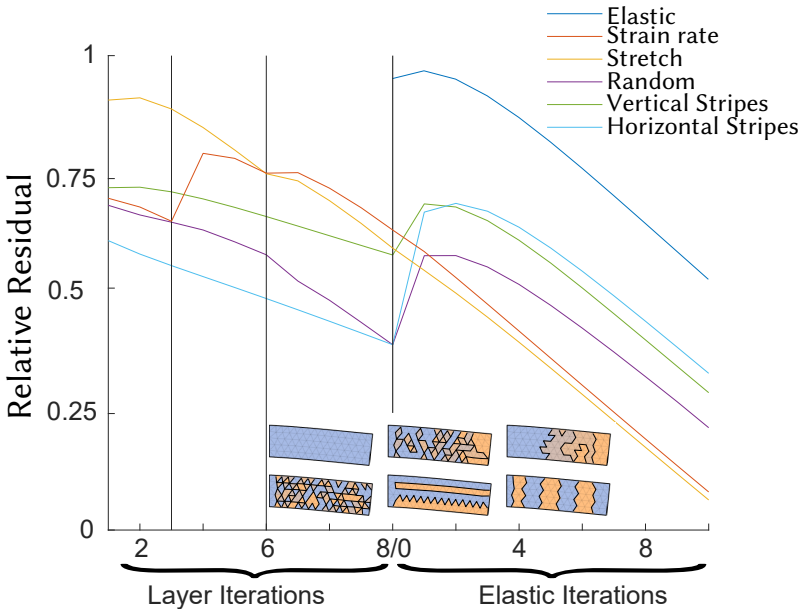


Figure 5.5: Comparison of convergence per coarsening pattern: On the left the plot shows convergence including the reduced iterations, while on the right, we see the convergence on the fully elastic layer. The vertical lines represent layer switches.

cases where an animator has domain knowledge for how a complex mesh would deform. That is, prior specification of layers may potentially be useful in niche applications. Hence, we suggest using strain-rate based ordering for the insertion of elements in the disjoint set to generate the layers. We note that the error sometimes increases on layer switch, which is not unexpected because of the stepping of residual velocities in the system on layer switch.

5.4.2 Choice of Layer Group Sizes

Because large deformations are more likely to impact global simulation than tiny elastic vibrations, we suggest that starting with more aggressively rigidified layers first is more likely to lead to faster convergence than coarsening after spending iterations on a fine resolution solve. Hence, we always start our test from rigid to elastic. Likewise, there are different ways to select the change in rigidification group sizes per layer. In Figure 5.6 we compare different types of layer selection for a standard cantilever example. We break the elements into groups with an equal number of elements in each such that the number

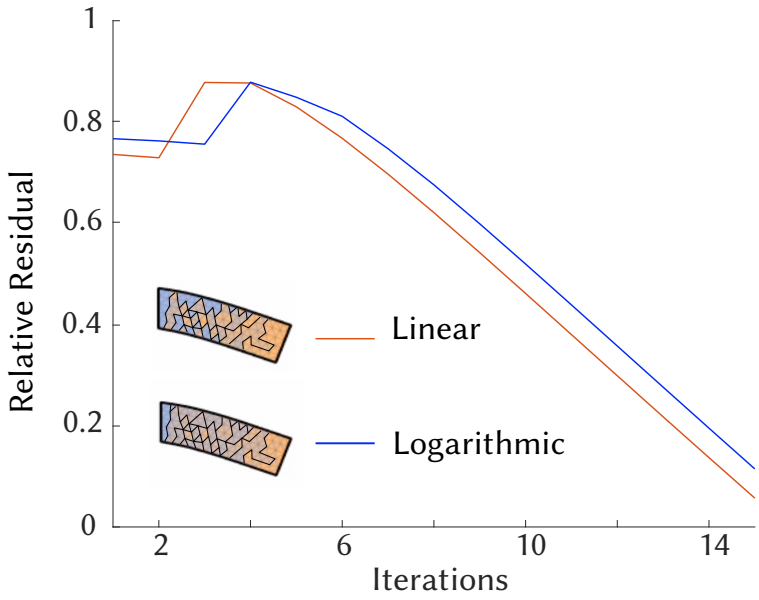


Figure 5.6: Linear or logarithmic: choosing linear changes between the size of layer groups lead to better convergence compared to logarithmic halving group size for this cantilever example.

of rigid elements across the different layers changes linearly. We try starting rigid and increasingly elastifying elements with constant size jumps of 25% rigidification. We also test halving the number of elements rigid for each layer, starting at 100% rigidification and ending at around 12% before going fully elastic. The logarithmic approach underperforms due to the change in group sizes being too steep initially, missing the opportunity to propagate important rigid motions.

We note that steps with purely rigid motions can instantly terminate early as the symplectic stepping of rigid bodies generate low error solutions before solving the elastic constraints. We show this phenomenon in Figure 5.9a.

5.4.3 Number of Layers

For each layer we must build the current rigid body and compute the properties like the rotation and center of mass. As such, there is a small linear overhead over rigid bodies on layer switch. Switching between many small rigid layer group sizes with a low number

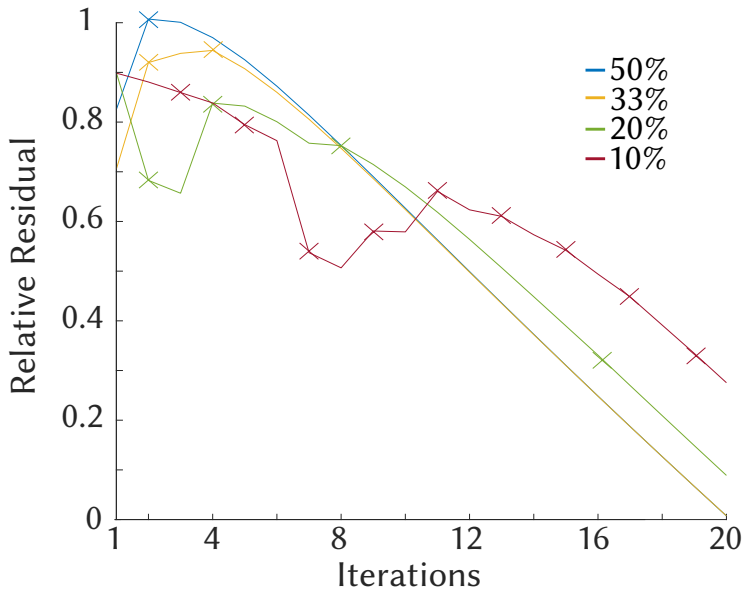


Figure 5.7: Test of the impact of the layer size in the cantilever scene for linearly distributed layers. The cross markers show the iterations where layer switches happen.

of iterations could hinder performance. Hence, choosing an appropriate number of layers is important.

A case-by-case selection can allow optimal performance. For instance, if we know that 30% of the elements are stiffer, like the rim of a wheel, then that proportion could be included in the layers. A single initial 100% layer is only a good start for scenes without pinned vertices to instantly propagate global rigid motions (otherwise, with pinned vertices, the fully rigid layer does not move and provides no benefit). In Figure 5.7 we show the cantilever test with different numbers of layers. Because the cantilever is pinned, we start after the first reduction in percentages. We also double the number of iterations done per layer. For instance, the 50% plot does one iteration in the layer 50%, while the 33% plot does 1 iteration in the layer 66% and 2 in the layer with 33% rigidification. Because the plot for the 10% decrease in group size has too many layers to double the iteration count, we evenly distribute the computation using 2 iterations per layer.

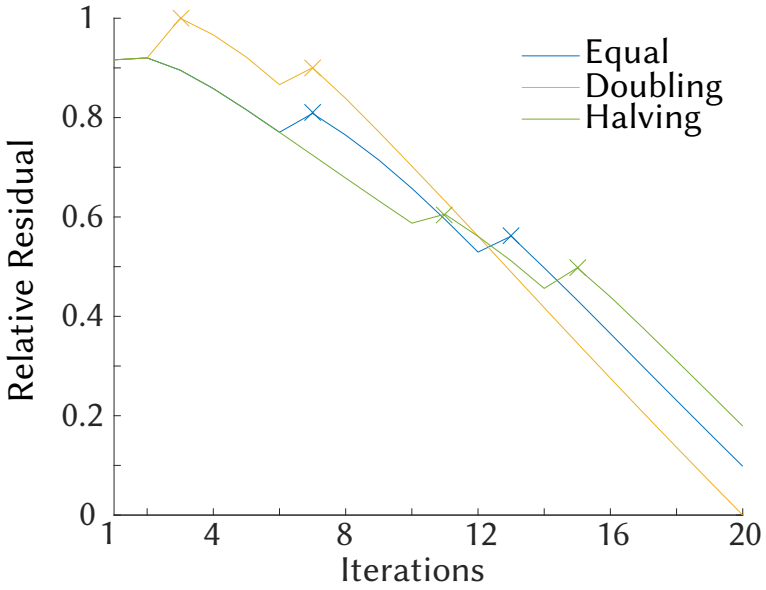


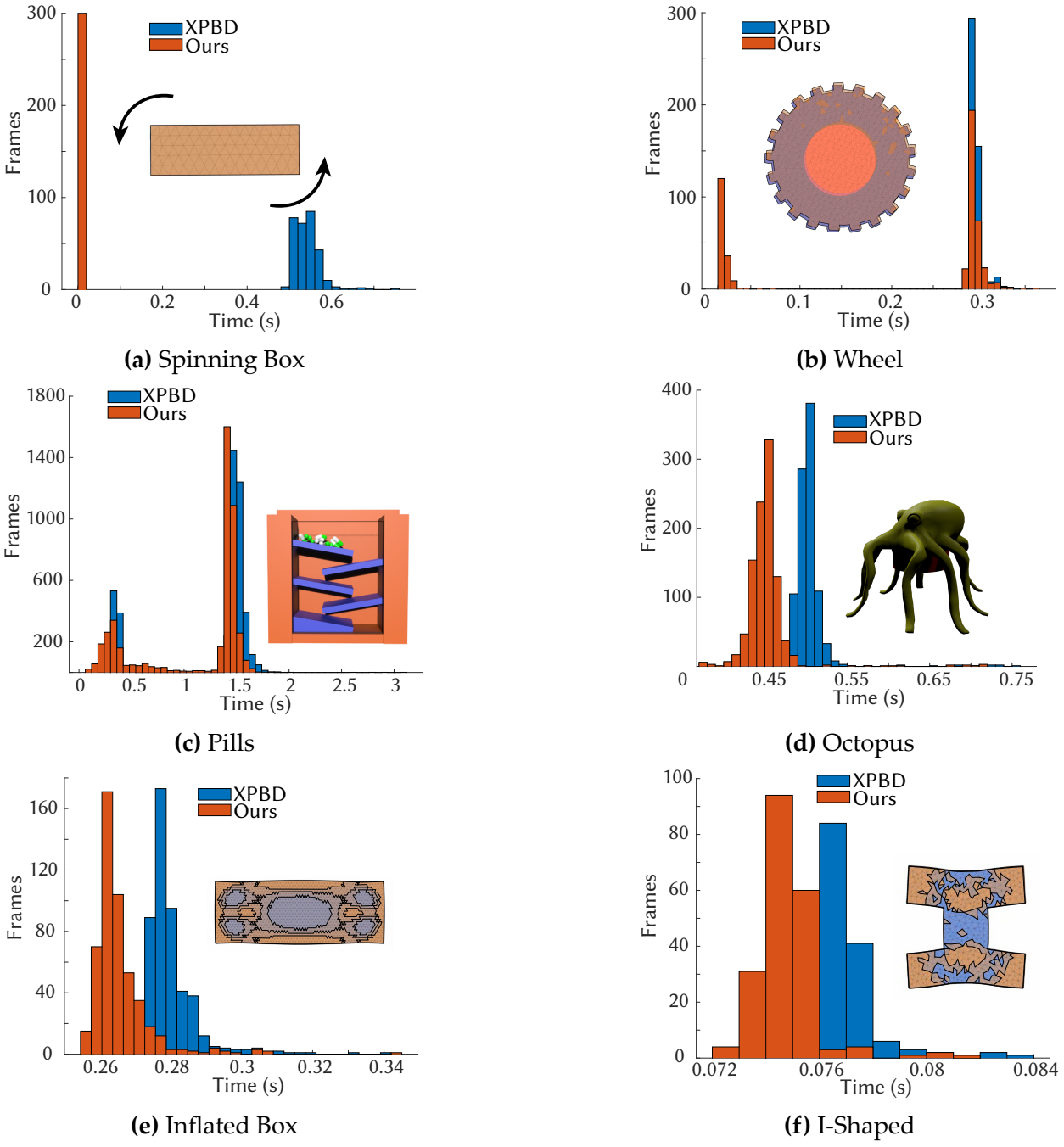
Figure 5.8: For the cantilever example, increasing the number of iterations as we elastify the model yields the best performance. The cross markers show layer switches.

5.4.4 Number of Iterations per Layers

For more aggressively rigidified layers, it is reasonable to assume that fewer iterations are needed, because the problem is effectively smaller. Likewise, doing too many iterations in any partially rigidified layer can lead to convergence plateaus. In Figure 5.8, we explore different iteration numbers per layer to validate our hypothesis. In our test, we use equal number of iterations (6 per rigid layer), increasing iterations by doubling the number of iterations per layer, and decreasing iterations by halving the number of iterations per layer. We notice how the doubling approach outperforms the other by doing less iterations in aggressive layers, while spending more time in finer layers.

5.4.5 Example Simulations

We designed challenging test scenes to evaluate the performance of our method, featuring varying degrees of deformations and contacts. Figure 5.9 shows our example models and compares the run times of our approach to XPBD with histograms of the solve times



(a) Spinning Box **(b)** Wheel

(c) Pills **(d)** Octopus

(e) Inflated Box **(f)** I-Shaped

Figure 5.9: Histograms of wall clock time to simulate a frame with a stop criterion of 90% improvement in residual error for the frame compared to the initial residual of the elastic simulation. In all these various examples, we see improvements in performance.

necessary to achieve a desired amount of error for each time step in a typical simulation trajectory.

We run our experiments on CPU, with efforts made for parallelization of the code using vectorized MATLAB code. Our tests use relative measures so as to provide a fair evaluation of the performance. While our current implementation could be ported to GPU, modifications are necessary to adapt the dynamic nature (e.g., problem sizes) of the method. A compiled language could also provide performance improvement especially for the computation of constraint gradients.

While our solver offers ground truth solutions, the difference in convergence and constraint error distribution can lead to different behaviors when compared to a standard XPBD solve. We note that the standard XPBD method can introduce error due to the linear discretization of impulses, leading to inflation during rigid motions. This is not the case in our solver when using a fully rigid first layer, as demonstrated by the spinning box shown in Figure 5.9a. The simulation of a spinning box instantly terminates because the rigid motions are solved accurately in the symplectic step of the first layer.

In Figure 5.9b we notice two different distributions due to the fast solve of purely rigid motions. Because of the imprecision of elastic simulations, we can see a divergence in the simulations. Our solver preserves the purely rigid motions in this wheel example much like it does in the spinning box example.

Our solver can speed up contact heavy scenes like the pills machine of Figure 5.9c. Due to improved propagation, our method consistently yield better performance even in a contact heavy scene. We note that performance could be further improved by analyzing the residual error locally instead of globally, allowing early stop for various independent regions like individual pills.

Even in active scenes like Figure 5.9d, motions can often be represented as a set of rigid body motions, leading to a handful of nearly free time steps and overall cheaper solves. A simpler example with global deformation is shown in Figure 5.9e where a box is stretched and released, providing no undeforming region. Even in this fully deforming scene, the residual velocity solver outperforms the elastic simulation at all time. Likewise, in Figure 5.9f, global deformation in wildly diverging directions lead to adequate rigid body

Table 5.1: Comparison across resolutions. For the simulation of a box stretched horizontally by 10%, the proportion of time taken sorting and computing connected components is minimal, while the overall performance improvement in comparison to the full elastic solve becomes large at higher resolution.

Elements	651	1550	3304
Sort and connected components	0.01%	0.017%	0.02%
Performance improvement	0.1%	5.8%	11.93%

formation, and ultimately an enhancement in performance. In both cases, the simulation benefits from the rigid motions of the first layers, propagating information to distant elements similar to long-range constraints.

Much like adaptive rigidification, our algorithm shows increased benefits when used with finer meshes. For instance, while simulating a 3 by 1 box stretch horizontally by 10% before being released, we see that refining the model increases the proportion of computation time saved. In Table 5.1, we show the percentages of improvement for different resolutions of this scene. We also monitor the proportion of time that the sort and the connected components occupy in the solve. We note that while it is not currently a bottleneck, scenes featuring very large numbers of elements would benefit from parallelization of this algorithm.

Using the octopus, we also evaluate scalability across stop criteria in Figure 5.10. We see that the benefits of using the multi-layer method becomes more apparent for higher residual-reduction stop thresholds. Obviously solving the system more precisely requires more time, but we note that the performance gain increases as we raise the improvement percentage threshold.

5.5 Discussion and Limitations

Our implementation comes with the same drawbacks as its overarching method. Based on XPBD, it suffers from some of the same convergence issues. While our method does improve convergence throughout our examples, the worst case scenario remains in the same order of magnitude as the original XPBD method. This happens when the solver hits a plateau for many iterations without using an early stop for the layer. However, our

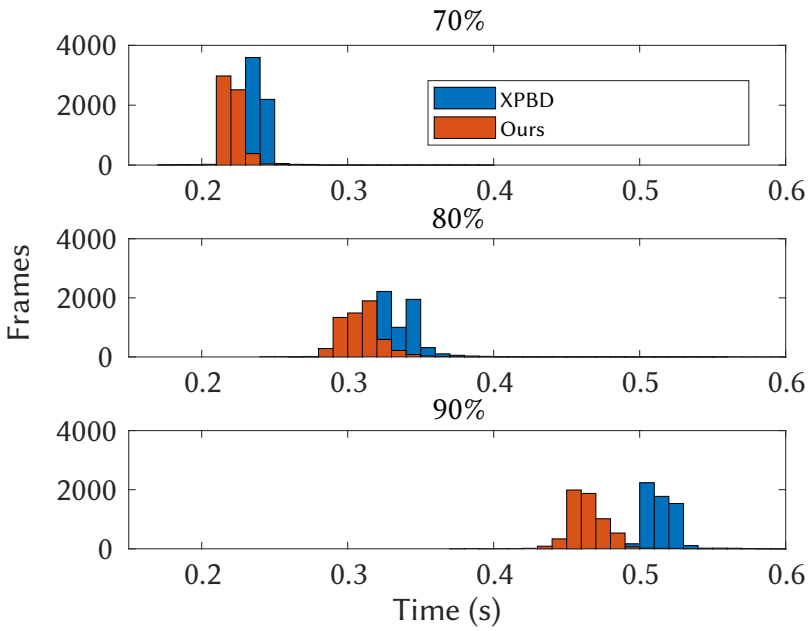


Figure 5.10: In the octopus scene, the gap between the multi-layer and XPBD solvers in terms of computation time increases proportionally to precision.

method also makes the generation of long range constraints trivial by not requiring any domain knowledge, and adapting itself to the current environmental inputs. Likewise, our method can propagate contacts efficiently by treating them as both elastic and rigid while we iterate through the layers. For rigid motions, our solver instantly reduces the error up to relevant numerical accuracy making the simulations efficient, while retaining ground truth accuracy.

Our method uses concepts similar to adaptive rigidification for the choice of layers, but without an oracle. As such the last layer must always be elastic to generate good strain rates at the next time step. Future work could create an oracle for XPBD simulations that would render this method compatible with the approximate simulations of adaptive rigidification, further improving the speedups when accuracy is not critical. Because XPBD benefits from not assembling the system's Hessian matrix, the original oracle of adaptive rigidification is incompatible with such a framework. We theorize that an alternative oracle could be to do elastic iterations first, then proceed with rigidified parts, but this would likely be subject to localized propagation issues.

Likewise, the current implementation does not include co-dimensional shell simulations. Such simulation requires extra care for the bending components [77]. It is not obvious that locking bending angles would yield good first steps in the partially rigid layers. A potential alternative could be to only use multi-layers on the 2D projection of the shells, leaving the bending components fully simulated.

There are potential opportunities to parallelize the union-find [46, 23] connected component builder. Likewise, we use a standard sorting algorithm, which could be parallelized using forms of radix parallel sorts. Furthermore, we do not need a full sort. That is, the grouped elements at a fine layer do not need to be sorted and are only expected to have lower strain rate than those elements in the group at the next coarser layer. Nevertheless, the sort and union-find are not the current bottlenecks of our implementation.

While early stops allow fast simulations of deformables, setting a constant number of layers and iterations can have potential uses for real time applications where constant time solves are important. With our simulator, it is easy to allocate a constant amount of resources for elasticity and still benefit from improved convergence. This is an important feature for real-time application as resources are often limited.

Our method shares similarities with multigrid methods, which would make the introduction of a v-cycle pattern intuitive, starting elastic to rigid and back to elastic. However, such pattern lose the benefits of residual velocity layers, and the possibility to instantly terminate on rigid motions during the first iteration.

While we use strain-based energies for our elasticity, the overarching multi-layer solver is independent of such constraint and could be used with other popular formulations like the stable Neo-Hookean from Macklin et al. [69].

5.6 Conclusions

We present a multi-layer approach for the simulation of soft bodies with XPBD. Our method converges faster than standard XPBD by automatically generating cheap coupling

similar to long-range constraints through the use of rigid bodies. Without using a stop criterion for layers, our solver provides an iterative solution that offers steady performance across frames.

While iterating through coarse layers, the solver provides an efficient propagation of information to distant elements with a significantly reduced number of constraints to solve. We hope that this new method will inspire the community to build upon our work and generate novel multi-layer solvers working outside of the typical algebraic or geometric multigrid approaches.

References for Chapter 5

[1] M. Adams and J. W. Demmel. Parallel Multigrid Solver for 3D Unstructured Finite Element Problems. In *Proceedings of the 1999 ACM / IEEE Conference on Supercomputing*, SC '99, pages 27–45, Portland, Oregon, USA. Association for Computing Machinery, Jan. 1999. DOI: 10.1145/331532.331559.

[8] L. Barbié, I. Ramière, and F. Lebon. An Automatic Multilevel Refinement Technique Based on Nested Local Meshes for Nonlinear Mechanics. *Computers & Structures*, 147:14–25, 2015. DOI: 10.1016/j.compstruc.2014.10.008. CIVIL-COMP.

[11] N. Bell, Y. Yu, and P. J. Mucha. Particle-based Simulation of Granular Materials. In *Proceedings of the 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '05, pages 77–86, Los Angeles, California. Association for Computing Machinery, 2005. DOI: 10.1145/1073368.1073379.

[13] J. Bender, D. Koschier, P. Charrier, and D. Weber. Position-based simulation of continuous materials. *Computers & Graphics*, 44:1–10, 2014. DOI: 10.1016/j.cag.2014.07.004.

[15] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Transactions on Graphics*, 33(4):1–11, July 2014. DOI: 10.1145/2601097.2601116.

[20] O. Cetinaslan. ESBD: Exponential Strain-based Dynamics using XPBD Algorithm. *Computers & Graphics*, 116:500–512, Nov. 2023. DOI: 10.1016/j.cag.2023.09.014.

[23] J. Chen, K. Nonaka, H. Sankoh, R. Watanabe, H. Sabirin, and S. Naito. Efficient Parallel Connected Component Labeling With a Coarse-to-Fine Strategy. *IEEE Access*, 6:55731–55740, 2018. DOI: 10.1109/ACCESS.2018.2872452.

[36] M. Fratarcangeli and F. Pellacini. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum*, 34(2):405–413, 2015. DOI: 10.1111/cgf.12570.

[45] H. Guillard. Node-nested Multi-grid Method with Delaunay Coarsening. Research Report RR-1898, INRIA, 1993.

[46] S. Gupta, D. Palsetia, M. M. A. Patwary, A. Agrawal, and A. Choudhary. A New Parallel Algorithm for Two-Pass Connected Component Labeling. In *IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1355–1362, 2014. DOI: 10.1109/IPDPSW.2014.152.

[59] T.-Y. Kim, N. Chentanez, and M. Müller-Fischer. Long Range Attachments a Method to Simulate Inextensible Clothing in Computer Games. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '12, pages 305–310, Lausanne, Switzerland. Eurographics Association, 2012.

[65] T. Liu. Projective dynamics/simulation. Symposium on Geometry Processing Graduate School, 2021.

[68] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Kim. Primal/Dual Descent Methods for Dynamics. *Computer Graphics Forum*, 39(8):89–100, 2020. DOI: 10.1111/cgf.14104.

[69] M. Macklin and M. Muller. A Constraint-based Formulation of Stable Neo-Hookean Materials. In *Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games*, MIG '21, pages 1–7, Virtual Event, Switzerland. Association for Computing Machinery, 2021. DOI: 10.1145/3487983.3488289.

[70] M. Macklin, M. Müller, and N. Chentanez. XPBD: Position-Based Simulation of Compliant Constrained Dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, pages 49–54, Burlingame, California. Association for Computing Machinery, 2016. DOI: 10.1145/2994258.2994272.

[71] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 33(4):1–12, July 2014. DOI: 10.1145/2601097.2601152.

[74] J. Marins, X. Yun, E. Bachmann, R. McGhee, and M. Zyda. An Extended Kalman Filter for Quaternion-based Orientation Estimation Using MARG Sensors. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 4, pages 2003–2011, 2001. DOI: 10.1109/IROS.2001.976367.

[75] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Transactions on Graphics*, 30(4):1–12, July 2011. DOI: 10.1145/2010324.1964932.

[76] A. Mercier-Aubin and P. G. Kry. A Multi-layer Solver for XPBD. *Computer Graphics Forum*, 43(8), 2024. DOI: 10.1111/cgf.15186.

[77] A. Mercier-Aubin and P. G. Kry. Adaptive Rigidification of Discrete Shells. *Proc. ACM Comput. Graph. Interact. Tech.*, 6(3):1–17, Aug. 2023. DOI: 10.1145/3606932.

[78] A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124.

[80] M. Müller. Hierarchical Position Based Dynamics. In F. Faure and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation*, volume 8, pages 1–10, Jan. 2008. DOI: 10.2312/PE/vriphys/vriphys08/001-010.

[82] M. Müller, N. Chentanez, M. Macklin, and S. Jeschke. Long Range Constraints for Rigid Body Simulations. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 1–10, Los Angeles, California. Association for Computing Machinery, 2017. DOI: 10.1145/3099564.3099574.

[83] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position Based Dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, Apr. 2007. DOI: 10.1016/j.jvcir.2007.01.005.

[84] M. Müller, M. Macklin, N. Chentanez, S. Jeschke, and T.-Y. Kim. Detailed Rigid Body Simulation with Extended Position Based Dynamics. *Computer Graphics Forum*, 39(8):101–112, 2020. DOI: 10.1111/cgf.14105.

[85] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., USA, 1st edition, Aug. 1994. DOI: 10.1201/9781315136370.

[94] L. Sacht, E. Vouga, and A. Jacobson. Nested Cages. *ACM Transactions on Graphics*, 34(6):1–14, Nov. 2015. DOI: 10.1145/2816795.2818093.

[98] M. Servin, C. Lacoursière, and N. Melin. Interactive Simulation of Elastic Deformable Materials. In *Proceedings of SIGRAD*, pages 22–32, Jan. 2006.

[103] R. E. Tarjan and J. van Leeuwen. Worst-Case Analysis of Set Union Algorithms. *Journal of the ACM*, 31(2):245–281, Mar. 1984. DOI: 10.1145/62.2160.

[107] Q.-M. Ton-That, P. G. Kry, and S. Andrews. Parallel block Neo-Hookean XPBD using graph clustering. *Computers & Graphics*, 110:1–10, 2023. DOI: 10.1016/j.cag.2022.10.009.

[108] R. Tonge, F. Benevolenski, and A. Voroshilov. Mass Splitting for Jitter-Free Parallel Rigid Body Simulation. *ACM Transactions on Graphics*, 31(4):1–8, July 2012. DOI: 10.1145/2185520.2185601.

[113] Z. Xian, X. Tong, and T. Liu. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Transactions on Graphics*, 38(6):1–13, Nov. 2019. DOI: 10.1145/3355089.3356486.

Chapter 6

Discussion

Physics-based animations have seen significant advancements over the past few years, frequently due to the demand for realistic and computationally efficient models in a wide range of applications, from computer graphics to engineering simulations. Among these advancements, adaptive rigidification has emerged as a powerful technique for optimizing the simulation of deformable bodies, offering a means to balance computational load with physical accuracy. Adaptive rigidification is especially valuable in scenarios where certain regions of a model undergo minimal deformation and can be treated as rigid without sacrificing the overall realism of the simulation.

Tangentially, extended position-based dynamics (XPBD) is a popular framework for real-time simulation of soft bodies. It is favored for its stability and ease of integration with constraint-based methods. XPBD is particularly well suited for applications that require high-speed computations, like real-time interactive simulations and virtual reality. However, as the complexity of simulations increases the need for improved convergence becomes apparent. This is the case in systems with numerous constraints or detailed interactions. Such speed and stability are requirements for the simulations of our industrial partner.

This discussion digs into the evolution of adaptive rigidification as a research idea, exploring our application of the concept to elastic solids and discrete shells, and culminating in the development of a multi-layer solver for XPBD. We compare the contributions to highlight their noteworthy differences. Next, we discuss the applications of our methods as well as their limitations. From there, we suggest potential improvements to the implementations. Following this, we address the methodology challenges, focusing on the reasoning behind our choices during the development and implementation of the

algorithms. Finally, we outline the steps taken to ensure that our methodology can be consistently replicated in other simulations.

6.1 Research Insights

The research that we present in this work traces the evolution of adaptive simulation techniques through a series of methodologies designed to balance computational efficiency and accuracy in physics-based simulations. Adaptive rigidification of elastic solids improves simulation performance by identifying low-strain regions within a deformable body and treating them as rigid. In a way, this is similar to adaptive merging, where rigid bodies with similar motions are merged to save on computation time. This allows the simulation to focus resources on actively deforming areas, thus reducing the computational load without sacrificing realism. The method involves using an oracle to guide when elements should switch from rigid back to elastic. A primary challenge lies in setting the correct criteria for rigidification, as over-rigidifying can lead to unnatural stiffness, while under-rigidifying negates performance benefits. Although there is some computational overhead in maintaining the oracle and strain-rate calculations, this overhead is negligible compared to the overall performance gains from reducing the number of degrees of freedom by treating parts of the mesh as rigid. However, adaptive rigidification of elastic solids only features the model reduction on soft bodies. To show the versatility of our work, we explore alternative types of models.

Building on the success of adaptive rigidification for elastic solids, the method now extends to discrete shells, which introduces unique challenges due to the distinct deformation characteristics of thin-walled structures like clothes or membranes. Unlike volumetric solids, discrete shells present no deformation in the normal direction and are prone to bending. Shells necessitate careful handling of geometric properties as they exacerbate the drawbacks of the oracle. This is why we use an edge filter inspired by image kernels common to computer vision in order to quickly diffuse contact information along neighbouring vertices. An innovation in this extension is the adaptation of our oracle to account for curvature, which allows for a smooth transition between rigid and

elastic regions, avoiding artefacts like unnaturally rigid creases. We prefer to minimize the amount of parameter tuning, hence we set the bending thresholds automatically from bending equivalences. However, adaptive rigidification still relies on an oracle that introduces complexity and the potential for errors, motivating exploration of fundamentally different approaches based on the concept of rigidification.

We then introduce the multi-layer solver, inspired by geometric and algebraic multigrid methods, as a way to eliminate the need for an oracle. Because we do not have predictions for elastification, we need the solution to be accurate. Likewise, our industrial partner uses XPBD for their simulations. In such framework, the preconditioner that we use for the oracle does not make sense as part of its appeal is the lack of the Hessian matrix assembly. As such, our multi-layer method solves two major issues at once. Unlike traditional adaptive rigidification, which selectively rigidifies regions based on strain rates, the multi-layer solver operates across hierarchical layers. This new solver quickly addresses large-scale dynamics in the early rigid layers and progressively adds new degrees of freedom for more detailed modelling. This approach eliminates the reliance on an external oracle because the refinement process is handled internally and the last layer is fully elastic. This leads to accurate solutions.

6.2 Comparing Approaches

There are several key differences and advantages that emerge when comparing the adaptive rigidification methods for elastic solids and discrete shells with the multi-layer solver for XPBD. Adaptive rigidification, while effective in reducing computational overhead, is heavily reliant on the oracle's ability to accurately predict regions of low deformation. This reliance introduces a level of complexity and potential inaccuracy that can limit the method's applicability, particularly in scenarios where deformation patterns are unpredictable or highly variable. However, because of these approximations, the performance can be pushed to levels beyond the ability of the multi-layer solver.

While rigidification improves computational performance, it can lead to a loss of detail in regions that are simplified into rigid bodies. This is particularly problematic if those

regions undergo unexpected deformation, which the rigid model might not be able to accurately capture. The potential locking artefacts for bending angles were concrete examples of such a problem, which we fixed in the second contribution by using an adequate contact filter. Special care might be required for other types of materials that we have not implemented in our work, such as snow, dirt, or fluids.

The multi-layer solver, on the other hand, offers a more generalized approach that does not depend on an external oracle. By leveraging a hierarchical refinement process, the multi-layer solver can dynamically adjust the level of detail in the simulation, ensuring that computational resources are focused where they are most needed. This method cheaply improves the initial elastic solution, similar to a warmstart heuristic. It also enhances the solver's ability to handle complex, dynamic interactions, such as contacts and collisions. We note, that performance is bounded by the fully elastic layer, preventing it from reaching the same levels of improvement as adaptive rigidification. If we instead opted to use an oracle for approximate solutions, then we could terminate at lower resolutions, but that is not the case here.

Tuning the adaptive rigidification threshold usually requires some level of intuition built over many simulations, even if we provide information on how to set reasonable threshold values. The multi-layer solver does not require such tuning, but does add the selection of layers and iteration count per layer. Regardless of the method, extra parameters are needed for the adaptive algorithms to work. In terms of ease of use, the multi-layer solver shows a clear advantage, particularly in large-scale simulations where the complexity of the system can vary significantly across different regions. By starting with a coarse, rigid approximation and gradually refining the model, the solver reduces the overall computational burden, allowing for faster and more accurate simulations. Because the layers are generated based on rigidification percentages, it is usually easier to tune.

Moreover, the hierarchical approach of the multi-layer solver could be extended to other types of simulations beyond XPBD. For example, it could be applied to finite element methods for physics-based animation or other constraint-based systems, potentially improving convergence and performance across a wide range of applications.

6.3 Applications

The introduction of the multi-layer solver is an innovative contribution to physics-based animation. The solver has the capacity to efficiently manage large, complex systems with varying degrees of rigidity and elasticity. It unlocks new opportunities for real-time applications, particularly in domains such as virtual reality and interactive simulations such as surgical training simulations. This innovation is valuable for our industrial partner, who specializes in surgery simulations integrated with haptic feedback devices. Their simulations must rapidly generate hundreds of frames per second to ensure precise synchronization with the haptic devices. Equally important is the need for stability in these simulations, as it is essential for providing accurate and plausible feedback through the device. Instabilities could lead to serious issues, such as the machine reacting unpredictably and potentially causing harm, like striking the surgeon due to poorly managed contact feedback. Moreover, the simulations must maintain a high degree of realism to fully immerse the user, as this is critical for effective learning and training. In such a context, there is no room for compromise, every aspect of the simulation, from performance to stability and realism, must be meticulously balanced. Fortunately, our multi-layer solver method is particularly well-suited to this application, it enables optimal performance, stability, and realism in localized regions like where the surgery happens.

6.4 Limitations

In the current adaptive rigidification implementations, we recompute rigid body properties only when elements are added or removed, but for small changes, there is potential to apply inexpensive incremental updates to mass properties and the rigid state. The main challenge arises when a rigid body splits into multiple components, and developing efficient algorithms for this remains a promising area of future work. Although the linear algorithm for generating connected components is not a bottleneck, incremental updates could further improve adaptive rigidification's efficiency. We currently only merge adjacent elements to form rigid bodies, but merging elements in contact could also enhance

performance. This approach, similar to Coevoet et al. [26], which merges rigid bodies to reduce the cost of collision detection and contact force computations, could speed up simulations involving stacked elastic bodies becoming rigid.

While adaptive rigidification of discrete shells is effective for simulating wrinkled, non-deforming regions, it lacks the capability to remesh actively deforming areas. However, adaptive rigidification could complement remeshing by efficiently reducing the size of the linear pass over elements and coarsening wrinkle-free areas. Unlike remeshing, which depends on the quality of coarsening, adaptive rigidification speedups are determined by thresholds and scene dynamics. While remeshing can efficiently handle flat, actively stretching areas, adaptive rigidification excels in rigidly moving dense folds.

Handling new contacts reduces the likelihood of missing elastification of slow and constant motions that still present challenges. Our contact handling, however, suffers from performance bottlenecks with large numbers of contacts, which could be improved by employing contact handling methods like that of Verschoor et al. [110] or pruning contacts on rigid bodies. Additionally, caching internal forces for rigidified elements could reduce computational costs.

Our multi-layer solver, built on XPBD, inherits some of its limitations, including similar convergence issues. While our method improves convergence in many cases, in the worst-case scenarios, it performs similarly to XPBD, particularly when the solver hits a plateau without early stopping.

Shells simulations are not yet designed for the multi-layer solver, as handling bending components with partial rigid layers is uncertain. It is likely that this method may encounter challenges in co-dimensional cases, such as in XPBD cloth simulations, where the material exhibits both stiff stretching constraints and very soft bending constraints. A possible solution could involve using rigidification layers only in the 2D projection of the shells while fully simulating bending. This would potentially be at the cost of larger bending energy computation time.

There exist opportunities to parallelize certain algorithms, like the union-find connected component builder. We could use approximate parallel sorting methods to optimize performance, noting that the elements need to be sorted with respect to the previous

set, but not within themselves. However, sorting and union-find are not the main bottlenecks in our implementation. Early stopping enhances simulation speed, but setting a fixed number of layers and iterations could be beneficial for real-time applications that require constant solve times.

While our multi-layer solver has similarities to multigrid methods, we did not see significant improvements to the performance while using a v-cycle pattern compared to to the more straightforward coarse-to-fine patterns. The v-cycle pattern, which involves transitioning from elastic to rigid states and back to elastic, sacrifices some of the advantages inherent in our current approach. More specifically, it loses the benefits of the residual velocity layers, which retain important velocity information as the solver transitions between layers and benefits from the ability to quickly terminate on rigid motions during the early layers, significantly reducing the computational load by addressing large-scale dynamics efficiently.

While this challenge is not unique to our approaches, the use of cutting techniques that require remeshing would also introduce significant complications in graph coloring as well as the adjacency graph. Dynamically updating the graph coloring at runtime is computationally expensive, which can constrain the practicality of implementing finite element methods, particularly in scenarios involving frequent cuts and topological changes. There are opportunities to improve graph coloring methods in the context of physical simulations, which are outside of the focus of this thesis.

6.5 Implementation Limitations

All of our contributions were developed within a unified codebase written in MATLAB. The MATLAB programming language offers an excellent environment for rapid prototyping. However, it lacks the compilation optimizations necessary for high-performance simulations, resulting in slower execution times. This performance gap shows the potential benefits of transitioning to a more optimized programming language. Julia, for instance, offers a compelling alternative. It combines the ease of prototyping found in MATLAB with the execution speed of more traditional compiled languages. One of Julia's core

strengths is that it allows you to write high-level code that is compiled to fast machine code with Just-In-Time compilation. Julia can approach or match the speed of C++ for many numerical tasks, without requiring the programmer to manage low-level details like memory allocation. Transitioning the codebase to a more open-source language like Julia could potentially yield even more impressive simulations with the possibility of truly achieving real-time performance, while not impeding development time. Likewise, it would also make the code more accessible to non-academic communities that may not have access to Matlab for free like we do in an academic setting.

Moreover, our current parallel processing is confined to CPU threads, which, while effective, is not optimal for achieving maximum performance. A proper GPU implementation could substantially accelerate the computations. During our preliminary exploration of GPU acceleration, we encountered a bottleneck due to the high number of kernel calls in the automatically generated CUDA code. This issue is likely resolvable with a manually optimized CUDA implementation, which would involve fine-tuning the GPU code to minimize kernel overhead and maximize parallel efficiency. Addressing this will require further work, but it represents a promising avenue for future optimization that could significantly enhance the speed and scalability of our simulations, but this is work for another time in potential future contributions.

We note that our implementations of adaptive rigidification feature standard collision detection and handling, but are not necessarily state-of-the-art. For instance, we use a one-level bounding volume hierarchy coupled with signed distance functions to generate collision points. We then handle contacts using projected Gauss-Seidel. Because our methods are compatible with other forms of collision detection and contact handling, we remove their computation time from our benchmarks and compare only the performance of the relevant pieces of the simulations. This is to ensure a fair comparison of only the relevant part of the simulation, independent of the collision handling and collision detection methods. Likewise, we also outsource the renders to Blender instead of considering them as a part of the simulation.

6.6 Evaluation Methodology Challenges

Throughout the entirety of our contributions, we consistently strive to provide a comprehensive and fair measure of performance by evaluating relative speeds. For example, the initial two papers feature speedup factors to highlight improvements. In a similar vein, the final paper takes a slightly different approach by evaluating the wall clock time required to improve the residual error by a specified percentage, rather than simply making them relative in terms of error reduction per step. This approach essentially serves to negate some of the potential impacts of our choice of programming language and the specific implementation strategies we employed.

In our results, we compare the adaptive solutions to the fully elastic simulations, referring to it as the ground truth. However, we note that this is an overstatement of the quality of the elastic simulations. In reality, we are undermining the performance of our algorithms for fairness of comparison. Because implicit time integrations are burdened by numerical damping, the fully elastic simulations are not all that accurate. Likewise, for XPBD, the constraint solve dissipates energy. However, the adaptive simulations are not as affected by these phenomena, making our simulations potentially more accurate than the standard elastic simulations in some scenarios. This is likely to be the case for most stiff simulations where models tend to act as rigid bodies throughout the animations. It is unclear how to properly assess the relevance of our adaptive simulations without downplaying their performance.

6.7 Reproducibility

Snapshots of the codebase are available online for all the papers. We hope that open-source access to the code will speed up adoption and inspire new ideas for research projects or future collaborative work. Likewise, we keep recordings of the full conference presentations to improve the quality of our research archives. Open-source research code plays an important role in ensuring the replicability of scientific discoveries. Researchers enable others to verify results and build upon existing work by making code publicly

available. This transparency cultivates trust in the scientific community and accelerates innovation by compelling researchers to collaborate. Moreover, open-source code helps to identify and correct bugs or errors, improving the overall quality and reliability of research. While we offer the code for our papers as is, we would be thrilled to see how the computer graphics community makes it evolve in time for their own contributions.

Chapter 7

Conclusion

The journey from adaptive rigidification to the development of a multi-layer solver for XPBD represents an interesting evolution in the field of physics-based simulation. Each step in this progression has addressed key challenges associated with balancing computational efficiency and simulation accuracy, leading to the creation of more robust and versatile simulation tools.

Adaptive rigidification provided a powerful method for reducing the computational load in simulations of elastic solids and discrete shells, allowing for more efficient simulations without sacrificing visual consistency. However, the reliance on an oracle to guide the rigidification process poses limitations, particularly in scenarios where deformation patterns are complex or consistently below the rigidification threshold over multiple steps.

The multi-layer solver for XPBD represents a novel approach to address these limitations. The multi-layer solver achieves faster convergence and greater accuracy, without the need for an external oracle. This is achieved by adopting a hierarchical approach that dynamically adjusts the level of rigidity with multiple rigid patterns evolving throughout the simulation. This approach not only improves performance in traditional XPBD simulations but also opens the door to new types of multigrid methods, generating resolutions based on environmental inputs rather than a predetermined set of geometrical models.

All the contributions of this thesis successfully achieve their goal of enhancing the simulation performances through adaptive reduction with rigidification. The adaptive rigidification and adjacent methods offer promising approaches to enhance the efficiency and stability of physics-based simulations, particularly in complex scenarios involving deformables. By dynamically identifying and leveraging rigid regions within deformable objects, this technique significantly reduces the computational burden without sacrific-

ing accuracy in capturing essential deformations. The ability to adaptively switch between rigid and elastic representations allows for more efficient simulations, particularly in scenes with an initially high degree of complexity or frequent interactions. Our methods improve the computation time often by orders of magnitude, while being relatively simple to implement and compatible with most existing frameworks. These approaches are intuitive at their core, focusing the attention only where needed the most in the simulations.

7.1 Future Work

Looking forward, there are numerous opportunities to expand upon the work presented here. Future work could include explore the applicability of our methods to other simulations models. Good candidates would be snow and dirt, which are subject to plasticity but quickly stop deforming after external interactions. These materials present an ideal use case for adaptive rigidification, as the ability to dynamically switch between deformable and rigid states could further optimize computational efficiency.

Additionally, integrating fast remeshing techniques within the adaptive rigidification framework could yield even more impressive speedups. This would be especially impactful for shell simulations, where remeshing excels at coarsening nearly flat regions, while rigidification allows coarsening in the bent regions.

Another promising direction is the development of a GPU-compatible version of our methods. Given the parallel nature of GPU architectures, this could substantially accelerate our simulations, improving scalability and making our approach even more suitable for large-scale applications. Such advancements would not only greatly increase simulation speed but also enable the handling of more complex physical interactions and higher-resolution models.

Future work could also explore an oracle for XPBD, making it compatible with adaptive rigidification for faster simulations when precision is less critical.

By continuing to refine and optimize the techniques presented in this thesis, we can push the boundaries of real-time physics-based simulations and open up new opportunities for both academic and industrial applications. The methods developed here could see

wide-ranging applications across fields such as interactive virtual environments, video games, movies, and training simulators. As the demands for both realism and performance continue to grow in these areas, the approaches outlined in this thesis have the opportunity to play a role in advancing the state of the art, making simulations more accessible and more powerful than ever before.

In conclusion, this thesis has laid the foundation for more efficient, scalable, and accurate simulations for deformable bodies. Our methods have the potential to significantly impact the field of physics-based simulation. By developing adaptive rigidification and the multi-layer solver approach, we have demonstrated that it is possible to achieve dramatic reductions in computational cost without sacrificing the accuracy and realism that are at the heart of modern simulations. The future of this field is bright, with countless opportunities for further innovation and exploration.

Bibliography

[1] M. Adams and J. W. Demmel. Parallel Multigrid Solver for 3D Unstructured Finite Element Problems. In *Proceedings of the 1999 ACM / IEEE Conference on Supercomputing*, SC '99, pages 27–45, Portland, Oregon, USA. Association for Computing Machinery, Jan. 1999. DOI: 10.1145/331532.331559.

[2] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. *Recent Advances in Remeshing of Surfaces*. In *Shape Analysis and Structuring*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pages 53–82. DOI: 10.1007/978-3-540-33265-7_2.

[3] S. Andrews, K. Erleben, and Z. Ferguson. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses*, SIGGRAPH '22, Vancouver, British Columbia, Canada. Association for Computing Machinery, 2022. DOI: 10.1145/3532720.3535640.

[4] L. Armijo. Minimization of Functions Having Lipschitz Continuous First Partial Derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, Jan. 1966. DOI: 10.2140/pjm.1966.16.1.

[5] S. Artemova and S. Redon. Adaptively Restrained Particle Simulations. *Physical Review Letters*, 109:1–5, 19, Nov. 2012. DOI: 10.1103/PhysRevLett.109.190201.

[6] D. Baraff and A. Witkin. Large Steps in Cloth Simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1998, pages 43–54, New York, NY, USA. Association for Computing Machinery, 1998. DOI: 10.1145/280814.280821.

[7] J. Barbič and D. L. James. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Transactions on Graphics*, 24(3):982–990, July 2005. DOI: 10.1145/1073204.1073300.

[8] L. Barbié, I. Ramière, and F. Lebon. An Automatic Multilevel Refinement Technique Based on Nested Local Meshes for Nonlinear Mechanics. *Computers & Structures*, 147:14–25, 2015. DOI: 10.1016/j.compstruc.2014.10.008. CIVIL-COMP.

[9] A. W. Bargteil, T. Shinar, and P. G. Kry. An Introduction to Physics-Based Animation. In *SIGGRAPH Asia 2020 Courses*, SA '20, pages 1–57, Virtual Event. Association for Computing Machinery, Nov. 2020. DOI: 10.1145/3415263.3419147.

[10] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering*, 1(1):1–16, 1972.

[11] N. Bell, Y. Yu, and P. J. Mucha. Particle-based Simulation of Granular Materials. In *Proceedings of the 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '05, pages 77–86, Los Angeles, California. Association for Computing Machinery, 2005. DOI: 10.1145/1073368.1073379.

[12] J. Bender and C. Deul. Efficient Cloth Simulation using an Adaptive Finite Element Method. In *Virtual Reality Interactions and Physical Simulations*, pages 21–30, Darmstadt, Germany. Eurographics Association, Dec. 2012. DOI: 10.2312/PE/vriphys/vriphys12/021-030.

[13] J. Bender, D. Koschier, P. Charrier, and D. Weber. Position-based simulation of continuous materials. *Computers & Graphics*, 44:1–10, 2014. DOI: 10.1016/j.cag.2014.07.004.

[14] D. Benson and J. Hallquist. Computation for Transient and Impact Dynamics. In S. Braun, editor, *Encyclopedia of Vibration*, pages 278–286. Elsevier, Oxford, 2001. DOI: 10.1006/rwvb.2001.0006.

[15] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Transactions on Graphics*, 33(4):1–11, July 2014. DOI: 10.1145/2601097.2601116.

[16] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004. DOI: 10.1017/CBO9780511804441.

[17] R. Bridson, R. Fedkiw, and J. Anderson. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Transactions on Graphics*, 21(3):594–603, July 2002. DOI: 10.1145/566654.566623.

[18] M. Bro-Nielsen and S. Cotin. Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation. *Computer Graphics Forum*, 15(3):57–66, 1996. DOI: 10.1111/1467-8659.1530057.

[19] S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popović. A Multiresolution Framework for Dynamic Deformations. In *Proceedings of the 2002 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '02, pages 41–47, San Antonio, Texas. Association for Computing Machinery, 2002. DOI: 10.1145/545261.545268.

[20] O. Cetinaslan. ESBD: Exponential Strain-based Dynamics using XPBD Algorithm. *Computers & Graphics*, 116:500–512, Nov. 2023. DOI: 10.1016/j.cag.2023.09.014.

[21] J. Chaskalovic. *Finite Element Methods for Engineering Sciences*. Springer Berlin, Heidelberg, 2008. DOI: 10.1007/978-3-540-76343-7.

[22] D. Chen, D. I. W. Levin, W. Matusik, and D. M. Kaufman. Dynamics-Aware Numerical Coarsening for Fabrication Design. *ACM Transactions on Graphics*, 36(4):1–15, July 2017. DOI: 10.1145/3072959.3073669.

[23] J. Chen, K. Nonaka, H. Sankoh, R. Watanabe, H. Sabirin, and S. Naito. Efficient Parallel Connected Component Labeling With a Coarse-to-Fine Strategy. *IEEE Access*, 6:55731–55740, 2018. DOI: 10.1109/ACCESS.2018.2872452.

[24] F. Cirak, M. Ortiz, and P. Schröder. Subdivision Surfaces: A New Paradigm for Thin-shell Finite-element Analysis. *International Journal for Numerical Methods in Engineering*, 47(12):2039–2072, 2000. DOI: 10.1002/(SICI)1097-0207(20000430)47:12<2039::AID-NME872>3.0.CO;2-1.

[25] M. Cline and D. Pai. Post-stabilization for Rigid Body Simulation with Contact and Constraints. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 3744–3751, 2003. DOI: 10.1109/ROBOT.2003.1242171.

[26] E. Coevoet, O. Benchekroun, and P. G. Kry. Adaptive Merging for Rigid Body Simulation. *ACM Transactions on Graphics*, 39(4):1–13, Aug. 2020. DOI: 10.1145/3386569.3392417.

[27] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018.

[28] H. Courtecuisse, J. Allard, C. Duriez, and S. Cotin. Asynchronous Preconditioners for Efficient Solving of Non-linear Deformations. In K. Erleben, J. Bender, and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation*. The Eurographics Association, 2010. DOI: 10.2312/PE/vriphys/vriphys10/059-068.

[29] A. R. Curtis and J. K. Reid. On the Automatic Scaling of Matrices for Gaussian Elimination. *IMA Journal of Applied Mathematics*, 10(1):118–124, Aug. 1972. DOI: 10.1093/imamat/10.1.118.

[30] G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Adaptive Simulation of Soft Bodies in Real-time. In *Proceedings of the Computer Animation*, pages 15–20, May 2000. DOI: 10.1109/CA.2000.889022.

[31] G. Debunne, M. Desbrun, M.-P. Cani, and A. 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, pages 31–36, New York, NY, USA. Association for Computing Machinery, Aug. 2001. DOI: 10.1145/383259.383262.

[32] E. English and R. Bridson. Animating Developable Surfaces using Nonconforming Elements. *ACM Transactions on Graphics*, 27(3):1–5, Aug. 2008. DOI: 10.1145/1360612.1360665.

[33] K. Erleben. *Stable, robust, and versatile multibody dynamics animation*. PhD thesis, University of Copenhagen, 2004.

[34] O. Etzmub, M. Keckeisen, and W. Straber. A Fast Finite Element Solution for Cloth Modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 244–251, USA. IEEE Computer Society, 2003. DOI: 10.1109/PCCGA.2003.1238266.

[35] Z. Ferguson, T. Schneider, D. Kaufman, and D. Panozzo. In-Timestep Remeshing for Contacting Elastodynamics. *ACM Transactions on Graphics*, 42(4):1–15, July 2023. DOI: 10.1145/3592428.

[36] M. Fratarcangeli and F. Pellacini. Scalable Partitioning for Parallel Position Based Dynamics. *Computer Graphics Forum*, 34(2):405–413, 2015. DOI: 10.1111/cgf.12570.

[37] J. Georgii and R. Westermann. Corotated Finite Elements Made Fast and Stable. In F. Faure and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation*. The Eurographics Association, 2008. DOI: 10.2312/PE/vriphys/vriphys08/011-019.

[38] B. Gilles, G. Bousquet, F. Faure, and D. K. Pai. Frame-Based Elastic Models. *ACM Transactions on Graphics*, 30(2):1–12, Apr. 2011. DOI: 10.1145/1944846.1944855.

[39] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient Simulation of Inextensible Cloth. *ACM Transactions on Graphics*, 26(3):49–57, July 2007. DOI: 10.1145/1276377.1276438.

[40] R. D. Gregory. *Classical Mechanics*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511803789.

[41] R. D. Gregory. *Lagrange's equations and conservation principles*. In *Classical Mechanics*. Cambridge University Press, 2006, pages 323–365. DOI: 10.1017/CBO9780511803789.013.

[42] M. Griebel, D. Oeltz, and M. A. Schweitzer. An Algebraic Multigrid Method for Linear Elasticity. *SIAM Journal on Scientific Computing*, 25(2):385–407, 2003. DOI: 10.1137/S1064827502407810.

[43] E. Grinspun, A. N. Hirani, M. Desbrun, and P. Schröder. Discrete Shells. In D. Breen and M. Lin, editors, *Proceedings of the 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '03, pages 62–67, San Diego, California. The Eurographics Association, 2003. DOI: 10.2312/SCA03/062-067.

[44] E. Grinspun, P. Krysl, and P. Schröder. CHARMS: A Simple Framework for Adaptive Simulation. *ACM Transactions on Graphics*, 21(3):281–290, July 2002. DOI: 10.1145/566654.566578.

[45] H. Guillard. Node-nested Multi-grid Method with Delaunay Coarsening. Research Report RR-1898, INRIA, 1993.

[46] S. Gupta, D. Palsetia, M. M. A. Patwary, A. Agrawal, and A. Choudhary. A New Parallel Algorithm for Two-Pass Connected Component Labeling. In *IEEE International Parallel & Distributed Processing Symposium Workshops*, pages 1355–1362, 2014. DOI: 10.1109/IPDPSW.2014.152.

[47] F. Hahn, B. Thomaszewski, S. Coros, R. W. Sumner, F. Cole, M. Meyer, T. DeRose, and M. Gross. Subspace Clothing Simulation Using Adaptive Bases. *ACM Transactions on Graphics*, 33(4):1–9, July 2014. DOI: 10.1145/2601097.2601160.

[48] F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien. Updated Sparse Cholesky Factors for Corotational Elastodynamics. *ACM Transactions on Graphics*, 31(5):1–13, Sept. 2012. DOI: 10.1145/2231816.2231821.

[49] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo. Tetrahedral Meshing in the Wild. *ACM Transactions on Graphics*, 37(4):1–14, July 2018. DOI: 10.1145/3197517.3201353.

[50] A. Jacobson. gptoolbox: Geometry Processing Toolbox, 2021.

[51] A. Jacobson, Z. Deng, L. Kavan, and J. Lewis. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, Vancouver, Canada. Association for Computing Machinery, 2014. DOI: 10.1145/2614028.2615427.

[52] J. Jansson and J. S. M. Vergeest. Combining Deformable and Rigid-Body Mechanics Simulation. *The Visual Computer*, 19(5):280–290, Aug. 2003. DOI: 10.1007/s00371-002-0187-6.

[53] X. Jiao, A. Colombi, X. Ni, and J. C. Hart. Anisotropic Mesh Adaptation for Evolving Triangulated Surfaces. In P. P. Pébay, editor, *Proceedings of the 15th International Meshing Roundtable*, pages 173–190, Berlin, Heidelberg. Springer Berlin Heidelberg, 2006. DOI: 10.1007/978-3-540-34958-7_11.

[54] B. Jones, N. Thuerey, T. Shinar, and A. W. Bargteil. Example-based Plastic Deformation of Rigid Bodies. *ACM Transactions on Graphics*, 35(4):1–11, July 2016. DOI: 10.1145/2897824.2925979.

[55] L. Kavan, D. Gerszewski, A. W. Bargteil, and P.-P. Sloan. Physics-Inspired Upsampling for Cloth Simulation in Games. In *ACM Transactions on Graphics*, volume 30 of number 4, pages 1–10, New York, NY, USA. Association for Computing Machinery, Jan. 2011. DOI: 10.1145/1964921.1964988.

[56] C. T. Kelley. *Solving Nonlinear Equations with Newton's Method*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718898.

[57] L. Kharevych, P. Mullen, H. Owhadi, and M. Desbrun. Numerical Coarsening of Inhomogeneous Elastic Materials. *ACM Transactions on Graphics*, 28(3):1–8, July 2009. DOI: 10.1145/1531326.1531357.

[58] R. Kikuuwe, H. Tabuchi, and M. Yamamoto. An Edge-based Computationally Efficient Formulation of Saint Venant-Kirchhoff Tetrahedral Finite Elements. *ACM Transactions on Graphics*, 28(1):1–13, Feb. 2009. DOI: 10.1145/1477926.1477934.

[59] T.-Y. Kim, N. Chentanez, and M. Müller-Fischer. Long Range Attachments a Method to Simulate Inextensible Clothing in Computer Games. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '12, pages 305–310, Lausanne, Switzerland. Eurographics Association, 2012.

[60] T. Kim and D. L. James. Skipping steps in deformable simulation with online model reduction. *ACM Transactions on Graphics*, 28(5):1–9, Dec. 2009. DOI: 10.1145/1618452.1618469.

[61] J. Lenoir and S. Fonteneau. Mixing Deformable and Rigid-body Mechanics Simulation. In *Proceedings Computer Graphics International*, pages 327–334, 2004. DOI: 10.1109/CGI.2004.1309229.

[62] D. I. Levin. Physics-based animation lecture 6: cloth simulation, Oct. 2020.

[63] L. Li and V. Volkov. Cloth Animation with Adaptively Refined Meshes. In *Proceedings of the Twenty-Eighth Australasian Conference on Computer Science*, ACSC '05, pages 107–113, Newcastle, Australia. Australian Computer Society, Inc., 2005.

[64] M. Li, D. M. Kaufman, and C. Jiang. Codimensional Incremental Potential Contact. *ACM Transactions on Graphics*, 40(4):1–24, Jan. 2021. DOI: 10.1145/3450626.3459767.

[65] T. Liu. Projective dynamics/simulation. Symposium on Geometry Processing Graduate School, 2021.

[66] T. Liu, S. Bouaziz, and L. Kavan. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Transactions on Graphics*, 36(4):1–16, May 2017. DOI: 10.1145/3072959.2990496.

[67] C. Loop. *Smooth Subdivision Surfaces Based on Triangles*. PhD thesis, The University of Utah, Jan. 1987.

[68] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Kim. Primal/Dual Descent Methods for Dynamics. *Computer Graphics Forum*, 39(8):89–100, 2020. DOI: 10.1111/cgf.14104.

[69] M. Macklin and M. Muller. A Constraint-based Formulation of Stable Neo-Hookean Materials. In *Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games*, MIG '21, pages 1–7, Virtual Event, Switzerland. Association for Computing Machinery, 2021. DOI: 10.1145/3487983.3488289.

[70] M. Macklin, M. Müller, and N. Chentanez. XPBD: Position-Based Simulation of Compliant Constrained Dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, pages 49–54, Burlingame, California. Association for Computing Machinery, 2016. DOI: 10.1145/2994258.2994272.

[71] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics*, 33(4):1–12, July 2014. DOI: 10.1145/2601097.2601152.

[72] P.-L. Manteaux, C. Wojtan, R. Narain, S. Redon, F. Faure, and M.-P. Cani. Adaptive Physically Based Models in Computer Graphics. *Computer Graphics Forum*, 36(6):312–337, Sept. 2017. DOI: 10.1111/cgf.12941.

[73] P.-L. Manteaux, F. Faure, S. Redon, and M.-P. Cani. Exploring the Use of Adaptively Restrained Particles for Graphics Simulations. In *Proceedings of the 10th Workshop on Virtual Reality Interaction and Physical Simulation*, pages 17–24, Lille, France. Eurographics Association, Nov. 2013. DOI: 10.2312/PE.vriphys.vriphys13.017-024.

[74] J. Marins, X. Yun, E. Bachmann, R. McGhee, and M. Zyda. An Extended Kalman Filter for Quaternion-based Orientation Estimation Using MARG Sensors. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 4, pages 2003–2011, 2001. DOI: 10.1109/IROS.2001.976367.

[75] A. McAdams, Y. Zhu, A. Selle, M. Empey, R. Tamstorf, J. Teran, and E. Sifakis. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Transactions on Graphics*, 30(4):1–12, July 2011. DOI: 10.1145/2010324.1964932.

[76] A. Mercier-Aubin and P. G. Kry. A Multi-layer Solver for XPBD. *Computer Graphics Forum*, 43(8), 2024. DOI: 10.1111/cgf.15186.

[77] A. Mercier-Aubin and P. G. Kry. Adaptive Rigidification of Discrete Shells. *Proc. ACM Comput. Graph. Interact. Tech.*, 6(3):1–17, Aug. 2023. DOI: 10.1145/3606932.

[78] A. Mercier-Aubin, A. Winter, D. I. W. Levin, and P. G. Kry. Adaptive Rigidification of Elastic Solids. *ACM Transactions on Graphics*, 41(4):1–11, July 2022. DOI: 10.1145/3528223.3530124.

[79] E. Miguel, D. Bradley, B. Thomaszewski, B. Bickel, W. Matusik, M. A. Otaduy, and S. Marschner. Data-Driven Estimation of Cloth Simulation Models. *Computer Graphics Forum*, 31(2pt2):519–528, May 2012. DOI: 10.1111/j.1467-8659.2012.03031.x.

[80] M. Müller. Hierarchical Position Based Dynamics. In F. Faure and M. Teschner, editors, *Workshop in Virtual Reality Interactions and Physical Simulation*, volume 8, pages 1–10, Jan. 2008. DOI: 10.2312/PE/vriphys/vriphys08/001-010.

[81] M. Müller and N. Chentanez. Wrinkle Meshes. In M. Popovic and M. Otaduy, editors, *Proceedings of the 2010 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '10, pages 85–92, Madrid, Spain. The Eurographics Association, 2010. DOI: 10.2312/SCA/SCA10/085-091.

[82] M. Müller, N. Chentanez, M. Macklin, and S. Jeschke. Long Range Constraints for Rigid Body Simulations. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 1–10, Los Angeles, California. Association for Computing Machinery, 2017. DOI: 10.1145/3099564.3099574.

[83] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position Based Dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, Apr. 2007. DOI: 10.1016/j.jvcir.2007.01.005.

[84] M. Müller, M. Macklin, N. Chentanez, S. Jeschke, and T.-Y. Kim. Detailed Rigid Body Simulation with Extended Position Based Dynamics. *Computer Graphics Forum*, 39(8):101–112, 2020. DOI: 10.1111/cgf.14105.

[85] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., USA, 1st edition, Aug. 1994. DOI: 10.1201/9781315136370.

[86] R. Narain, T. Pfaff, and J. F. O'Brien. Folding and Crumpling Adaptive Sheets. *ACM Transactions on Graphics*, 32(4):1–8, July 2013. DOI: 10.1145/2461912.2462010. Proceedings of ACM SIGGRAPH 2013, Anaheim.

[87] R. Narain, A. Samii, and J. F. O'Brien. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Transactions on Graphics*, 31(6):1–10, Nov. 2012. DOI: 10.1145/2366145.2366171.

[88] M. Nesme, P. G. Kry, L. Jeřábková, and F. Faure. Preserving Topology and Elasticity for Embedded Deformable Models. *ACM Transactions on Graphics*, 28(3):1–9, July 2009. DOI: 10.1145/1531326.1531358.

[89] R. W. Ogden and E. Sternberg. Nonlinear Elastic Deformations. *Journal of Applied Mechanics*, 52(3):740–741, Sept. 1985. DOI: 10.1115/1.3169137.

[90] T. Pfaff, R. Narain, J. M. de Joya, and J. F. O'Brien. Adaptive Tearing and Cracking of Thin Sheets. *ACM Transactions on Graphics*, 33(4):1–9, July 2014. DOI: 10.1145/2601097.2601132.

[91] T. Popa, Q. Zhou, D. Bradley, V. Kraevoy, H. Fu, A. Sheffer, and W. Heidrich. Wrinkling Captured Garments Using Space-Time Data-Driven Deformation. *Computer Graphics Forum*, 28(2):427–435, 2009. DOI: 10.1111/j.1467-8659.2009.01382.x.

[92] X. Provot. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behaviour. In *Proceedings of Graphics Interface*, GI '95, pages 147–154, Quebec, Quebec, Canada. Canadian Human-Computer Communications Society, 1995. DOI: 10.20380/GI1995.17.

[93] D. Rohmer, T. Popa, M.-P. Cani, S. Hahmann, and A. Sheffer. Animation Wrinkling: Augmenting Coarse Cloth Simulations with Realistic-Looking Wrinkles. *ACM Transactions on Graphics*, 29(6):1–8, Dec. 2010. DOI: 10.1145/1882261.1866183.

[94] L. Sacht, E. Vouga, and A. Jacobson. Nested Cages. *ACM Transactions on Graphics*, 34(6):1–14, Nov. 2015. DOI: 10.1145/2816795.2818093.

[95] W. Scherzinger and C. Dohrmann. A Robust Algorithm for Finding the Eigenvalues and Eigenvectors of 3×3 Symmetric Matrices. *Computer Methods in Applied Mechanics and Engineering*, 197(45):4007–4015, 2008. DOI: 10.1016/j.cma.2008.03.031.

[96] H. Schmidl and V. J. Milenkovic. A Fast Impulsive Contact Suite for Rigid Body Simulation. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):189–197, Mar. 2004. DOI: 10.1109/TVCG.2004.1260770.

[97] C. Schreck, D. Rohmer, S. Hahmann, M.-P. Cani, S. Jin, C. C. L. Wang, and J.-F. Bloch. Nonsmooth Developable Geometry for Interactively Animating Paper Crumpling. *ACM Transactions on Graphics*, 35(1):1–18, Dec. 2016. DOI: 10.1145/2829948.

[98] M. Servin, C. Lacoursière, and N. Melin. Interactive Simulation of Elastic Deformable Materials. In *Proceedings of SIGRAD*, pages 22–32, Jan. 2006.

[99] E. Sifakis and J. Barbic. FEM Simulation of 3D Deformable Solids: A Practitioner's Guide to Theory, Discretization and Model Reduction. In *ACM SIGGRAPH 2012 Courses*, SIGGRAPH 2012, pages 1–50, Los Angeles, California. Association for Computing Machinery, Aug. 2012. DOI: 10.1145/2343483.2343501.

[100] B. Smith, F. D. Goes, and T. Kim. Stable Neo-Hookean Flesh Simulation. *ACM Transactions on Graphics*, 37(2):1–15, July 2018. DOI: 10.1145/3180491.

[101] R. Smith et al. *Open dynamics engine user manual*. 2005.

[102] R. Tamstorf and E. Grinspun. Discrete bending forces and their Jacobians. *Graphical Models*, 75(6):362–370, 2013. DOI: 10.1016/j.gmod.2013.07.001.

[103] R. E. Tarjan and J. van Leeuwen. Worst-Case Analysis of Set Union Algorithms. *Journal of the ACM*, 31(2):245–281, Mar. 1984. DOI: 10.1145/62.2160.

[104] Y. Teng, M. Meyer, T. DeRose, and T. Kim. Subspace Condensation: Full Space Adaptivity for Subspace Deformations. *ACM Transactions on Graphics*, 34(4):1–9, July 2015. DOI: 10.1145/2766904.

[105] D. Terzopoulos and A. Witkin. Physically Based Models with Rigid and Deformable Components. *IEEE Computer Graphics and Applications*, 8(6):41–51, 1988. DOI: 10.1109/38.20317.

[106] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically Deformable Models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1987, pages 205–214, New York, NY, USA. Association for Computing Machinery, July 1987. DOI: 10.1145/37401.37427.

[107] Q.-M. Ton-That, P. G. Kry, and S. Andrews. Parallel block Neo-Hookean XPBD using graph clustering. *Computers & Graphics*, 110:1–10, 2023. DOI: 10.1016/j.cag.2022.10.009.

[108] R. Tonge, F. Benevolenski, and A. Voroshilov. Mass Splitting for Jitter-Free Parallel Rigid Body Simulation. *ACM Transactions on Graphics*, 31(4):1–8, July 2012. DOI: 10.1145/2185520.2185601.

[109] M. Tournier, M. Nesme, F. Faure, and B. Gilles. Seamless Adaptivity of Elastic Models. In *Proceedings of Graphics Interface*, GI 2014, pages 17–24, Montréal, Québec, Canada. Canadian Human-Computer Communications Society, 2014.

[110] M. Verschoor and A. C. Jalba. Efficient and Accurate Collision Response for Elastically Deformable Models. *ACM Transactions on Graphics*, 38(2):1–20, Mar. 2019. DOI: 10.1145/3209887.

[111] H. Wang, J. O'Brien, and R. Ramamoorthi. Multi-Resolution Isotropic Strain Limiting. *ACM Transactions on Graphics*, 29(6):1–10, Dec. 2010. DOI: 10.1145/1882261.1866182.

[112] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, and J. F. O'Brien. Dynamic Local Remeshing for Elastoplastic Simulation. *ACM Transactions on Graphics*, 29(4):1–11, July 2010. DOI: 10.1145/1778765.1778786.

[113] Z. Xian, X. Tong, and T. Liu. A Scalable Galerkin Multigrid Method for Real-Time Simulation of Deformable Objects. *ACM Transactions on Graphics*, 38(6):1–13, Nov. 2019. DOI: 10.1145/3355089.3356486.

[114] Y. Yang, G. Rong, L. Torres, and X. Guo. Real-time Hybrid Solid Simulation: Spectral Unification of Deformable and Rigid Materials. *Computer Animation and Virtual Worlds*, 21(3-4):151–159, 2010. DOI: 10.1002/cav.373.

[115] Y. Zhang, A. Azad, and A. Buluç. Parallel Algorithms for Finding Connected Components using Linear Algebra. *Journal of Parallel and Distributed Computing*, 144:14–27, 2020. DOI: 10.1016/j.jpdc.2020.04.009.

[116] Y. Zhu, E. Sifakis, J. Teran, and A. Brandt. An Efficient Multigrid Method for the Simulation of High-Resolution Elastic Solids. *ACM Transactions on Graphics*, 29(2):1–18, Mar. 2010. DOI: 10.1145/1731047.1731054.

[117] O. Zienkiewicz, R. Taylor, and J. Zhu. Appendix H: Matrix Diagonalization or Lumping. In O. Zienkiewicz, R. Taylor, and J. Zhu, editors, *The Finite Element Method: its Basis and Fundamentals*, pages 689–695. Butterworth-Heinemann, Oxford, seventh edition edition, 2013. DOI: 10.1016/B978-1-85617-633-0.00034-4.