University of California
Santa Barbara

Efficient Deformations Using Custom Coordinate Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Yun Teng

Committee in charge:

Professor Theodore Kim, Co-Chair
Professor Tobias Höllerer, Co-Chair
Professor Linda Petzold

September 2016

The Dissertation of Yun Teng is approved.

Professor Linda Petzold

Professor Tobias Höllerer, Committee Co-Chair

Professor Theodore Kim, Committee Co-Chair

July 2016

Efficient Deformations Using Custom Coordinate Systems

by

Yun Teng

Acknowledgements

I owe gratitude to a great many people for making this dissertation possible and for making my graduate school years such an amazing journey.

My deepest gratitude is to my advisor, Professor Theodore Kim. Your inspiring classes on various aspects of Computer Graphics has made me fall in love with this field. You have been guiding me through various stages of research, while allowing me the freedom to explore on my own. Your encouragement and support helped me overcome numerous difficulties over the years and led to this dissertation.

I am sincerely grateful to my co-advisor, Professor Tobias Höllerer. Your insightful comments and constructive suggestions greatly improved the quality of my research work as well as my understanding of the research field. I am very thankful to you for the fruitful discussions that helped me build up my research road map, clarify technical details, and refine presentation quality.

My sincere appreciation to Professor Linda Petzold for serving on my committee. Your class on numerical simulation prepared a sound theoretical foundation for all my later work and ignited my research interest in this area. You have been always there to listen, provide advice and navigate me through various technical difficulties and inspire my career development.

I have been lucky to have had many excellent collaborators: Professor Miguel A. Otaduy, Dr. Tony DeRose, Dr. Mark Meyer and Dr. David I.W. Levin. I am constantly inspired by your brilliant ideas and never-ending enthusiasm for research problems. Special thanks to Tony and Mark for two awesome summers at Pixar!

Many thanks to my dearest friends, for the long-time companionship and support.

Finally, I am deeply indebted to my family. To my mom and dad, thank you for all your unconditional love and support. To my dearest husband, those long distance drives

during the weekends are finally over.

To all of you I dedicate this thesis.

This research was sponsored by the a National Science Foundation CAREER award (IIS-1253948) and a Google Ph.D. Fellowship.

Curriculum Vitæ

Yun Teng

Education

2016	Ph.D. in Computer Science, University of California, Santa Barbara.
2011	B.S. in Computer Science, Xiamen University.

Publications

Yun Teng, David I.W. Levin and Theodore Kim. Eulerian solid-fluid coupling. (Under review for SIGGRAPH Asia 2016)

Yun Teng, Mark Meyer, Tony DeRose and Theodore Kim. Subspace condensation: full space adaptivity for subspace deformations. *ACM Transactions on Graphics*, 34(4) (2015).

Yun Teng, Miguel A.Otaduy and Theodore Kim. Simulating articulated subspace self-contact. *ACM Transactions on Graphics*, 33(4) (2014).

Yun Teng, Rashaad Jones, Laura Marusich, John O'Donovan, Cleotilde Gonzalez and Tobias Höllerer. Trust and situation awareness in a 3-player diner's dilemma game. *IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support* (2013).

John O'Donovan, Rashaad Jones, Laura Marusich, Yun Teng, Cleotilde Gonzalez and Tobias Höllerer. A model-based evaluation of trust and situation awareness in the diners dilemma game. *Proceedings of the 22th Behavior Representation in Modeling & Simulation* (2013).

Abstract

Efficient Deformations Using Custom Coordinate Systems

by

Yun Teng

Physics-based deformable object simulations have been playing an increasingly important role in 3D computer graphics. They have been adopted for humanoid character animations as well as special effects such as fire and explosion. However, simulations of large, complex systems can consume large amounts of computation and mostly remain offline, which prohibits their use for interactive applications.

We present several highly efficient schemes for deformable object simulation using custom spatial coordinate systems. Our choices span the spectrum of subspace to full space and both Lagrangian and Eulerian viewpoints.

Subspace methods achieve massive speedups over their "full space" counterparts by drastically reducing the degrees of freedom involved in the simulation. A long standing difficulty in subspace simulation is incorporating various non-linearities. They introduce expensive computational bottlenecks and quite often cause novel deformations that are outside the span of the subspace.

We address these issues in articulated deformable body simulations from a Lagrangian viewpoint. We remove the computational bottleneck of articulated self-contact handling by deploying a *pose-space cubature* scheme, a generalization of the standard "cubature" approximation. To handle novel deformations caused by arbitrary external collisions, we introduce a generic approach called *subspace condensation*, which activates full space simulation on the fly when an out-of-basis event is encountered. Our proposed framework efficiently incorporates various non-linearities and allows subspace methods to be used

in cases where they previously would not have been considered.

Deformable solids can interact not only with each other, but also with fluids. We design a new full space method that achieves a two-way coupling between deformable solids and an incompressible fluid where the underlying geometric representation is entirely Eulerian. No-slip boundary conditions are automatically satisfied by imposing a global divergence-free condition. We are able to simulate multiple solids undergoing complex, frictional contact while simultaneously interacting with a fluid. The complexity of the scenarios we are able to simulate surpasses those that we have seen from any previous method.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Recent years have seen wide adoption of physics-based deformable object simulation in 3D computer animations. For example, the contraction of Eret's biceps in *How to Train your Dragon 2* [1], the stunning snow phenomena in *Frozen* [2] and the amazing water effects in *The Good Dinosaur* [3] are all achieved through physical simulation. Any deformable object simulation involves a time and space discretization, and a huge amount of literature exists in the applied mathematics, mechanical engineering, as well as computer graphics community. This dissertation focuses on the selection of the spatial coordinate systems for efficient simulations. Based on the simulation domain, space discretization can be performed from either a Lagrangian or an Eulerian viewpoint. The Lagrangian viewpoint breaks the continuum itself into a set of particles (material points) and updates their states (positions, velocities, etc.) in each simulation step. It can be further categorized to finite element/volume methods, mass-spring systems and mesh-free particle systems. The Eulerian viewpoint looks at how measurements of the continuum such as density, velocity, etc. at fixed points in space change over time. The domain of interest in the world space is usually diced up into regular grid cells. Solids are typically simulated in a Lagrangian way and Eulerian approaches are more popular for fluids,

although the opposite is also actively being developed [4, 5]. Excellent surveys [6, 7] exist to outline the physics-based deformable models in computer graphics and discuss their pros and cons.

In either viewpoint, a high resolution discretization is required in order to reveal fine details such as creases around an elbow or complex vorticities in a smoke ring, which results in numerous variables that need to be accounted for. With advances in computer hardwares, we are allowed to do bigger simulations than ever before, while at the same time facing the demand for even higher resolutions. The high non-linearities and enormous degrees of freedom in a complex system render the simulation very expensive, and hence prevent its use in interactive applications. This is also a major reason that low-cost geometric skinning approaches [8, 9] are still very popular in the motion pictures and gaming industries for character animations, even though they lack flexibility in supporting contact behaviors or dynamic effects.

A large body of work has been devoted to improving the efficiency of physics based deformations. Classic spatially adaptive methods [10] or modern GPU implementations [11] can yield speedups of roughly an order of magnitude, which brings a simulation that takes 5 minutes per frame down to 30 seconds, but is still far from the 30 Hz requirement for interactive applications. The complexity of these algorithms depends on the resolution of the discretization, which inevitably scale poorly for high resolution simulations.

Subspace methods, also known as model reduction, or reduced order methods, provides a nice alternative to the *full space* acceleration techniques described above. They have been recently used to accelerate a variety of simulations in computer graphics, such as deformable bodies [12], fluids [13] and clothing [14]. These methods detect temporal redundancies in the simulation during a precomputation stage and use them to construct efficient, low-dimensional subspaces. Simulations are performed in these subspaces at runtime, and routinely yield speedups that are orders of magnitude faster than their

full-rank counterparts.

Until recently, the efficient use of these approaches have been largely restricted to linear problems [15, 16]. If a quantity involved in the simulation is non-linear (e.g. force response of a non-linear type of material), it needs to be recomputed in every simulation frame. A straightforward approach is to compute the non-linear quantity in full space and then project it into subspace [17, 18], but this re-introduces full-rank computation complexity and dials back the potential speedups to a single order of magnitude. When the non-linearities can be expressed as a tensor, projected tensor approaches [12, 13, 19] are applied to accelerate their computation. Unfortunately, this assumption rarely holds for general non-linear problems. Alternatively, the *cubature* scheme [20] can be applied to approximate arbitrary subspace non-linear quantities through sparse point sampling. It has been successfully applied to the subspace material forces and Jacobians of a variety of non-linear materials [20], and used to quickly assemble the subspace velocity field after the advection step in fluid simulation [21], as well as geometric reduction for material optimization [22]. However, its efficiency in approximating higher non-linearities is to be determined.

Additionally, subspace methods share a fundamental limitation: the motions are nothing but linear combinations of the basis vectors, and thus cannot represent deformations that are not well-captured by the subspace. This problem is usually referred to as *locking*. A variety of strategies have been proposed to alleviate this issue, including basis enrichment [23], adaptive basis construction [14], and falling back to brute force, full-order computation over the entire mesh [24].

The story gets even more complicated when it comes to multi-phase simulation, i.e. , solid-fluid coupling. As mentioned above, solids and fluids are typically simulated in Lagrangian and Eulerian coordinates, respectively. In such cases, the boundary handling across different objects requires a negotiation between different coordinate systems, which

typically involves sophisticated geometry operations [25, 26, 27]. Although subspace methods exist for both Lagrangian and Eulerian simulations [12, 28], no such work has been developed for coupled systems. The idea of a unified solver, where the underlying geometry is either entirely Lagrangian or entirely Eulerian, is certainly appealing. To date, most attempts at such solver have been Lagrangian [29, 30, 31, 32]. However, their underlying geometries are either particle systems or topologically-changing meshes, which are not amenable to subspace adoption either.

The first half of this dissertation tackle the limitations of subspace methods in the context of articulated self-contact and arbitrary external collisions that occur in deformable character simulations. The second half proposes a unified, Eulerian framework for simulating fully coupled deformable solids and an incompressible fluid that can potentially be adopted for subspace simulation. We show that these common non-linear phenomena in deformable object simulations can be computed efficiently using custom coordinate systems. The reason that these particular problems are of interest is because they play a key role in human simulations. Solving them efficiently will get us one step closer to interactive virtual humans, which is also one of the main challenges of this dissertation. Next, we give a brief description of each of these problems.

1.1 Articulated Self-Contact

Articulated self-contact refers to skin contact deformations caused by joint movement. They convey important visual cues for muscle movement, e.g. characteristic bulges that form as limbs fold or a hand closes into a fist. Dense self-contact routinely arises on an articulated mesh, and can consume large computation times. This is less an issue in full space simulation, since even more dominant factors such as formulating the material force Jacobians and solving the large linearized system exist. However, it truly

becomes a bottleneck in subspace simulation. As the other stages are sufficiently accelerated, self-contact handling can consume more than 90% of the computation time in a single simulation frame. Since these contacts are driven by the articulation and not unpredictable external forces, they display precisely the type of coherent behavior that subspace methods excel at exploiting. However, we are only aware of subspace techniques that accelerate the collision detection stage of subspace contact simulation [33, 34]; the contact resolution stage is still computed in a full-rank manner. In Chapter 3 we present our *pose-space cubature* scheme that leverages the coherence of articulated self-contact to accelerate the *entire* contact computation in subspace simulation.

1.2 Arbitrary External Collisions

For interactive applications, the motion of the character is less likely to stay in a prescribed set. For example, external collisions might occur, which in contrast to the structure of articulated self-contact, are highly unpredictable. If the contact deformation was not accounted for during subspace construction (i.e. the full space motion lies outside the span of the subspace), subspace simulation can *lock*, producing motions that are both very different from the full space solution and unrealistic in appearance (Figure 1.1). Recent work [23] handles these situations by dynamically updating the basis with analytic, Boussinesq solutions to approximate the contact shape, but its performance deteriorates as the deformation region approaches the size of the deformable body. Instead we address this issue by proposing a generic approach we call *subspace condensation* [35]. It allows full space computation to be activated in the neighborhood of novel events while the rest of body still computes in a subspace. We present the details of subspace condensation in Chapter 4 and demonstrate its effectiveness in handling novel collisions.

Figure 1.1: Novel collisions cause extreme locking in a subspace-only simulation.

1.3 Solid-Fluid Coupling

The creation of immersive environments such as virtual flying [36], virtual swimming [37, 38] and surfing involves real-time simulation of a coupled solid-fluid system. The fluid dynamics need to take into account the effect of body movement, and the deformable body needs to react to the pressure from the fluid. Although efficient subspace simulation algorithms for fluid [KD13] and solid [KJ11] objects have been developed separately over the last few years, none of them can handle fully coupled effects. This is due to the fact that fluids and solids are typically simulated in different coordinate systems. Even if we use separate bases for each object, the sophisticated geometry operations required for boundary handling [25, 26, 27] will dominate the simulation. Most existing attempts for a unified solver have been Lagrangian [29, 30, 31, 32]. They either require constantly remeshing of the fluid geometry or use particle systems, which are difficult to formulate for subspace simulation. We take the opposite perspective and explore the coupling of fully Eulerian solids and fluids. This is made possible by the recent work of Eulerian solid simulation [5], which presents a method for simulating hyperelastic solids within

an Eulerian framework. In Chapter 5 we present our unified Eulerian framework for deformable solid-fluid coupling and show complex contact scenarios between multiple solids and a single-phase fluid.

1.4 Thesis Statement and Main Results

The thesis statement is as follows:

> *Non-linear phenomena that occur in the simulation of deformable objects such as articulated self-contact, external collisions and solid-fluid coupling can be computed efficiently using custom coordinate systems.*

In support of this thesis, I have developed three efficient algorithms for deformable object simulation using different coordinate systems described in Chapter 3 through 5. The first algorithm specifically targets articulated self-contact computation in a purely subspace simulation. My main results are:

- I show how to efficiently apply the subspace cubature approach to the problem of self-contact;
- I identify two general conditions that will cause the standard cubature approach to fail: sparse training matrices and excessive discontinuities (e.g. Heaviside functions);
- I propose *pose-space* cubature, which directly addresses these issues and enables efficient subspace self-contact.

My second approach deals with the more difficult subspace *locking* problem caused by novel external collisions. Localized full space computation is adaptively added to the

underlying subspace simulation. The technique itself is quite generic and can be applied to general cases where out-of-basis deformations occur. My main results are:

- *Subspace condensation*, a new method that combines the generality of full space deformations with the speed of subspace computations.
- The main bottleneck of condensation methods is a large matrix inverse. I design a solver that sidesteps this problem using subspace coordinates, but still maintains a two-way coupling between the full space and subspace regions.
- Condensation is usually only applicable to linear materials, but the speed of my method allows it to be applied to non-linear materials as well.
- I demonstrate my algorithm on a physics-based skinning application. To this end, I propose several *oracles* that detect the regions where full space computation is needed and where the subspace approximation will suffice, and dynamically partitions the mesh into these regions at every frame.
- By exploiting the forces along the boundary of the full space regions, I show that an efficient, cubature-based method [20, 39] can be obtained for evaluating the forces inside the subspace regions.

My third development achieves a two-way coupling between deformable solids and an incompressible fluid where the underlying geometric representation is entirely Eulerian. My main results are:

- A unified, Eulerian framework for simulating fully coupled fluids and deformable solids
- A semi-implicit solver for Eulerian Solids

- A method for satisfying incompressibility for both the solid and fluid regions of the simulation
- A collision resolution scheme for multibody frictional contact

1.5 Organization

The organization of this dissertation is as follows. Chapter 2 provides an overview of deformable body simulation and discusses related work on subspace simulation as well as recent development in Eulerian solid simulation. In Chapter 3 we present a purely subspace approach for efficient articulate self-contact handling. Chapter 4 introduces subspace condensation, an adaptive full space - subspace simulation scheme. We show the effectiveness of this approach by applying it to a variety of articulated character scenarios. In Chapter 5 we develop a new full space method that achieves a two-way coupling between deformable solids and an incompressible fluid where the underlying geometric representation is entirely Eulerian. Chapter 6 presents our conclusions and suggests several directions for future work.

Chapter 2

Background

Simulating deformable objects is a well-studied subject in computer graphics, and excellent articles exist that discuss developments up through the 1990s [6], the 2000s [7] and approaching the present day [40]. To make this theis relatively self-contained, we lay out a few fundamentals of 3D elastic solids modeling using Finite Element Method (FEM) and incompressible fluid simulation. Excellent tutorials are available on each subject [41, 40, 42, 43].

Notation: Throughout this dissertation, we will denote scalars using unbolded lower case, e.g. w, vectors using bold lower-case, e.g. $\mathbf{f}$, and matrices and tensors using bold uppercase, e.g. $\mathbf{K}$. Arbitrary non-linear function will be denoted in script, e.g. $\mathcal{F}(\mathbf{x})$. The reserved symbol N represents the full-order rank of a mesh, i.e. the number of unconstrained vertices in a tetrahedral mesh. Conversely, the symbol r denotes the subspace rank. The matrix $\mathbf{U} \in \mathbb{R}^{3N \times r}$ then represents the subspace basis that efficient operations are performed in. Subspace quantities are denoted with a tilde, such as $\widetilde{\mathbf{f}} = \mathbf{U}^\top \mathbf{f}$, which is in $\mathbb{R}^r$, and $\widetilde{\mathbf{K}} = \mathbf{U}^\top \mathbf{K} \mathbf{U}$, which is in $\mathbb{R}^{r \times r}$. They are the counterparts of full space quantities $\mathbf{f} \in \mathbb{R}^{3N}$ and $\mathbf{K} \in \mathbb{R}^{3N \times 3N}$ respectively. The one exception is the displacement vector $\mathbf{u}$, whose subspace version is denoted as $\mathbf{q} = \mathbf{U}^\top \mathbf{u}$.

2.1 Full Space Simulation of Hyperelastic Solids

In this section, we focus on traditional FEM simulation in a Lagrangian point of view. Recent development on Eulerian formulation is discussed later in Section 2.4.

2.1.1 Strain Energy and Hyperelasticity

Let Ω be the volumetric domain occupied by the solid object at its undeformed state. This domain is usually referred to as the *rest* or *reference* space. We begin by establishing a relationship $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ between the rest space and world space, which maps a material point $\overline{\mathbf{x}}$ to its deformed world-space location $\mathbf{x} = \phi(\overline{\mathbf{x}})$. The *deformation gradient* $\mathbf{F} \in \mathbb{R}^{3\times3}$ measures how much a infinitesimal line segment in the rest space is stretched and rotated in the world space and its full expression is:

$$\mathbf{F} = \begin{bmatrix} \frac{\partial\phi_1}{\partial\overline{\mathbf{x}}_1} & \frac{\partial\phi_1}{\partial\overline{\mathbf{x}}_2} & \frac{\partial\phi_1}{\partial\overline{\mathbf{x}}_3} \\ \frac{\partial\phi_2}{\partial\overline{\mathbf{x}}_1} & \frac{\partial\phi_2}{\partial\overline{\mathbf{x}}_2} & \frac{\partial\phi_2}{\partial\overline{\mathbf{x}}_3} \\ \frac{\partial\phi_3}{\partial\overline{\mathbf{x}}_1} & \frac{\partial\phi_3}{\partial\overline{\mathbf{x}}_2} & \frac{\partial\phi_3}{\partial\overline{\mathbf{x}}_3} \end{bmatrix}. \tag{2.1}$$

Hyperelastic materials refer to those that deform elastically and deformation process is dependent only on the rest state and the final, deformed configuration. They are typically used to model human and animal skin (and flesh) in computer graphics. The *strain energy*, i.e. , energy stored by the object undergoing deformation, can be written as:

$$E = \int_{\Omega} \Psi(\overline{\mathbf{x}}, \mathbf{F}(\overline{\mathbf{x}}))d\overline{\mathbf{x}}, \tag{2.2}$$

where Ψ is the energy density function. We use the notation $\mathbf{F}(\overline{\mathbf{x}})$ to express that the deformation gradient varies spatially over Ω (except when the body is undergoing pure rigid motion). One common type of material used in computer animation is *co-rotational*

elasticity [44, 11], for which we have:

$$\Psi = \mu||\mathbf{F} - \mathbf{R}||_F^2 + \frac{\lambda}{2}tr^2(\mathbf{R}^\top\mathbf{F} - \mathbf{I}), \tag{2.3}$$

where μ and λ are the Lamé coefficients, $||\cdot||_F$ is the Frobenius norm, $\mathbf{I}$ is the 3×3 identity matrix, and $\mathbf{R}$ is the rotation from the polar decomposition $\mathbf{F} = \mathbf{RS}$. We also use the co-rotational material for all the examples in this dissertation. However, our proposed techniques can be applied to other types of hyperelastic materials as well.

2.1.2 Quasistatic Simulation

Numerical simulation requires a discretizion of the domain Ω. We focus on the Finite Element Method (FEM) in this dissertation, but the concepts can also be applied to other discretization approaches such as the Finite Difference Method, Finite Volume Method and mass-spring systems. In particular, in Chapter 3 and 4, we use linear tetrahedral elements, while noting that our algorithms are independent of the choice of element type.

For quasistatic simulation with external forces $\mathbf{f}_{ext}$ (e.g., body forces), we ask that the resulting force vanishes at all N unconstrained nodal positions in the discrete setting. Using a first-order Taylor expansion at the rest state for the material force, we have:

$$\mathbf{f}(\mathbf{x}) + \mathbf{f}_{ext} = \mathbf{f}(\overline{\mathbf{x}} + \mathbf{u}) + \mathbf{f}_{ext} \tag{2.4}$$

$$\approx \mathbf{f}(\overline{\mathbf{x}}) - \mathbf{K}\mathbf{u} + \mathbf{f}_{ext} = \mathbf{0}, \tag{2.5}$$

In the above equation, $\mathbf{f} := -\frac{\partial E}{\partial \mathbf{x}}$ is the nodal material force vector, $\mathbf{K} := \left.\frac{\partial^2 E}{\partial \mathbf{x}^2}\right|_{\overline{\mathbf{x}}}$ is the $3N \times 3N$ Hessian matrix of the strain energy at the rest pose and $\mathbf{u} \in \mathbb{R}^{3N}$ is the displacement vector of the mesh vertices. $\mathbf{K}$ is also referred to as the *stiffness* matrix. Note that from now on, $\overline{\mathbf{x}}$ and $\mathbf{x}$ no longer represent individual material points and

positions, but the concatenated positions of the unconstrained N mesh vertices. For character animations, the positions of some vertices might be fixed or prescribed (e.g. , driven by the skeleton), therefore constrained, so these degrees of freedom are not involved in the simulation.

In the case of linear materials, the stiffness matrix $\mathbf{K}$ is constant and the approximation in Equation 2.5 becomes exact. Thus we can precompute and factorize $\mathbf{K}$ once and reuse it at run time to solve $\mathbf{Ku} = \mathbf{f}_{ext}$ for $\mathbf{u}$ (by definition $\mathbf{f}(\overline{\mathbf{x}}) = \mathbf{0}$). Unfortunately, linear elastic materials are only suitable for small deformations and otherwise introduce large distortions.

Any realistic material model must incorporate non-linearities to some certain degree. Thus, the stiffness matrix $\mathbf{K}$ is no longer constant and Equation 2.5 is replaced by a Newton-Raphson process:

$$\mathbf{K}(\mathbf{x}^k)\mathbf{u}^k = \mathbf{f}(\mathbf{x}^k) + \mathbf{f}_{ext}. \tag{2.6}$$

Here $\mathbf{K}(\mathbf{x}^k) := \left.\frac{\partial^2 E}{\partial \mathbf{x}^2}\right|_{\mathbf{x}^k}$ and $\mathbf{u}^k := \mathbf{x}^{k+1} - \mathbf{x}^k$. As the demand for high-resolution simulation grows, the size of the linearized system does as well. Although highly efficient iterative linear system solvers exist such as preconditioned conjugate gradient [45] and multigrid methods [46, 47], they cannot achieve real-time performance for large systems involving hundreds of thousands, or even millions, of degrees of freedom.

2.1.3 Dynamic Simulation

Here we layout the equations of motion (called the *Euler-Lagrange equations*) for dynamic simulation:

$$\mathbf{M\ddot{u}} + \mathbf{D\dot{u}} + \mathbf{Ku} = \mathbf{f}_{ext}. \tag{2.7}$$

In the above equation, $\mathbf{M} \in \mathbb{R}^{3N \times 3N}$ is the mass matrix. It is often simplified into a diagonal matrix by lumping the mass in each element onto its vertices. $\mathbf{D} \in \mathbb{R}^{3N \times 3N}$ is the damping matrix. One commonly used damping model is the Rayleigh damping $\mathbf{D} = \alpha\mathbf{M} + \beta\mathbf{K}$. The α and β are coefficients that control the amount of damping for lower frequency and higher structure modes respectively. The values we use are $\alpha = 0.01$ and $\beta = 0.001$.

A time integration scheme is needed to step the simulation forward. Explicit integration schemes (e.g., forward Euler) are fast, but less stable compared to implicit schemes, and demand small timesteps. Implicit schemes (e.g., backward Euler) remain stable even when using large timesteps, at the expense of larger computation cost per simulation frame. Implicit schemes also suffer from numerical damping. Mixed explicit-implicit schemes have also been proposed to optimize the trade-off between the two [48]. The integration scheme used in this dissertation is backward Euler. Note that this choice is irrelevant to our techniques.

2.2 Subspace Simulation

Subspace methods, also known as reduced-order, reduced coordinate, or model reduction methods, were first introduced to computer graphics by Pentland and Williams [15] in 1989 and have recently been used to accelerate a variety of simulations in computer graphics, such as deformable bodies [12], fluids [13] and clothing [14]. In this section, we describe the basic concept of subspace methods and discuss its strengths and limitations, followed by a summary of recent developments on this subject.

To give a concrete example, we will introduce the concept on the deformable solid simulation described in the previous section. Let us first assume that there is a r-dimensional subspace that captures the typical deformations of the solid object and the

matrix $\mathbf{U} \in \mathbb{R}^{3N \times r}$ is an orthonomal basis that spans this subspace. We apply a change of variable $\mathbf{q} = \mathbf{U}^T\mathbf{u}$ and left multiply the linear system of Equation 2.5 by $\mathbf{U}^\top$ and arrive at:

$$\widetilde{\mathbf{K}}\mathbf{q} = \mathbf{U}^\top \mathbf{f}_{ext}. \tag{2.8}$$

in which $\widetilde{\mathbf{K}} = \mathbf{U}^\top\mathbf{K}\mathbf{U}$ is the *reduced* version of $\mathbf{K}$ with dimension $r \times r$. The material properties naturally limit the shapes that a solid model can take on. Thus, it is reasonable to expect that a broad span of relevant deformations can still be efficiently captured when $r \ll N$. Now we only need to solve a small $r \times r$ dense linear system instead of a large $3N \times 3N$ sparse system, gaining large speedups [49].

The subspace basis $\mathbf{U}$ can be either obtained via a static analysis of the underlying deformable body mesh, such as linear modal analysis [15, 50] and modal derivatives [12], or a data driven process, such as principal component analysis (PCA) of existing full space simulation snapshots. Analytic approaches do not require any training overhead, but only give a low-order approximation of the true solution, and produce a rubbery look for the solid model. Therefore, we prefer PCA based approaches that capture the realistic appearance of the human flesh.

2.2.1 Previous Work on Incorporating Non-linearities

Equation 2.8 describes a subspace simulation of linear elasticity. The nonlinear version can be obtained from Equation 2.6 via the same projection:

$$\widetilde{\mathbf{K}}(\widetilde{\mathbf{x}}^k)\mathbf{q}^k = \widetilde{\mathbf{f}}(\widetilde{\mathbf{x}}^k) + \mathbf{U}^\top \mathbf{f}_{ext}, \tag{2.9}$$

in which $\widetilde{\mathbf{f}}(\widetilde{\mathbf{x}}^k) = \mathbf{U}^\top\mathbf{f}(\mathbf{x}^k)$ is the subspace material force. $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{f}}$ are no longer constant and need to be recomputed for every Newton iteration. Strictly following the definitions,

we can compute these quantities by first computing their full space counterparts and then project them into the subspace. We call this process *Krysl projection* [17]. It re-introduces $O(N)$ complexity into the subspace simulation and dials the potential speedup back to a single order of magnitude.

Barbič and James [12] developed a *projected tensor* approach to quickly compute the subspace stiffness matrix and material force for St. Venant-Kirchhoff (StVK) materials, whose energy density function is defined as:

$$\Psi = \mu \mathbf{E} : \mathbf{E} + \frac{\lambda}{2} tr^2(\mathbf{E}). \tag{2.10}$$

Here $\mathbf{E} := \frac{1}{2}(\mathbf{F}^\top \mathbf{F} - \mathbf{I})$ is the Green strain tensor.

From Equation 2.10 we can tell that the energy density function for StVK material is a fourth order multivariate polynomial function in the components of $\mathbf{x}$, thus the material force is a third-order multivariate polynomial function in $\mathbf{x}$. The analytical solutions for the the polynomial coefficients are derived in [12]. They can be precomputed and stored together as a third-order tensor. This tensor is projected once into subspace and reused at run time for fast evaluation of the reduced material force and stiffness matrix.

2.2.2 Cubature Approximation

Projected tensor approaches are used in subspace fluid simulation as well [13, 19]. However, a variety of non-linear materials cannot be written in this form. Alternatively, the *cubature* approach to subspace simulation [20] can be used. Instead of projecting a tensor, the cubature approach approximates the underlying material force function $\widetilde{\mathbf{f}}(\widetilde{\mathbf{x}})$

using a carefully weighted set $\mathfrak{C}$ of C *cubature points*:

$$\widetilde{\mathbf{f}}(\widetilde{\mathbf{x}}) \approx \sum_{i \in \mathfrak{C}} w_i \cdot \mathbf{U}_i^\top \mathbf{f}_i(\mathbf{U}_i \widetilde{\mathbf{x}}). \tag{2.11}$$

Here, w_i is a scalar cubature weight applied to sample i, $\mathbf{U}_i \in \mathbb{R}^{12 \times r}$ is the rows of $\mathbf{U}$ that correspond to the vertices of tet i, and $\mathbf{f}_i$ is a point-sampled version of the material force function that computes the material forces on the vertices of the ith tet. Similarly, the reduced stiffness matrix $\widetilde{\mathbf{K}}(\widetilde{\mathbf{x}})$ can be approximated as:

$$\widetilde{\mathbf{K}}(\widetilde{\mathbf{x}}) \approx \sum_{i \in \mathfrak{C}} w_i \cdot \mathbf{U}_i^\top \mathbf{K}_i(\mathbf{U}_i \widetilde{\mathbf{x}}) \mathbf{U}_i^\top, \tag{2.12}$$

with $\mathbf{K}_i$ being the 12×12 stiffness matrix of the ith tet.

Connections between Equation 2.11 and 2.12 and the compressive sensing literature have been observed [51], and this perspective provides some intuition. Similar to how the Fourier transform of a Dirac delta yields a function with global support, transforming a point sample $\mathbf{f}_i(\mathbf{U}_i \widetilde{\mathbf{x}})$ into the subspace $\mathbf{U}$ also yields a global function. The question is whether a global function $\widetilde{\mathbf{f}}(\widetilde{\mathbf{x}})$ can be approximated by projecting a small number of samples. The subspace modes of $\mathbf{U}$ were constructed by performing PCA, which succinctly parameterizes all previously seen examples of $\mathbf{f}(\mathbf{x})$, so it is reasonable to expect that within this subspace, some succinct version $\widetilde{\mathbf{f}}(\widetilde{\mathbf{x}})$ can be discovered.

The cubature approach has been successfully applied to several non-linear material models [20]. In particular, we have found that it can be applied to the co-rotational material model [McAdams et al. 2011] used in this dissertation. It has also been generalized to other non-linear problems, including evaluating the quadratic term in the inviscid Euler equations [21], as well as geometric reduction for material optimization [22]. We also use this approach extensively in our subspace simulations. One detail we have not

mentioned so far is how to choose the cubature points and their weights. In fact, several approaches have been proposed to select the cubature points in the last few years, which we summarize in the next section.

2.2.3 Cubature Training Schemes

We will start with the common features of the three algorithms covered in this section and then describe each one individually.

The cubature optimization is modeled as a discrete subset selection problem whose goal is to select cubature points that, when weights are optimized using non-negative least squares (NNLS), tends to minimize fitting error of a training set. Given T training snapshots, $\{\widetilde{\mathbf{f}}^t\}_{t=1\ldots T}$ where $\widetilde{\mathbf{f}}^t$ is computed using Krysl projection [17] (i.e., $\widetilde{\mathbf{f}}^t = \mathbf{U}^\top \mathbf{f}^t$), the fitting error is chosen as the root mean square (RMS) relative L2-norm error over all samples, as described by the following error metric:

$$\varepsilon = \sqrt{\frac{1}{T}\sum_{t=1}^{T} \frac{||\overline{\mathbf{f}}^t - \widetilde{\mathbf{f}}^t||^2}{||\widetilde{\mathbf{f}}^t||^2}}, \tag{2.13}$$

where $\overline{\mathbf{f}}^t$ is the cubature estimation of $\widetilde{\mathbf{f}}^t$. If we fix a set of n cubature points temporarily, minimizing the fitting error ε is equivalent to solving the weights $\mathbf{w}$ in:

$$\begin{bmatrix} \frac{\widetilde{\mathbf{f}}_1^1}{||\widetilde{\mathbf{f}}^1||} & \cdots & \frac{\widetilde{\mathbf{f}}_i^1}{||\widetilde{\mathbf{f}}^1||} & \cdots & \frac{\widetilde{\mathbf{f}}_n^1}{||\widetilde{\mathbf{f}}^1||} \\ \vdots & & \vdots & & \vdots \\ \frac{\widetilde{\mathbf{f}}_1^t}{||\widetilde{\mathbf{f}}^t||} & \cdots & \frac{\widetilde{\mathbf{f}}_i^t}{||\widetilde{\mathbf{f}}^t||} & \cdots & \frac{\widetilde{\mathbf{f}}_n^t}{||\widetilde{\mathbf{f}}^t||} \\ \vdots & & \vdots & & \vdots \\ \frac{\widetilde{\mathbf{f}}_1^T}{||\widetilde{\mathbf{f}}^T||} & \cdots & \frac{\widetilde{\mathbf{f}}_i^T}{||\widetilde{\mathbf{f}}^T||} & \cdots & \frac{\widetilde{\mathbf{f}}_n^T}{||\widetilde{\mathbf{f}}^T||} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_i \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} \frac{\widetilde{\mathbf{f}}^1}{||\widetilde{\mathbf{f}}^1||} \\ \vdots \\ \frac{\widetilde{\mathbf{f}}^t}{||\widetilde{\mathbf{f}}^t||} \\ \vdots \\ \frac{\widetilde{\mathbf{f}}^T}{||\widetilde{\mathbf{f}}^T||} \end{bmatrix} \leftrightarrow \mathbf{A}\mathbf{w} = \mathbf{b}, \tag{2.14}$$

subject to $\mathbf{w} \geq \mathbf{0}$. The non-negativity constraints are added to avoid over-fitting and

preserve the spectral properties of the stiffness matrices (Equation 2.12). Here, $\widetilde{\mathbf{f}}_i = \mathbf{U}_i^\top \mathbf{f}_i^T(\mathbf{U}_i\widetilde{\mathbf{x}})$ is the reduced material force on the ith sample point, $\mathbf{A}$ is a dense rT by n matrix, and $\mathbf{b}$ is an rT-vector. The relative residual error $||\mathbf{r}||/||\mathbf{b}||$ is equivalent to the fitting error in Equation 2.13. Highly efficient NNLS implementations exist for solving such problems. For example, the standard Lawson-Hanson [52] has a complexity of rTn^3. As observed by [21], the Bro and de Jong [53], also known as Fast Non-Negative Least Squares (FNNLS), yields a significant constant speedup over [52].

The entire optimization is a iterative process where during each iteration, a subset of cubature candidates are added to the existing cubature set $\mathfrak{C}$, and after the NNLS solve, only the ones with positive weights are kept. The entire process is terminated when the relative residual is below a certain threshold. The major difference between the three algorithms we describe next is how the candidates are selected in each iteration.

Greedy Estimation was proposed by An et al. [20] along with the overall cubature approach. During each iteration, they select candidate e such that $\widetilde{\mathbf{f}}_e$ is the most positively parallel to the current NNLS residual. Their algorithm is outlined in Algorithm 1. In case the total number of candidates is large, the candidate is picked from a random subset of the remaining candidates (Line 3-4 in Algorithm 1) to reduce the storage and computation cost. The total complexity of greedy estimation is $O(rTn^4)$ where n is the number of cubature points.

Importance Sampled Cubature was developed by Kim and Delaney [21] in which cubature is applied to estimate the fluid velocity after the advection step in fluid simulation. The authors observed that much of the work of the greedy algorithm is redundant, as the matrix $\mathbf{A}$ between two consecutive iterations only differ by one column. They add C samples each iteration instead of one, which are selected using an importance sampling approach. The importance probability distribution function for each candidate

Algorithm 1 GreedyCubature($\mathbf{A}, \mathbf{b}, TOL$)

1: $\mathfrak{C} \leftarrow \emptyset$
2: $\mathbf{r} \leftarrow \mathbf{b}$
3: **while** $||\mathbf{r}||/||\mathbf{b}|| > TOL$ **do**
4: $\quad \mathfrak{S} \leftarrow$ SelectCandidatePoints($\mathfrak{C}$)
5: $\quad e \leftarrow \arg\max_{e \in \mathfrak{S}} \frac{\widetilde{\mathbf{f}}_e \mathbf{r}}{||\widetilde{\mathbf{f}}_e|| ||\mathbf{r}||}$
6: $\quad \mathfrak{C} \leftarrow \mathfrak{C} \cup \{e\}$
7: $\quad \mathbf{w} \leftarrow NNLS(\mathbf{A}_{\mathfrak{C}}, \mathbf{b})$
8: $\quad \mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}_{\mathfrak{C}} \mathbf{w}$
9: **end whilereturn** $(\mathfrak{C}, \mathbf{w})$

e is defined as:

$$\text{PDF}(e) = R\Big(\frac{|\mathbf{a}_e \cdot \mathbf{r}|}{\mathbf{r} \cdot \mathbf{r}}\Big). \tag{2.15}$$

where R is the number of points that are not yet in the cubature set.

Line 4-6 of Algorithm 1 are replaced with a call to Algorithm 2.

Algorithm 2 ImportanceSampledCubature(C)

1: **while** C points have not been added to $\mathfrak{C}$ **do**
2: $\quad$ Randomly select a candidate e not in $\mathfrak{C}$
3: $\quad$ Add e to $\mathfrak{C}$ with probability PDF(e)
4: **end while**

The overall complexity of importance-sampled cubature is $O(rTn^3)$. Detailed analysis was given in [21].

Similar ideas were also explored in [54, 23].

Non-Negativity-Constrained Hard Thresholding Pursuit (NN-HTP) [39] is an extension of the Hard Thresholding Pursuit (HTP) [55], a popular algorithm for best subset selection problems in the field of compressed sensing. Let $g(\mathbf{w}) = T\varepsilon(\mathbf{w})^2$ be the objective function, where $\varepsilon(\mathbf{w})$ is the fitting error defined in Equation 2.13. The optimization problem is defined as:

$$\min_{\mathbf{w}} g(\mathbf{w}) \text{ subject to } \mathbf{w} \geq \mathbf{0}, |\text{supp}(\mathbf{w})| \leq s, \tag{2.16}$$

where supp($\mathbf{w}$) refers to those with positive values in $\mathbf{w}$ and $|\text{supp}(\mathbf{w})|$ denotes its cardinality. s denotes the upper bound on the size of the cubature and is given as a user input.

An iteration of the NN-HTP algorithm is given by:

$$\mathbf{w}^{(i+1)} = \mathcal{H}_s^+(\mathbf{w}^{(i)} - \mu^{(i)}\nabla g(\mathbf{w}^{(i)})) \tag{2.17}$$

where $\nabla g(\mathbf{w}^{(i)}) = 2\mathbf{A}^T(\mathbf{A}\mathbf{w}^{(i)} - b)$ and $\mathcal{H}_s^+$ is a combinatorial projection that sets all but the s largest positive entries of a vector to zero. The step length $\mu^{(i)}$ is chosen adaptively at every iteration to increase stability. Before we can perform this step, the support of $\mathbf{w}^{(i+1)}$ needs to be contained in the $2s$-element set

$$\mathfrak{S}^{(i)} = \text{supp}(\mathbf{w}^{(i)}) \cup \text{supp}(\mathcal{H}_s^+(-\nabla_{\mathfrak{J}\setminus\text{supp}(\mathbf{w}^{(i)})} g(\mathbf{w}^{(i)}))) \tag{2.18}$$

Here $\nabla_{\mathfrak{K}} g$ denotes the gradient of g with all entries not in the set $\mathfrak{K}$ set to zero and $\mathfrak{J} = \{1, \ldots, m\}$ with m being the total number of cubature candidates. Based on $\mathfrak{S}^{(i)}$, the step length is computed as follows:

$$\mu^{(i)} = \frac{||\nabla_{\mathfrak{S}^{(i)}} g(\mathbf{w}^{(i)})||^2}{||\mathbf{A}\nabla_{\mathfrak{S}^{(i)}} g(\mathbf{w}^{(i)})||^2} \tag{2.19}$$

The NN-HTP step is followed by a standard NNLS solve to determine the weights of the current candidates. The entire algorithm is outlined in Algorithm 3.

Note that there is a typo at Line 10 of Algorithm 1 in the original paper and here we fix it with Line 7 of Algorithm 3. The lazy evaluation at Line 2 refers to an approximation of the gradient $\nabla g(\mathbf{w})$. The time and storage complexity of evaluation $\nabla g(\mathbf{w})$ are both $O(rTm)$, which can be a huge cost. They instead randomly choose C of the remaining candidates and calculate only the corresponding columns of $\mathbf{A}$. In practice, they found

Algorithm 3 NN-HTP($\mathbf{A}, \mathbf{b}, s, TOL$)

Randomly initialize $\mathbf{w}^{(1)}$ with s positive entries and set the rest to 0
for $i = 1, 2, \ldots$ **do**
    (Lazy) evaluate $\nabla g(\mathbf{w}^{(i)})$
    if $||\nabla g(\mathbf{w}^{(i)})|| \leq TOL$ **then return** $\mathbf{w}^{(i)}$
    end if
    Determine $\mathfrak{S}^{(i)}$ via Equation 2.18
    Determine $\mu^{(i)}$ via Equation 2.19
    $\mathbf{w}^{(i+1)} \leftarrow \mathcal{H}_s^+(\mathbf{w}^{(i)} - \mu^{(i)} \nabla g(\mathbf{w}^{(i)}))$
    $\mathfrak{C} \leftarrow \text{supp}(\mathbf{w}^{(i+1)})$
    if $\mathfrak{C}^{(i+1)} = \mathfrak{C}^{(i)}$ **then return** $\mathbf{w}^{(i)}$;
    end if
    $\mathbf{w}^{(i+1)} \leftarrow NNLS(\mathbf{A}_{\mathfrak{C}^{(i+1)}}, \mathbf{b})$
end for

choosing $C \approx 5s$ gives the best results. We also observed similar behavior.

The time complexity of Algorithm 3 is $O(krTs^3)$ with k being the number of iterations. No theoretical estimation on the convergence rate was given, although comparison with greedy estimation on a few examples showed that NN-HTP converged to a smaller fitting error using the same number of cubature points.

Newton-Cotes rule [56] is used to accelerate cubature computation in Yang et al. [57]. Cubature points are sampled uniformly in the solid body while the weights are optimized inividually and in parallel for each basis vector. They also accelerate training dataset generation. By incrementally adding training samples that minimizing the expected error of the subspace simulation, they are able to avoid generating unnecessary training samples and reduce the precomputation cost.

2.3 Multi-Domain Subspace Deformations

Coupled multibody simulation and substructuring techniques have been well studied in the engineering literature and excellent survey exists [58]. The intuition for domain

decomposition is straightforward: break the original large, complex model into small, simple ones that are easier to solve, and impose hard or soft constraints on the domain boundaries to couple the interfaces together. For example, it is inituitive to divide the body according to bone structure in character animation. In the case of subspace simulation, instead of constructing a large, global basis that captures the entire articulation motion space, we can assign each sub-domain its own low-rank basis, greatly improving the efficiency of subspace methods.

Hard constraint methods utilize Lagrange multipliers to ensure that the domain-interface vertices align. Popular full space integration methods such as Finite Element Tearing and Interconnect (FETI) and its variants [59], and Balancing Domain Decomposition by Constraints (BDDC) [60] both fall into this category. Unfortunately, they cannot be applied directly to subspace simulation, because the actual DOFs required to satisfy the constraints may be close to or exceed the subspace DOFs, producing locking artifacts or even worse, singular Jacobians. Yang and colleagues [61] constructed boundary-aware bases that contain boundary deformation modes, which allow the interface constraints to be handled using Lagrange multipliers. However, the rank of their subspace grows linearly with the size of the boundary, limiting the potential speedup of their method.

Due to the above issues, interface coupling in multi-domain subspace deformation is usually handled using soft constraints such as spring forces [62, 63, 64]. Barbič and Zhao [63] partition the deformable models into tree-structured subdomains. The interfaces are small and/or can be treated as rigid bodies. Their model deformed passively according to external forces, and is very suitable for plant simulation [65]. Kim and James [64] apply multi-domain subspace simulation to articulated characters where the mesh is decomposed into bone-associated domains, and domain articulation is specified via character pose. In contrast to [63], the inter-domain interfaces in [64] can be both large

and have significant deformations. The reduced interface spring forces and Jacobians are quickly evaluated using a technique called the Fast Sandwich Transform (FST). We also use their domain decomposition approach and its implementation in the open-source software *Cubica*[66] when simulating articulated characters. Next, we give a brief description of the FST algorithm.

In Kim and James [64], the mesh is partitioned using a simple nearest-neighbor strategy to associate each each tet with the nearest bone. Each pair of duplicated vertices are connected by a linear spring. Note that the algorithm itself is agnostic to the partitioning method, so other methods could be used. The pose is configured by updating the rigid rotation $\mathbf{R}$ and translation $\mathbf{t}$ for each domain. The basis for each domain is computed by first transforming the training data to the bone's local frame, then perform PCA. For example, for vertex m belonging to domain i, its world-frame position is

$$\mathbf{x}_m^i = \mathbf{R}^i(\overline{\mathbf{x}}_m + \mathbf{U}_m^i\mathbf{q}^i) + \mathbf{t}^i = \mathbf{R}^i\underline{\mathbf{U}}_m^i\underline{\mathbf{q}}^i + \mathbf{t}^i. \tag{2.20}$$

Here we use superscripts for domain index and subscripts for vertex index. $\mathbf{U}_m^i \in \mathbb{R}^{3\times r_i}$ is the rows of $\mathbf{U}^i$ that correspond to vertex m. $\underline{\mathbf{q}}$ and $\underline{\mathbf{U}}$ are homogeneous coordinate notations $\underline{\mathbf{q}} = (\mathbf{q}^\top|1)^\top$ and $\underline{\mathbf{U}} = [\mathbf{U}|\overline{\mathbf{x}}]$.

The full space spring force for interface vertex k coupled between domain i and j is

$$\mathbf{f}_k^{ij} = -\kappa(\mathbf{x}_k^i - \mathbf{x}_k^j) \tag{2.21}$$

$$= -\kappa(\mathbf{R}^i\underline{\mathbf{U}}_k^i\underline{\mathbf{q}}^i + \mathbf{t}^i - \mathbf{R}^j\underline{\mathbf{U}}_k^j\underline{\mathbf{q}}^j - \mathbf{t}^j) \tag{2.22}$$

where κ is the spring constant. For simplicity we dropped the area weighted term a_k in

the original paper. The reduced force transformed back to the local frame of domain i is

$$\widetilde{\mathbf{f}}_k^{ij} = (\mathbf{U}_k^i)^\top (\mathbf{R}^i)^\top \mathbf{f}_k^{ij} \tag{2.23}$$

$$= -\kappa[(\mathbf{U}_k^i)^\top \underline{\mathbf{U}}_k^i \underline{\mathbf{q}}^i - (\mathbf{U}_k^i)^\top (\mathbf{R}^i)^\top \mathbf{R}^j \underline{\mathbf{U}}_k^j \underline{\mathbf{q}}^j + (\mathbf{U}_k^i)^\top (\mathbf{R}^i)^\top (\mathbf{t}^i - \mathbf{t}^j)]. \tag{2.24}$$

The reduced spring force on domain i summed over all n coupled vertices between domain i and j is

$$\widetilde{\mathbf{f}}^{ij} = -\kappa[(\mathbf{U}^{ij})^\top \underline{\mathbf{U}}^{ij} \underline{\mathbf{q}}^i - (\mathbf{U}^{ij})^T (diag_n(\mathbf{R}^i))^\top diag_n(\mathbf{R}^j) \underline{\mathbf{U}}^{ji} \underline{\mathbf{q}}^j + (\mathbf{U}^{ij})^\top \mathbf{I}_n (\mathbf{R}^i)^\top (\mathbf{t}^i - \mathbf{t}^j)]. \tag{2.25}$$

Here $\mathbf{U}^{ij}$ is a submatrix of rows in $\mathbf{U}^i$ corresponding to vertices along the i, j interface and $\mathbf{I}_n \in \mathbb{R}^{3n \times 3}$ is a column vector of n 3-by-3 identity matrices. $diag_n(\mathbf{R})$ is the 3n-by-3n block-diagonalized version of $\mathbf{R}$ ($\mathbf{R}$ repeating n times). Although $(\mathbf{U}^{ij})^\top \underline{\mathbf{U}}^{ij}$ and $(\mathbf{U}^{ij})^\top \mathbf{I}_n$ can be precomputed, $(diag_n(\mathbf{R}^i))^\top diag_n(\mathbf{R}^j)$ is a time-varying quantity (in red) *sandwiched* between the basis matrices (blue), which appears to precludes precomputation and becomes a computation bottleneck. Fortunately, we can still preprocess away most of the complexity by using tensor expressions.

The Fast Sandwiched Transform (FST) is based on the observation that terms like $\mathbf{U}^T(diag_n(\mathbf{R}))\mathbf{V}$, where $\mathbf{U}$ and $\mathbf{V}$ are constant matrices with dimensions $3n$-by-r_1 and $3n$-by-r_2 respectively, are linear in the only time-varying parameters: the 9 entries of $\mathbf{R}$. We can write $\mathbf{R}$ as

$$\mathbf{R} = \sum_{\mu,\nu=1}^{3} \mathbf{E}^{\mu\nu} R_{\mu\nu}, \tag{2.26}$$

where $R_{\mu\nu}$ are the elements of $\mathbf{R}$ and $\mathbf{E}^{\mu\nu}$ are 3×3 basis matrices:

$$\mathbf{E} = \begin{bmatrix} \delta_{\mu 1}\delta_{\nu 1} & \delta_{\mu 1}\delta_{\nu 2} & \delta_{\mu 1}\delta_{\nu 3} \\ \delta_{\mu 2}\delta_{\nu 1} & \delta_{\mu 2}\delta_{\nu 2} & \delta_{\mu 2}\delta_{\nu 3} \\ \delta_{\mu 3}\delta_{\nu 1} & \delta_{\mu 3}\delta_{\nu 2} & \delta_{\mu 3}\delta_{\nu 3} \end{bmatrix}. \quad \begin{matrix} \mu = 1 \ldots 3 \\ \nu = 1 \ldots 3 \end{matrix} \tag{2.27}$$

Thus

$$\begin{aligned} \mathbf{U}^T diag_n(\mathbf{R})\mathbf{V} &= \sum_{\mu,\nu=1}^{3} \mathbf{U}^T diag_n(\mathbf{E}^{\mu\nu})R_{\mu\nu}\mathbf{V} \\ &= \sum_{\mu,\nu=1}^{3} \Big(\mathbf{U}^T diag_n(\mathbf{E}^{\mu\nu})\mathbf{V}\Big)R_{\mu\nu} \\ &= \sum_{\mu,\nu=1}^{3} \mathbf{A}^{\mu\nu}R_{\mu\nu}, \end{aligned} \tag{2.28}$$

which is just a linear superposition of 9 r_1-by-r_2 precomputed matrices, $\mathbf{A}^{\mu\nu}$. This reduced the runtime cost of computing the sandwiched term in equation 2.23 from $O(r^2 n)$ to $O(r^2)$. Kim and James [64] observed that this process can also be viewed in terms of tensors, $\mathbf{A} \bigotimes \mathbf{r}$, where $\mathbf{A} \in \mathbb{R}^{r_1 \times r_2 \times 9}$, $\mathbf{r} \in \mathbb{R}^9$ and $\bigotimes$ denotes a mode-3 tensor product.

In section 4.4 we will see that this process can be further simplified by using *skinning correction* bases.

2.4 Eulerian Solid Simulation

Lagrangian methods are known to suffer from instability when the mesh elements (tetrahedral, hexahedral, etc) become near singular under extreme deformations. Remeshing schemes have been developed to remedy this problem [67, 68]. Unfortunately, they add extra computational cost and can be hard to implement. It is also difficult for

the modified mesh to return to the original reference configuration. Mesh-free particle-based methods such as the Material Point Method (MPM) [69, 70, 71, 72] eliminate the requirement for an explicit volumetric mesh completely, but they are rarely used for hyperelastic deformations. The reason is that the Taylor expansion of $\mathbf{F}$ in MPM leads to drift, therefore allowing artificial plasticity to creep into the simulation.

In order to avoid complicated meshing schemes, simulate elastic objects accurately, and robustly resolve complicated collisions, Levin et al. developed the Eulerian Solids methodology [5]. A brief overview is given here, while full details can be found in previous work [5, 73] and tutorial [42]. Eulerian solids discretize the physical space as a regular grid and store the material space coordinates $\overline{\mathbf{x}}$ as a field to be advected. Rather than storing the full material coordinates, we follow Fan et al. [73], and advect the displacement field $\mathbf{u}$, which improves the robustness of the method under large time steps.

The advected $\mathbf{u}$ is the key to the Eulerian solids approach, as it allows the direct computation of the deformation gradient $\mathbf{F}$ using

$$\mathbf{F} = \left(\mathbf{I} - \frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right)^{-1}. \tag{2.29}$$

With this $\mathbf{F}$, we can compute the forces inside a solid using any arbitrary constitutive model. The original work[5] uses FVM to integrate the material force inside each grid cell and FEM was used in the follow-up[73]. Crucially, only displacement fields with zero deformation can yield $\mathbf{F} = \mathbf{I}$. This guarantees that an elastic constitutive model will always generate forces that attempt to return to a zero deformation state, and ensure an accurate hyper-elastic simulation.

Plastic deformation can be also added if desired. Fan et al. [73] uses the popular multiplicative plasticity model [70, 67] which decomposes the total deformation gradient

into

$$\mathbf{F}_{total} = \mathbf{F}_e \mathbf{F}_p, \tag{2.30}$$

where $\mathbf{F}_e$ is the elastic deformation gradient and $\mathbf{F}_p$ is the plastic deformation gradient. Some part of $\mathbf{F}_e$ might transfer to $\mathbf{F}_p$ according to some yield criterion. The plastic deformation gradients are stored in a material space grid and initially set to identity. The full updating process is given by:

$$\mathbf{F}_e = \mathbf{U}\Sigma\mathbf{V}^T, \tag{2.31}$$

$$\Delta\mathbf{F}_p = \mathbf{V}^T \left(\frac{\Sigma}{(det\Sigma)^{1/3}} \right)^{\gamma} \mathbf{V}, \tag{2.32}$$

$$\mathbf{F}_p^{t+1} = \Delta\mathbf{F}_p \mathbf{F}_p^t. \tag{2.33}$$

Here $\gamma = \nu\left(\frac{||\sigma|| - ||\sigma_y||}{||\sigma||}\right)$, $0 \leq \gamma \leq 1$, σ is the Cauchy stress tensor and $||\sigma_y||$ is the plastic yield.

2.5 Eulerian Incompressible Fluid Simulation

The famous incompressible Navier-Stokes equations describe most fluid flow of interest in animation:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla\mathbf{v} + \frac{1}{\rho}\nabla p = \mathbf{b} + \nu\nabla \cdot \nabla\mathbf{v}, \tag{2.34}$$

$$\nabla \cdot \mathbf{v} = 0. \tag{2.35}$$

The first line is the momentum equation and the second line makes the fluid incompressible. Here $\mathbf{v}$ is the velocity of the fluid, ρ stands for the density, p stands for pressure, $\mathbf{b}$ accounts for accelerations caused by body forces such as gravity and ν is the kinematic

viscosity. Several spatial derivatives are involved: ∇ is the gradient operator, $\nabla\cdot$ is the divergence and $\nabla \cdot \nabla$ is the Laplacian. Due to the ease of deriving these spatial derivatives, as well as numerically approximating them, Eulerian approaches are popular for simulating fluids. Here, we briefly summarize the steps of a basic fluid solver and refer interested readers to Bridson's book [43].

A first-order splitting scheme is often used:

$$\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} = 0, \tag{2.36}$$

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{b}, \tag{2.37}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \quad \text{such that} \quad \nabla \cdot \mathbf{v} = 0. \tag{2.38}$$

Note that the viscosity term $\nu\nabla \cdot \nabla \mathbf{v}$ in Equation 2.34 is dropped here. Equation 2.37 adds the body forces and is often integrated explicitly. Equation 2.38 is usually resolved by a projection step – we compute a pressure field that can be used to project out the divergent part of the velocity, thus making the fluid incompressible. This step typically requires solving a Poisson system.

The advection step, Equation 2.36, requires more attention as it plays a key role in the stability of the simulation. The pioneering work of Stam [74] first made this step unconditionally stable in computer graphics. The proposed *semi-Lagrangian* advection scheme operates by tracing from each grid position back in time using the grid velocity, and updates it with the velocity at the traced back position. Since we do not necessarily land exactly on a grid position, interpolation is required. The linear interpolation used originally in Stam [74] causes severe numerical dissipation which leads to loss of details. Less dissipative schemes were developed along this thread [75] but high-frequency details can still be lost.

The Fluid-Implicit-Particle (FLIP) approach [76] was first applied in computer graph-

ics for animating sand [77]. The basic idea is to store fluid particles on a grid. Advection can be carried out on the particles without loss of information. Velocities (and other necessary fields) are transferred between the grid and the particles through interpolation schemes. FLIP owes its popularity to "the ability to represent large variations in data" even though it is limited to first-order accuracy and can also suffer from dissipation [43]. Variants [72] are actively being developed to remedy these issues.

Chapter 3

Articulated Subspace Self-Contact

Note: A large portion of this chapter has previously appeared as [78].

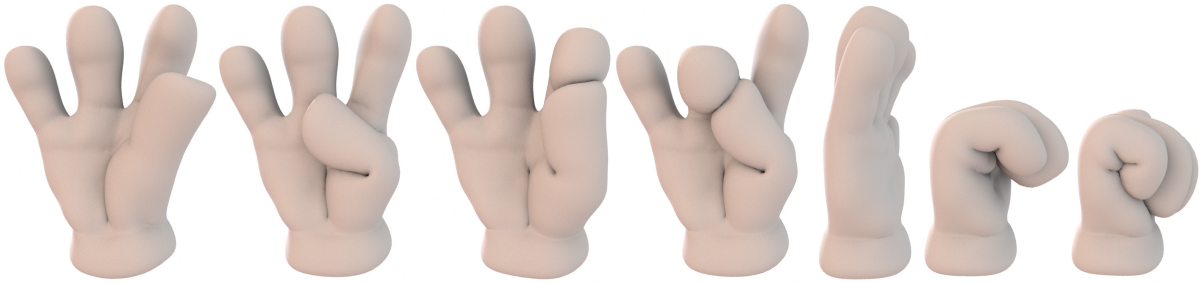


Figure 3.1: A hand mesh composed of 458K tetrahedra, running at 5.8 FPS (171 ms), including both self-contact detection and resolution. Our algorithm accelerates the computation of complex self-contacts by a factor of 5× to 52× over other subspace methods and 166× to 391× over full-rank simulations. Our self-contact computation never dominates the total time, and takes up at most 46% of a single frame.

In this chapter, we present an efficient new subspace method for simulating the self-contact of articulated deformable bodies, such as characters. Self-contact is highly structured in this setting, as the limited space of possible articulations produces a predictable set of coherent collisions. Subspace methods can leverage this coherence, and have been used in the past to accelerate the collision detection stage of contact simulation [33, 34].

We show that these methods can be used to accelerate the *entire* contact computation, and allow self-contact to be resolved *without looking at all of the contact points*. Our analysis of the problem yields a broader insight into the types of non-linearities that subspace methods can efficiently approximate, and leads us to design a *pose-space cubature* scheme. Our algorithm accelerates self-contact by up to an order of magnitude over other subspace simulations, and accelerates the overall simulation by two orders of magnitude over full-rank simulations. We demonstrate the simulation of high resolution (100K – 400K elements) meshes in self-contact at interactive rates (5.8 – 50 FPS). Figure 3.1 shows a few snapshots from a hand animation sequence simulated using our subspace method.

3.1 The Subspace Self-Collision Problem

The equilibrium state for a deformable object undergoing self-contact and external loadings is given as follows:

$$\mathbf{f}(\mathbf{x}) = -\mathbf{f}_s(\mathbf{x}) - \mathbf{f}_{ext}. \tag{3.1}$$

The meaning of $\mathbf{f}(\mathbf{x})$ and $\mathbf{f}_{ext}$ remain the same as the quasistatic formulation of Equation 2.5. $\mathbf{f}_s(\mathbf{x}) \in \mathbb{R}^{3N}$ denotes forces that arise from self-contact, and is the primary quantity that we are connected with in this chapter. We will introduce dynamics later (§3.3.2), but all of the simulation difficulties are already present in this formulation. The linearized system for Equation 3.1 for one Newton iteration is:

$$(\mathbf{K}(\mathbf{x}^k) + \mathbf{K}_s(\mathbf{x}^k))\mathbf{u}^k = \mathbf{f}(\mathbf{x}^k) + \mathbf{f}_s(\mathbf{x}^k) + \mathbf{f}_{ext}, \tag{3.2}$$

where $\mathbf{K}_s := -\frac{\partial \mathbf{f}_s}{\partial \mathbf{x}}$ is the negative self-contact force Jacobian. The reduced version can be obtained using the same subspace projection described in Section 2.2:

$$(\widetilde{\mathbf{K}} + \widetilde{\mathbf{K}}_s)\mathbf{q} = \widetilde{\mathbf{f}} + \widetilde{\mathbf{f}}_s + \mathbf{U}^\top \mathbf{f}_{ext}. \tag{3.3}$$

Here we drop the superscript k and dependence on $\mathbf{x}$ for brevity. $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{f}}$ can be computed efficiently using cubature approximation (Equation 2.11 and 2.12. The external forces can be efficiently computed using methods from previous works [79, 80], so the question we are interested in is: Can we compute $\widetilde{\mathbf{f}}_s$ (and subsequently $\widetilde{\mathbf{K}}_s$) efficiently?

Krysl projection (i.e. $\mathbf{U}^\top \mathbf{f}_s$ and $\mathbf{U}^\top \mathbf{K}_s \mathbf{U}$) could be applied, but this would discard much of the knowledge we have of $\mathbf{f}_s$. The variety of possible self-contact forces is largely constrained by the articulation limits on the skeleton, and while extreme self-contact configurations are possible (e.g. the character being crushed in a trash compactor) the vast majority of forces will repetitively occur at a predictable set of surface vertices. We will use this knowledge to design an efficient cubature scheme for $\mathbf{U}^\top \mathbf{f}_s$ in Section 3.2.

Penalty Force Formulation: Like many recent works [81, 11, 82], we use a penalty force formulation to resolve contacts. For each penetrating surface triangle and vertex, $(\mathbf{t}_p, \mathbf{v}_p)$, we find the surface point $\mathbf{v}_s$ on the triangle $\mathbf{t}_p$ that is closest to $\mathbf{v}_p$ and apply a force to $\mathbf{v}_p$,

$$\mathbf{f}_p = k_c \cdot k_a \cdot \mathbf{N}(\mathbf{v}_s - \mathbf{v}_p), \tag{3.4}$$

where k_c is a constant spring stiffness and k_a is the average of the area of $\mathbf{t}_p$ and the area of the surface triangles in the one-ring surrounding $\mathbf{v}_p$. The $\mathbf{N} \in \mathbb{R}^{3\times 3}$ anisotropy term is from McAdams et al. [11], and defined as

$$\mathbf{N} = (1-\alpha)\mathbf{n} \cdot \mathbf{n}^\top + \alpha \mathbf{I}, \tag{3.5}$$

where $\alpha \in [0,1]$, $\mathbf{I} \in \mathbb{R}^{3\times3}$ is an identity matrix, and $\mathbf{n}$ is the outward unit contact normal, $\mathbf{n} = \frac{(\mathbf{v}_s - \mathbf{v}_p)}{|\mathbf{v}_s - \mathbf{v}_p|}$. The force at each ith vertex of the triangle $\mathbf{t}_p$ is:

$$\mathbf{f}_i = -\beta_i \cdot k_c \cdot k_a \cdot \mathbf{N}(\mathbf{v}_s - \mathbf{v}_p), \tag{3.6}$$

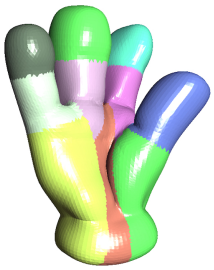
where β_i is the barycentric coordinate associated with triangle vertex i and $\mathbf{v}_s$. Section 3.3.1 contains further details on collisions.

Domain Decomposition: We use the domain decomposition technique from [64] described in the previous chapter for our articulated subspace simulations, but our algorithm is agnostic to the partitioning method, so other methods could be used. Each subdomain has its own subspace basis and the interfaces are coupled using penalty-based forces.

3.2 A Self-Collision Cubature Scheme

Self-contacts on articulated meshes occur mainly due to the motion of the underlying skeleton. We still start by directly applying the cubature approach to these self-collisions. The algorithm will produce unacceptable results, but motivate a more sophisticated scheme inspired by Pose Space Deformation [83].

3.2.1 A Direct Cubature Scheme



For expository purposes, we will focus on a single domain, x, and examine the collisions that arise between itself and another domain, y. We emphasize that these include *intra*-domain collisions between the surface of x and itself that arise due to the motion of y, not just the collisions between x and y. An example of our final algorithm

handling such collisions is inset on the left. The hand contains 10 domains, visualized by different colors. Note that the red domain in the middle of the palm is colliding with itself. We denote the full-coordinate ranks of x and y as N_x and N_y, and their reduced ranks as r_x and r_y. Each domain will have its own basis, $\mathbf{U}_x$ and $\mathbf{U}_y$.

The $\mathbf{f}_s$ self-collision term for domain x is intrinsically a force. Since the cubature approach was successfully applied to the other force terms such as the $\mathbf{f}(\mathbf{x})$ material force term, it is reasonable to expect that it will be able to generalize to this term as well. Within this context, a single cubature point p is represented by the self-collision force, $\mathbf{f}_s(\mathbf{t}_p, \mathbf{v}_p) \in \mathbb{R}^{N_x+N_y}$, with only 12 non-zero entries arising from the collision of a single vertex, $\mathbf{v}_p \in \mathbb{R}^3$, with a triangle, $\mathbf{t}_p \in \mathbb{R}^9$. Note that in this context, a *cubature point* no longer corresponds to a literal spatial point or tetrahedron, but rather a geometric pair. With this definition established, we can compute a self-collision cubature scheme using the same training algorithm from An et al. [20].

At a high level, the training problem we are trying to solve is as follows. We have a set of self-collision forces that were generated from a large number of contact pairs, and would like to approximate these forces by instead using a weighted subset of these pairs. The primary question of interest is: how many pairs are needed to achieve a desired accuracy?

We will limit ourselves to training details that will be relevant when later examining why this direct scheme fails. We have a set of T training snapshots (i.e. example deformations), which we obtain by running a series of full-rank simulations. Each t-th snapshot possesses a self-contact force vector, $\mathbf{f}_s^t \in \mathbb{R}^{N_x+N_y}$. A reduced version of this vector can be obtained via projection, $\widetilde{\mathbf{f}}_s^t = \mathbf{S}^\top \mathbf{f}_s^t$, where $\mathbf{S} \in \mathbb{R}^{(N_x+N_y)\times(r_x+r_y)}$ and is constructed as the block matrix:

$$\mathbf{S} = \begin{bmatrix} \mathbf{U}_x & 0 \\ 0 & \mathbf{U}_y \end{bmatrix}. \tag{3.7}$$

The goal of the training stage is to locate a small cubature set $\mathfrak{C}$ consisting of C collision pairs that, when evaluated,

$$\widetilde{\mathbf{f}}_s^t \approx \sum_{p=1}^{C} w_p \cdot \mathbf{S}_p^\top \mathbf{f}_s(\mathbf{t}_p^t, \mathbf{v}_p^t), \tag{3.8}$$

approximate all T self-collision forces to some desired degree of accuracy. Here, $(\mathbf{t}_p^t, \mathbf{v}_p^t)$ denotes a collision pair deformed according to the displacement vector from the corresponding snapshot t, and $\mathbf{S}_p$ denotes a concatenation of the rows in $\mathbf{S}$ that correspond to $\mathbf{v}_p$ and the vertices in $\mathbf{t}_p$.

As with material forces (Equation 2.11), there is reason to believe that a sparse approximation exists. For example, imagine that only a single self-collision example had been observed, and its $\mathbf{f}_s$ was added as the final column in $\mathbf{U}_x$. A single cubature point would suffice, provided its projection produced a vector composed entirely of zeros, except for a final non-zero component. The w_p would account for any scalar discrepancy, and the global force would be effectively represented by a single point.

Promising candidates for $\mathfrak{C}$ can be located in a variety of ways described in Chapter 2, including a greedy search [20], importance sampling [21, 80], or HTP [51]. We used importance sampling in this work. Once candidates are located, a Non-Negative Least Squares (NNLS) solve assigns each candidate a weight (w_p in Equation 3.8). First, the projected self-collision force produced by each p-th candidate at each t-th snapshot is computed,

$$\begin{bmatrix} \mathbf{S}_p^\top \mathbf{f}_s(\mathbf{t}_p^1, \mathbf{v}_p^1) \\ \mathbf{S}_p^\top \mathbf{f}_s(\mathbf{t}_p^2, \mathbf{v}_p^2) \\ \cdots \\ \mathbf{S}_p^\top \mathbf{f}_s(\mathbf{t}_p^T, \mathbf{v}_p^T) \end{bmatrix} = \mathbf{g}_p. \tag{3.9}$$

These columns are concatenated to form the $\mathbf{Aw} = \mathbf{b}$ system

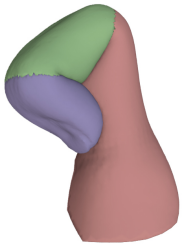
$$\begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \dots & \mathbf{g}_C \end{bmatrix} \mathbf{w} = \begin{bmatrix} \left(\widetilde{\mathbf{f}}_s^1\right)^\top & \left(\widetilde{\mathbf{f}}_s^2\right)^\top & \dots & \left(\widetilde{\mathbf{f}}_s^T\right)^\top \end{bmatrix}^\top, \tag{3.10}$$

where $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_C \end{bmatrix}^\top$ is the vector of weights that is solved for. This system can then be sent to any standard NNLS solver [84].

3.2.2 Analysis of Negative Results

We found that the cubature sets discovered by this direct cubature method are not immediately useful. In order to converge to an acceptable level of relative error of 1% to 5%, between 30% and 40% of all of the collision pairs in the training snapshots had to be added to the cubature set. These sets clearly did not have $O(r)$ cardinality, and did not significantly accelerate self-contact computation. Alternate candidate selection approaches such as HTP [51] may be able to decrease the cardinality by a small constant, but the asymptotic performance is unlikely to change.

Infrequent events in the training set were captured poorly.



In the figure on the right for example, the top pad of the finger (left image, in blue) comes into contact with the bottom third of the finger (left image, in red) only in the final frames of our training set. Since these forces do not appear in many snapshots, the NNLS solver treats them as unimportant, and the final cubature set approximates them poorly. While it might be possible to re-weight these examples, such an approach would essentially second-guess the PCA and NNLS solvers, which seems unappealing. Clearly, this naïve application of the cubature approach is not sufficient.

The intuition from §3.2.1 breaks down because it applied to a single self-collision, $\widetilde{\mathbf{f}}_s^t$,

not the set of all self-collisions. While a single approximation may be sparse, the set of all approximations may be spatially disjoint, since the contact regions can be disjoint for two radically different poses. The union of the sparse approximations can then be dense if sparsity structures do not share any commonality. We can quantify this intuition as follows.

Sparsity of the training matrix A: The training column $\mathbf{g}_p$ for any collision pair is likely to contain mostly zeros, since the pair is likely in collision for only a small number of snapshots. Therefore, many of the $O(T)$ rows in the $\mathbf{A}$ matrix (Equation 3.10) will also consist solely of zeros. Unless a collision pair is added to the cubature set that specifically introduces non-zero entries into these rows, their relative errors will be 100%. This explains why the error only became acceptable for cubature sets of size $O(N)$, or more precisely $O(P)$, where P is the total number of collision pairs in all of the training snapshots; it is threshold where the number of non-zero rows in $\mathbf{A}$ had been sufficiently increased. We can conclude from this that *a sparse training matrix will produce a dense cubature set.*

Difficulty of learning the non-linear function: Equation 3.4, the non-linear force being learned, appears to be essentially a cubic function. However, the function actually contains an additional non-linearity. We can write a more general force term,

$$\mathbf{f}_p(\mathbf{t}_p, \mathbf{v}_p) = H(\mathbf{t}_p, \mathbf{v}_p) \cdot k_c \cdot k_a \cdot \mathbf{N}(\mathbf{v}_s - \mathbf{v}_p), \tag{3.11}$$

where $H(\mathbf{t}_p, \mathbf{v}_p)$ is a Heaviside function defined as:

$$H(\mathbf{t}_p, \mathbf{v}_p) = \begin{cases} 1 & \text{if } \mathbf{t}_p \text{ and } \mathbf{v}_p \text{ collide} \\ 0 & \text{otherwise} \end{cases}. \tag{3.12}$$

This new term significantly increases the non-linearity, as its dependence on $\mathbf{x}$ indirectly

introduces the complexity of the material model from Equation 3.1. The binary nature of the term also shows that while the force is C^0 continuous, it contains a C^1 discontinuity. We will need to address the complexity introduced by this term.

3.2.3 A Pose-Space Cubature Scheme

The preceding analysis led us to design a method that leverages the same intuition that underlies Pose Space Deformation (PSD) [83]. Instead of trying to capture a difficult non-linearity in its entirety, we interpolate over a sparse set of solutions where the non-linearity is known to be efficiently resolvable. In the case of PSD, a set of artist-sculpted examples are interpolated. In our case, we interpolate over a set of *per-pose* cubature schemes.

Per-Pose Cubature: Instead of computing a *single* cubature scheme that fits a large, sparse training matrix, we compute *multiple* cubature schemes that each fit small, dense matrices. At a high level, the primary difference between this training regimen and the previous one is that we are no longer trying to find a single cubature scheme that can approximate every self-collision force in the training set. Instead, we compute one cubature scheme for every example pose in the training set. If a pose is encountered at runtime that was not in the original training set, we can use a PSD-like process to interpolate between the nearest schemes.

The simplest training matrix that we can construct that is guaranteed to be dense is the one for a single training snapshot. For example, if we define the projected force for the 1^{st} snapshot for the collision pair p as $\mathbf{S}_p^\top \mathbf{f}_s(\mathbf{t}_p^1, \mathbf{v}_p^1) = \mathbf{g}_p^1$, we can write the training matrix:

$$\begin{bmatrix} \mathbf{g}_1^1 & \mathbf{g}_2^1 & \dots & \mathbf{g}_C^1 \end{bmatrix} \mathbf{w}^1 = \widetilde{\mathbf{f}}_s^1. \tag{3.13}$$

We abbreviate this system to $\mathbf{A}^1\mathbf{w}^1 = \widetilde{\mathbf{f}}_s^1$. Note that the right hand side now contains a

single self-contact force sample, and that the training matrix dimensions are $\mathbf{A}^1 \in \mathbb{R}^{r \times C}$ instead of $\mathbf{A} \in \mathbb{R}^{rT \times C}$, essentially making $\mathbf{A}^1$ a subset of rows from $\mathbf{A}$. This training matrix will automatically be dense. All of the cubature selection strategies (greedy, importance sampled, or HTP) project potential candidates onto the residual vector $\mathbf{r} = \mathbf{A}^1\mathbf{w}^1 - \widetilde{\mathbf{f}}_s^1$. A collision pair that produced a zero-valued training column would also produce a zero-valued projection, and would therefore never be considered promising enough to be added to the cubature set. As predicted, we found that the training stage now produced cubature sets with $O(r)$ cardinality for each input snapshot (Figure 3.2).

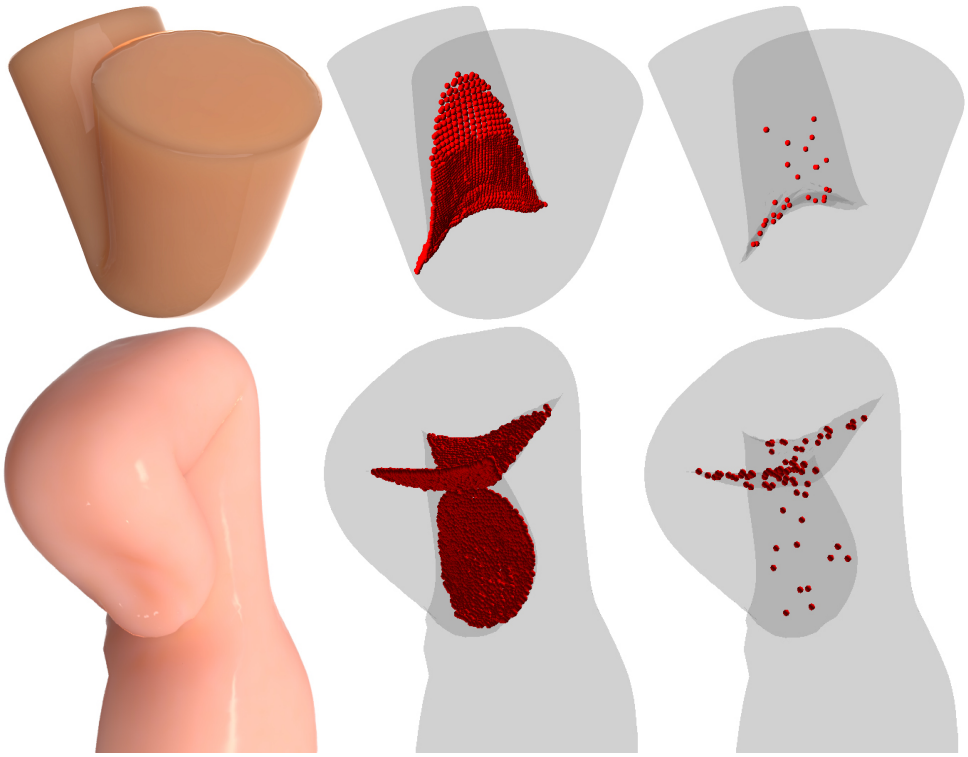


Figure 3.2: Left: Final frames from Figs. 3.3 and 3.4. **Middle:** Dense, full-rank contact points. **Right:** Our cubature scheme only needs to detect and resolve a sparse subset of these points.

Cubature Interpolation: We now have T cubature sets to interpolate at runtime. While the direct cubature scheme contained a difficult-to-learn Heaviside function, we

now have the opposite problem that this non-linearity has been aggressively pre-processed away. At runtime, each cubature set assumes that the Heaviside functions have already been resolved, and all that remains is for the contact forces (Eqns. 3.4 and 3.6) to be computed. This is not an issue if the character only takes on poses that are exactly the same as the training snapshots, but much of the appeal of subspace methods is that they can generalize to motions that are similar to, but distinct from, those in the training set. A method is needed for interpolating between cubature sets when a pose is encountered that was not seen during training. During this interpolation, $H(\mathbf{t}_p, \mathbf{v}_p)$ is evaluated at each cubature point, as it is unknown whether the collision occurs at this intermediate pose.

We perform the interpolation as follows. At runtime, we locate the closest cubature set $\mathfrak{C}_t$ of all T cubature sets, by computing the distance between some feature vector of the current pose, $\mathbf{q}_c \in \mathbb{R}^6$, and the corresponding $\mathbf{q}_i \in \mathbb{R}^6$ for each of the cubature sets:

$$t = \underset{i \in T}{\arg\min} \left(\text{dist}(\mathbf{q}_c, \mathbf{q}_i) \right). \tag{3.14}$$

where *dist* denotes a distance function. We have some freedom in designing both *dist* and the pose vectors, $\mathbf{q}$. If domains x and y are connected by a joint, quaternions could be used ($\mathbf{q} \in \mathbb{R}^4$), and a natural distance measure would be $1 - \|\mathbf{q}_c \cdot \mathbf{q}_i\|$. However, if x and y are not adjacent, a translation needs to be added to $\mathbf{q}$. This complicates the distance measure, because while quaternions are normalized, translations are not, and a large value can spuriously dominate the distance measurement. Separate weightings could be added to rotation and translation, but such a scheme would be ad hoc.

We use a PCA method [85] to address this issue. In an offline stage, for each training snapshot, we transform the surface positions of domain y into x's local coordinates system. We perform PCA on these positions and extract the six most significant com-

ponents, $\mathbf{U}_r$. For an unconstrained mesh, these components will have principal values of zero, as they correspond to the zero-energy, infinitesimal rigid transformation modes of the mesh (see e.g. [86], Appendix D in [12], or §3.1 in [87]). The subspace coordinates $\mathbf{q}_i$ for each training snapshot i are computed and stored along with $\mathbf{U}_r$. The distance between two poses c and i can now be defined as $\text{dist}(\mathbf{q}_c, \mathbf{q}_i) = ||\mathbf{q}_c - \mathbf{q}_i||_2$. At run time, we again transform the surface positions of y into the local coordinates of x, and project it using $\mathbf{U}_r$ to obtain $\mathbf{q}_c$.

We find the two nearest neighbors from Equation 3.14 by using a brute force search. The subspace forces from the two schemes are then interpolated based on their relative distances. Because the number of snapshots T is small in all of our examples (see Table 3.1), the brute force search takes a negligible amount of time. If T were larger and created a bottleneck, more efficient methods [88] could be used.

Discussion: This approach was inspired by Pose Space Deformation, but the resulting algorithm is fairly different. We use a linear blend instead of radial basis functions, as we found they were more appropriate for our non-linearities. Each cubature scheme is localized to a pair of domains, which is reminiscent of Weighted PSD [89], and suggests that a non-partitioned algorithm that can be applied to monolithic subspace methods [90, 91] should also possible. The final outputs of our algorithm are generic subspace forces and stiffnesses, which do not fundamentally depend on the existence of a spatial decomposition. We leave a more in-depth investigation into these issues and their relationship with PSD as future work.

3.3 Implementation and Results

3.3.1 Implementation

Solvers: All of our simulations used implicit Newmark as the time discretization. We used the interior point optimizer IPOPT [92, 93] as our non-linear solver, as we found its wall-clock time to be faster than Newton-Raphson. In the inner loops of the full-rank and reduced-rank solves, we respectively used the highly tuned HSL solvers `MA86` [94] and `MA57` [95]. We found that these were faster than conjugate gradient, and outperformed even an improved Incomplete Cholesky preconditioner [96].

Hardware: All simulations were run on a 12-core, 2.66 Ghz Mac Pro with 96 GB of RAM. Our own code and the HSL solvers both leveraged OpenMP for parallelization. All of the full-rank simulations ran on 12 threads. For each subspace example, we manually optimized the number of threads to fit the number of domains.

Basis Construction: To obtain meaningful forces, we explicitly enriched the subspace basis $\mathbf{U}$ with the self-contact forces. We computed the self contact forces for a variety of training poses, and perform a PCA on the resulting $\mathbf{f}_s$ vectors. We then concatenate the results to $\mathbf{U}$ and re-orthogonalize using modified Gram-Schmidt. We found it necessary to add the self-contact force vectors to the *front* of $\mathbf{U}$. Otherwise, the locality of the forces would become entangled with the global support from other columns in $\mathbf{U}$.

Collision Detection: During full-rank simulation, we used DeformCD [97] for collision detection. This algorithm uses a bounding volume hierarchy, specifically axis-aligned bounding boxes (AABBs), for the broad phase, and the Representative Triangles during the narrow phase.

For subspace collision detection, we also used AABBs for the broad phase, and then brute-force point-tet intersection tests for the narrow phase. We built two bounding volume hierarchies (BVH) for each domain, one for the cubature vertices and one for

the tetrahedra. At each timestep, the cubature vertices from one domain x was tested against *all* of the tetrahedra from another domain y (for intra-domain collisions, $x = y$). We could have only performed collisions against the triangles $\mathbf{t}_p$ from the training set, but we found this to be too conservative for high-resolution meshes, as the cubature vertex $\mathbf{v}_p$ could easily penetrate a nearby triangle instead. The cubature weights were still effective, since nearby triangles generate similar penalty forces. We also attempted to use BD-Trees [33], but found our BVH was faster. The deformations required many leaf-level updates, and the $\mathbf{U}\tilde{\mathbf{x}}$ multiply at each BD-Tree node update quickly became a bottleneck. As we were simulating high resolution meshes, we assumed that edge-edge and vertex-edge contacts are negligible, but our algorithm could be extended to handle these case as well.

As anticipated, the cardinality of the cubature sets was much smaller than the complete set of contact points, and reduced the number of necessary collision tests considerably (Table 3.2). The cubature schemes sufficiently accelerated the simulation that collision detection dominated the running time ($\sim$90%). We experimented with diminishing this bottleneck by using a low-resolution proxy of the tetrahedral mesh for the broad phase of subspace collision detection. This accelerated the phase significantly, and as the high-resolution mesh is queried during the narrow phase, the impact on the final results was minimal. To facilitate comparisons, all of the timings given in Table 3.2 and Figs. 3.6 and 3.7 use this proxy geometry. We did not use the low-resolution proxy during the full-rank precomputation, as collision detection was not the main bottleneck, and instead took between 16-27% of the running time. The increase in computational cost instead appeared in the form of more complex non-linear solves.

Collision Resolution: We called the same collision resolution code for both the reduced- and full-rank simulations. For each penetrating vertex $\mathbf{v}_p$ and penetrated tet $\mathbf{h}$, we used barycentric interpolation to transform $\mathbf{v}_p$ to its position inside of the rest-pose

version of $\mathbf{h}$, $\overline{\mathbf{v}}_{\mathbf{h}}$. We obtained the surface point $\overline{\mathbf{v}}_s$ and surface triangle $\mathbf{t}_p$ closest to $\overline{\mathbf{v}}_{\mathbf{h}}$ using the signed distance field of the rest pose [98, 99]. Barycentric interpolation was used to obtain the deformed surface position, $\mathbf{v}_s$. The resulting $\mathbf{v}_s$ and $\mathbf{v}_p$ were then sent to Eqns. 3.4 and 3.6.

Material Model: In all of our examples, we used the co-rotational material, along with indefiniteness correction, from McAdams et al. [11]. Our simulations were performed over tetrahedral meshes instead of hexahedral meshes, so we did not find it necessary to perform their stabilization step, as the deformation nullspace of a tetrahedron is much smaller than that of hexahedron (3 instead of 15). The cubature approach successfully approximated the co-rotational material model with minimal intervention.

Example	Full-Rank Time Per Frame	Reduced-Rank Time Per Frame	Speedup
Cylinder	7.82 s	20 ms	391×
Finger	15.90 s	53 ms	300×
Arm	20.86 s	77 ms	271×
Hand	28.45 s	171 ms	166×

Table 3.1: Algorithm performance compared to full-rank solves.

3.3.2 Results

In our examples, we sparsely sampled the articulation by computing T samples in joint space (Table 3.1). For each pose, we ran a full-rank quasistatic simulation with self-contact detection and resolution, and a pose-space cubature scheme was computed to within 5% training error. In all cases, the subspace simulations ran *two orders of magnitude* faster than full-rank solves (Table 3.1).

Cylinder: We deformed an articulated cylinder containing 118,389 tetrahedra into a highly colliding state in Figure 3.3. The mesh contained 2 domains, and was trained on

Example	Using Reduced Self-Contact?	Mean SC Time Per Frame	Max SC Time Per Frame	Max # of Collisions	Collision Complexity Reduction	Mean Speedup	Max Speedup
Cylinder	No	114 ms	415 ms	4127			
Cylinder	**Yes**	7 ms	8 ms	15	275×	16×	52×
Finger	No	168 ms	517 ms	4930			
Finger	**Yes**	17 ms	26 ms	81	61×	10×	20×
Arm	No	182 ms	407 ms	1264			
Arm	**Yes**	26 ms	37 ms	74	17×	7×	11×
Hand	No	59 ms	782 ms	3025			
Hand	**Yes**	32 ms	155 ms	445	7×	2×	5×

Table 3.2: Performance of our self-contact (SC) algorithm compared to a subspace simulation that does not use self-contact cubature.

$T = 12$ examples for a final rank of $r = 20$. Without cubature, the subspace simulation spent *up to 96%* of its frame computation time detecting and resolving collisions. With it, self-contact computation *never exceeded 46%* (Figure 3.6). Self-contact in the most highly colliding frame was *accelerated by a factor of 52*×.

Finger: We simulated the complex self-collision of a finger containing 248,869 tetrahedra in Figure 3.4. The mesh contained 3 domains, and trained on $T = 17$ examples for a final rank of $r = 60$. This example shows our pose-space cubature scheme capturing infrequent contacts between initially non-adjacent domains; the exact configuration that defeated the direct cubature scheme (§3.2.2). Without cubature, self-contact consumed *up to 96%* of the frame time, but the cubature version *never exceeded 44%*. The most highly colliding frame was *accelerated by a factor of 20*×.

Arm: We simulated the collision of an arm with a body on a mesh composed of 171,492 tetrahedra (Figure 3.5). The mesh contained 5 domains, and trained on $T = 27$ examples for a final rank of $r = 161$. All of the initial training data was quasistatic. Without cubature, self-contact took *up to 89%* of the time, while our version *never*

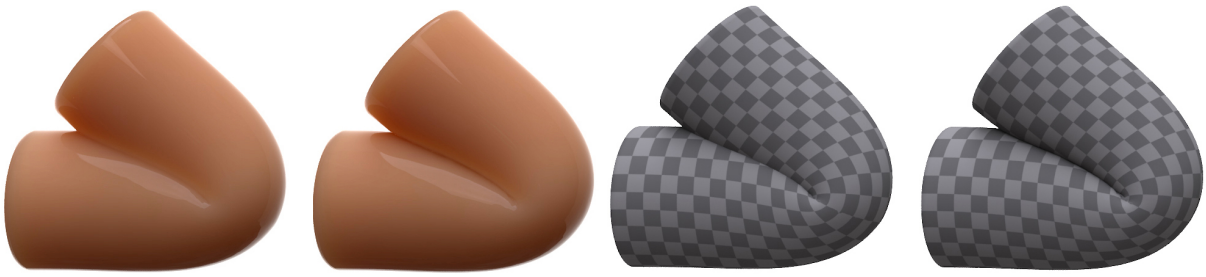


Figure 3.3: A cylinder articulated with a single joint. **Left in each pair:** Without self-contact. **Right in each pair:** With self-contact. Penetrations obscure the characteristic crease along the contact, especially in the subsurface scattering rendering. We accelerate self-contact by 52× and only take up 46% of the total computation time instead of 96%.

exceeded 43%. The most highly colliding frame was *accelerated by 11*×.

This example shows self-contacts resolved over a wide spatial extent, and subspace dynamics added as a post-process. Dynamics were activated using standard methods, i.e. by adding reduced-order mass and damping terms to the left hand side of Equation 3.3: $\widetilde{\mathbf{M}}\ddot{\widetilde{\mathbf{x}}} + \widetilde{\mathbf{C}}\dot{\widetilde{\mathbf{x}}} + \widetilde{\mathbf{f}}(\widetilde{\mathbf{x}}) = -\mathbf{U}^\top\mathbf{f}_e - \mathbf{U}^\top\mathbf{f}_s$. Newmark integration (see Appendix C of Barbič and James [12]) was then applied. Other methods could also have been used, as our method is agnostic to the underlying time integration and domain decomposition methods.

Hand: We simulated a hand containing 458,071 tetrahedra in a variety of poses. The mesh contained 10 domains, and was trained on $T = 120$ examples for a final rank of $r = 300$. The self-contacts in this example were sparser and less persistent than those in the other examples, so the performance is noisier (Figure 3.6). Without cubature, self-contact took *up to 72%* of computation, while our computation *never exceeded 42%*. The most highly colliding frame was *accelerated by 5*×. We also show an example of a collision very different from those seen during cubature training in the video[1]. As

[1] http://youtu.be/mjT9ZfIYN-s

expected, the collision is missed entirely, and the deformations begin to exhibit subspace artifacts. However, the simulation remains stable.

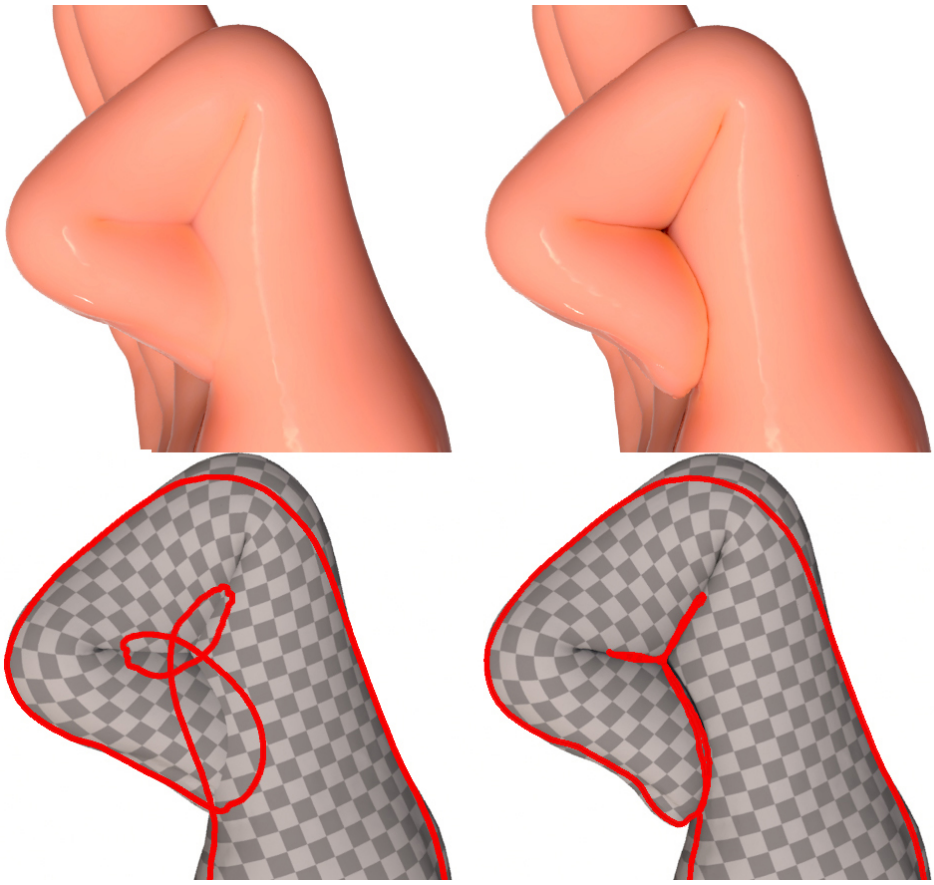


Figure 3.4: Left column: Without self-contact. **Right column:** With self-contact. A finger is shown with subsurface scattering (top), and with checkerboard texturing and a highlighted cross-section (bottom). Extreme penetrations occur without our self-collision cubature; the full extent can be seen in the cross-section. As with Figure 3.3, the characteristic crease from the contact is far less visible.

Discussion: Predictably, the largest speedups occurred when the largest number of primitives were in self-collision (Table 3.2). Faster collision detection could improve the performance of the simulations without cubature, but these methods would apply equally well to the cubature versions, so we expect that the relative performances would remain similar.

Direct comparisons to other methods are difficult due to implementation and hardware specifics, but rough, order-of-magnitude comparisons are possible. McAdams et al. [11] perform full-rank simulations of equivalent geometric complexity as ours (∼100K elements), but our performance is a clear improvement (5 s vs. 53 ms). Their algorithm is complementary and could be used to accelerate our precomputation. Other, faster methods use geometries that are orders of magnitude simpler, e.g. 4160 [100] and 8619 [91] elements. Our method offers clear advantages for both computational and geometric complexity.

3.4 Summary

We have shown that the cubature approach can be used to accelerate self-contact computation in articulated simulations. The standard approach can fail if the non-linearity of the underlying function is too severe, so we formulated a pose space variant to address this limitation. Interestingly, we found that the sparsity of the cubature training matrix serves as a useful heuristic for determining if the underlying function is "too non-linear." Our main results are:

- We show how to efficiently apply the subspace cubature approach to the problem of self-contact;
- We identify two general conditions that will cause the standard cubature approach to fail: sparse training matrices and excessive discontinuities (e.g. Heaviside functions);
- We propose *pose-space* cubature, which directly addresses these issues and enables efficient subspace self-contact.

Our pose-space cubature scheme is based on the observation that articulated self-contact is highly structured and predictable. This is no longer the case for arbitrary external collisions, which we will address in the next chapter.

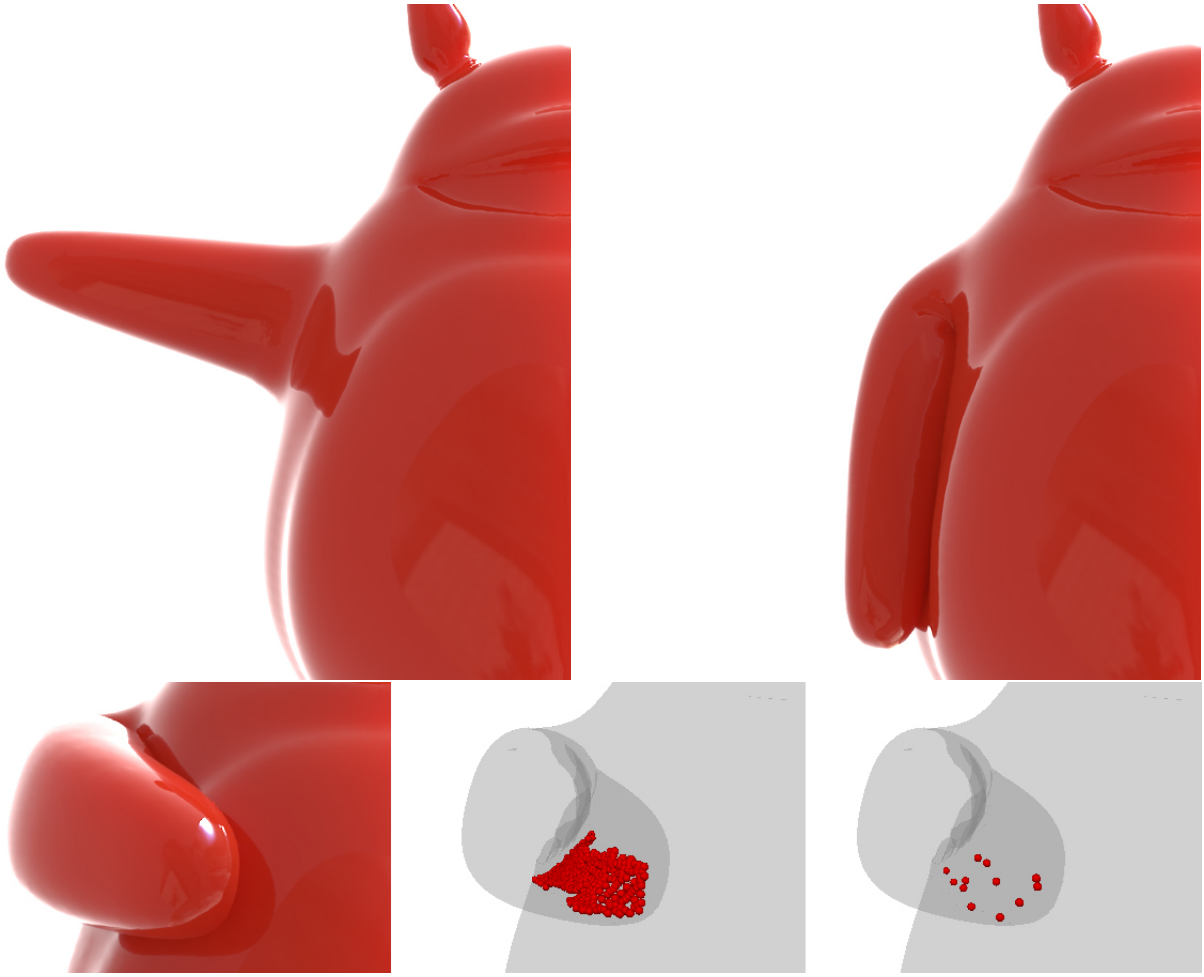


Figure 3.5: We can quickly simulate subspace dynamics, even if the input data was quasistatic. The character's arm begins outstretched (top left), but later hits the body (top right), causing it to shake. Please see the video to view the overall dynamics. As in Figure 3.2, the cubature scheme effectively sparsifies the contact (bottom row).

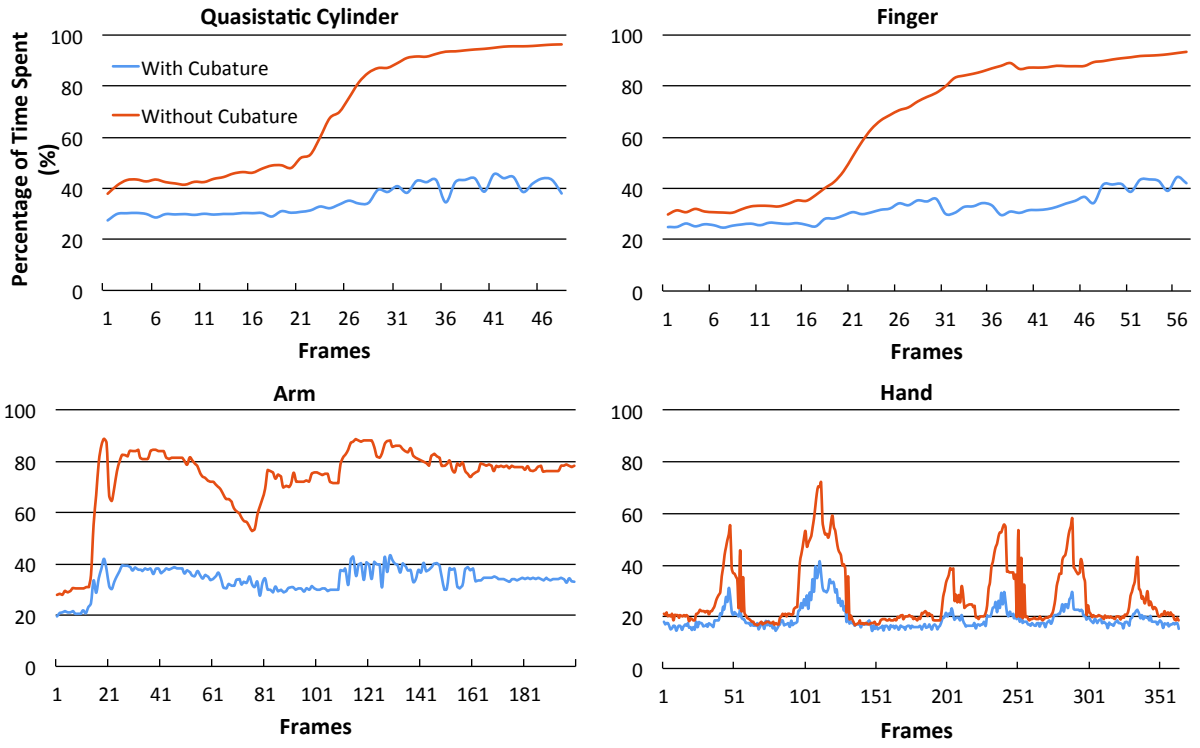


Figure 3.6: Percentage of time the subspace simulation spends in self-contact detection and resolution, with and without cubature. Without cubature, the time spent routinely exceeds 90%. Our method consumes a maximum of 46%, bringing it to parity with the rest of the simulation.

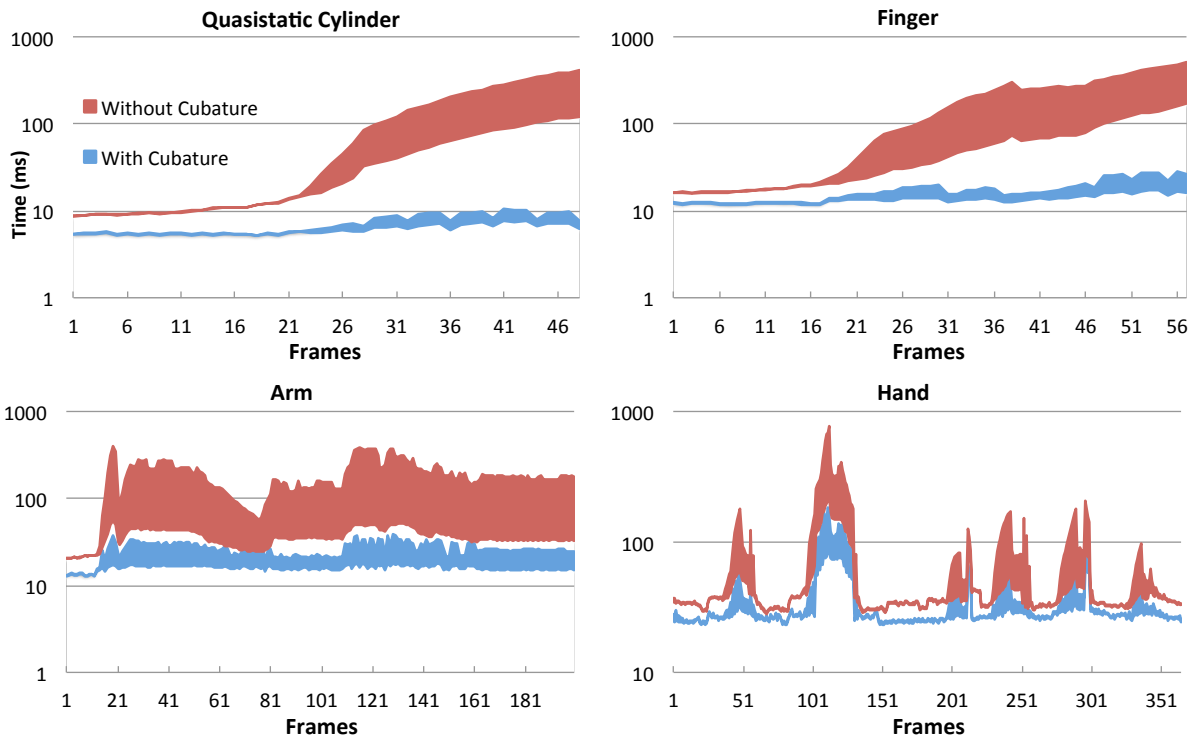


Figure 3.7: Time spent in self-collision detection and resolution, with and without cubature. The lower boundary of each shaded area is the time spent in self-collision detection, and the upper boundary shows the total time when resolution is added. Note that we have used a log scale, because otherwise the additional time added by collision resolution when using cubature would be difficult to see.

Chapter 4

Handling Arbitrary External Collisions with Subspace Condensation

Note: A large portion of this chapter has previously appeared as [35].

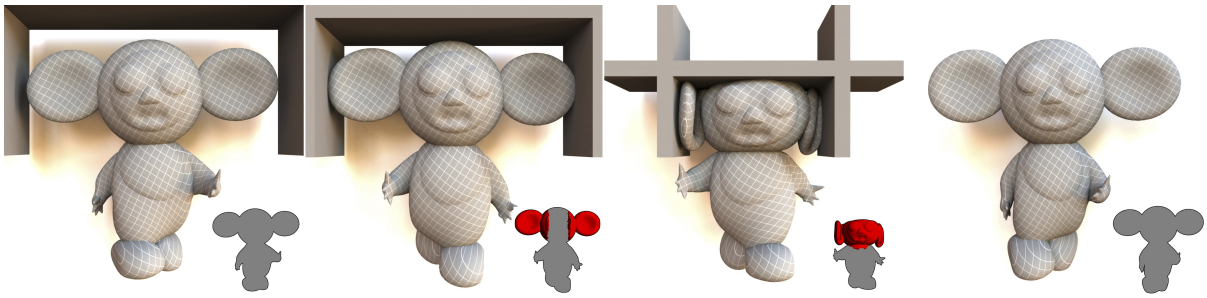


Figure 4.1: **(a)** The simulation runs at 16 FPS, entirely within the subspace, **67**× faster than a full space simulation over the entire mesh. **(b)** Novel wall collisions begin, activating full space tets, shown in red in the inset. The simulation still runs at 2.1 FPS, a **7.7**× speedup. **(c)** Collisions produce a deformation far outside the basis, and 49% of the tets are simulated in full space. The step runs at 0.5 FPS; still a **1.9**× speedup. **(d)** The collisions are removed, and the **67**× speedup returns.

The pose-space cubature algorithm described in the previous chapter successfully ac-

celerates highly structured self-contact for articulated bodies. However, it does not handle collisions outside the training set. In fact, subspace simulations can behave unrealistically when behaviors outside the prescribed subspace, such as novel external collisions, are encountered.

In this chapter, we address this limitation by presenting a fast, flexible new method that allows full space computation to be activated in the neighborhood of novel events while the rest of the body still computes in a subspace. We achieve this using a method we call *subspace condensation*, a variant on the classic *static condensation* precomputation. However, instead of a precomputation, we use the speed of subspace methods to perform the condensation at *every frame*. This approach allows the full space regions to be specified arbitrarily at runtime, and forms a natural two-way coupling with the subspace regions. While condensation is usually only applicable to linear materials, the speed of our technique enables its application to non-linear materials as well.

We show the effectiveness of our approach by applying it to a variety of articulated character scenarios. Figure 4.1 shows a few snapshots from a animation sequence simulated using subspace condensation.

4.1 Introduction

Subspace methods achieve large speedups by constraining the motion to a subspace spanned by a compact, but expressive, set of basis vectors. Inevitably, deformations that are not well-captured by the subspace arise. A straightforward example of this is an external collision, such as a cannonball hitting the mesh in a novel location that was not accounted for when constructing the subspace. In these cases, subspace methods can *lock*, producing motions that are both very different from the full space solution and unrealistic in appearance (Figure 4.2).

Figure 4.2: Novel collisions cause extreme locking in a subspace-only simulation.

A variety of strategies have been devised to address this limitation. The basis vectors usually have global support, so domain decomposition techniques described in Section 2.3 have been used to localize their influence and reduce the likelihood that a novel local deformation will trigger a non-physical, global artifact [101, 64, 61]. To avoid confusion with other decompositions of the simulation domain that we use in this chapter, we will refer to these more specifically as *skeletal decomposition* techniques.

Many enrichment techniques have also been proposed, such as the use of approximate, analytic Boussinesq solutions [23], or the construction of a large database that is used to build a custom subspace model at every frame [14, 102, 103]. While these methods are successful at making subspace methods more general, they can still be defeated by novel deformations that are not well-captured by the Boussinesq approximation, or not present in the database. Our method complements these existing ones; it can be activated at the moment that they fail.

In this chapter, we present a distinctly different approach. We observe that in many cases, particularly when dealing with external collisions, subspace methods only diverge from the full space solution in spatially localized patches. Unfortunately, there is no way

to know during the precomputation stage where these patches will be, and how their locations will change over time. Therefore, we propose an approach that activates full space computation along these patches at run-time, but allows the rest of the model, where the subspace approximation is still valid, to continue computing efficiently in the subspace.

We accomplish this using a method we call *subspace condensation*, which is a variation on the widely known *static condensation* method from the domain decomposition literature [104]. While condensation has traditionally been deployed as a precomputation for linear materials, we show that it can be efficiently computed at runtime, even for non-linear materials, by leveraging the reduced dimensionality of the subspace. The method does not require any constraint mechanisms such as spring forces [64] or Lagrange multipliers [61], to couple the subspace and full space regions.

Our method allows subspace methods to be used in cases where they previously would not have been considered. Even if it is known in advance that novel behaviors will arise, we provide a mechanism that allows the simulation to only "pay for" the novel components in each frame, while other, more familiar behaviors are solved efficiently. Our method gracefully degrades, so in the worst case scenario, it merely falls back to a full space simulation over the entire mesh. We demonstrate the effectiveness of our algorithm on a variety of character animation examples.

4.2 Static Condensation

Static condensation has been known in structural mechanics and civil engineering for some time [105, 106, 107] and was originally developed for static vibration analysis, i.e. the eigenmodes of a structure. For this reason, the procedure is also sometimes referred to as *eigenvalue economization* [108]. Practitioners are naturally also interested

in dynamics, so *dynamic condensation* was developed to take inertial effects into account [108, 109]. In graphics, we are by no means the first researchers to leverage this technique, as it was used successfully by Bro-Nielsen and Cotin [110] for real-time surgery. More recently, it was leveraged successfully in the context of physically-based skinning [111], where a very nice connection to the Steklov-Poincaré operator was also drawn, and was applied in a novel material optimization context by Xu and colleagues [112]

We will first review the basics of static condensation before describing our novel variant. For consistency, we adhere to the notation of Bro-Nielsen and Cotin [110] where possible. Let $\mathbf{Ku} = \mathbf{f}$ be the linearized, quasistatic system that models a solid object. If we reorder the vertices so that the surfaces vertices (V_e) come before the internal vertices (V_i), we can rewrite the system as a block structure:

$$\begin{bmatrix} \mathbf{K}_{ee} & \mathbf{K}_{ei} \\ \mathbf{K}_{ie} & \mathbf{K}_{ii} \end{bmatrix} \begin{bmatrix} \mathbf{u}_e \\ \mathbf{u}_i \end{bmatrix} = \begin{bmatrix} \mathbf{f}_e \\ \mathbf{f}_i \end{bmatrix} \tag{4.1}$$

Here, e indicates external (i.e. surface) and i indicates internal, where $|V_e| + |V_i| = N$. The $\mathbf{K}_{ie}$ and $\mathbf{K}_{ei}$ blocks represent the couplings between the two. By using 2×2 block Gaussian elimination, we obtain a system that only involves the surface vertices,

$$\mathbf{K}^*_{ee}\mathbf{u}_e = \mathbf{f}^*_e, \tag{4.2}$$

where

$$\mathbf{K}^*_{ee} = \mathbf{K}_{ee} - \mathbf{K}_{ei}\mathbf{K}^{-1}_{ii}\mathbf{K}_{ie} \tag{4.3}$$

$$\mathbf{f}^*_e = \mathbf{f}_e - \mathbf{K}_{ei}\mathbf{K}^{-1}_{ii}\mathbf{f}_i. \tag{4.4}$$

Equation 4.3 is the well-known Schur complement, a widely used expression in domain

decomposition, and Eqns. 4.3 and 4.4 together form its standard application. After solving for $\mathbf{u}_e$, if $\mathbf{u}_i$ is desired, it can still be retreived via

$$\mathbf{u}_i = \mathbf{K}_{ii}^{-1}(\mathbf{f}_i - \mathbf{K}_{ie}\mathbf{u}_e). \tag{4.5}$$

However, if only the external surface positions $\mathbf{u}_e$ are needed, the degrees of freedom of the internal vertices have been condensed away. In the case of linear materials, both Equation 4.3 and the $\mathbf{K}_{ei}\mathbf{K}_{ii}^{-1}$ term in $\mathbf{f}_e^*$ can be precomputed and reused at runtime [110]. The runtime cost is then drastically reduced, as the inverse at runtime now depends on $|V_e|$, the number of surface vertices, not N, the number of total vertices. Adding dynamics is then a straightforward application of the same block reordering to the mass and damping matrices, $\mathbf{M}$ and $\mathbf{C}$ (see §2.4.2 in [110]).

Discussion: Static condensation works best in the context of linear materials. In the non-linear case, $\mathbf{K}$ is no longer constant and becomes $\mathbf{K}(\mathbf{u})$, and repeatedly computing the $\mathbf{K}_{ii}(\mathbf{u})^{-1}$ in Eqs. 4.3 and 4.4 can be prohibitively expensive. Some progress has been made on this limitation, as Gao and colleagues [111] recently formulated an *ex-rotated* (extrinsically rotated) material model that is specifically tailored to efficiently approximate $\mathbf{K}_{ii}(\mathbf{u})^{-1}$.

Our work differs from the two most related graphics works [110, 111] in two key ways. First, due to the presence of an expensive matrix inverse, they either perform the condensation as a pre-process or build a special material model whose inverse is easier to compute. We show that by leveraging fast subspace inverses, condensation can be performed quickly and dynamically at run-time for an arbitrary material.

Second, they assume that the surface degrees of freedom (DOFs) are the most important ones, and use condensation to project away the interior DOFs. Our method allows *any* subset of nodes to be designated as important, allows these designates to be changed

at every frame, and can quickly project away the complexity of their complement. This distinction is significant, because in the case of local, non-trivial contacts, DOFs on the interior of the mesh can play a significant role in the appearance of the final deformation, and should not always be be projected away. Our general approach allows any subset of interior DOFs to be simulated as necessary, and the surface-only variant becomes a special case. We elaborate on these distinctions in the next section.

Partition Oracle: The issue of how the vertices are divided into full space and subspace regions is a separate question that we delegate to an external *partition oracle*. We will propose several oracles in §4.4.2, but for now will put this issue aside and describe a generic condensation technique that is not dependent on the specifics of any particular oracle.

4.3 Subspace Condensation: Combining Full Space and Subspace Simulations

The main bottleneck of static condensation is computing the inverse, $\mathbf{K}_{ii}^{-1}(\mathbf{u})$. Subspace methods excel at quickly inverting compact approximations to these kinds of matrices, $\widetilde{\mathbf{K}}_{ii}^{-1}(\mathbf{q})$. Their applicability looks promising, but several non-trivial issues must be addressed before the algorithm becomes practical.

We define *full vertices* as those undergoing full space simulation, and denote their set as V_f. The rest are referred to as *subspace vertices* and denoted as V_s. Again, we assume that some external partition oracle has provided these labels. Let us consider quasistatic deformations without any external forces. Assuming that V_f is not empty, we reorder

the vertices and write the system $\mathbf{Ku} = \mathbf{f}$ as,

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fs} \\ \mathbf{K}_{sf} & \mathbf{K}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{u}_s \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{f}_s \end{bmatrix}. \tag{4.6}$$

The above equation is the linearized system for one Newton iteration, where we have abbreviated $\mathbf{K}(\mathbf{u}) = \mathbf{K}$, $\mathbf{K}_{ff}(\mathbf{u}) = \mathbf{K}_{ff}$, and so on, for brevity. Applying the condensation technique now requires the inversion of $\mathbf{K}_{ss}$, much like in Eqns. 4.3 and 4.4.

A Matrix Formulation: Fast subspace inverses can be directly, but naïvely, applied to this problem. For example, an analog to Equation 4.4,

$$\mathbf{f}_f^* = \mathbf{f}_f - \mathbf{K}_{fs}\mathbf{K}_{ss}^{-1}\mathbf{f}_s, \tag{4.7}$$

can incorporate the subspace matrix $\widetilde{\mathbf{K}}_{ss}^{-1}(\mathbf{q})$ thusly,

$$\mathbf{f}_f^* \approx \mathbf{f}_f - \mathbf{K}_{fs}\left(\mathbf{U}_s\widetilde{\mathbf{K}}_{ss}^{-1}(\mathbf{q})\mathbf{U}_s^T\right)\mathbf{f}_s, \tag{4.8}$$

where $\mathbf{U}_s$ is a basis matrix composed of the rows of $\mathbf{U}$ that correspond to the vertices in V_s. The inverse is quickly computed in the subspace, expanded into a larger matrix, and used to compute $\mathbf{f}_f^*$. An analogous method can be used for Equation 4.3.

Unfortunately, this formulation is highly inaccurate. If we think of the $\mathbf{K}_{fs}\mathbf{K}_{ss}^{-1}\mathbf{f}_s$ term in Equation 4.7 as a corrective force vector to $\mathbf{f}_f$, it becomes clear why this occurs. It is reasonable to expect that the vector $\mathbf{K}_{ss}^{-1}\mathbf{f}_s$ lies in the span of $\mathbf{U}_s$, since that corrective force, or something similar, was visible to the SVD that constructed the subspace $\mathbf{U}_s$. However, it is too optimistic to expect that $\mathbf{U}_s\widetilde{\mathbf{K}}_{ss}^{-1}(\mathbf{q})\mathbf{U}_s^T$, the expanded version of $\mathbf{K}_{ss}^{-1}$, will also be a meaningful approximation. This would imply that $\mathbf{K}_{ss}^{-1}$ is low rank, and that $\mathbf{U}_s$ spans its dominant eigenvectors. Simple numerical experiments verify that neither of

these assumptions are true; if $\mathbf{K}_{ss}$ were not full rank, its inverse would not exist.

Instead, it is clear that $\widetilde{\mathbf{K}}_{ss}^{-1}(\mathbf{q})$ should not be expanded, as it is only a meaningful Jacobian of $\widetilde{\mathbf{f}}_s$. The basis $\mathbf{U}_s$ was constructed to approximate $\mathbf{f}_s$, but not its full-rank Jacobian. Therefore, we need to structure our algorithm so that not only the inverse is computed quickly, but that the result is carried forward in the subspace until a reduced version of the entire corrective force, $\mathbf{K}_{ss}^{-1}\mathbf{f}_s$, is obtained.

A Vector Formulation: We instead examine Equation 4.6 by expanding it from its block form into

$$\mathbf{K}_{ff}\mathbf{u}_f + \mathbf{K}_{fs}\mathbf{u}_s = \mathbf{f}_f$$
$$\mathbf{K}_{sf}\mathbf{u}_f + \mathbf{K}_{ss}\mathbf{u}_s = \mathbf{f}_s.$$

If we project the entire second equation using $\mathbf{U}_s^T$, and perform the substitution $\mathbf{u}_s \approx \mathbf{U}_s\mathbf{q}_s$ on both equations, we obtain

$$\mathbf{K}_{ff}\mathbf{u}_f + \mathbf{K}_{fs}\mathbf{U}_s\mathbf{q}_s = \mathbf{f}_f \tag{4.9}$$
$$\mathbf{U}_s^T\mathbf{K}_{sf}\mathbf{u}_f + \widetilde{\mathbf{K}}_{ss}\mathbf{q}_s = \widetilde{\mathbf{f}}_s. \tag{4.10}$$

The subspace regions now communicate with the full space through $\mathbf{q}_s$, namely the $\mathbf{K}_{fs}\mathbf{U}_s\mathbf{q}_s$ product, not through a rank-deficient expansion of the $\widetilde{\mathbf{K}}_{ss}^{-1}$ matrix. The system can be returned to block form:

$$\begin{bmatrix} \mathbf{K}_{ff} & \mathbf{K}_{fs}\mathbf{U}_s \\ \mathbf{U}_s^T\mathbf{K}_{sf} & \widetilde{\mathbf{K}}_{ss} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{q}_s \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \widetilde{\mathbf{f}}_s \end{bmatrix}. \tag{4.11}$$

Solving this new system (Equation 4.11) yields exactly the subspace-to-full space coupling that we seek. In order to solve it efficiently, one additional component is needed, which

we will now describe.

Efficient Force Evaluation: It is not immediately obvious how to efficiently compute $\widetilde{\mathbf{f}}_s$, the internal forces on the subspace vertices. A brute-force method would be to compute the full space force, $\mathbf{f}_s$ and then project it [17]. However, this would make its evaluation $O(N)$, a complexity that we explicitly want to avoid.

One approach would be to compute the force over *all* vertices using an existing method [12, 20], and to subtract off the contribution from the full space vertices, $\widetilde{\mathbf{f}}_s \approx \widetilde{\mathbf{f}} - \mathbf{U}_f^T \mathbf{f}_f$, where, $\mathbf{U}_f$ denotes the rows of $\mathbf{U}$ that correspond to the full vertices, V_f. However, since the full space force can contain components that are not well-captured by $\mathbf{U}_f$, the projection produces unusable results.

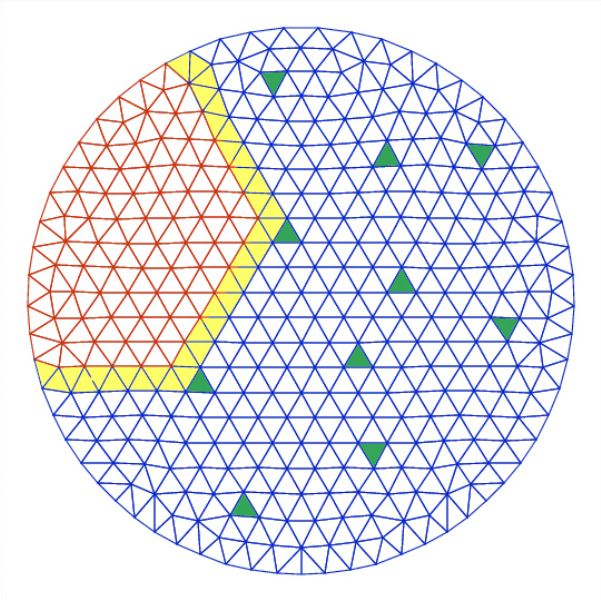


Figure 4.3: A 2D illustration of Equation 4.12. The $\widetilde{\mathbf{f}}_s$ vector is computed by projecting the force exerted by boundary tets (yellow) onto the subspace region (blue) and adding the weighted forces from the cubature tets (green) that reside completely in the subspace region (blue).

Fortunately, we are able to devise an efficient, alternative, cubature-based [20, 39] method. First, we only evaluate the cubature tets that lie inside the set of subspace

tets, T_s. Second, we project the forces exerted by the *boundary* tets onto the subspace region (i.e. tets on the boundary between the full space and subspace regions), which are already available from evaluating $\mathbf{f}_f$. Unlike in the previous case, these boundary forces are very likely to be well-captured by the basis, as they have already been partially constrained to the subspace during previous timesteps. The remainder of the full space region (Figure 4.3, red), whose out-of-basis projections will likely just produce locking artifacts, is correctly ignored. More formally:

$$\widetilde{\mathbf{f}}_s \approx \sum_{j \in V_b} \mathbf{U}_j^T \mathbf{f}_j + \sum_{i \in T_c} \delta \cdot w_i \cdot \mathbf{U}_i^T \mathbf{f}_i(\mathbf{q}) \begin{cases} \delta = 1 & \text{if } i \in T_s \\ \delta = 0 & \text{if } i \in T_f \end{cases}. \tag{4.12}$$

In the first summation, V_b is the set of subspace vertices on the boundary between the full space and subspace regions, $\mathbf{f}_j$ is the internal force on the jth boundary vertex exerted by its adjacent boundary tets, and $\mathbf{U}_j$ is the rows of $\mathbf{U}$ that correspond to that vertex. In the second summation, T_c is the set of cubature tets, w_i denotes the weight on cubature tet i, $\mathbf{U}_i$ the rows of $\mathbf{U}$ that correspond to the vertices in tet i, and $\mathbf{f}_i$ is the internal force function evaluated at tet i. Figure 4.3 shows a 2D illustration of this process. The total time to evaluate Equation 4.12 is $O(|T_c| + |V_b|)$, which is proportional to the complexity of the subspace and the currently activated full vertices. Any $O(N)$ dependency on the full complexity of the model has been removed.

Solving the System: While it is be possible to apply the traditional Schur complement to Equation 4.11, it is both undesirable and unnecessary. We always want to solve for $\mathbf{q}_s$, as it is needed to update the surface positions in the subspace region. Unlike the classic static condensation case, $\mathbf{q}_s$ is very small, so computing it in addition to $\mathbf{u}_f$ does not add prohibitive complexity. This also allows us to trivially apply the method to non-linear materials, as we can quickly solve several Newton iterations of $\mathbf{q}_s$.

We use preconditioned conjugate gradients (PCG) to solve Equation 4.11. We use a

Jacobi preconditioner for the upper diagonal block, and precondition the lower diagonal block with its explicit inverse ($\widetilde{\mathbf{K}}_{ss}^{-1}$). The solver typically converges in fewer than 100 iterations. Using a more sophisticated preconditioner such as Incomplete Cholesky (IC) is difficult, because each factorization sees limited re-use before V_f changes and it has to be recomputed. Ideally a multigrid method would be applied to this system, but we leave this problem as future work.

The entire solution procedure now depends on the rank r of the subspace, and $|V_f|$, the number of active full space vertices. The $\mathbf{K}_{fs}\mathbf{U}_s$ product and its transpose in Equation 4.11 at first appears to be $O(N \times r)$, but $\mathbf{K}_{fs}$ is very sparse, and contains $O(|V_f|)$ non-zero entries. Since $\mathbf{K}_{fs}$ encodes the coupling between $O(|V_f|)$ vertices and the rest of the mesh, this sparsity is to be expected. The overall solver is outlined in Algorithm 4, and it is clear that no $O(N)$ computation is needed at any stage.

Algorithm 4 Integration using subspace condensation

construct active full space region V_f
for $i := 1$ **to** n **do** ▷ n = # of Newton iterations
  if $V_f \neq \emptyset$ **then**
    initialize $\mathbf{q}_s = \mathbf{0}$
    compute $\mathbf{f}_f, \mathbf{K}_{ff}, \mathbf{U}_s^T\mathbf{K}_{sf}$ $\left(= (\mathbf{K}_{fs}\mathbf{U}_s)^T\right)$
    compute $\widetilde{\mathbf{f}}_s$ and $\widetilde{\mathbf{K}}_{ss}$ using cubature (Equation 4.12)
    solve Equation 4.11 for $\mathbf{u}_f$ and $\mathbf{q}_s$.
    update full space region using $\mathbf{u}_f$
    update subspace region using $\mathbf{q}_s$
  else
    perform subspace-only Newton step over entire mesh
  end if
  end if
end for
end for

4.4 Physics-Based Skinning

Our simulation technique is general enough to be applied to any subspace deformable body simulation. As a proof-of-concept, we have applied it to the problem of novel external collisions in physics-based skinning. We will describe the features of the skinning technique here, while noting that the subspace condensation technique does not fundamentally rely on any of them.

4.4.1 Basis Construction

We elected to use a skinning correction basis, similar to that used by EigenSkin [90] and the kinematic correction employed by Hahn et al. [14]. Keeping the notation similar to the latter paper where possible, let us use φ to denote a skinning function, and express the final deformed, world-space position $\mathbf{x}$ for a single vertex m as

$$\begin{aligned}\mathbf{x}^m &= \overline{\mathbf{x}}^m + \mathbf{u}^m \\ &= \varphi(\mathbf{p}, \overline{\mathbf{x}}^m + \overline{\mathbf{u}}^m) = \mathbf{R}(\overline{\mathbf{x}}^m + \overline{\mathbf{u}}^m) + \mathbf{t}.\end{aligned}$$

Here, $\mathbf{p}$ is the current skeleton configuration, e.g. as expressed by joint angles, $\overline{\mathbf{x}}^m$ denotes the rest pose, $\mathbf{u}^m$ is the world-space displacement, and $\overline{\mathbf{u}}^m$ is the same displacement prior to the skinning transformation. $\mathbf{R}$ and $\mathbf{t}$ together represent the affine transformation determined by the skinning function. In order to obtain this transform, we used dual-quaternion skinning [8] and volumetric heat diffusion [113] to propagate the weights throughout the tetrahedral mesh. One of the nice properties of dual-quaternion skinning is that $\mathbf{R}$ is guaranteed to be a pure rotation matrix. This will later be used to simplify the skeletal decomposition forces.

In order to build our basis $\mathbf{U}$, we first obtained samples of the global displacement

vector $\mathbf{u}$ using full space simulations over the entire mesh. We then inverted the skinning transform φ^{-1} to pull each $\mathbf{u}$ back to its untransformed version $\overline{\mathbf{u}}$. Our subspace basis $\mathbf{U}$ is then constructed by performing a truncated PCA over these samples of $\overline{\mathbf{u}}$. The world-space, full space position is then computed as:

$$\mathbf{x} = \varphi(\mathbf{p}, \overline{\mathbf{x}} + \mathbf{U}\mathbf{q}). \tag{4.13}$$

As observed in previous work, this basis is significantly more flexible than the one obtained by performing PCA directly over $\mathbf{u}$. The skinning transforms have been factored out, so the correctives that remain can be applied over a wider range of scenarios, not just to the original skinning transform that produced it.

Skeletal Decomposition: The role of the skinning transform φ in the basis could potentially complicate the use of skeletal decomposition techniques, such as the one from Kim and James [64]. In that work, penalty springs were inserted between bone-centered domains to ensure their compatibility. A 3$^{\text{rd}}$-order *fast sandwich transform* (FST) tensor had to be introduced to efficiently incorporate each domain's local rotation into the subspace computation.

In our work, we found that the choice of a skinning correction basis *removes* the need for any special transforms. To see why this is so, we examine the penalty spring energy E^m between two domains, i and j. Let $\mathbf{v}^m$ be an interface vertex between these two

domains with respective positions of $\mathbf{x}_i^m$ and $\mathbf{x}_j^m$. The spring energy is then written as:

$$E^m = \frac{1}{2}k\ (\mathbf{x}_i^m - \mathbf{x}_j^m)^T(\mathbf{x}_i^m - \mathbf{x}_j^m) \tag{4.14}$$

$$\begin{aligned} = \frac{1}{2}k\ &[\varphi(\mathbf{p}, \overline{\mathbf{x}}^m + \overline{\mathbf{u}}_i^m) - \varphi(\mathbf{p}, \overline{\mathbf{x}}^m + \overline{\mathbf{u}}_j^m)]^T \\ &[\varphi(\mathbf{p}, \overline{\mathbf{x}}^m + \overline{\mathbf{u}}_i^m) - \varphi(\mathbf{p}, \overline{\mathbf{x}}^m + \overline{\mathbf{u}}_j^m)] \end{aligned} \tag{4.15}$$

$$\begin{aligned} = \frac{1}{2}k\ &[\mathbf{R}_j^m(\mathbf{U}_i^m\mathbf{q}_i - \mathbf{U}_j^m\mathbf{q}_j)]^T \\ &[\mathbf{R}_i^m(\mathbf{U}_i^m\mathbf{q}_i - \mathbf{U}_j^m\mathbf{q}_j)]. \end{aligned} \tag{4.16}$$

Here k is the spring constant, $\mathbf{U}^m \in \mathbb{R}^{3\times r}$ is the basis for vertex $\mathbf{v}^m$, and $\mathbf{R}_i^m$ and $\mathbf{R}_j^m$ are the rotations for the vertex in partitions i and j. The solver then requires the gradient (i.e. the spring force) and the Hessian of E^m with respect to $\mathbf{q}_i$:

$$\frac{\partial E^m}{\partial \mathbf{q}_i} = k(\mathbf{U}_i^m)^T(\mathbf{R}_j^m)^T\mathbf{R}_i^m(\mathbf{U}_i^m\mathbf{q}_i - \mathbf{U}_j^m\mathbf{q}_j) \tag{4.17}$$

$$\frac{\partial^2 E^m}{\partial {\mathbf{q}_i}^2} = k(\mathbf{U}_i^m)^T(\mathbf{R}_j^m)^T\mathbf{R}_i^m\mathbf{U}_i^m. \tag{4.18}$$

In the previous work [64], the composition of rotations, $(\mathbf{R}_i^m)^T\mathbf{R}_j^m$, was then fed into an FST tensor described in section 2.3. However, under the current basis, the skinning guarantees the two rotations to always be the same, $\mathbf{R}_i^m = \mathbf{R}_j^m$, so the composition $(\mathbf{R}_i^m)^T\mathbf{R}_j^m$ is always the identity matrix. The gradient then becomes,

$$\frac{\partial E^m}{\partial \mathbf{q}_i} = k((\mathbf{U}_i^m)^T\mathbf{U}_i^m\mathbf{q}_i - (\mathbf{U}_i^m)^T\mathbf{U}_j^m\mathbf{q}_j), \tag{4.19}$$

where everything aside from $\mathbf{q}_i$ and $\mathbf{q}_j$ can be precomputed, and the Hessian resolves to a constant matrix,

$$\frac{\partial^2 E^m}{\partial {\mathbf{q}_i}^2} = k(\mathbf{U}_i^m)^T\mathbf{U}_i^m. \tag{4.20}$$

Other gradient and Hessian terms can be deduced similarly.

4.4.2 Contact and Dynamics Oracles

We applied subspace condensation to the problem of contact handling in both quasistatic and dynamic simulations, and used the penalty-based collision force model from McAdams et al. [11] for both external and self-collisions.

Contact Oracle: As mentioned in §4.2, some form of *partition oracle* is needed to label the full space and subspace regions. When contacts are the main source of novel deformations, it is natural to build an oracle that is based on collisions. We labelled the vertices that are in collision as belonging to the full space region, and additionally applied a simple, distance-based criterion. Specifically, we conducted a breadth-first search that started from each colliding vertex and terminated when the vertices within an influence radius ρ of the starting vertex had been included. All of the vertices encountered during this search were added to the full space region. Consequently, all collision-related force and Hessian terms only exist in the $\mathbf{f}_f$ and $\mathbf{K}_{ff}$ terms in Equation 4.11. The radius ρ provided a speed-quality tradeoff that will be discussed in detail in §4.5.

When the mesh is recovering from a complex contact, we found that it is inadvisable to deactivate the full space region as soon as no collisions are detected. The subspace basis has no knowledge of the deformation created by the collision, and can have trouble generating the detailed, localized, restoring force necessary to untangle a configuration, e.g. a fold that formed in the palm of a hand. Therefore, the oracle was modified to also include the vertices discovered by the breadth-first search from the previous frame. After two frames, the search results time out, which allow the full space region to shrink.

Dynamics Oracle: Our subspace condensation approach can be applied to dynamics simulation by adding another modification to the oracle. Even if a body is no longer in

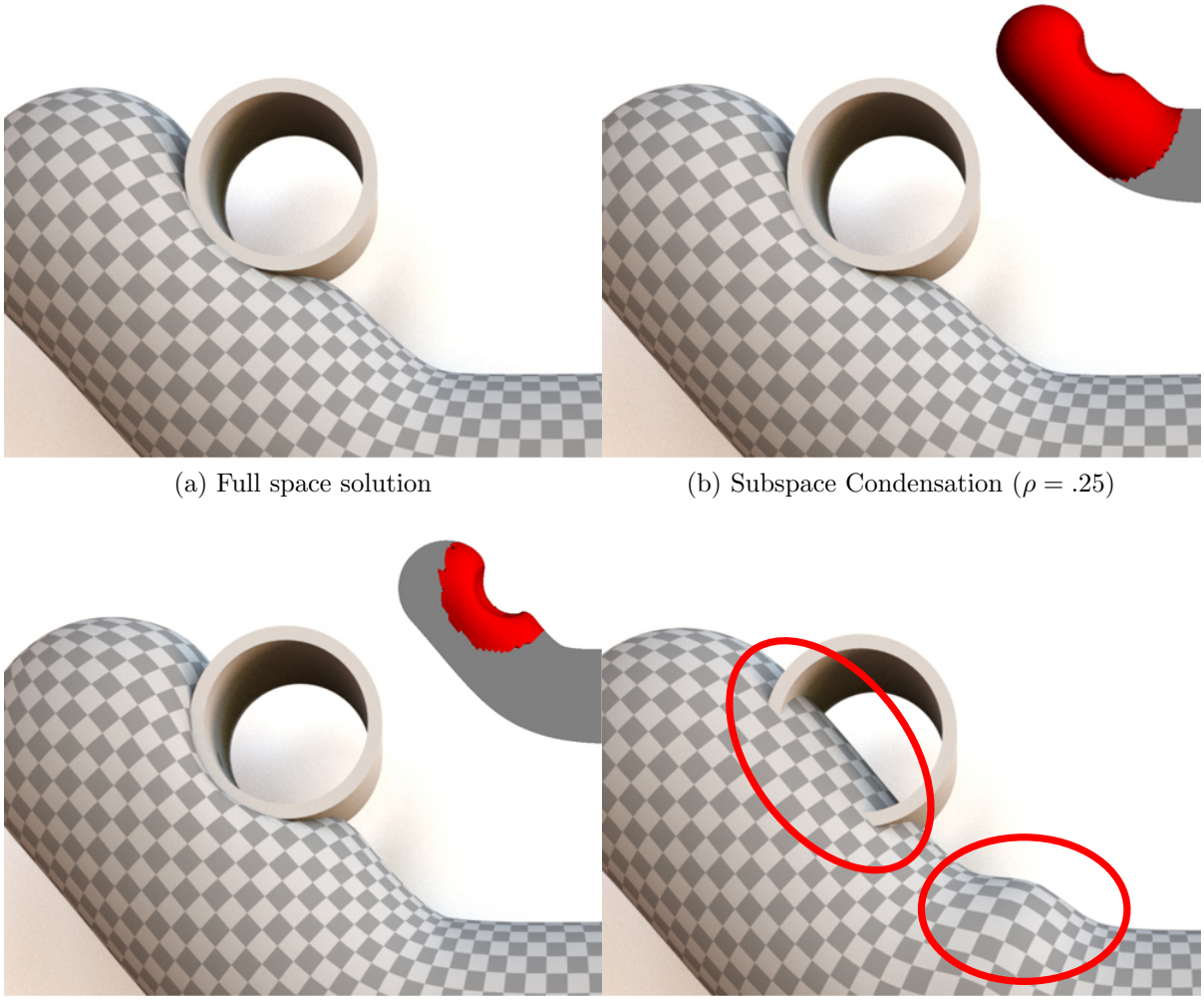
collision, accelerations caused by the contact forces can lead to interesting deformations that are not captured by the subspace basis. Therefore, when simulating dynamics, the oracle only folds a full space region back into the subspace if both the average velocity and acceleration for the full vertices are below a certain threshold. We found this simple strategy to be effective, though somewhat conservative.

4.5 Results

We use Newton's method as our non-linear solver and implicit Euler for our time discretization. Our full rank and subspace condensation solvers use Jacobi-preconditioned Conjugate Gradients for solving the linearized systems. Dense linear algebra routines, such as the direct subspace inverse, use the Eigen [114] library. All simulations were run on a relatively modest 8-core, 2.4 Ghz MacBook Pro with 8 GB of RAM using 8 threads. When possible, we leverage OpenMP for parallelization. All of our results used the co-rotational material from McAdams et al. [11]. A performance summary of all the examples is shown in Table 4.1 and 4.2.

Capsule: As an initial test case, we simulated a capsule containing 29,343 vertices and 160,960 tetrahedra. We rigged the capsule with 2 bones and trained the subspace using 12 bending poses, producing a basis with rank 11. Figure 4.4(a) shows the full space simulation of the capsule bending and colliding against a pipe. The subspace-only simulation cannot resolve the novel contact deformation, so we observe severe locking (Figure 4.4(d)).

Our subspace condensation method easily handles this situation by adaptively activating full space vertices near the contact region. With a small influence radius ($\rho = 0.08$) we obtain a useful approximation to the full solution with an average speedup of 76$\times$ (Figure 4.4(c)). Increasing the ρ to 0.25 refines the approximation further (Figure 4.4(b))



(a) Full space solution

(b) Subspace Condensation ($\rho = .25$)

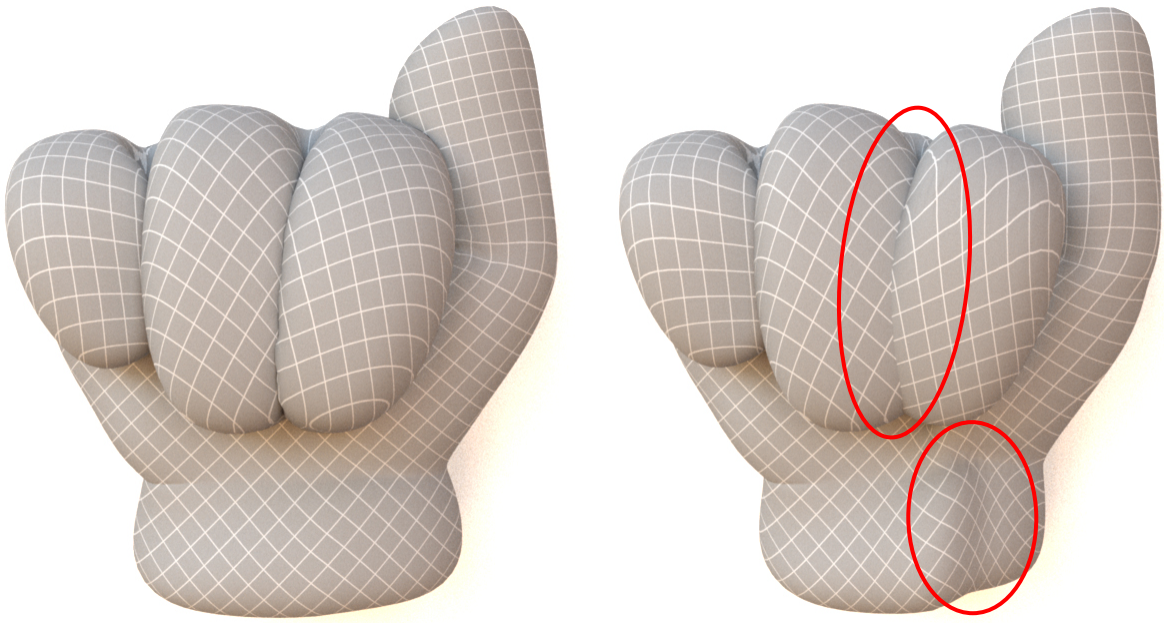
(c) Subspace Condensation ($\rho = .08$)

(d) Subspace-only solution

Figure 4.4: For (b) and (c), activated full space region is shown in red in the inset. While the deformation in (c) is less pronounced than the ground truth, the contact is still resolved. (d) Subspace-only simulation cannot resolve the contact. Locking artifacts are circled in red.

and still maintains an average speedup of nearly an order of magnitude (9.4×). In the worst case, 50.6% of the vertices are simulated in full space, but we still see a 3.5× speedup.

Hand: The hand mesh in chap4figs. 4.5 and 4.8 consists of 95,746 vertices and 458,071 tetrahedra, and is rigged with 10 bones. The subspace was constructed using



(a) Subspace Condensation (b) Subspace-only solution

Figure 4.5: Comparison between our approach with subspace-only simulation on a highly novel contact configuration. The subspace-only solution produces significant locking artifacts, circled in red.

a simple, sparse sampling of 53 snapshots obtained by actuating each bone in isolation. The basis has no knowledge of multiple bones moving in tandem. The total rank of the subspace bases, summed over all the skeletal domains, is 156. Self-collisions were taken into account during training, so we used these samples to compute self-collision cubature (SCC) [103] in order to quickly compute similar joint collisions at runtime. The SCC was quickly computed using the NN-HTP algorithm [39].

We then put the hand through a quasistatic, calisthenic exercise regimen. The complete sequence is shown in the supplement video[1]. The subspace-only simulation easily handled different combinations of individual joint motions as long as there was no novel contact. However, it immediately failed when novel collisions occurred (Figure 4.5(b)).

[1]https://youtu.be/rS-DxWyi5z4

In contrast, our approach was able to resolve arbitrary contact by adaptively activating full space vertices near the contact areas (Figure 4.5(a)). Collisions seen during training were handled using self-collision cubature unless they overlapped with the full space region. In this case, the entire region was treated as a novel contact. On average, our simulation ran at 0.28 s/frame, accelerating the full space simulation by 48×. Even in the worst case, the most novel collision-induced deformation was computed 8.1× faster than a full space simulation over the entire mesh. Figure 4.8 shows the time per frame, as well as the percentage of vertices being simulated in full space, for a fist clenching sequence. The simulation time is clearly proportional to the size of the full space region.

To test for convergence of our approach, we compared the simulation errors of using different influence radii against the full space simulation solution for a subsequence of the hand motion (Fig. 4.6). We used relative $\mathbf{L}_2$ error. It is clear that the error decreases as the influence radius increases.

Cheb: The "Cheb" mesh [113], shown in Figures 4.1, 4.2 and 4.7, contains 26,652 vertices and 123,464 tetrahedra, and is rigged with 17 bones. We constructed the subspace using 38 evenly spaced samples of a walk cycle. The total rank of the subspace basis, summed over all the skeletal domains, is 301. SCC was used to resolve the predictable collisions between Cheb's feet.

As an extreme collision test, we had three walls gradually crush Cheb's head. The forces caused extreme locking in the subspace-only simulation (Figure 4.2), while our simulation gracefully handled the deformation by activating full space computation in half of the vertices (Figure 4.1). A large influence radius was needed in this example ($\rho = 0.39$) due to the geometric and material properties of the model: an impulse at the tip of the ear propagates quickly through the entire ear. On average, this difficult scenario was accelerated by a factor of 4.8×. Even when half of the mesh was being simulated in full space, we saw a 1.9× speedup; the other half of the mesh was simulated

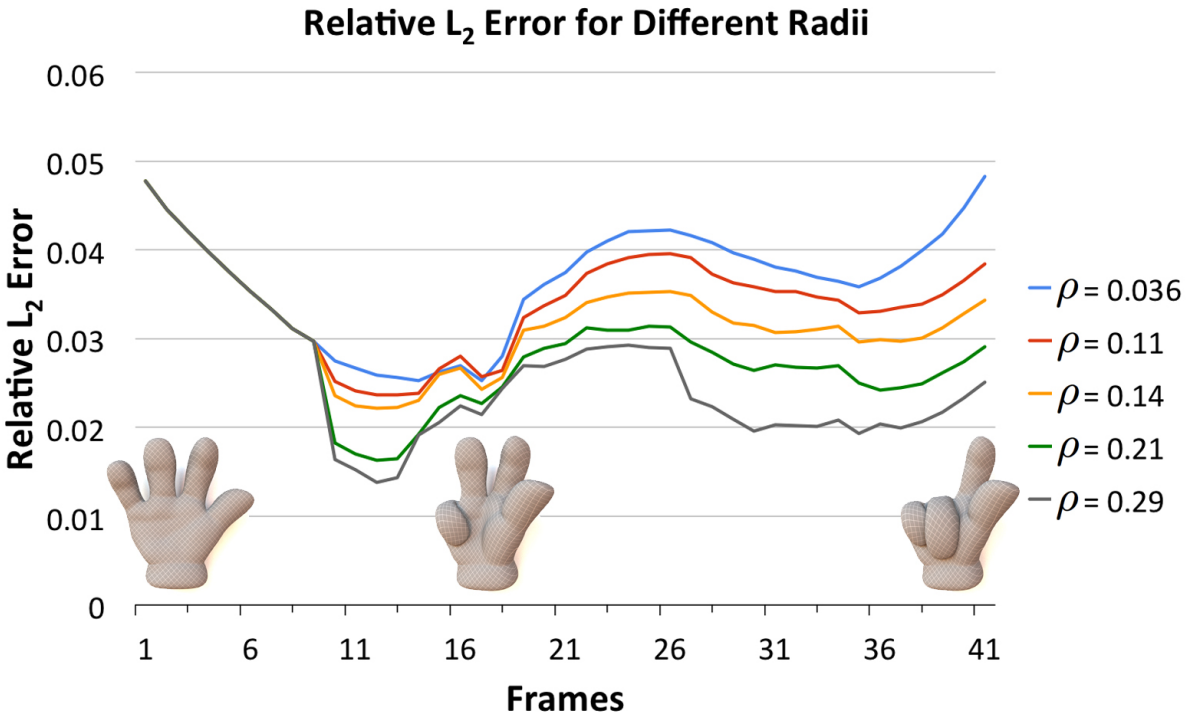


Figure 4.6: The relative error of our approach using different influence radii, plotted with different colors, compared to the full simulation. Full space regions start appearing at frame 9. The relative error clearly decreases as the radius increases.

in the subspace for a negligible additional cost.

Dynamics were also enabled in this example, so the accelerations caused by the contact forces also induced out-of-basis deformations. The dynamics oracle correctly kept the head in full space until the motion had sufficiently damped out (Figure 4.7).

Example	Full Space Time/Frame	Subspace Time/Frame	Best Case Speedup
Capsule	7.95 s	15 ms	**506**×
Hand	13.54 s	100 ms	**135**×
Cheb	3.69 s	55 ms	**67**×

Table 4.1: Performance of full space simulations over the entire mesh, and subspace-only simulations.

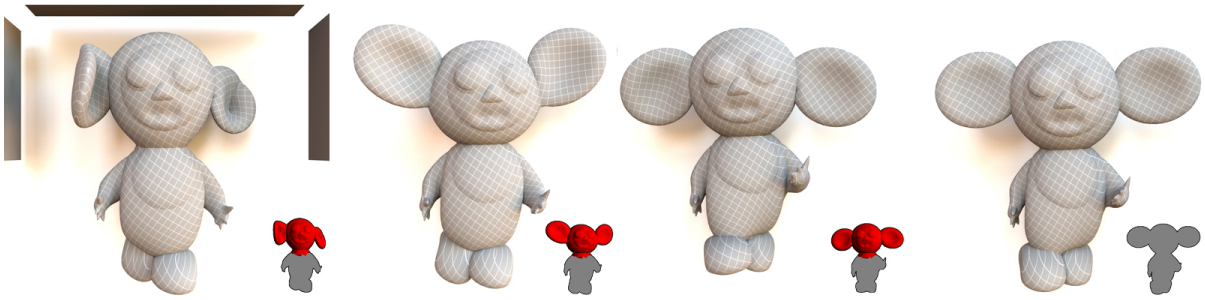


Figure 4.7: In reading order, Cheb's head is simulated in full space (inset, in red) even after the walls are removed, because accelerations still produce out-of-basis deformations. Once the dynamics damp out (lower right), the subspace-only simulation re-activates.

Example	Influence Radius	Average			Worst Case		
		Fullsim %	Time/Frame	Speedup	Fullsim %	Time/Frame	Speedup
Capsule	0.25	18.6%	0.84 s	**9.4**×	50.6%	2.7 s	**3.5**×
	0.08	2.6%	0.11 s	**76**×	10.2%	0.42 s	**22**×
Hand	0.046	1.7%	0.28 s	**48**×	14.3%	1.67 s	**8.1**×
Cheb	0.39	26%	0.77 s	**4.8**×	49.4%	1.90 s	**1.9**×

Table 4.2: Performance of our subspace condensation algorithm. The influence radii assume that the mesh has been normalized to a unit cube. We report the % of vertices that were activated in full space (Fullsim %), the time/frame and the speedup averaged over the entire sequence as well as in the worst, most complex frame.

4.6 Summary

We have presented an efficient algorithm that addresses a key limitation of subspace simulations. When an out-of-basis event is encountered, we activate full space computation on-the-fly in the neighborhood of the event. Our method is fast, generic and applies to non-linear materials. Our main contributions include:

- *Subspace condensation*, a new method that combines the generality of full space deformations with the speed of subspace computations.

- The main bottleneck of condensation methods is a large matrix inverse. We design

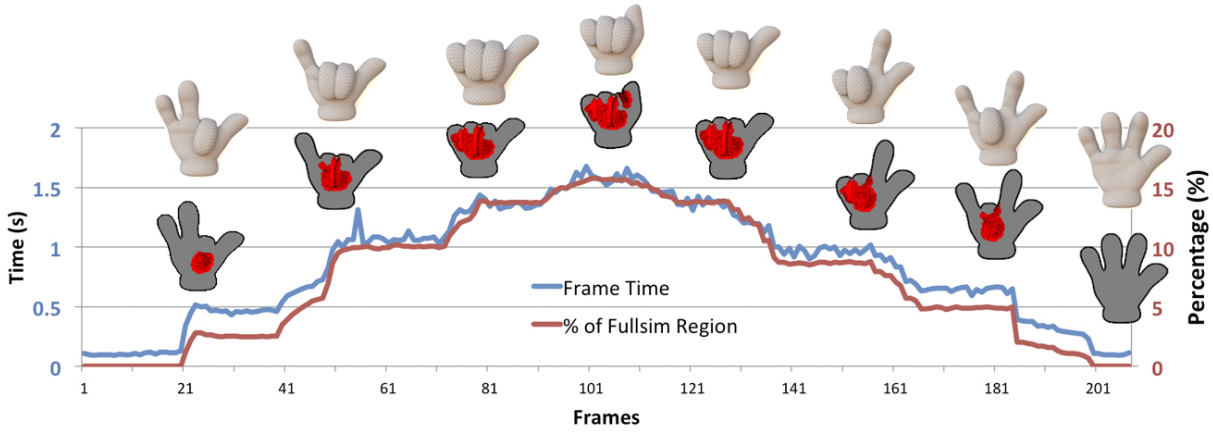


Figure 4.8: Simulation time per frame for a fist clenching and unclenching. The fingers clench in a different sequence from which they unclench, so many novel collision configurations that were not seen during training are encountered. The full space regions are drawn in red in the gray silhouette images. The simulation time is clearly proportional to the size of the active full space.

a solver that sidesteps this problem using subspace coordinates, but still maintains a two-way coupling between the full space and subspace regions.

- Condensation is usually only applicable to linear materials, but the speed of our method allows it to be applied to non-linear materials as well.

- We demonstrate our algorithm on a physics-based skinning application. To this end, we propose several *oracles* that detect the regions where full space computation is needed and where the subspace approximation will suffice, and dynamically partitions the mesh into these regions at every frame.

- By exploiting the forces along the boundary of the full space regions, we show that an efficient, cubature-based method [20, 39] can be obtained for evaluating the forces inside the subspace regions.

Our method allows subspace deformable solids to be interactive rather than just staying in a prescribe motion space. Actually, deformable solids can interact not only

with each other, but also with fluids. In the next chapter, we present an efficient algorithm for solid-fluid coupling.

Chapter 5

Eulerian Solid-Fluid Coupling

Note: A large portion of this chapter has previously appeared as [115].

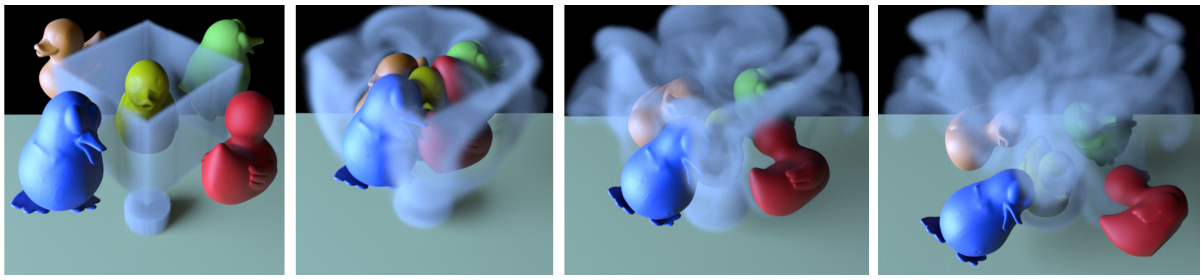


Figure 5.1: "Party" scene: Three hyper-elastic and two elasto-plastic objects are squashed into a complex contact configuration, all while fully two-way coupled with the surrounding fluid. All of the objects and the fluid are represented on a $200 \times 180 \times 200$ Eulerian grid.

Chapter 3 presents a purely subspace approach for efficient articulated self-contact. When the contact scenario gets more complex, we observe that subspace alone becomes insufficient and design an adaptive full space - subspace algorithm (Chapter 4). We now go to the other extreme to handle even more sophisticated systems. We present a new full space method that achieves a two-way coupling between deformable solids and an incompressible fluid where the underlying geometric representation is entirely Eulerian.

Using the recently developed Eulerian Solids approach [5], we are able to simulate multiple solids undergoing complex, frictional contact while simultaneously interacting with a fluid. The complexity of the scenarios we are able to simulate surpasses those that we have seen from any previous method (Figure 5.1). Eulerian Solids have previously been integrated using explicit schemes, but we develop a semi-implicit scheme that allows large time steps to be taken. The incompressibility condition is satisfied in both the solid and the fluid, which has the added benefit of simplifying collision handling.

5.1 Introduction

Two-way solid-fluid coupling produces visually and mechanically distinctive behaviors such as a ball pushing smoke and water away while it simultaneously deforms under the fluid's load. The correct handling of this coupling also leads to realistic behavior such as a bowling ball falling more quickly than a feather due to drag forces. Thus, methods for simulating complex scenes that couple solids and fluids can be critical for generating compelling visual effects and accurately simulating the world around us.

Solid simulation, particularly hyperelastic solids, predominantly uses a Lagrangian representation [116, 117, 118, 119, 120], while single-phase fluids such as smoke are often simulated on an Eulerian grid [121, 122, 123]. As a result, a variety of methods have been developed that attempt to couple these two disparate representations [25, 124] using a suite of numerical techniques and geometric operations.

However, the idea of a unified solver, where the underlying geometry is either entirely Lagrangian or entirely Eulerian, is an appealing one. It removes the need to negotiate between different coordinate systems, and promises to simplify both the design and implementation of the overall algorithms. To date, most attempts at such a solver have been Lagrangian [29, 30, 31, 32], because SPH-like [125, 4] and FLIP-like [72] methods

can be used to make the fluid representation Lagrangian as well.

In this chapter, we take the opposing perspective and explore the coupling of fully Eulerian solids and fluids. This is made possible by the recent work of Levin et al. [5], which presents a method for simulating hyperelastic solids within an Eulerian framework. By incorporating this method into an Eulerian fluid solver, we are able to resolve complex contact scenarios between multiple solids and a single-phase fluid (Figure 5.1). In order to enable large timesteps in the presence of difficult contact configurations, we present a semi-implicit method for stepping the system forward in time. As in the cases of cloth and hair [126, 127], we empirically observe that the incompressibility of the fluid naturally assists in collision handling. The final algorithm does not require the generation or maintenance (i.e. remeshing) of a well-conditioned tetrahedral mesh, and maintains the robustness and ease of use of a purely Eulerian technique. Our method is able to simulate complex collision scenarios between solids and fluids that we have not seen from any previous approach.

5.2 Related Work

Beginning with the immersed boundary method [128], simulating the coupled motion of solids and fluids has a long history in both graphics and engineering. In graphics, there has been much work coupling fluids to rigid bodies [129, 130, 131, 132], as well as rigid and deformable shells [133].

Chentanez et al. [25] modeled the deformable solid as an unstructured tetrahedral mesh and showed how to couple it to an Eulerian fluid, which could be represented as either a regular grid or another unstructured mesh. This approach requires a mesh generation stage, and the specific formulation required an asymmetric system to be solved. This approach has been extended to fully Lagrangian simulations of both the solid and

fluid [134, 135, 136], and has incorporated additional phenomena such as phase transitions [31] and porous flow [137]. Other methods have further investigated Eulerian fluid discretizations and used sophisticated geometric operations [124, 26] as well as overlapping grids [138] to couple the grid velocities to a Lagrangian solid. Fast, approximate, position-based methods have also been recently developed for real-time applications [32], which can often need careful parameter tuning to generate realistic results.

Recently, the Material Point Method (MPM) [71, 72] has become popular for simulating a number of mixed-phase phenomena. It shares some of the same advantages of our approach, as it avoids the need for complex remeshing schemes and geometric conversions. However, as mentioned by Jiang et al. [139], these schemes are known to have issues representing hyperelastic materials, as artificial plasticity can creep into the simulation. Our scheme naturally handles hyperelastic response, even in the presence of a fluid, and still allows the user to add plasticity if desired.

In order to avoid complicated meshing schemes, simulate elastic objects accurately, and robustly resolve complicated collisions, Levin et al. developed the Eulerian Solids methodology [5]. With this technique in hand, it is natural to ask whether we can now perform solid-fluid coupling in a purely Eulerian fashion. The closest work to ours in the engineering literature is Kamrin et al. [140], which showed that a similar "reference map" method can be used to couple deformable, elastic solids to weakly compressible fluids. The approach has also been extended to handle non-frictional contact between two objects [141]. However, this method does not handle incompressible fluids, large time steps or complex contacts between a multitude of objects. Crucially, their efficacy has also only been demonstrated on coarse 2D grids. In contrast, we present a fully 3D Eulerian Solids-based solver that couples an incompressible fluid to multiple deformable objects undergoing frictional contact. By using an implicit time integration scheme, we are able to take large timesteps.

5.3 Coupled Solid–Fluid Simulation

Notation: Our notation is coherent with the rest of this dissertation. Additionally, we use a $\star$ superscript to denote intermediate, before-advection state. A superscript to the left of a variable is used to distinguish solid objects from fluid. Unlabelled vectors are considered global, i.e. they contain both solid and fluid entries.

5.3.1 The Continuous Formulation

In this work we focus on the coupled simulation of multiple incompressible, hyper-elastic, and elasto-plastic solids immersed in an incompressible fluid. This requires us to solve the momentum equation, given by

$$\left.\begin{aligned} \rho\frac{d\mathbf{v}}{dt} &= \nabla\cdot\sigma+\mathbf{f} \\ \nabla\cdot\mathbf{v} &= \mathbf{0} \end{aligned}\right\} \forall\mathbf{x}\in\Omega \qquad\qquad {}^{f}\mathbf{v} = {}^{s}\mathbf{v} \quad \forall\mathbf{x}\in\Gamma \tag{5.1}$$

where Ω denotes a region in world space, $\mathbf{v}$ is the velocity of a particle at $\mathbf{x}$, σ is the Cauchy stress and $\mathbf{f}$ are external forces such as gravity. For each $\mathbf{x}$ containing a solid, we compute σ using a standard hyper-elastic or elasto-plastic constitutive model, and for each $\mathbf{x}$ containing a fluid we set $\sigma = \mathbf{0}$. For these cells, the divergence-free condition, $\nabla\cdot\mathbf{v} = \mathbf{0}$, is sufficient. We also enforce a no slip condition along the solid-fluid boundary Γ, where fluid and solid velocities are respectively denoted ${}^{f}\mathbf{v}$ and ${}^{s}\mathbf{v}$. We use the hyper-elastic model of McAdams et al. [11] for the elastic component of all of our examples.

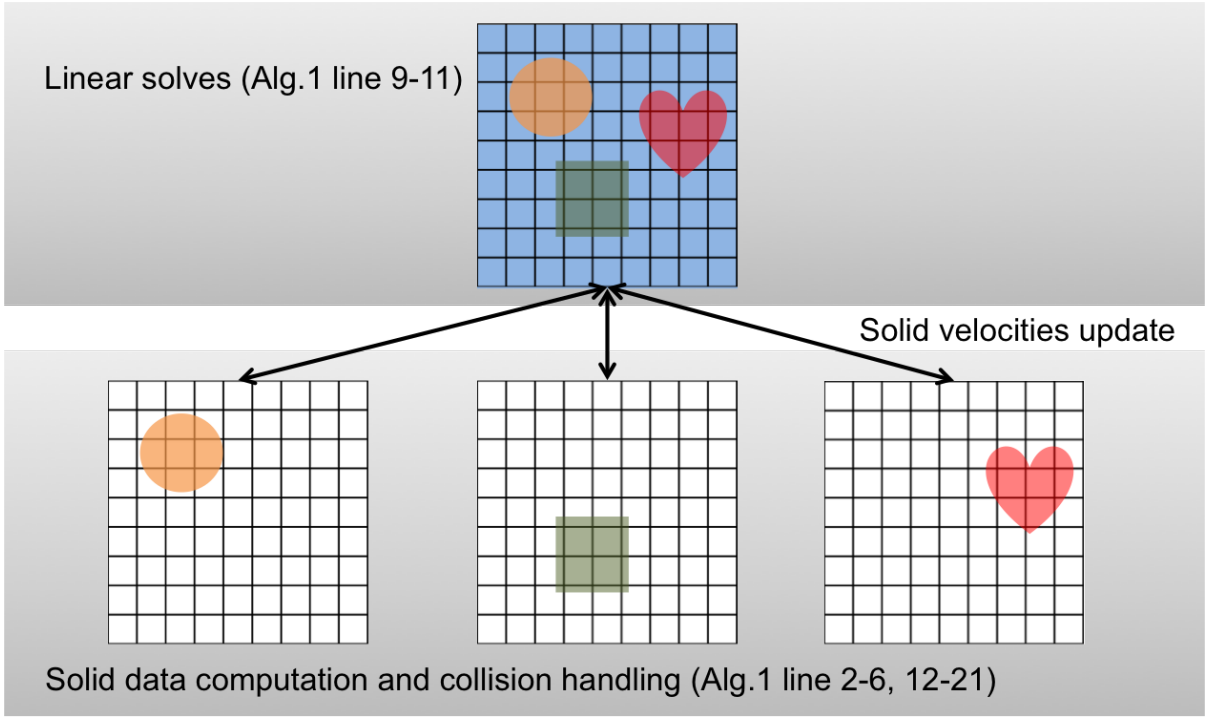


Figure 5.2: The high-level structure of our data storage and computation scheme. To assist advection and contact handling, we keep separate velocity fields for each solid object. For efficiency, only values near the solid are updated.

5.3.2 Spatial Discretization and Constraints

Our method relies on fixed discretizations of both $\bar{\mathbf{x}}$ and $\mathbf{x}$. In order to solve Equation 5.1, we discretize Ω using regular, hexahedral finite elements. Velocity, displacement and forces are co-located at the grid nodes, while pressure values is stored at grid centers. In order to incorporate equality constraints we rely on a mixed formulation in which incompressibility is applied as a point constraint at the cell center. This can be considered an under-integrated finite element, which is commonly used to prevent locking [142]. We then compute per-element mass and stiffness matrices, based on whether each cell contains a solid or a fluid, using an eight-point quadrature rule, and then assemble into global $\mathbf{M}$ and $\mathbf{K}$ operators. Our discretized divergence-free constraint is expressed as

$\mathbf{Jv} = \mathbf{0}$ where $\mathbf{J}$ is the constant constraint gradient. Note that due to the continuity of the velocity field, the no slip condition on solid-fluid boundaries is implicitly enforced, and no special spatial coupling terms need to be formulated.

To facilitate velocity advection and collision detection, each solid stores a copy of the velocity field, but only values near the solid are ever updated. Fig. 5.2 shows our high level data storage and computation structure. Our algorithm also requires a discretization of $\bar{\mathbf{x}}$ if plastic deformation is desired. For each plastic solid, we create an auxiliary grid of that solid's reference coordinate system ${}^{l}\bar{\mathbf{x}}$, where $l \in [1, N_s]$ indexes each solid in the simulation.

5.3.3 Time Integration

We use a splitting scheme to advance our system in time. First, we use implicit integration to compute a divergence-free velocity field for the solid cells, and then perform an advection that resolves collisions. Algorithm 5 gives an overview of our time integration scheme. In the next sections, we describe the key components of our algorithm: a semi-implicit update for Eulerian Solids, pressure projection, and a collision resolution scheme.

5.3.4 Semi-implicit Update

The original Eulerian Solids scheme [5] used explicit force integration to compute the velocity field, followed by a first-order finite difference scheme for advection. Both of these design decisions resulted in small time steps. While Fan et al. [73] introduced Lagrangian modes on top of the Eulerian motion in order to reduce this restriction, we seek to ease it in a way that maintains the convenience of a single spatial discretization. First, we replace the explicit force integration with a semi-implicit scheme that is similar

Algorithm 5 Eulerian solids and fluids simulation

Compute Δt based on CFL condition
for each solid object, l **do**
    Compute mass ${}^{l}\mathbf{M}$ and volume fraction ${}^{l}\mathbf{V}$ on the grid
    Compute material force ${}^{l}\mathbf{f}$ and stiffness matrix ${}^{l}\mathbf{K}$
    ▷ (§5.3.4)
end for
Compute fluid mass ${}^{f}\mathbf{M}$
Assemble $\mathbf{M}$, $\mathbf{K}$, $\mathbf{C}$ and $\mathbf{f}^{\star}$ ▷ (§5.3.4)
$\mathbf{v}^{\star} = \mathbf{G}^{-1}\mathbf{f}^{\star}$ ▷ (Eqn. 5.3)
Compute pressure $\mathbf{p}$ ▷ (Eqn. 5.7)
Pressure project $\mathbf{v}^{\star}$ to get pre-advection $\mathbf{v}^{n+1}$ ▷ (Eqn. 5.8)
for each solid object, l **do**
    Update solid particle velocities using FLIP from ${}^{l}\mathbf{v}^{n+1}$
    Add repulsions and frictions to particles in collision
    ▷ (§5.3.6)
end for
for each solid object, l **do**
    Rasterize particle velocities to get final velocity ${}^{l}\mathbf{v}^{n+1}$
    Update displacement ${}^{l}\mathbf{u} = {}^{l}\mathbf{u}^{n} + \Delta t\,{}^{l}\mathbf{v}^{n+1}$
    Semi-Lagrangian advect ${}^{l}\mathbf{u}$ to get ${}^{l}\mathbf{u}^{n+1}$
end for
Semi-Lagrangian advect fluid velocity ${}^{f}\mathbf{v}^{n+1}$

to that of Stomakhin et al. [143].

We denote the change in velocity at each grid node, due to internal forces $\mathbf{f}_{\text{int}}$, over the time interval $[t, t + \Delta t]$ as:

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\ \mathbf{f}_{\text{int}}\left(\mathbf{u}^{n+1}\right).$$

A standard Taylor expansion around $\mathbf{x}$ yields,

$$\mathbf{f}(\mathbf{u}^{n+1}) = \mathbf{f}_{\text{int}}(\mathbf{u}^n + \Delta t\mathbf{v}^{n+1}) \approx \mathbf{f}_{\text{int}}^n + \frac{\partial \mathbf{f}_{\text{int}}^n}{\partial \mathbf{x}}\Delta t\ \mathbf{v}^{n+1}, \tag{5.2}$$

which we can further abbreviate to $\mathbf{f}(\mathbf{u}^{n+1}) = \mathbf{f}_{\text{int}}^n + \mathbf{K}\Delta t\mathbf{v}^{n+1}$. By combining this with a first order discretization of acceleration, $\mathbf{a}^\star = (\mathbf{v}^{n+1} - \mathbf{v}^n)/\Delta t$, and the equations of motion for a deformable solid, $\mathbf{M}\mathbf{a}^\star + \mathbf{C}\mathbf{v}^\star + \mathbf{f}_{\text{int}}^\star = \mathbf{f}_{\text{ext}}$, we obtain the semi-implicit update equation:

$$(\mathbf{M} + \Delta t\mathbf{C} + \Delta t^2\mathbf{K})\mathbf{v}^\star = \mathbf{M}\mathbf{v}^n + \Delta t(\mathbf{f}_{\text{ext}} - \mathbf{f}_{\text{int}}^n). \tag{5.3}$$

Here, $\mathbf{C}$ is a Rayleigh damping matrix. The external force term $\mathbf{f}_{\text{ext}}$ includes body, buoyancy and vorticity forces. For fluid-only cells, $\mathbf{C}$ and $\mathbf{K}$ disappear and only the diagonal mass matrix $\mathbf{M}$ remains. Therefore, the system can be solved efficiently if the simulation domain is dominated by a fluid.

Note that there is no advective term in the stiffness matrix. In a purely Eulerian sense, the force arises from a chain of variables: $\mathbf{f}_{\text{int}}\left(\mathbf{F}\left(\mathbf{u}(\mathbf{x}(t), t)\right)\right)$. The derivative should then be:

$$\frac{\partial \mathbf{f}_{\text{int}}\left(\mathbf{F}\left(\mathbf{u}(\mathbf{x}(t), t)\right)\right)}{\partial t} = \frac{\partial \mathbf{f}_{\text{int}}}{\partial \mathbf{F}}\frac{\partial \mathbf{F}}{\partial \mathbf{u}}\left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial t} + \frac{\partial \mathbf{u}}{\partial t}\right). \tag{5.4}$$

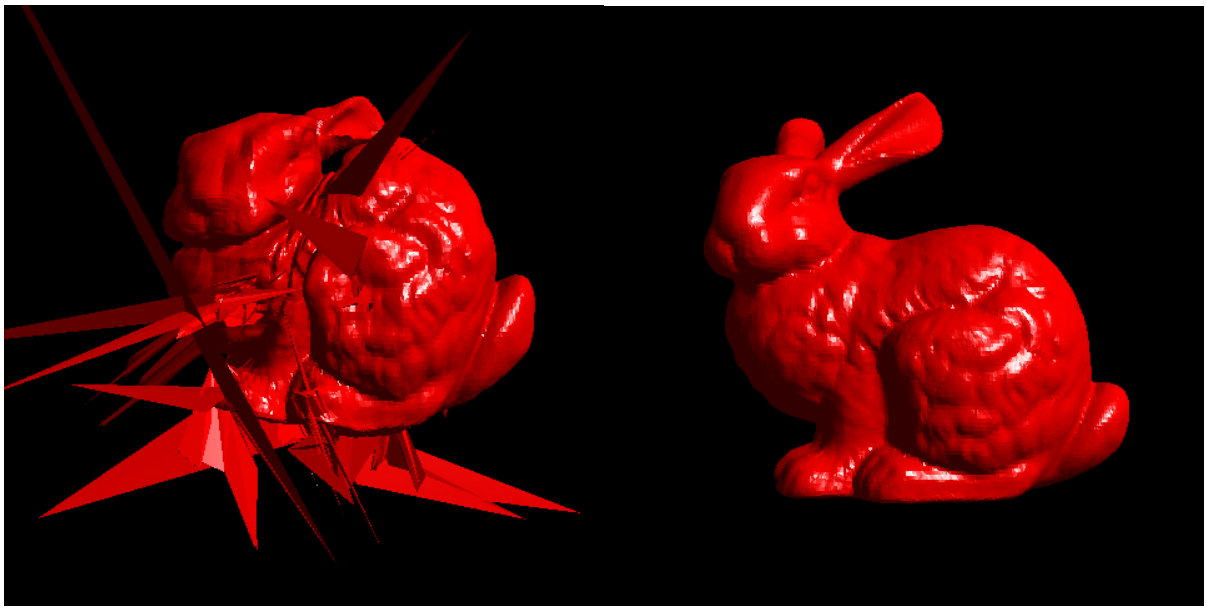
The $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v} \cdot \nabla \mathbf{u}$ appears to introduce an advective term, but we can observe that $\left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial t} + \frac{\partial \mathbf{u}}{\partial t}\right) = \frac{D\mathbf{u}}{Dt}$, i.e. the total derivative of $\mathbf{u}$. This then reduces to the Lagrangian

case,

$$\frac{\partial \mathbf{f}_{\text{int}}\left(\mathbf{F}\left(\mathbf{u}(t)\right)\right)}{\partial t} = \frac{\partial \mathbf{f}_{\text{int}}}{\partial \mathbf{F}} \frac{\partial \mathbf{F}}{\partial \mathbf{u}} \frac{D\mathbf{u}}{Dt}, \tag{5.5}$$

Taking a perspective similar to Stomakhin et al. [143] that the nodes of the Eulerian mesh are fictitiously deforming in a Lagrangian manner, the Lagrangian $\mathbf{K}$ in Equation 5.3 suffices.

Our semi-implicit integration scheme can handle large time steps under severe deformations. In Figure 5.3 we initially squished a bunny by half. We did not respect the CFL condition and set $\Delta t = \frac{1}{24}$. Explicit integration blows up almost immediately, while semi-implicit integration correctly returns the bunny to its rest shape.



(a) (b)

Figure 5.3: We scale the bunny by half and let it expand. Using $\Delta t = \frac{1}{24}$, explicit integration blew up after 4 frames while our semi-implicit scheme is extremely stable.

5.3.5 Incompressibility Constraints

We enforce incompressibility constraints using a primal-dual algorithm. First, we form the Karush-Kuhn-Tucker (KKT) system prescribed by our semi-implicit scheme (§5.3.4),

$$\begin{bmatrix} \mathbf{G} & \mathbf{J}^T \\ \mathbf{J} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v}^{n+1} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f}^\star \\ \mathbf{b} \end{bmatrix} \tag{5.6}$$

where $\mathbf{G} = \mathbf{M} + \Delta t\mathbf{C} + \Delta t^2\mathbf{K}$, $\mathbf{f}^\star = \mathbf{M}\mathbf{v}^n + \Delta t(\mathbf{f}_{\text{ext}} - \mathbf{f}_{\text{int}}^n)$ and $\mathbf{b}$ contains the boundary conditions. We first solve $\mathbf{G}\mathbf{v}^\star = \mathbf{f}^\star$ for the unconstrained velocity $\mathbf{v}^\star$ and then solve the dual problem to eliminate the divergent part of the velocity field. We replace $\mathbf{G}^{-1}$ with $\mathbf{M}^{-1}$ in the pressure solve to avoid an expensive matrix inversion:

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{p} = \mathbf{J}\mathbf{v}^\star - \mathbf{b}. \tag{5.7}$$

Finally, we correct $\mathbf{v}^\star$ to get the pre-advection velocity field $\mathbf{v}^{n+1}$:

$$\mathbf{v}^{n+1} = \mathbf{v}^\star - \mathbf{G}^{-1}(\mathbf{J}^T\mathbf{p}). \tag{5.8}$$

The substitution in Eqn. 5.7 is can be interpreted in two ways. First, if all the cells contain fluid, the Schur complement in Eqn. 5.7 naturally yields $\mathbf{M}^{-1}$. Thus, we can interpret this substitution as momentarily approximating the solid cells as fluid. Second, if the solid cells are integrated explicitly, Eqn. 5.7 again yields $\mathbf{M}^{-1}$, as the material forces still appear on the right hand side. So, we can interpret the substitution as only integrating the *volume* terms implicitly, while treating the solid strain energies explicitly.

We also attempted to solve the KKT system (Eqn. 5.6) directly, but initial test showed that our primal-dual version ran over 3× faster in 2D. The investigation of more sophisticated solution methods for this problem is left as future work.

Complexity compared to explicit integration: In previous work, [5, 73], two quadratic problems of the same form as Equation 5.6 were solved to determine the time step size. In our formulation, we instead solve three linear systems (Eqs. 5.3, 5.7 and 5.8). In our experiments, we found that the increase in time step size far outweighed the cost of this additional linear solve. Thus, we are able to compute a large, implicit step at a cost that is proportional to a small, explicit step.

5.3.6 Contact and Collision Response for Solids

As noted in previous work [126, 127], the presence of divergence-free constraints help to maintain a collision-free state. However, some collision handling is still needed to avoid solids from "sticking" if the advection stage introduces overlaps. In order to address this, we apply the repulsion forces of Bridson et al. [144] during our advection.

For this stage, it is necessary to employ an auxiliary Lagrangian variable. While this seems slightly at odds with the goal of a fully Eulerian simulation, our underlying geometric representation remains Eulerian. Like the Eulerian grid projection stage of the Lagrangian FLIP method [77], or the semi-Lagrangian particle traces of grid-based Stable Fluids [74], we leverage the advantages of the other coordinate system during time integration without fully commiting to the representation.

Collision resolution begins by copying each solid velocity $\mathbf{v}^{n+1}$ from our spatial grid to the individual solid grids ${}^{l}\mathbf{v}^{n+1}$, where l indexes each solid in the scene. Next, we instantiate particles for each solid, using 8-16 particles per cell. In order to avoid collisions we check the distance between the initial particles of one solid against all of the other solids. If it is closer than a distance h (typically the grid resolution) the normal velocity

of the particle is modified by an impulse r, defined as:

$$r = -\min\left(\Delta t\ k\ d, m\left(\frac{0.1d}{\Delta t} - v_N\right)\right). \tag{5.9}$$

Here, d is the overlap distance, k is a spring stiffness, m is the mass of the particle and v_N is the relative velocity in the direction of the contact normal. The change of the particle velocity in the normal direction is then defined as $\Delta v_N = r/m$. Friction can also be applied by modifying the relative tangential velocity:

$$\mathbf{v}_T = \max\left(1 - \mu\frac{\Delta v_N}{|\mathbf{v}_T^{\text{pre}}|}, 0\right)\mathbf{v}_T^{\text{pre}}, \tag{5.10}$$

where $\mathbf{v}_T^{pre}$ is the pre-friction relative tangential velocity. The values of k and μ we used are listed in Table 5.1.

Computing the contact normal: Each solid object has an embedded surface mesh and a signed distance field ϕ defined in its material domain. The mesh is advected *passively* in the same way as Fan et al. [73]. When checking for the collision of material particle s of solid i against solid j, we interpolate the material position field ${}^j\bar{\mathbf{x}}$ and lookup ${}^j\phi$. A repulsion is added if the overlap $d = h - {}^j\phi({}^j\bar{\mathbf{x}}({}^i\mathbf{p}_s)) > 0$. We find the nearest surface element to ${}^j\bar{\mathbf{x}}({}^i\mathbf{p}_s)$ and use its world space normal as the contact normal. If a surface mesh is not available, the gradient of the background volume grid can be used to compute the normal [5].

5.4 Implementation and Results

We solved Equation 5.3, 5.7 and 5.8 using Preconditioned Conjugate Residuals (PCR) with a Jacobi preconditioner because the matrices are semi-definite. Warm starting was used when solving the pressure (Equation 5.7). We use the Eigen [114] library for linear

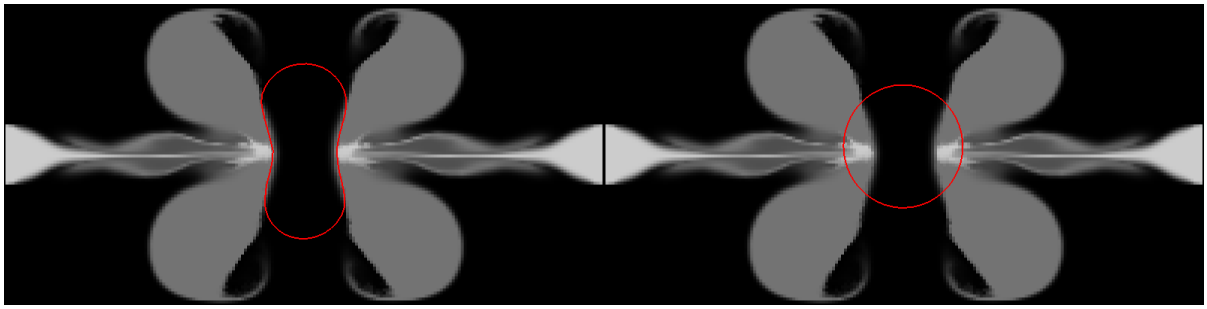
algebra routines. All simulations were run on a 8-core, 3GHz MacPro with 32 GB of RAM using 16 threads. An advantage of Eulerian simulation is that most stages can be embarrassingly parallelized, so OpenMP was used whenever possible. Both our 2D and 3D examples used the co-rotational material from McAdams et al. [11]. Tables 5.2 and 5.3 shows a performance summary of our 3D examples and Table 5.1 shows their simulation parameters. We chose a high contact spring stiffness for more bouncy contact and lower value for dampened contact. We used a PCR threshold of 0.02 for the pressure solve (Equation 5.7) in all the examples. For the two velocity solves (Eqs. 5.3 and 5.8) we used a PCR threshold of $1e^{-4}$ for Cheb and $1e^{-3}$ for the others.

5.4.1 Simulating a Single Solid with a Fluid

In all of the following examples, we found that the linear solves, particularly the pressure solve, consumed the largest fraction of the running time (Table 5.3). The collision assistance provided by the divergence-free constraint can also be seen in the timings, as very little time needs to be spent in collision resolution.

Elasticity validation: Figure 5.4a shows two jets compressing an elastic circle. External forces, including fluid forces, are then removed and the simulation of the circle continues in isolation. The circle correctly returns to its reference configuration, which is a hallmark of the Eulerian Solids approach (Figure 5.4b).

Ball: We dropped a heavy elastic ball to a carpet of smoke. The only external forces applied to the fluids are gravity and vorticity confinement of Fedkiw et al. [145]. The smoke gets pushed away when the ball hits the flow. As the ball rebounds, a plume of smoke is drawn up by the low-pressure eddy formed in its wake. The simulation is inside a box with Dirichlet boundary conditions, so four vortices form along the four quadrants of the x-z plane (Fig. 5.5).



(a) (b)

Figure 5.4: (a) A elastic circle deforms under the influence of two jets. (b) We removed all external forces and simulated the circle by itself. The circle is able to return to its rest shape.

Buoyancy: Figure 5.6 illustrates that solids with different densities behave differently when interacting with the fluid. The heavy bunny falls at the rate of gravity and the light bunny falls much slower due to drag forces. A buoyant jet is used to push the light bunny to the ceiling while the heavy bunny remains on the ground.

Cheb's Glaucoma Test: For plastic deformation, we use a multiplicative plasticity model [70]. The plastic deformation gradients are stored in a material space grid, initially set to identity, and updated according to a yield condition (Section 2.4). We take Cheb to an eye exam where a high pressure puff of air is shot at his head. Figure 5.7c and 5.7d shows the comparison between elastic and plastic deformation. In the supplemental video[1], we show that by varying whether the feet are constrained, different motions are obtained.

5.4.2 Simulating Multiple Solids with a Fluid

Collision Response: In the supplement video[1] we show two 2D circles moving towards each other. By adding collision response to the solid particles, the circles are able to separate after collision. Without this response, the circles stick.

[1]https://youtu.be/jobwwpLriDQ

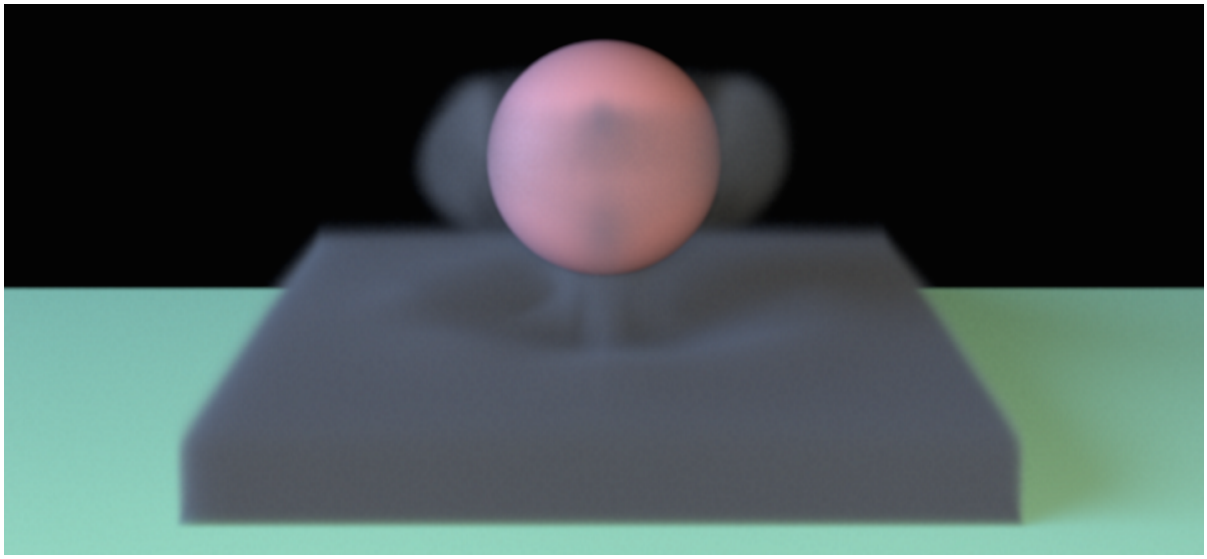


Figure 5.5: A plume rises as an elastically deforming ball bounces up from a smoky floor.

Example	Solid-fluid density ratios	k spring stiffness	μ friction coefficient
Ball	1000:1	10000	0.5
Bunny (light)	1.3:1	2000	0.2
Bunny (heavy)	1000:1	2000	0.5
Cheb	1.3:1	2000	0.5
Party	1000:1, 500:1	10000	0.2

Table 5.1: Simulation parameters used for each example.

"Party": We simulate multiple solids with various densities and material parameters interacting with each other and the fluid (Figure 5.1). Figure 5.9 shows the time steps used throughout the simulation. We use a CFL number of 0.6 when two solids are less than 2 grid cells apart and 1.8 otherwise. Some small time steps occur after second 6 in order to complete the current render frame, not due to the CFL number. Smoke and side walls are not rendered so that the solid and fluid motions can be seen more clearly. Figure 5.8 shows the final shape of each solid. Note the extreme deformations such as

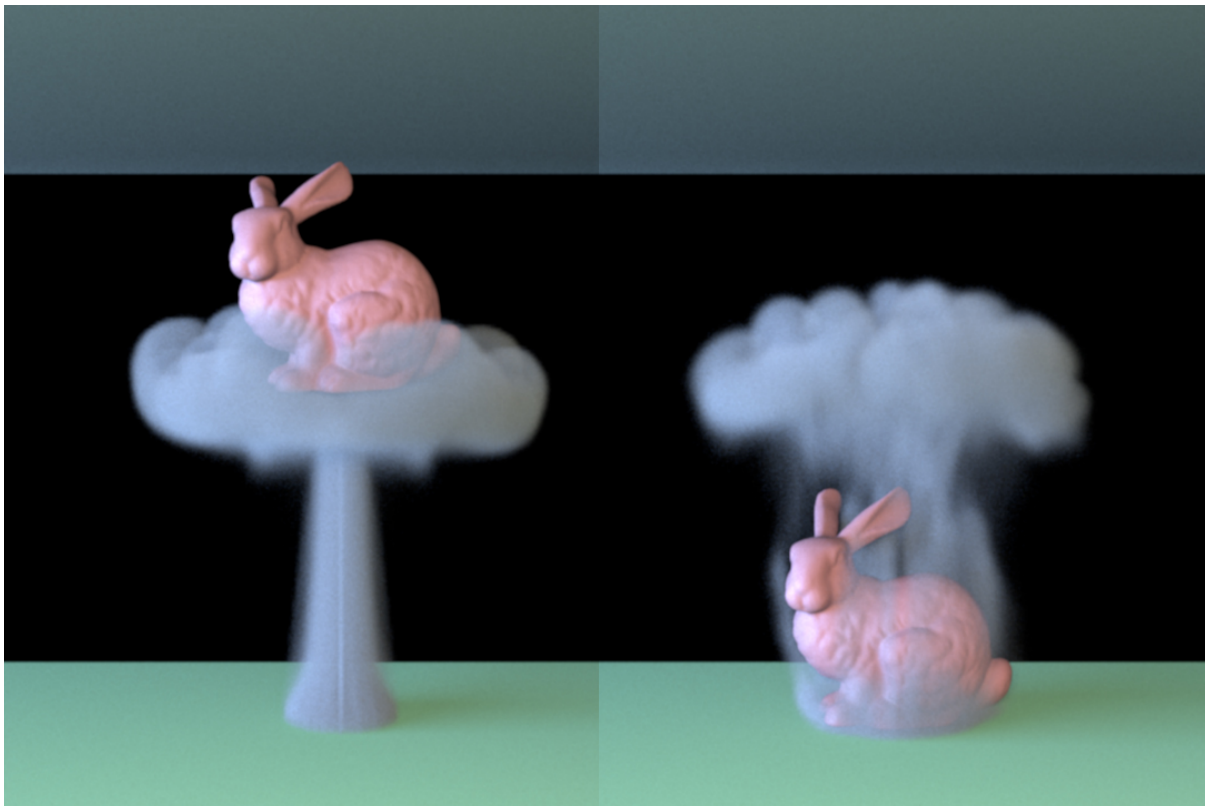


Figure 5.6: Different solid densities behave differently under buoyant flow. When the solid-fluid density ratio is 1.3:1 (left), the smoke plumes causes the bunny to rise like a balloon. When the ratio is 1000:1 (right), the bunny drops like a rock.

the plastic duck's wing being compressed to be flush with its body.

Due to the presence of multiple objects, the timing breakdown differs from that of the other examples. More time is spent in advection and constructing $\mathbf{G}$. This is because a $\mathbf{K}$ matrix must be constructed for each solid, and each must also be converted to FLIP particles and then re-rasterized to the grid. Our current implementation parallelizes these operations internally for each solid object; it does not create a separate thread for each solid. Therefore, many opportunities for further accelerating these operations still remain.

Example	Grid Dimensions	Avg. timestep	Min. timestep	Avg. time/frame
Ball	$120 \times 140 \times 120$	0.025	0.0016	6.54s
Light Bunny	$144 \times 200 \times 144$	0.0369	0.0064	17.4s
Heavy Bunny	$144 \times 200 \times 144$	0.0236	0.00058	16.9s
Cheb	$120 \times 100 \times 80$	0.021	0.0023	6.97s
Party	$200 \times 180 \times 200$	0.0133	0.00083	54.1s

Table 5.2: For each example, the size of the spatial grid, average / minimum simulation time step sizes and average per-frame simulation times are reported.

Example	Compute $\mathbf{G}$ (line 8)	Velocity solves (lines 9, 11)	Pressure solve (line 10)	Advection (lines 13, 18)	Collision (line 14)
Ball	0.84	0.12	2.83	1.12	0.01
Light Bunny	1.32	5.02	6.21	1.72	0.016
Heavy Bunny	1.35	2.47	8.31	1.68	0.016
Cheb	1.55	1.10	1.10	1.37	0.01
Party	13.6	6.72	12.1	14.46	0.25

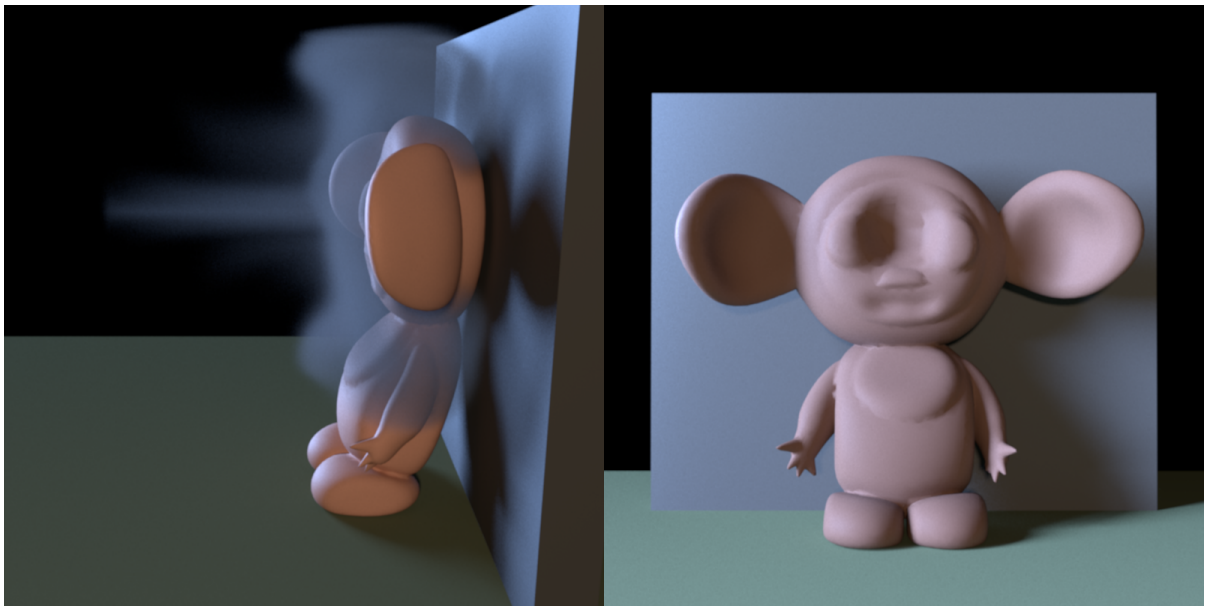
Table 5.3: The computation times of key stages of Algorithm 5. All timings are reported in seconds.

5.5 Summary

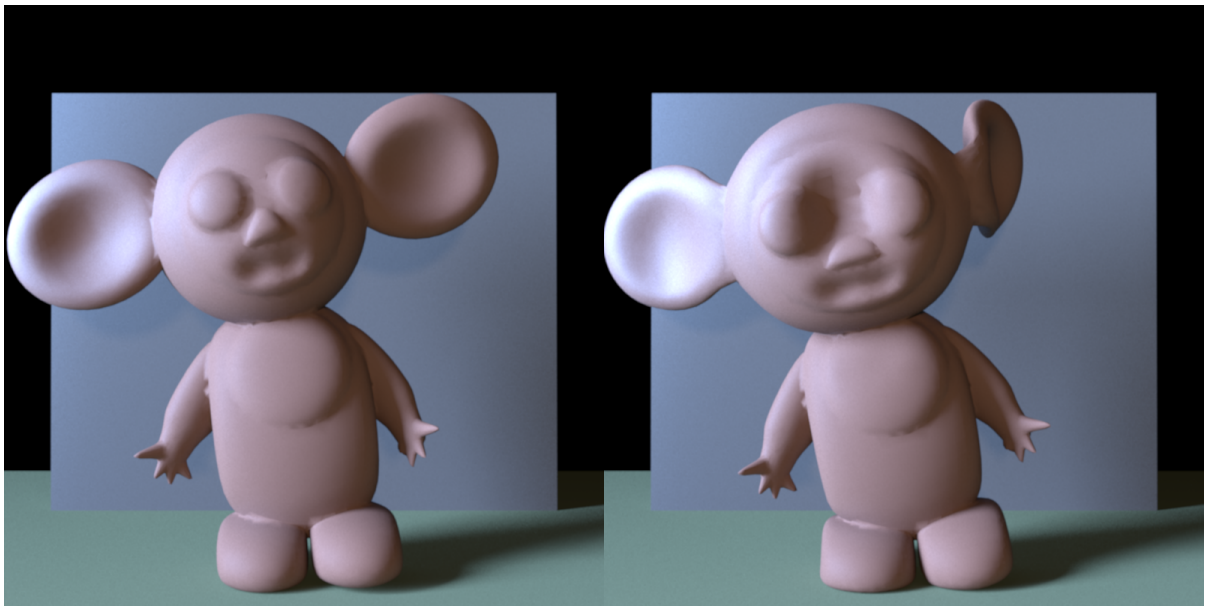
In this chapter we have shown how to simulate a two-way coupling between solids and a fluid where the underlying representation is entirely Eulerian. This allows us to generate simulations that feature large deformations and frictional contact, all the while capturing visually interesting fluid effects. We believe our method produces examples that are more complex than previous approaches and avoids the complexities of Lagrangian, mesh-based simulation. Our work makes the following technical contributions:

- A unified, Eulerian framework for simulating fully coupled fluids and deformable solids

- A semi-implicit solver for Eulerian Solids
- A method for satisfying incompressibility for both the solid and fluid regions of the simulation
- A collision resolution scheme for multibody frictional contact



(a) (b)



(c) (d)

Figure 5.7: (a) Cheb has a high pressure puff of air shot at his head. (b) Same frame, viewed from the front, with smoke removed to make the deformation more visible. (c) Final frame, elastic deformation. (d) Final frame, plastic deformation. Note that the dent in his head persists, as well as the deformations to his ears.

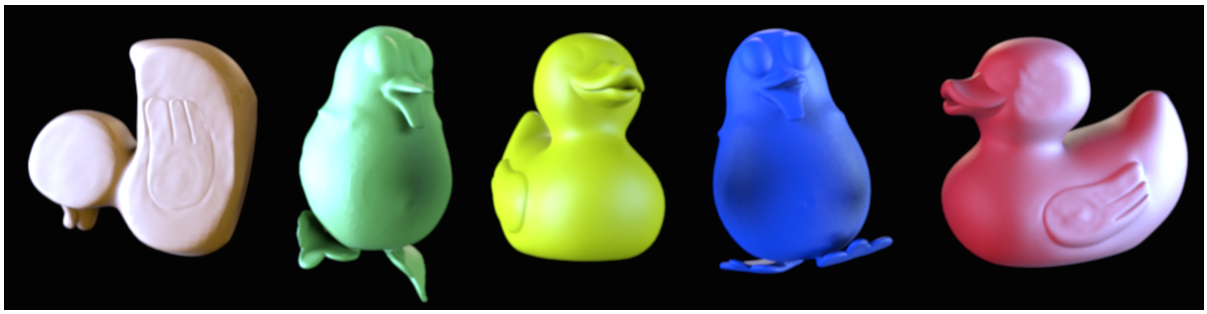


Figure 5.8: The final shapes of party members. The duck and penguin on the left are plastic while the other three are elastic.

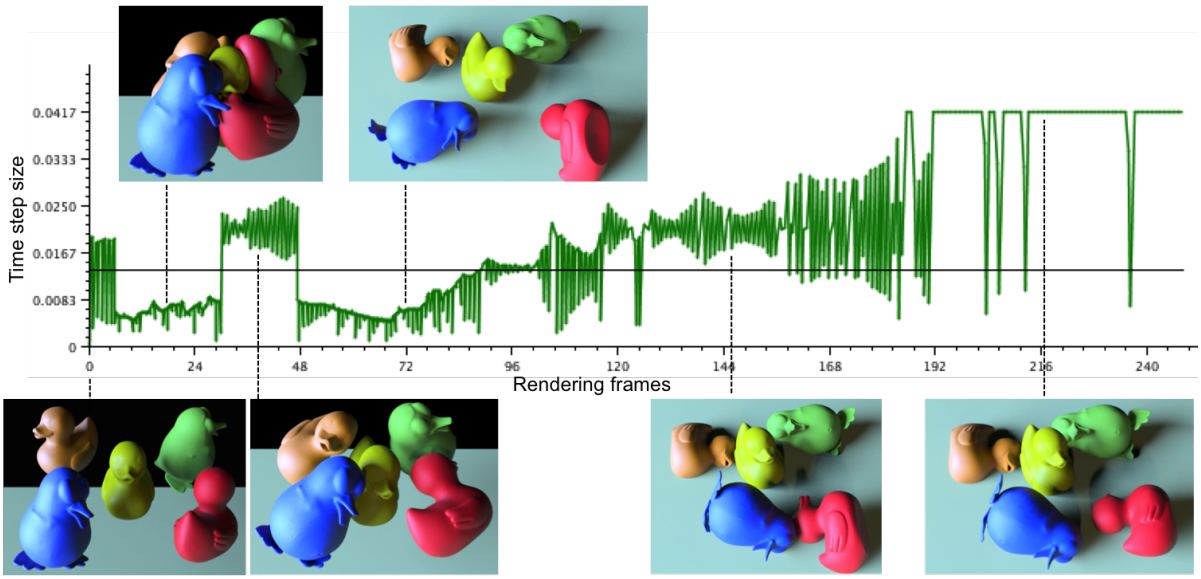


Figure 5.9: Simulation timestep sizes as the party progresses. The rendering rate is fixed at $\Delta t = \frac{1}{24}$ throughout. The horizontal black line denote the average timestep during the simulation. Frames from events corresponding to significant changes in the timestep size are shown. From left to right: the solids are given an initial impulse, they collide and form complex contacts, they fall to the floor, bounce on the floor, and finally come to rest.

Chapter 6

Conclusion

In this dissertation, we have presented several methods for efficient deformable object simulation. Options along the 2D spectrum of subspace space to full space and Lagrangian to Eulerian coordinate systems were explored of different levels of complexity (Figure 6.1). Our proposed techniques have been verified with experimental results in character animation, articulated self-contact, arbitrary external collisions, and solid-fluid coupling. These problems are crucial to human simulations. Solving them efficiently will get us one step closer to real-time applications such as interactive virtual humans.

6.1 Summary of Results

We have presented a pose-space cubature approach in the context of articulated subspace self-contact. Collision detection and resolution could easily become a computational bottleneck, especially in the setting of subspace simulation, where the other stages have been sufficiently accelerated. Since articulated self-contact is highly structured, it possesses precisely the type of coherent behavior that subspace methods excel at exploiting. The original cubature approach [20] showed superior performance for approximating

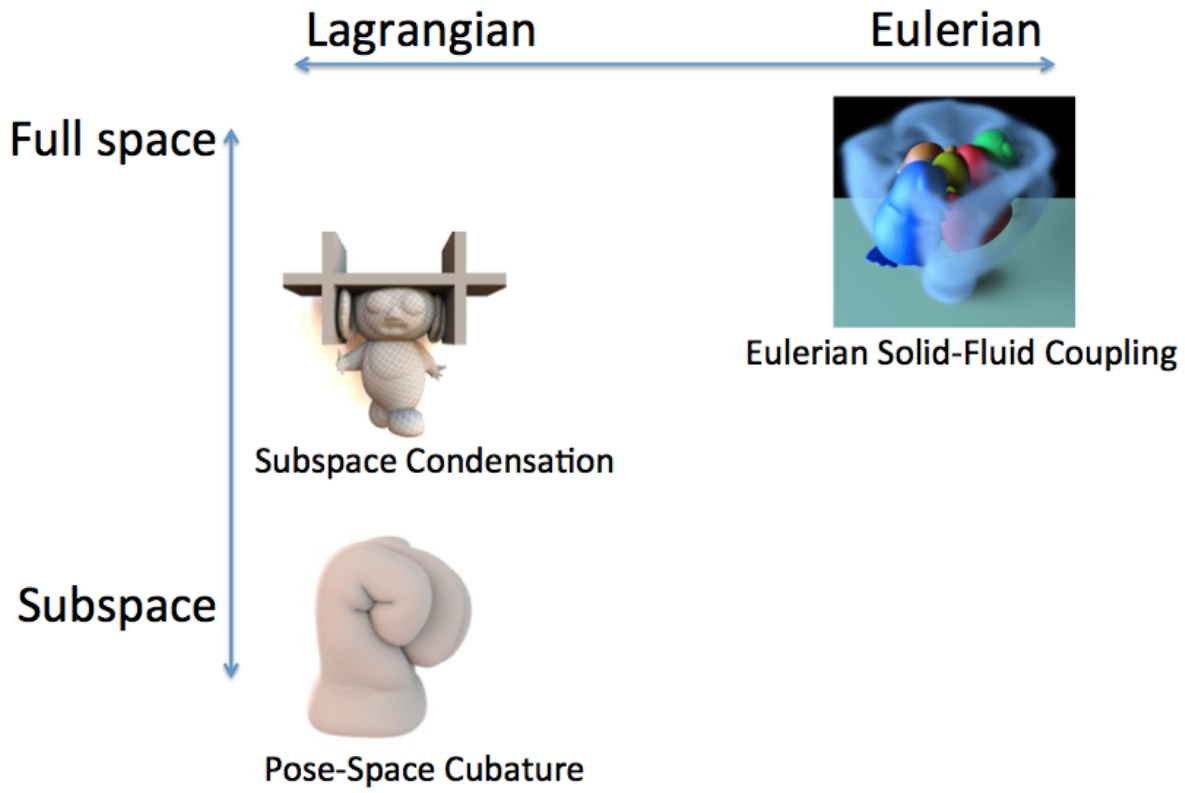


Figure 6.1: We achieve efficient deformation by sampling 3 points on this 2D spectrum of coordinate systems.

subspace non-linear material forces and Jacobians through sparse point sampling. Given sufficient amount of training data, a single set of cubature points worked well on a wide range of deformations. Unfortunately, this is no longer the case for contact forces. First, contact forces are highly non-linear (only C^0 continuous). Second, the training matrix that contains all the data is inevitably sparse, since each collision point is only active in a few frames. Our analysis shows that the above two conditions make cubature training inefficient and a dense cubature set is produced. Based on these observations, we designed the pose-space cubature scheme that trains cubature points in a more localized fashion and associates them with a pose-space lookup. Our algorithm accelerates self-contact by up to an order of magnitude over other subspace simulations, and accelerates the overall

simulation by two orders of magnitude over full-rank simulations.

The pose-space cubature approach accelerates the computation of subspace non-linear terms. But a subspace method itself could fail when the basis become insufficient. This could happen when, for example, the solid undergoes arbitrary external collisions. For a basis to include all possible deformations, its rank would approach the original full space degrees of freedom and there would be no performance gain at all. To overcome this limitation, we proposed subspace condensation, an adaptive full space - subspace simulation scheme. It allows full space computation to be activated in the neighborhood of novel events while the rest of body still computes in a subspace. No constraint mechanisms are required to couple the subspace and full space regions. We designed several oracles to determine the full space region and provided a control parameter to balance the trade-off between accuracy and performance.

Finally, we designed a new full space method that achieves a two-way coupling between deformable solids and an incompressible fluid where the underlying geometric representation is entirely Eulerian. Using the recently developed Eulerian Solids approach [5], we were able to simulate multiple solids undergoing complex, frictional contact while simultaneously interacting with a fluid. Eulerian Solids have previously been integrated using explicit schemes. Instead, we developed an semi-implicit scheme that allows large time steps to be taken. No-slip boundary conditions are automatically satisfied by imposing a global divergence-free condition. The complexity of the scenarios we are able to simulate surpasses those that we have seen from any previous method.

6.2 Limitations

Our pose-space cubature scheme improves upon state-of-the-art subspace simulations by providing a quick way to approximate highly non-linear quantities. As observed

in Harmon and Zorin [80], a comprehensive theory of cubature sampling is still an open problem. We did not present such a theory here, but we hope our results and observations can assist in the formation of such a theory in the future.

Although subspace methods enjoy considerable speedup at runtime, the benefit might potentially be offset by the amount of precomputation. A PCA-based basis typically requires training data generated through full space simulation. The effort involved in setting up these examples might make it less appealing. Model analysis-based bases eliminate this requirement and recent advances [39, 57] improve their ability to handle non-linear deformations. How to compute our pose-space cubatures without explicit training data remains a very interesting question.

While our condensation algorithm allows subspace acceleration to be applied to new scenarios, some of the usual limitations of subspace methods still remain. If there are global deformations that are not well-captured by the subspace, some artifacts can appear. For example, collisions that produced near-rigid translations of the capsule sometimes produced slight swimming in the checkerboard texture if an equivalent quasi-translation was not captured by the basis. So, while basis construction becomes less onerous with our method, the subspace must still be constructed with care.

Our method can support any oracle, but the quality of that oracle will directly determine the amount of acceleration experienced by the simulation. We have proposed two oracles based on spherical distance and global velocity and acceleration and set their parameters heuristically. Our experimental results show that the quality of the simulation improves as the coverage of the full space region expands. A theory on this convergence property would definitely assist in designing more sophisticated oracles.

We used penalty forces to resolve collisions, but full space adaptivity opens the door to constraint mechanisms that subspace methods have had trouble with in the past. The degrees of freedom needed for hard constraints can now be added on the fly, and enable

such phenomena as adhesive contact [146].

Currently our Eulerian solid-fluid coupling method is limited to single-phase fluid. Extending this approach include high-quality, fully Eulerian liquid simulations [147] is an interesting direction for future work. As the technique is Eulerian, handling features that are smaller than a single grid cell, e.g. rods and thin shells [133], remains a challenge. This same difficulty extends to representing detailed fracture patterns, though extended finite element approaches [148] offer a potential solution.

Furthermore, while we have shown that an implicit integration scheme can be effective when simulating this coupling problem, other schemes that enable even larger timesteps [149], or preserve more structure given the same timestep [121], would be welcome additions to this work. Finally, it remains to be seen whether a combination of Eulerian-on-Lagrangian [73] and subspace methods [28, 150] could be used to accelerate the overall simulation.

6.3 Future Work

The scope of physics-based simulation in computer graphics has expanded tremendously in the past three decades. Simulations of sophisticated natural phenomena come at a cost of intensive computations. How to improve their efficiency remains an active research topic. By sampling three points on the 2D spectrum of Lagrangian to Eulerian and subspace to full space coordinate systems, we are able to accelerate a variety of deformable object simulations. There are still lots of empty spaces left on this spectrum that are worth exploring (Figure 6.2).

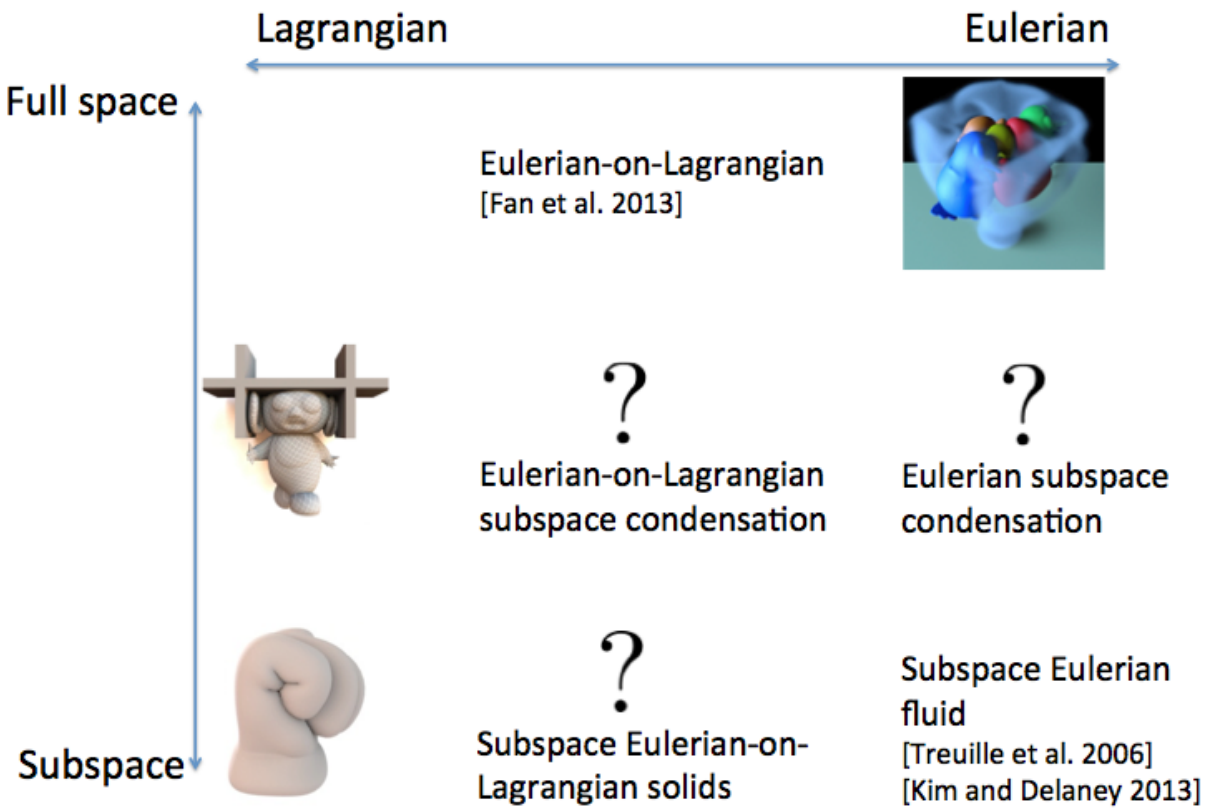


Figure 6.2: A lot of empty space still remains on this 2D spectrum. We list a few possibilities for future research.

6.3.1 Eulerian Subspace Condensation For Fluid Simulation

The subspace fluid simulation method developed by Kim and Delaney [28] not only faithfully reproduces the full space training result, but also allows re-simulating novel variations when small changes in the parameters (e.g. vorticity confinement, buoyancy) are made. Small, scripted moving obstacles are also supported by this approach. The questions remains whether we can further generalize it to quickly handle novel solid motion by adding full space adaptivity. Here we give a brief discussion of our initial attempt to solve this problem.

We applied subspace condensation to one-way coupling of an Eulerian fluid to a

scripted rigid body [132]. We built the subspace basis for the fluid as in Kim and Delaney [28] and set the full space region to be a narrow band around the solid-fluid boundary. In Kim and Delaney [28], the reduced matrices (e.g., divergence operator, Poisson matrix) for different stages can be precomputed and combined as a single reduced system matrix. Efficient incremental update can be applied to account for small scripted moving obstacles using the the Iterated Orthogonal Projection (IOP) approach [151]. Unfortunately, this is no longer the case for our scenario where the obstacle is large. The size of the full space region is significant and recomputing those reduced matrices for every frame offsets the performance gained in the linear solve. For a 2D simulation of size 64×64, subspace condensation ran $5\times$ slower than its full space counterpart. Alternatively, we could divide the domain into blocks and at runtime activate relevant blocks instead of arbitrary full space - subspace partitions. The reduced matrices can be assembled from the precomputed ones for each block. The problem can hence be formulated as one of finding the block size that optimizes the performance and accuracy trade-off.

6.3.2 Subspace Eulerian-on-Lagrangian Solids

An Eulerian-on-Lagrangian simulation (e.g. [73]) allows the Eulerian grid to always tightly enclose the solid being simulated and therefore can potentially be efficiently represented by a low-rank basis. One complication lies in that the actual cells containing the solids varies across different frames and representing the values (e.g. displacements, velocities) in the empty cells in a meaningful way is crucial to the quality of the resulting basis. For example, setting the velocities in the empty cells to zero would result in high discontinuities in the current frame as well as across different training frames. A PCA basis constructed from such data might be overly sensitive to the boundaries and hard

to generalize. Representing the displacements is more difficult, as neither zero-values nor linear extrapolation seems to be a good choice. Applying subspace condensation should be straight forward provided that the pure subspace version works.

Bibliography

[1] D. DeBlois and C. Cowell, *How to train your dragon 2*. Fox-Paramount Home Entertainment, 2014.

[2] C. Buck and J. Lee, *Frozen*. Walt Disney Studios Motion Pictures, 2013.

[3] P. Sohn, *The good dinosaur*. Walt Disney Studios Motion Pictures, 2015.

[4] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, *Sph fluids in computer graphics*, in *Eurographics: State of the Art Reports*, The Eurographics Association, 2014.

[5] D. I. Levin, J. Litven, G. L. Jones, S. Sueda, and D. K. Pai, *Eulerian solid simulation with contact*, in *ACM Transactions on Graphics (TOG)*, vol. 30, p. 36, ACM, 2011.

[6] S. F. Gibson and B. Mirtich, *A Survey of Deformable Models in Computer Graphics*, Tech. Rep. TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA, November, 1997.

[7] A. Nealen, M. Muller, R. Keiser, E. Boxerman, and M. Carlson, *Physically based deformable models in computer graphics*, in *Eurographics: State of the Art Report*, The Eurographics Association, 2005.

[8] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan, *Skinning with dual quaternions*, in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pp. 39–46, ACM, 2007.

[9] A. Jacobson, I. Baran, J. Popovic, and O. Sorkine, *Bounded biharmonic weights for real-time deformation.*, *ACM Transactions on Graphics (TOG)* **30** (2011), no. 4 78.

[10] P. Krysl, E. Grinspun, and P. Schrder, *Natural hierarchical refinement for finite element methods*, *International Journal for Numerical Methods in Engineering* **56** (2001) 2003.

[11] 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 (TOG)* **30** (July, 2011) 37:1–37:12.

[12] J. Barbič and D. L. James, *Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models*, *ACM Trans. on Graphics* **24** (Aug., 2005) 982–990.

[13] A. Treuille, A. Lewis, and Z. Popović, *Model reduction for real-time fluids*, *ACM Transactions on Graphics (TOG)* **25** (July, 2006) 826–834.

[14] 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 (TOG)* **33** (July, 2014) 105:1–105:9.

[15] A. Pentland and J. Williams, *Good vibrations: Modal dynamics for graphics and animation*, in *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, pp. 215–222, ACM, 1989.

[16] M. Shinya and A. Fournier, *Stochastic motionmotion under the influence of wind*, in *Computer Graphics Forum*, vol. 11, pp. 119–128, Wiley Online Library, 1992.

[17] P. Krysl, S. Lall, and J. E. Marsden, *Dimensional model reduction in non-linear finite element dynamics of solids and structures*, *International Journal for Numerical Methods in Engineering* **51** (2001) 479–504.

[18] M. Meyer and H. G. Matthies, *Efficient model reduction in non-linear dynamics using the karhunen-love expansion and dual-weighted-residual methods*, *Computational Mechanics* **31** (2003), no. 1-2 179–191.

[19] M. Wicke, M. Stanton, and A. Treuille, *Modular bases for fluid dynamics*, in *ACM Transactions on Graphics (TOG)*, vol. 28, p. 39, ACM, 2009.

[20] S. S. An, T. Kim, and D. L. James, *Optimizing Cubature for Efficient Integration of Subspace Deformations*, *ACM Trans. on Graphics* **27** (Dec., 2008) 165.

[21] T. Kim and J. Delaney, *Subspace fluid re-simulation*, *ACM Transactions on Graphics (TOG)* **32** (July, 2013).

[22] S. Li, J. Huang, F. de Goes, X. Jin, H. Bao, and M. Desbrun, *Space-time editing of elastic motion through material optimization and reduction*, *ACM Transactions on Graphics* **33** (2014), no. 4.

[23] D. Harmon and D. Zorin, *Subspace integration with local deformations*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 4 107.

[24] T. Kim and D. L. James, *Skipping steps in deformable simulation with online model reduction*, *ACM Transactions on Graphics* **28** (Dec., 2009) 123:1–123:9.

[25] N. Chentanez, T. G. Goktekin, B. E. Feldman, and J. F. O'Brien, *Simultaneous coupling of fluids and deformable bodies*, in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 83–89, Eurographics Association, 2006.

[26] A. Robinson-Mosher, R. E. English, and R. Fedkiw, *Accurate tangential velocities for solid fluid coupling*, in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, (New York, NY, USA), pp. 227–236, ACM, 2009.

[27] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw, *Two-way coupling of fluids to rigid and deformable solids and shells*, *ACM Transactions on Graphics (TOG)* **27** (Aug., 2008) 46:1–46:9.

[28] T. Kim and J. Delaney, *Subspace fluid re-simulation*, *ACM Transactions on Graphics (TOG)* **32** (July, 2013) 62:1–62:9.

[29] B. Solenthaler, J. Schläfli, and R. Pajarola, *A unified particle model for fluid–solid interactions*, *Computer Animation and Virtual Worlds* **18** (2007), no. 1 69–82.

[30] N. Akinci, J. Cornelis, G. Akinci, and M. Teschner, *Coupling elastic solids with smoothed particle hydrodynamics fluids*, *Computer Animation and Virtual Worlds* **24** (2013), no. 3-4 195–203.

[31] P. Clausen, M. Wicke, J. R. Shewchuk, and J. F. O'Brien, *Simulating liquids and solid-liquid interactions with lagrangian meshes*, *ACM Transactions on Graphics* **32** (April, 2013) 17:1–15.

[32] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, *Unified particle physics for real-time applications*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 153:1–153:12.

[33] D. L. James and D. K. Pai, *BD-Tree: Output-sensitive collision detection for reduced deformable models*, *ACM Transactions on Graphics* **23** (Aug., 2004) 393–398.

[34] J. Barbič and D. L. James, *Subspace self-collision culling*, *ACM Transactions on Graphics (TOG)* **29** (July, 2010) 81:1–81:9.

[35] Y. Teng, M. Meyer, T. DeRose, and T. Kim, *Subspace condensation: full space adaptivity for subspace deformations*, *ACM Transactions on Graphics (TOG)* **34** (2015), no. 4 76.

[36] M. Rheiner, *Birdly an attempt to fly*, in *ACM SIGGRAPH 2014 Emerging Technologies*, p. 3, ACM, 2014.

[37] N. Kwatra, C. Wojtan, M. Carlson, I. Essa, P. J. Mucha, and G. Turk, *Fluid simulation with articulated bodies*, *Visualization and Computer Graphics, IEEE Transactions on* **16** (2010), no. 1 70–80.

[38] W. Si, S.-H. Lee, E. Sifakis, and D. Terzopoulos, *Realistic biomechanical simulation and control of human swimming*, *ACM Transactions on Graphics (TOG)* **34** (2014), no. 1 10.

[39] C. von Tycowicz, C. Schulz, H.-P. Seidel, and K. Hildebrandt, *An efficient construction of reduced deformable objects*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 6 213.

[40] E. Sifakis and J. Barbič, *Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction*, in *ACM SIGGRAPH Courses*, pp. 20:1–20:50, 2012.

[41] J. Bonet and R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, New York, second ed., 2008.

[42] D. K. Pai, D. I. Levin, and Y. Fan, *Eulerian solids for soft tissue and more*, in *ACM SIGGRAPH 2014 Courses*, p. 22, ACM, 2014.

[43] R. Bridson, *Fluid Simulation for Computer Graphics*. AK Peters, 2008.

[44] I. Chao, U. Pinkall, P. Sanan, and P. Schröder, *A simple geometric model for elastic deformations*, in *ACM Transactions on Graphics (TOG)*, vol. 29, p. 38, ACM, 2010.

[45] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU Press, 2012.

[46] W. Briggs, *A Multigrid Tutorial*. SIAM, 2000.

[47] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*. Academic press, 2000.

[48] U. M. Ascher, S. J. Ruuth, and B. Wetton, *Implicit-explicit methods for time-dependent PDE's*. University of British Columbia, Department of Computer Science, 1993.

[49] P. G. Kry, D. L. James, and D. K. Pai, *Eigenskin: real time large deformation character skinning in hardware*, in *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 153–159, ACM, 2002.

[50] K. K. Hauser, C. Shen, and J. F. O'Brien, *Interactive deformation using modal analysis with constraints*, in *Graphics Interface 2003*, pp. 247–256, June, 2003.

[51] C. von Tycowicz, C. Schulz, H.-P. Seidel, and K. Hildebrandt, *An efficient construction of reduced deformable objects*, *ACM Transactions on Graphics (TOG)* **32** (Nov., 2013) 213:1–213:10.

[52] C. L. Lawson and R. J. Hanson, *Solving Least Square Problems*. Prentice Hall, Englewood Cliffs, NJ, 1974.

[53] R. Bro and S. De Jong, *A fast non-negativity-constrained least squares algorithm*, *Journal of Chemometrics* **11** (1997), no. 5 393–401.

[54] D. L. James, J. Barbič, and D. K. Pai, *Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources*, *ACM Transactions on Graphics (TOG)* **25** (2006), no. 3 987–995.

[55] S. Foucart, *Hard thresholding pursuit: an algorithm for compressive sensing*, *SIAM Journal on Numerical Analysis* **49** (2011), no. 6 2543–2563.

[56] K. E. Atkinson, *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[57] Y. Yang, D. Li, W. Xu, Y. Tian, and C. Zheng, *Expediting precomputation for reduced deformable simulation*, *ACM Transactions on graphics (TOG)* **34** (2015), no. 6.

[58] T. Wasfy and A. Noor, *Computational strategies for flexible multibody systems*, *Applied Mechanics Reviews* **56** (2003), no. 6.

[59] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen, *Feti-dp: a dual–primal unified feti methodpart i: A faster alternative to the two-level feti method*, *International journal for numerical methods in engineering* **50** (2001), no. 7 1523–1544.

[60] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, *SIAM Journal on Scientific Computing* **25** (2003), no. 1 246–258.

[61] Y. Yang, W. Xu, X. Guo, K. Zhou, and B. Guo, *Boundary-aware multi-domain subspace deformation*, *IEEE Transactions on Visualization and Computer Graphics* (2013).

[62] P. Kaufmann, S. Martin, M. Botsch, and M. Gross, *Flexible simulation of deformable models using discontinuous galerkin fem*, *Graphical Models* **71** (2009), no. 4 153–167.

[63] J. Barbič and Y. Zhao, *Real-time large-deformation substructuring*, in *ACM transactions on graphics (TOG)*, vol. 30, p. 91, ACM, 2011.

[64] T. Kim and D. L. James, *Physics-based character skinning using multi-domain subspace deformations*, in *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, pp. 63–72, 2011.

[65] Y. Zhao and J. Barbič, *Interactive authoring of simulation-ready plants*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 4 84.

[66] T. Kim, "Cubica: a toolkit for subspace deformations."

[67] M. Wicke, D. Ritchie, B. M. Klingner, S. Burke, J. R. Shewchuk, and J. F. O'Brien, *Dynamic local remeshing for elastoplastic simulation*, in *ACM Transactions on graphics (TOG)*, vol. 29, p. 49, ACM, 2010.

[68] R. Narain, A. Samii, and J. F. O'Brien, *Adaptive anisotropic remeshing for cloth simulation*, *ACM transactions on graphics (TOG)* **31** (2012), no. 6 152.

[69] D. Sulsky, S.-J. Zhou, and H. L. Schreyer, *Application of a particle-in-cell method to solid mechanics*, *Computer Physics Communications* **87** (1995), no. 12 236 – 252. Particle Simulation Methods.

[70] A. W. Bargteil, C. Wojtan, J. K. Hodgins, and G. Turk, *A finite element method for animating large viscoplastic flow*, *ACM Transactions on Graphics (TOG)* **26** (2007), no. 3.

[71] A. Stomakhin, C. Schroeder, C. Jiang, L. Chai, J. Teran, and A. Selle, *Augmented mpm for phase-change and varied materials*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 138:1–138:11.

[72] C. Jiang, C. Schroeder, A. Selle, J. Teran, and A. Stomakhin, *The affine particle-in-cell method*, *ACM Transactions on Graphics (TOG)* **34** (July, 2015) 51:1–51:10.

[73] Y. Fan, J. Litven, D. I. Levin, and D. K. Pai, *Eulerian-on-lagrangian simulation*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 3 22.

[74] J. Stam, *Stable fluids*, in *SIGGRAPH 1999*, pp. 121–128, 1999.

[75] A. Selle, R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac, *An unconditionally stable maccormack method*, *J. Sci. Comput.* **35** (June, 2008) 350–371.

[76] J. Brackbill and H. Ruppel, *Flip: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions*, *Journal of Computational Physics* **65** (1986), no. 2 314–343.

[77] Y. Zhu and R. Bridson, *Animating sand as a fluid*, *ACM Transactions on Graphics (TOG)* **24** (July, 2005) 965–972.

[78] Y. Teng, M. A. Otaduy, and T. Kim, *Simulating articulated subspace self-contact*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 106:1–106:9.

[79] J. Barbič and D. L. James, *Six-DoF haptic rendering of contact between geometrically complex reduced deformable models*, *IEEE Transactions on Haptics* **1** (jan.-june, 2008) 39–52.

[80] D. Harmon and D. Zorin, *Subspace integration with local deformations*, *ACM Transactions on Graphics* **32** (July, 2013).

[81] M. Tang, D. Manocha, M. A. Otaduy, and R. Tong, *Continuous penalty forces*, *ACM Transactions on Graphics (TOG)* **31** (2012), no. 4.

[82] D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun, *Asynchronous contact mechanics*, *ACM Transactions on Graphics (TOG)* (2009).

[83] J. P. Lewis, M. Cordner, and N. Fong, *Pose Space Deformations: A Unified Approach to Shape Interpolation and Skeleton-Driven Deformation*, in *Proceedings of SIGGRAPH*, pp. 165–172, July, 2000.

[84] D. Chen and R. Plemmons, *Nonnegativity constraints in numerical analysis*, in *Symposium on the Birth of Numerical Analysis*, 2007.

[85] A. A. Shabana, *Theory of Vibration, Volume II: Discrete and Continuous Systems*. Springer–Verlag, NY, 1990.

[86] S. Idelsohn and A. Cardona, *A reduction method for nonlinear structural dynamic analysis*, *Computer Methods in Applied Mechanics and Engineering* **49** (1985) 253–279.

[87] J. F. O'Brien, C. Shen, and C. M. Gatchalian, *Synthesizing sounds from rigid-body simulations*, in *ACM SIGGRAPH Sym. on Computer Animation*, pp. 175–181, July, 2002.

[88] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd ed., 2008.

[89] T. Kurihara and N. Miyata, *Modeling deformable human hands from medical images*, in *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, pp. 357–366, 2004.

[90] P. G. Kry, D. L. James, and D. K. Pai, *EigenSkin: Real Time Large Deformation Character Skinning in Hardware*, in *ACM SIGGRAPH Sym. on Computer Animation*, pp. 153–160, July, 2002.

[91] J. Kim and N. S. Pollard, *Fast simulation of skeleton-driven deformable body characters*, *ACM Transactions on Graphics (TOG)* **30** (Oct., 2011) 121:1–121:19.

[92] A. Wächter and S. Vigerske, "Interior point optimizer." https://projects.coin-or.org/Ipopt, 2005.

[93] A. Wächter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2002.

[94] J. Hogg and J. Scott, *An indenite sparse direct solver for multicore machines*, Tech. Rep. TR-RAL-2010-011, Rutherford Appleton Laboratory, Chilton, Oxfordshire, UK, 2010.

[95] I. S. Duff, *MA57—a code for the solution of sparse symmetric definite and indefinite systems*, *ACM Trans. Math. Softw.* **30** (June, 2004) 118–144.

[96] M. T. Jones, P. E. Plassmann, and P. Mcs-p, *An improved incomplete cholesky factorization*, *ACM Trans. Math. Software* **21** (1995) 5–17.

[97] S. Curtis, R. Tamstorf, and D. Manocha, *Fast collision detection for deformable models using representative-triangles*, in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pp. 61–69, 2008.

[98] G. Hirota, S. Fisher, A. State, C. Lee, and H. Fuchs, *An implicit finite element method for elastic solids in contact*, in *Proceedings of the Fourteenth Conference on Computer Animation*, pp. 136 –254, 2001.

[99] J. Teran, E. Sifakis, G. Irving, and R. Fedkiw, *Robust quasistatic finite elements and flesh simulation*, in *ACM SIGGRAPH Symp. on Computer Animation*, pp. 181–190, 2005.

[100] L. Liu, K. Yin, B. Wang, and B. Guo, *Simulation and control of skeleton-driven soft body characters*, *ACM Transactions on Graphics (TOG)* **32** (Nov., 2013) 215:1–215:8.

[101] J. Barbič and Y. Zhao, *Real-time large-deformation substructuring*, *ACM Trans. on Graphics* **30** (2011).

[102] W. Xu, N. Umentani, Q. Chao, J. Mao, X. Jin, and X. Tong, *Sensitivity-optimized rigging for example-based real-time clothing synthesis*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 107:1–107:11.

[103] Y. Teng, M. A. Otaduy, and T. Kim, *Simulating articulated subspace self-contact*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 106:1–106:9.

[104] K.-J. Bathe, *Finite Element Procedures.* Prentice Hall, third ed., 2007.

[105] R. J. Guyan, *Reduction of stiffness and mass matrices*, *AIAA journal* **3** (1965), no. 2 380–380.

[106] B. Irons, *Structural eigenvalue problems-elimination of unwanted variables*, *AIAA journal* **3** (1965), no. 5 961–962.

[107] E. L. Wilson, *The static condensation algorithm*, *International Journal for Numerical Methods in Engineering* **8** (1974), no. 1 198–203.

[108] A. Y.-T. Leung, *An accurate method of dynamic condensation in structural analysis*, *International Journal for Numerical Methods in Engineering* **12** (1978), no. 11 1705–1715.

[109] M. Paz, *Modified dynamic condensation method*, *Journal of Structural Engineering* **115** (1989), no. 1 234–238.

[110] M. Bro-Nielsen and S. Cotin, *Real-time volumetric deformable models for surgery simulation using finite elements and condensation*, in *Computer graphics forum*, vol. 15, pp. 57–66, Wiley Online Library, 1996.

[111] M. Gao, N. Mitchell, and E. Sifakis, *Steklov-poincaré skinning*, in *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pp. 139–148, The Eurographics Association, 2014.

[112] H. Xu, Y. Li, Y. Chen, and J. Barbič, *Interactive material design using model reduction*, *ACM Trans. on Graphics* (2014).

[113] I. Baran and J. Popović, *Automatic rigging and animation of 3d characters*, in *ACM Transactions on Graphics (TOG)*, vol. 26, p. 72, ACM, 2007.

[114] G. Guennebaud, B. Jacob, *et. al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[115] Y. Teng, D. I. Levin, and T. Kim, *Eulerian solid-fluid coupling*, .

[116] G. Irving, C. Schroeder, and R. Fedkiw, *Volume conserving finite element simulations of deformable models*, in *ACM Transactions on Graphics (TOG)*, vol. 26, 2007.

[117] H. Wang, J. O'Brien, and R. Ramamoorthi, *Multi-resolution isotropic strain limiting*, *ACM Transactions on Graphics* **29** (2010), no. 6 156:1–156:10.

[118] A. Stomakhin, R. Howes, C. Schroeder, and J. M. Teran, *Energetically consistent invertible elasticity*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 25–32, 2012.

[119] F. S. Sin, D. Schroeder, and J. Barbič, *Vega: Non-linear fem deformable object simulator*, *Computer Graphics Forum* **32** (2013), no. 1 36–48.

[120] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, *Projective dynamics: Fusing constraint projections for fast simulation*, *ACM Transactions on Graphics (TOG)* **33** (July, 2014) 154:1–154:11.

[121] P. Mullen, K. Crane, D. Pavlov, Y. Tong, and M. Desbrun, *Energy-preserving integrators for fluid animation*, *ACM Transactions on Graphics (TOG)* **28** (July, 2009) 38:1–38:8.

[122] B. Zhu, W. Lu, M. Cong, B. Kim, and R. Fedkiw, *A new grid structure for domain extension*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 4 63:1–63:12.

[123] X. Zhang, R. Bridson, and C. Greif, *Restoring the missing vorticity in advection-projection fluid solvers*, *ACM Transactions on Graphics (TOG)* **34** (2015), no. 4 52:1–52:8.

[124] A. Robinson-Mosher, T. Shinar, J. Gretarsson, J. Su, and R. Fedkiw, *Two-way coupling of fluids to rigid and deformable solids and shells*, in *ACM Transactions on Graphics (TOG)*, vol. 27, p. 46, ACM, 2008.

[125] M. Macklin and M. Müller, *Position based fluids*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 4 104.

[126] E. Sifakis, S. Marino, and J. Teran, *Globally coupled collision handling using volume preserving impulses*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 147–153, 2008.

[127] A. McAdams, A. Selle, K. Ward, E. Sifakis, and J. Teran, *Detail preserving continuum simulation of straight hair*, *ACM Transactions on Graphics (TOG)* **28** (2009), no. 3 62:1–62:6.

[128] C. S. Peskin, *The immersed boundary method*, *Acta Numerica* **11** (1, 1972) 479–517.

[129] T. Takahashi, H. Ueki, A. Kunimatsu, and H. Fujii, *The simulation of fluid-rigid body interaction*, in *ACM SIGGRAPH 2002 Conference Abstracts and Applications*, SIGGRAPH '02, (New York, NY, USA), pp. 266–266, ACM, 2002.

[130] M. Carlson, P. J. Mucha, and G. Turk, *Rigid fluid: Animating the interplay between rigid bodies and fluid*, *ACM Transactions on Graphics (TOG)* **23** (Aug., 2004) 377–384.

[131] B. M. Klingner, B. E. Feldman, N. Chentanez, and J. F. O'Brien, *Fluid animation with dynamic meshes*, *ACM Transactions on Graphics (TOG)* **25** (July, 2006) 820–825.

[132] C. Batty, F. Bertails, and R. Bridson, *A fast variational framework for accurate solid-fluid coupling*, *ACM Transactions on Graphics (TOG)* **26** (2007), no. 3 100.

[133] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw, *Coupling water and smoke to thin deformable and rigid shells*, *ACM Transactions on Graphics (TOG)* **24** (July, 2005) 973–981.

[134] X. He, N. Liu, G. Wang, F. Zhang, S. Li, S. Shao, and H. Wang, *Staggered meshless solid-fluid coupling*, *ACM Transactions on Graphics (TOG)* **31** (Nov., 2012) 149:1–149:12.

[135] M. Souli and D. J. Benson, *Arbitrary Lagrangian Eulerian and Fluid-Structure Interaction: Numerical Simulation*. John Wiley & Sons, 2013.

[136] T. Wick, *Coupling of fully eulerian and arbitrary lagrangian–eulerian methods for fluid-structure interaction computations*, *Computational Mechanics* **52** (2013), no. 5 1113–1124.

[137] T. Lenaerts, B. Adams, and P. Dutré, *Porous flow in particle-based fluid simulations*, in *ACM Transactions on Graphics*, vol. 27, (New York, NY, USA), pp. 49:1–49:8, ACM, 2008.

[138] F. P. T. Baaijens, *A fictitious domain mortar element method for fluid/structure interaction*, *International Journal for Numerical Methods in Fluids* **35** (2001), no. 7 743–761.

[139] C. Jiang, C. Schroeder, J. Teran, A. Stomakhin, and A. Selle, *The material point method for simulating continuum materials*, in *ACM SIGGRAPH Courses*, 2016.

[140] K. Kamrin, C. H. Rycroft, and J.-C. Nave, *Reference map technique for finite-strain elasticity and fluid–solid interaction*, *Journal of the Mechanics and Physics of Solids* **60** (Nov., 2012) 1952–1969.

[141] B. Valkov, C. H. Rycroft, and K. Kamrin, *Eulerian method for multiphase interactions of soft solid bodies in fluids*, *Journal of Applied Mechanics* **82** (2015), no. 4.

[142] T. Belytschko, W. K. Liu, B. Moran, and K. Elkhodary, *Nonlinear finite elements for continua and structures*. John Wiley & Sons, 2013.

[143] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, and A. Selle, *A material point method for snow simulation*, *ACM Transactions on Graphics (TOG)* **32** (2013), no. 4 102.

[144] R. Bridson, R. Fedkiw, and J. Anderson, *Robust treatment of collisions, contact and friction for cloth animation*, in *ACM Transactions on Graphics (ToG)*, vol. 21, pp. 594–603, ACM, 2002.

[145] R. Fedkiw, J. Stam, and H. W. Jensen, *Visual simulation of smoke*, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 15–22, ACM, 2001.

[146] J. Gascón, J. S. Zurdo, and M. A. Otaduy, *Constraint-based simulation of adhesive contact*, in *Proc. of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2010.

[147] N. Heo and H.-S. Ko, *Detail-preserving fully-eulerian interface tracking framework*, *ACM Transactions on Graphics (TOG)* **29** (Dec., 2010).

[148] P. Kaufmann, S. Martin, M. Botsch, E. Grinspun, and M. Gross, *Enrichment textures for detailed cutting of shells*, *ACM Transactions on Graphics (TOG)* **28** (July, 2009) 50:1–50:10.

[149] M. Lentine, M. Cong, S. Patkar, and R. Fedkiw, *Simulating free surface flow with very large time steps*, in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 107–116, 2012.

[150] B. Liu, G. Mason, J. Hodgson, Y. Tong, and M. Desbrun, *Model-reduced variational fluid simulation*, *ACM Transactions on Graphics (TOG)* **34** (Oct., 2015) 244:1–244:12.

[151] J. Molemaker, J. M. Cohen, S. Patel, and J. Noh, *Low viscosity flow simulations for animation*, in *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, pp. 9–18, 2008.