NOVEL DEGREES OF FREEDOM, CONSTRAINTS, AND STIFFNESS FORMULATION FOR PHYSICALLY BASED ANIMATION

A Dissertation

by

NICHOLAS J. WEIDNER

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2022

Major Subject: Computer Science

ABSTRACT

I identify and improve upon three distinct components of physically simulated systems with the aim of increasing both robustness and efficiency for the application of computer graphics: A) the degrees of freedom of a system; B) the constraints put on that system; C) and the stiffness that derives from force differentiation and in turn enables implicit integration techniques. These three components come up in many implementations of physics-based simulation in computer animation. From a combination of these components, I explore four novel ideas implemented and experimented on over the course of my graduate degree. *Eulerian-on-Lagrangian Cloth Simulation* resolves a longstanding problem of simulating contact-mediated interaction of cloth and sharp geometric features by exploring a combination of all three of our components. *Bilateral Staggered Projections for Joints* explores the constrained degrees of freedom of articulated rigid bodies in a reduced state to extend the popular Staggered Projects technique into a novel formulation for rapid evaluation of frictional articulated dynamics. *Condensation Jacobian with Adaptivity* looks at using reduction methods to improve the efficiency of soft body deformations by allowing larger time step in dynamics simulations. Finally, *Ldot: Boosting Deformation Performance with Cholesky Extrapolation* explores the inner workings of sparse direct solvers to introduce a Cholesky factorization that is linearly extrapolated in time, which can improve the performance when encapsulated inside an iterative nonlinear solver.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of advising Professor Shinjiro Sueda, Professor John Keyser and Professor Timothy Davis of the Department of Computer Science and Professor Viniyak Krishnamurthy of the Department of Mechanical Engineering.

This work includes the research, implementation, and experimentation done on EoL Cloth in Chapter 2. Preliminary to this was experimentation done in the EoL Cloth domain by Kyle Piddington. Chapter 3 is based off of the "Frictional Joints" section of the REDMAX paper, while sections on "REDMAX Flexibility" and "Projected Block Jacobi Preconditioner" in the original paper were written by Ying Wang and Margaret Baxter respectively. All of the work covered in Chapter 4 was solely conducted by the student with contributions to the analytical derivations of the stiffness and deformations provided by Dr. Theodore Kim. Finally, all of the work on Ldot covered in Chapter 5 was done by the student alongside Bethany Witemeyer's implementations of various nonlinear solvers.

Funding Sources

Graduate study was supported by a SEC Engineering Deans Graduate fellowship from Texas A&M University and the National Science Foundation (CAREER-1846368).

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Physics-based animation is an important area of research in the field of computer graphics. In the early days of the visual effects (VFX) industry, many non physics based tool were created. Although very useful, these tools tended to be overly tedious or unsuited when it came to replicating physical phenomena, so it was not before long that physics became a staple in the development of many tools. With the advancement of algorithms, solvers, and hardware, simulating physical motion became the standard way to generate complex animation. The continued industry explosion in VFX accelerated this development even further. Now many of the tools and algorithms developed for artistic purposes have significantly impacted various cross disciplinary fields such as robotics and biomechanics.

Of the phenomena that is much simpler to describe using the laws of physics compared to pure artistic control, a few examples are the simulations of cloth, soft deformable bodies, and rigid bodies, which are the topic of this dissertation. Cloth simulation has proven to be one of the major success stories for physics-based animation as it is phenomenological behavior is difficult to capture for an artist. Cloth simulation itself has a long history in computer graphics starting with work by Terzopoulos et al. [1]. Driving cloth with physics allows artist-controlled inputs, such as geometry and initial conditions, to generate life-like results with much less human effort. Baraff and Witkin [2] increased the efficiency of dynamic simulations in their seminal work by enabling large time steps and making cloth much more useful in production. To this same extent, soft deformable body simulation uses many of the same fundamentals as thin shelled cloth in solids. Representing intricate geometry with diverse material properties offers a lot of versatility on the artistic side. When the artist has more interest in the overall object motion compared to the internal deformations, rigid body simulation simplifies the approach. More specifically, articulated rigid body dynamics connects a series of (potentially branching) bodies together to create complex and intertwined systems. All of these simulation types are heavily multidisciplinary and have applications in various disciplines outside of just computer graphics including biomechanics, robotics, and aerospace.

From these motivations, research over the last few decades has focused on improving both the performance and physical realism of these simulations. Understandably, the development of this research has not been a linear path. That is because there are often different acceptable solutions to these problems depending on the intent. Ultimately, at some level a simulation is only an estimation of a true physical phenomena down to a level of computational precision or specific algorithmic/mathematical abstractions. Often times simulating at the highest level of physical detail is expensive and overkill when the output is just a believable or plausible sequence of images image on a screen. With cloth for example, early work often described an object as a collection of connected springs. These simulations, while often inaccurate and unstable, are still used today for their relative simplicity in understanding and implementation and their performance when the results are good enough. In modern day computer graphics, research has accelerated to the point where adapting more stable and realistic implementations drawing and extending from fields such as mechanical engineering have become viable. This has increased the toolbox in which modern day artists have the option to choose from, and expanded the relevant research approaches to continuing the development of the field.

Because there are numerous angles at which to approach simulating dynamic motion, in this dissertation I focus on three primary components of a simulated system: A) the degrees of freedom of a system; B) the constraints put on that system; C) and the stiffness that derives from force differentiation and in turn enables implicit integration techniques.

A. Degrees of freedom, or DOFs, refer to the independent parameters defining some configuration state. This configuration can vary from problem to problem. With the Lagrangian view, objects are divided into discrete elements that represent a portion of the physical material. Each of these elements have their own configuration states representing their movement through space. Alternatively, with the Eulerian view, the space itself is divided into discrete cells. The simulated material moves through and are tracked on these cells [3]. The majority of the topics covered in this dissertation deal with the Lagrangian view, but some do touch on the use of the Eulerian view. In deformable bodies for example our degrees of freedom are usually nodal positions in 3D that connect to form a geometric object. The engineering field of modal analysis, which has grown prominent in computer graphics in recent years, studies a set of low-dimensional DOFs that are capable of reproducing the salient deformations of the object. In a rigid body system, the system is usually represented with 6 DOFs for each body, relating to 3D position and rotation. For an articulated rigid body system, the DOFs can be further reduced to just the relative motion between the bodies (*i.e.,* the joint angles).

B. In most "visually interesting" simulations, the interesting motion in a scene is induced by the introduction of constraints on the aforementioned DOFs of which the most prominently are collision constraints. Simulated collisions included everything from a soft body bounding off a floor, to cloth sliding across static scenery or itself, to the rotation of a rigid body around a joint. Furthermore, contact often implies the introduction of a frictional force into our simulated system. There exist relatively simple solutions to simulating contact and friction that do not involve constraints. Penalty force methods, for example, do not constrain our DOFs and instead introduce forces into our system to force contacting objects away from each other. This approach is efficient and can still be powerful, but often requires careful tuning of the penalty parameters, which can be tedious. Alternatively, we can apply explicit restrictions on the position relationships of our DOFs to constraint our system without introducing non-physical forces. When these restrictions require the constrained DOFs to take on a certain (potentially time-varying) value, this is referred to as a bilateral or equality constraint. As an example, imagine an object, rigid or deformed, fixed to a ceiling. These constrained DOFs are taking on the value of the ceiling in order to remain fixed. More broadly, some DOFs of an object may be constrained to lie on a lower-dimensional subspace. For example, some nodes may be constrained to slide along a rail on the ceiling. When these restrictions require the constrained DOFs to stay to a particular side of a (again potentially time-varying) value, this is referred to as a unilateral or inequality constraint. Unlike fixing an object to another, a simple example here is an object colliding with a floor. Due to the much larger solution space of inequality constraints and due to the fact that the constraints need to be turned on and off depending on the system configuration, they are numerically more challenging to solve compared to equality constraints.

C. Solving for these degrees of freedom under a set of constraints efficiently is at the core of many physics simulations. To do so, one is faced with the often-difficult task of choosing a time integrator, with each choice giving a different set of advantages and disadvantages in terms of complexity and efficiency. Many simulation problems related to deformable objects are categorized as "stiff" problems [4]. Most effective at solving these stiff problems is the implicit integration scheme that evaluates the forces at the end of the time step. This is in direct comparison to the explicit time stepping scheme that evaluates the force at the beginning of the time step, but is highly susceptible to overshooting our target on stiffer problems and requiring a relatively small step size to remain stable. The disadvantage of implicit schemes is that we now need the differential of our forces which get compiled into a (tangent) stiffness matrix. Depending on which material type is used in our object, we may have the option of leveraging precomputation approaches for constructing this matrix but this is not a guaranteed luxury [5]. This derivative of the force, or stiffness matrix, makes solving for our DOFs computationally more complicated. This stiffness matrix is usually indefinite, but previous work has explored analytical techniques for clamping the eigenvalues to be non-zero, making these matrices positive semi-definite [6]. This opens us up a field of efficient matrix solvers, but also introduces more choices into our problem-solving process. Iterative solvers are widely popular when the exact solution is not necessarily required [7]. These techniques approach a solution to within a specific tolerance relatively quickly. If the error from this tolerance range does not destabilize the simulation, this might be all that is needed [8, 2]. On the other hand sometimes the exact solution to a problem is needed and a direct solve approach is required [9]. Modern day direct solvers are incredibly efficient, but require a large amount of computational memory and become unusable on large scale problems. All in all, the introduction of stiffness into our system is often necessary to solve

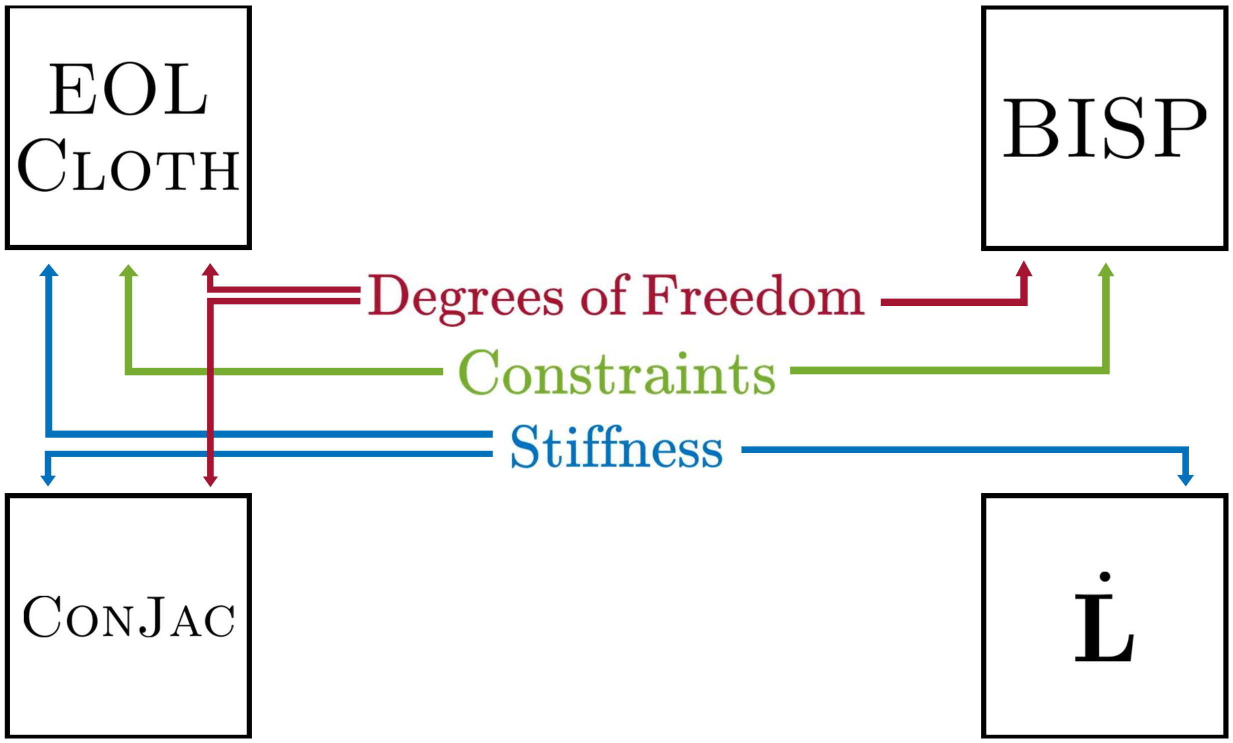


Figure 1.1: A subcategory of dynamic motion components that relate across the connected research discussed in this work.

those "visually interesting" problems. This introduction opens up new avenues for exploring physical simulation efficiency and stability.

These three components only cover a selective breadth of the entire field of physics based animation, but are still broad enough to cover a variety of topics. This dissertation aims to assess the contributed research across four different projects that each touched on these three categories, as shown in Fig. 1.1.

I. *Eulerian-on-Lagrangian Cloth Simulation* (SIGGRAPH 2018) [10] deals with all three of the prescribed categories. Where traditional approaches to cloth simulation employ a Lagrangian approach and limit the discretized nodes to three spacial DOFs (x, y, z), the introduction of two new DOFs in an Eulerian domain (u, v) resolves a long standing contact problem for cloth. Constraining cloth in regards to external objects is realizable only by introducing constraints that do not fully represent the underlying geometry and discretization. In order to more accurately define these constraints, Eulerian space is introduced to the problem. The Eulerian component of the problem must also account for the stiffness of our integration and must be properly constrained in its own space. With this technique cloth can properly interact with sharp geometry that traditionally snags cloth and leads to simulated instabilities.

II. *Bilateral Staggered Projections for Joints* (a part of the larger REDMAX project) (SIGGRAPH 2019) [11] explores the constrained DOFs of articulated bodies in a reduced space. A popular approach for applying friction to contact constraints is the Staggered Projections algorithm [12]. Using a reduced coordinate approach, we eliminate the expensive step of solving for the contact impulse. As a byproduct of this approach, though, we can no longer compute the constraint forces directly, which are needed when solving for the frictional forces. Extending Staggered Projections to a new algorithm called Bilateral Staggered Projections for articulated rigid bodies uses this reduced approach, but introduces an alternative method for computing the constraint forces by solving the reduced and unconstrained maximal system. For articulated rigid bodies, this approach proves much more efficient than using traditional Staggered Projections.

III. *Condensation Jacobian with Adaptivity* (MIG 2020) [13] looks at using reduction methods to improve the efficiency of soft body deformations. Previous work has explored the concept of condensation [14]. This approach to dynamic simulation takes a globalized problem and solves for the reduced modes of the structure. Originally developed in structural engineering for static vibration analysis and later expanded to dynamics, reducing the modes of the problem helps speed up large problems and reduce the accompanying numerical drift. While others like Gao et al. [15] have adapted this technique for the graphics field, we introduce condensation at the velocity level of a simulation to improve the efficiency of soft body simulations. Reducing problems down to a subset of driver nodes that still encapsulate the entirety of the material stiffness improves traditional stability even at larger time steps.

IV. *Ldot: Boosting Deformation Performance with Cholesky Extrapolation* [16] examines the solvers driving our simulations. These solvers are often treated as a black box by our simulation, but there is room to explore them to more specifically suit our problems. Almost always, the input matrices to these solvers undergo a Cholesky decomposition into the product of a lower triangular matrix and its transpose. We examine the derivative of this lower triangular matrix with respect to time in order to predict a future solution to minimize the total number of decompositions.

$$\mathbf{L} = \text{chol}(\mathbf{A}) \qquad \Rightarrow \qquad (\mathbf{L}, \dot{\mathbf{L}}) = \text{dchol}(\mathbf{A}, \dot{\mathbf{A}}) \tag{1.1}$$

While this work highlights some shortcomings in the approach, it does explore an interesting experimental avenue for cutting down on the typically most expensive portion of a physical simulation.

Across all of these works we examine how they tie into our overarching topics of exploration: Degrees of Freedom, Constraints, and Stiffness. Each of these topics ties into the even larger network of physical simulation work that continues to be developed by researchers from graphics and beyond.

2. EULERIAN-ON-LAGRANGIAN CLOTH SIMULATION [*]

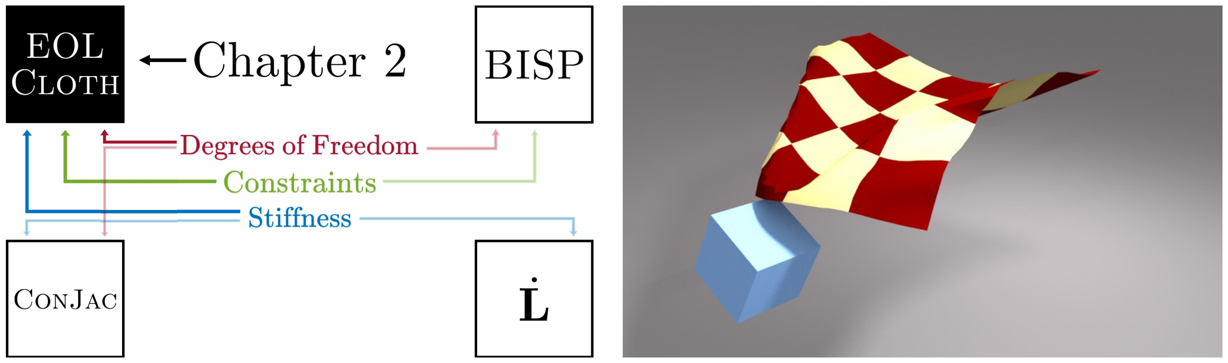


Figure 2.1: (Left) Eulerian-on-Lagrangian Cloth Simulation deals with all three of the prescribed categories. (Right) Still frame of a simulation showcasing the smooth sliding of cloth over sharp features such as the edges and corners of a dynamically moving box. Reprinted from [10].

We resolve the longstanding problem of simulating the contact-mediated interaction of cloth and sharp geometric features by introducing an Eulerian-on-Lagrangian (EOL) approach to cloth simulation. Unlike traditional Lagrangian approaches to cloth simulation, our EOL approach permits bending exactly at and sliding over sharp edges, avoiding parasitic locking caused by over-constraining contact constraints. Wherever the cloth is in contact with sharp features, we insert EOL vertices into the cloth, while the rest of the cloth is simulated in the standard Lagrangian fashion. Our algorithm manifests as new equations of motion for EOL vertices, a contact-conforming remesher, and a set of simple constraint assignment rules, all of which can be incorporated into existing state-of-the-art cloth simulators to enable smooth, inequality-constrained contact between cloth and objects in the world.

[*] Reprinted with permission from "Eulerian-on-Lagrangian Cloth Simulation" by N. J. Weidner, K. Piddington, D. I. W. Levin, and S. Sueda, *ACM Tans. Graph.*, vol. 37, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201281. Copyright 2018 by Association for Computing Machinery

2.1 Introduction

Cloth simulation has a long history in computer graphics. Starting with the work by Terzopoulos et al. [1], cloth simulation algorithms have steadily evolved from research curiosities to an integral part of the visual effects pipeline. In fact, cloth simulation has proven to be one of the major success stories for physics-based animation as its phenomenological behavior is difficult to capture for an artist and far simpler to describe using the laws of physics. This allows artist-controlled inputs, such as geometry and initial conditions, to generate life-like results with much less human effort. Motivated by this, research over the last few decades has focused on improving both the performance and physical realism of cloth-simulation.

Due to its highly-deformable nature, cloth undergoes complex colliding interactions with other geometry and itself. Therefore, it is no surprise that much of the research in the field focuses on resolving these collisions. Despite this concentration of effort, there is one crucial scenario which, so far, has yet to be treated: the interaction of cloth with sharp geometric features. Such interactions are common when cloth interacts with the world, from a table cloth pulled over a table edge, to a sheet dragged off a clothing line to the unveiling of a sculpture. Yet this literal (but not figurative) edge case still baffles today's state-of-the-art approaches.

The difficulty arises because all previous cloth simulation algorithms rely on a Lagrangian discretization of the cloth, which only permits bending along its edges. As a pedagogical example, imagine draping the cloth over a sharp edge of a table. Unless the cloth mesh has edges exactly aligned with the table edge, the cloth will not be able to bend sharply, leading to unappealing visual artifacts. For a stationary piece of cloth, this can be resolved by remeshing the cloth, inserting edges that directly align with the table edge. However, what happens if we pull the cloth? We suffer from unacceptable jitter since the cloth can only be remeshed between time steps. This occurs even if we remesh the cloth during the sliding motion, as shown in Fig. 2.2. In certain scenarios we may even experience catastrophic locking, wherein the cloth, unable to slide over the edge, simply gets stuck. Today, standard approaches to fixing this problem involve perturbing the underlying geometry by approximating the underlying surface as smooth (*e.g.,* by averaging surface normals).

However, this approach is unsatisfactory because it prevents us from visually capturing the correct physical sliding behavior of the cloth over the sharp feature, even if this is precisely what an artist wants to achieve. This goes against the very philosophy of physics-based animation as we are unintentionally adding counterintuitive, non-physical behavior to our simulations which can prevent achieving a desired physical effect.

In this work, we propose the first Eulerian-on-Lagrangian [17, 18, 19] cloth framework—an augmentation of Lagrangian cloth solvers that allows cloth to behave correctly when interacting with both smooth and sharp features based solely on the input geometry. We develop a new contact-conforming remeshing algorithm and propose a new set of contact constraints that, together, correctly handle sliding contact at sharp edges and corners by ensuring degrees of freedom (DOFs) exist at contact boundaries and automatically allowing a contact solver to apportion motion between Lagrangian and Eulerian DOFs in a principled way. These contributions mean that our method avoids constraint jitter and catastrophic locking due to contact (Fig. 2.2). Furthermore, because our technique works in conjunction with, rather than supplanting, Lagrangian simulation schemes, our algorithm gracefully elides to standard cloth simulation in all other cases. We demonstrate the efficacy and applicability of our method by integrating our framework with the state-of-the-art cloth remesher found in ARCSim [20] and using it to resolve a number of challenging scenarios involving close contact of cloth with sharp geometric features.

2.2 Related Work

Cloth simulation has received tremendous amount of attention from the graphics community, starting with the seminal work by Terzopoulos et al. [1]. Here we focus on works that are most relevant to our work.

Considerable amount of work has focused on improving the efficiency of cloth simulators, using, for example: linearly implicit integration [2]; implicit-explicit integration [21]; subspace integration [22]; and multigrid [23]. Some works are focused on optimizing the cloth behavior and collisions specifically for character animation [24, 25, 26, 27]. There have also been numerous works on improving the mechanical behavior and material models of the cloth [28, 29, 30, 31, 32,

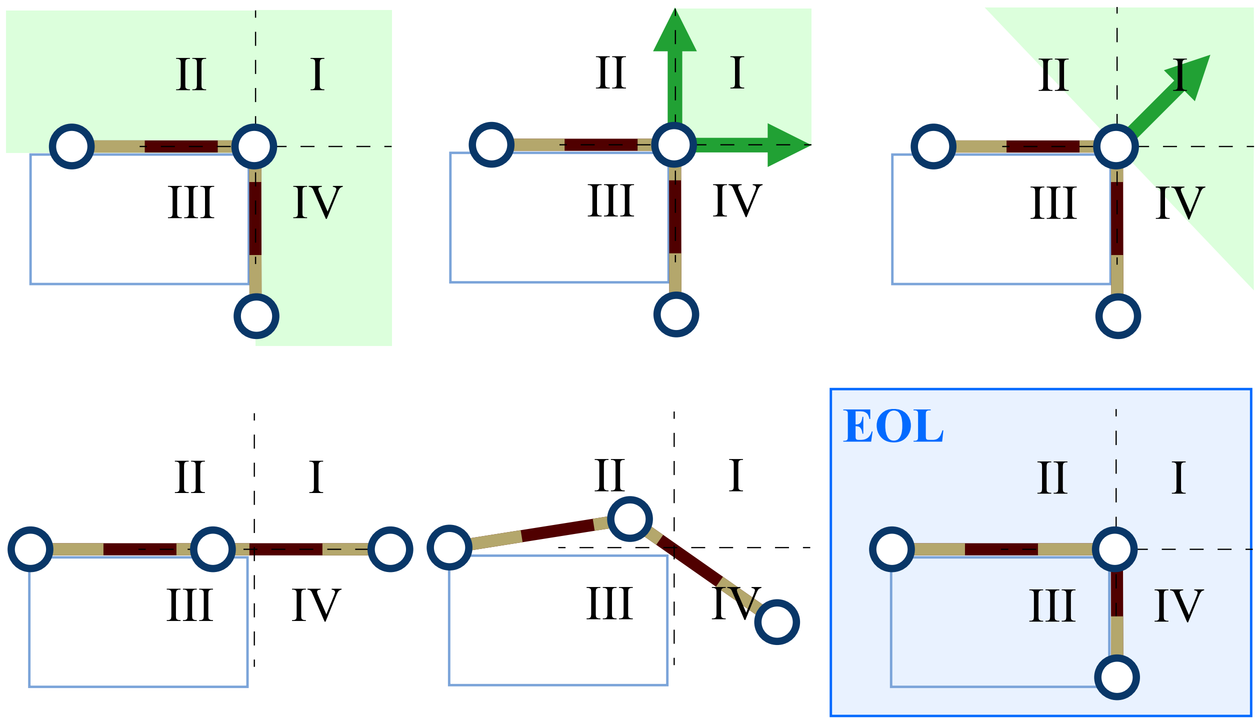


Figure 2.2: (Top) Velocity constraints. Left: ignoring neighboring elements, the colliding vertex should be able to move into quadrants I, II, and IV. Center: if we use both contact normals, the strand gets stuck and cannot be pulled left or down. Right: if we use an averaged contact normal, the strand would lift off of the table unexpectedly. (Bottom left & center) If we pull the strand to the left, the vertices would need to jump to remain collision free. (Bottom right) With the EOL approach, the center node does not need to move but can let the material pass through. Reprinted from [10].

33, 34] and data-driven materials [35, 36, 37, 38]. Some have focused on simulating or adding wrinkles and other high-frequency details [39, 40, 41, 42, 43, 44, 45, 46, 47, 48]. Recently, some works have focused on: interactive editing [49]; precomputation [50]; and simulating individual yarns [51, 52, 53, 54]. Creasing and adaptive remeshing for cloth and paper has also garnered much attention [55, 20, 56, 57, 58, 59, 60, 61].

Perhaps the most important topic for cloth simulation is collision handling. Indeed, from the early days of cloth animation, much of the attention has been focused on resolving collisions, using, for example: collision zones and consistency checking [62]; hybrid constraint forces [63]; rigid impact zones [64, 65]; tangle removal as a post-process [66]; constraint projection [67]; inelastic projection [68]; globally coupled impulses [69]; asynchronous variational integrators and nested barrier potentials [70, 71]; and air meshes [72]. None of these previous works can handle sliding around sharp features due to the *fundamental limitations of the Lagrangian discretization*.

Several previous works attempt to overcome the limitations of purely Lagrangian approaches. The most famous is the Arbitrary Lagrangian-Eulerian (ALE) method, which introduces an additional computational domain, the reference domain, to the standard continuum mechanics picture, which includes the material domain and the spatial domain [73]. This extra domain allows independent movement of the simulation mesh with respect to the material it is tracking. ALE has been used to great effect for simulating complex effects such as solid fluid coupling as well as for simple contacting scenario for 2D or 3D objects [74]. Typically, ALE mesh movement relies on defining an additional interpolation function or energy which drives mesh movement [74]. This is necessary because introducing referential DOFs creates a singularity in the motion description which must be resolved [18]. The Eulerian-on-Lagrangian (EOL) method [17, 18] is a close cousin to ALE with three important differences. First, EOL does not rely on a referential domain, instead chaining together Eulerian or Lagrangian domains into kinematic hierarchies; second, EOL can work with generalized DOFs; and third, EOL methods do not rely on additional functions to determine mesh movement, instead solving for it simultaneously as a function of the physics of the simulated system. Such techniques also require dealing with the inherent motion singularity and the manner in which this is accomplished is part of EOL algorithm design, be it using a least-squares partition of velocities [18] or a reduced coordinate / constraint based approach [17, 19, 51, 52, 75].

In this work, we follow an EOL approach and make several important contributions. First, we extend EOL to the case of general cloth simulation, which is an object with a 2D material domain in 3D world space. This is unique to all previous ALE and EOL methods. (Skin simulation work by Li et al. [19] did not have Lagrangian DOFs; Yarn-level simulation work by Cirio et al. [51, 52] did not have a 2D material domain.) Second, we develop a simple set of rules that allow for automatic computation of Eulerian and Lagrangian motions. Third, we extend a state-of-the-art remesher (ARCSim) [20] to allow conformal remeshing. Finally, our formulation works with contact inequality constraints, allowing not just sliding, but separation of contacting objects, something which is yet to be demonstrated for previous ALE or EOL methods.

2.3 Overview

At each vertex i of the mesh, we store the Lagrangian DOF, $\boldsymbol{x}_i \in \mathbb{R}^3$, and the Eulerian DOF, $\boldsymbol{X}_i \in \mathbb{R}^2$. The Lagrangian DOF represents the world coordinates of the vertex, and the Eulerian DOF represents the material coordinates of the vertex. (One interpretation is that the Eulerian DOFs represent the vertex texture coordinates, and a texel represents a cloth material point.) In traditional purely Lagrangian methods, the material coordinates of all vertices are fixed, whereas in our method, the material coordinates of *some* vertices can vary over time. We call such vertices EOL vertices, and we use these where the cloth is in contact with sharp geometric features. The rest of the cloth is discretized with standard Lagrangian vertices. We follow the standard notation and use $\boldsymbol{q}_i$ to denote the full DOF of a vertex. For a Lagrangian vertex, $\boldsymbol{q}_i = \boldsymbol{x}_i \in \mathbb{R}^3$, whereas for an EOL vertex, $\boldsymbol{q}_i = (\boldsymbol{x}_i \quad \boldsymbol{X}_i)^T \in \mathbb{R}^5$.

We use these EOL vertices wherever the cloth is in contact with sharp edges or corners of another body. For example, if the cloth is in contact with a box, box-face vs. cloth-vertex collisions are handled using standard Lagrangian approaches, whereas box-edge vs. cloth-edge and box-vertex vs. cloth-face collisions are handled using our EOL framework. The border of the cloth must be handled in a special way: cloth corner vertices are always purely Lagrangian, and cloth edge vertices are Eulerian only in the direction along the edge of the cloth. (Using the texture mapping interpretation from above, these conditions imply that the texture of the cloth cannot slide outside of the border of the cloth.) For clarity of exposition, we will assume that all non-cloth bodies are rigid *boxes*, though any rigid/deformable body would work well with our method, as long as there is a way to distinguish between hard and soft edges.

Alg. 2.1 shows the procedure for taking a time step. In the rest of this work, we will go over the important steps of this time stepper. We will first present the core of our framework—EOL cloth dynamics in §2.4. Then, we will describe the set of simple geometric rules for constructing the constraints on the Lagrangian and Eulerian velocities in §2.5. Finally, we will go over our conformal remeshing solution in §2.6.

Algorithm 2.1 EOL Cloth Time Stepper

```
1: while simulating do
2:     Detect collisions                              ▷ External call
3:     Preprocess & Remesh                            ▷ Section 2.6
4:     Compute EOL constraints on new mesh            ▷ Section 2.5
5:     Integrate velocities and positions             ▷ Section 2.4
6: end while
```

2.4 EOL Cloth Dynamics

We discretize the cloth with triangles that conform to the hard edges obtained from the collision detector. Let the three vertices of a triangle be denoted (a, b, c), and $\boldsymbol{X} \in \mathbb{R}^2$ be an arbitrary material point inside this triangle. From the Eulerian DOFs of the triangle $(\boldsymbol{X}_a, \boldsymbol{X}_b, \boldsymbol{X}_c)$ we can compute the barycentric coordinates (α, β, γ) of this material point, $\boldsymbol{X}$, using the standard expression for barycentric coordinates (Eq. A.3). The world position, $\boldsymbol{x} \in \mathbb{R}^3$, corresponding to this material point can then be computed as

$$\boldsymbol{x}(\boldsymbol{X}) = \alpha \boldsymbol{x}_a + \beta \boldsymbol{x}_b + \gamma \boldsymbol{x}_c, \tag{2.1}$$

where $(\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{x}_c)$ are the Lagrangian DOFs of the triangle. The world position is not only a function of the Lagrangian DOFs of the triangle but also of the Eulerian DOFs, since the barycentric coordinates depend on these Eulerian DOFs. Texture mapping again gives us a useful analogy for intuition. Even if we keep the nodal positions (Lagrangian DOFs) fixed, if we modify the nodal texture coordinates (Eulerian DOFs), the cloth moves in world space.

This dependence of the barycentric coordinates on the Eulerian DOFs becomes important when we take the time derivative of Eq. 2.1, since we need to account for the time derivative of the Eulerian coordinates as well. After some rearranging (see Appendix A.1 for the derivation), we obtain

$$\dot{\boldsymbol{x}} = (\alpha \dot{\boldsymbol{x}}_a + \beta \dot{\boldsymbol{x}}_b + \gamma \dot{\boldsymbol{x}}_c) - F(\alpha \dot{\boldsymbol{X}}_a + \beta \dot{\boldsymbol{X}}_b + \gamma \dot{\boldsymbol{X}}_c), \tag{2.2}$$

where $F \in \mathbb{R}^{3\times 2}$ is the deformation gradient of the triangle:

$$\begin{aligned} F &= D_x D_X^{-1} \\ D_x &= \begin{pmatrix} \boldsymbol{x}_b - \boldsymbol{x}_a & \boldsymbol{x}_c - \boldsymbol{x}_a \end{pmatrix} \\ D_X &= \begin{pmatrix} \boldsymbol{X}_b - \boldsymbol{X}_a & \boldsymbol{X}_c - \boldsymbol{X}_a \end{pmatrix}. \end{aligned} \tag{2.3}$$

Here, $D_x \in \mathbb{R}^{3\times 2}$ and $D_X \in \mathbb{R}^{2\times 2}$ are the matrices constructed from the edge vectors of the triangle. The appearance of the deformation gradient here should not surprise us, since its purpose is to map deformations from material space to world space. The negative sign is due to the derivative of the barycentric coordinates with respect to the Eulerian DOFs. This implies that when we change the Eulerian DOF of a node, the cloth material moves in the opposite direction, just like how the motion of texture is the opposite to the motion of texture coordinates.

Finally, for any material point, $\boldsymbol{X}$, inside a triangle, the Jacobian, $\mathsf{J} \in \mathbb{R}^{3\times 15}$, for mapping the generalized velocities of the three vertices of the triangle to the world velocity of the material point is

$$\mathsf{J} = \begin{pmatrix} \alpha I & \beta I & \gamma I & -\alpha F & -\beta F & -\gamma F \end{pmatrix}, \tag{2.4}$$

where I is the 3×3 identity matrix, and F is the deformation gradient from Eq. 2.3. The world space velocity of a material point is then $\dot{\boldsymbol{x}} = \mathsf{J}\dot{\mathsf{q}}$, where $\dot{\mathsf{q}} = (\dot{\boldsymbol{x}}_a\ \dot{\boldsymbol{x}}_b\ \dot{\boldsymbol{x}}_c\ \dot{\boldsymbol{X}}_a\ \dot{\boldsymbol{X}}_b\ \dot{\boldsymbol{X}}_c)^T \in \mathbb{R}^{15}$ is the concatenation of the Lagrangian and Eulerian DOFs of the triangle. The transpose of the Jacobian maps a world force to a generalized force: $\mathsf{f} = \mathsf{J}^T \boldsymbol{f}$. If the material point happens to be at a triangle node, the Jacobian simplifies to $\mathsf{J} = (I\ \ -F) \in \mathbb{R}^{3\times 5}$.

2.4.1 Generalized Inertia

The kinetic energy of a triangle can be expressed as

$$T = \frac{1}{2} \int_A \rho\, \dot{\boldsymbol{x}}^T \dot{\boldsymbol{x}}\, dA, \tag{2.5}$$

where ρ is the area density, and the integral is over the area of the triangle in material space spanned by $(\boldsymbol{X}_a, \boldsymbol{X}_b, \boldsymbol{X}_c)$. Using α and β as the variables of integration over the triangle, we obtain

$$T = \frac{1}{2} \int_{\alpha=0}^{1} \int_{\beta=0}^{1-\alpha} \rho\, \dot{\boldsymbol{x}}^T \dot{\boldsymbol{x}}\, A\, d\beta\, d\alpha, \tag{2.6}$$

where $A = |\det(D_X)|/2$ is the area of the triangle in material space. Integrating out α and β and rearranging the terms, we arrive at

$$T = \frac{1}{2} \dot{\mathsf{q}}^T \mathsf{M} \dot{\mathsf{q}}, \tag{2.7}$$

where M is the generalized inertia, obtained by using the Jacobian from Eq. 2.4 and then integrating the result:

$$\mathsf{M} = \frac{\rho A}{12} \begin{pmatrix} 2I & I & I & -2F & -F & -F \\ \cdot & 2I & I & -F & -2F & -F \\ \cdot & \cdot & 2I & -F & -F & -2F \\ \cdot & \cdot & \cdot & 2F^TF & F^TF & F^TF \\ \cdot & \cdot & \cdot & \cdot & 2F^TF & F^TF \\ \cdot & \cdot & \cdot & \cdot & \cdot & 2F^TF \end{pmatrix}, \tag{2.8}$$

where the dots indicate symmetric terms. This generalized inertia matrix is 15×15, corresponding to the 9 Lagrangian DOFs and 6 Eulerian DOFs of the triangle. The upper left 3×3 blocks of the inertia matrix correspond to the Lagrangian DOFs and are constant over time, an advantage exploited by Lagrangian simulators. The rest of the inertia matrix must be computed at every time step, since F is a function of both Lagrangian and Eulerian DOFs.

2.4.2 Generalized Forces

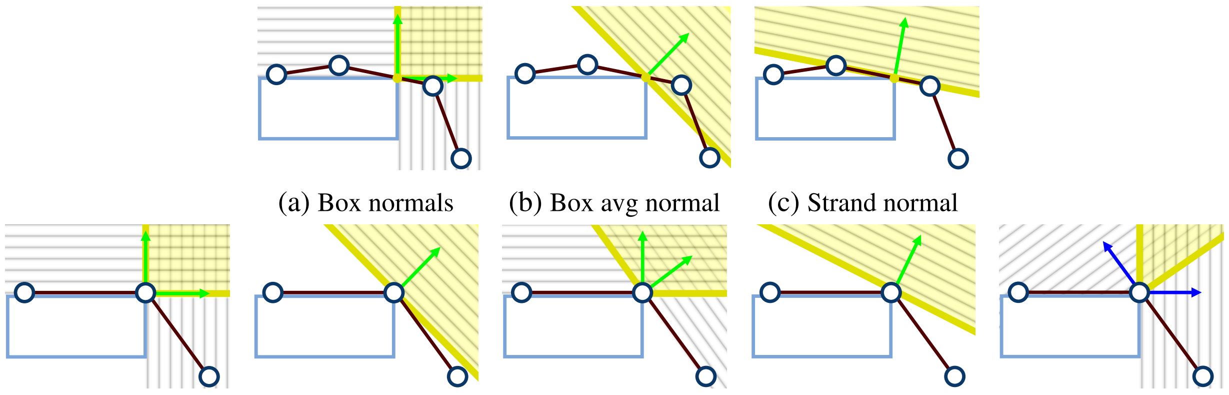
The EOL framework works with any set of forces. For each node, we compute its world force, $\boldsymbol{f}$, and then use the Jacobian transpose from Eq. 2.4 to map this world force into its Lagrangian and Eulerian force components:

$$\begin{pmatrix} \boldsymbol{f}^L \\ \boldsymbol{f}^E \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ -F^T \boldsymbol{f} \end{pmatrix}. \tag{2.9}$$

The corresponding stiffness matrix, K, for a node is similarly transformed into its Lagrangian and Eulerian components as

$$\begin{pmatrix} K^{LL} & K^{LE} \\ K^{EL} & K^{EE} \end{pmatrix} = \begin{pmatrix} K & -KF \\ -F^T K & F^T K F \end{pmatrix}. \tag{2.10}$$

In our current implementation, we use the corotated linear material model for membrane forces [76, 32, 57] and discrete Willmore energy for bending forces [31, 34]. Whenever we need the deformation gradient at a vertex, we simply take an element-wise average. An alternative approach would be to take the polar decomposition to interpolate the rotation separately.



(a) Box normals (b) Box avg normal (c) Strand normal

(d) Box normals (e) Box avg normal (f) Strand normals (g) Strand avg normal (h) Strand binormals

Figure 2.3: Possible contact constraints for a strand-box collision without conformal remeshing (a-c) and with conformal meshing (d-h). (a) If we use both normals of the box corner, the strand would be over constrained. (b & c) If we use the averaged box normal or the strand normal, it would result in an unnatural lift off. (d-g) Similar choices exist with conformal remeshing. Reprinted from [10].

2.4.3 Equations of Motion

In our current implementation, we use the linearly implicit integration scheme at the velocity level, popularized by the work on efficient cloth simulation by Baraff and Witkin [2]:

$$\left(\mathsf{M} - h^2\mathsf{K}\right)\dot{\mathsf{q}}^{(k+1)} = \mathsf{M}\dot{\mathsf{q}}^{(k)} + h\mathsf{f}, \tag{2.11}$$

where h is the time step, and the superscript k indicates the current time step. The EOL framework is not tied to a specific integration scheme, and should work equally well with other schemes.

Adding the EOL equality and inequality constraints, which are described later in §2.5, and applying Gauss's Principle of Least Constraint [77], we arrive at the following, which is solved for the new velocities, $\dot{\mathsf{q}}^{(k+1)}$.

$$\begin{aligned} \underset{\dot{\mathsf{q}}}{\text{minimize}} \quad & \frac{1}{2}\dot{\mathsf{q}}^T\tilde{\mathsf{M}}\dot{\mathsf{q}} - \dot{\mathsf{q}}^T\tilde{\mathsf{f}} \\ \text{subject to} \quad & \mathsf{A}_{\text{eq}}\dot{\mathsf{q}} = 0 \\ & \mathsf{A}_{\text{ineq}}\dot{\mathsf{q}} \geq 0, \end{aligned} \tag{2.12}$$

where $\tilde{\mathsf{M}} = \mathsf{M} - h^2\mathsf{K}$, and $\tilde{\mathsf{f}} = \mathsf{M}\dot{\mathsf{q}}^{(k)} + h\mathsf{f}$. Finally, the Lagrangian and Eulerian DOFs are both updated as $\mathsf{q}^{(k+1)} = \mathsf{q}^{(k)} + h\dot{\mathsf{q}}^{(k+1)}$.

2.5 EOL Cloth Constraints

Before we describe the constraints we apply to the EOL vertices (§2.5.2), we first review how one would constrain a Lagrangian cloth in contact with sharp geometric features (§2.5.1). As we saw in Fig. 2.2, there are difficulties in dealing with position-level as well as velocity-level constraints. Here, however, we expand only on velocity-level constraint problems.

2.5.1 Constraints for Lagrangian Cloth

Let us consider a 1D strand for illustration. Without conformal remeshing, the strand contacts the box corner at some element, as shown in Fig. 2.3(a-c). There are two normals at the box corner, as shown in (a), as well as the single normal from the strand element, as shown in (c). If we choose (a), then the strand becomes over-constrained and locked, unable to move left or down. Instead, we must average the box normals (b) or use the strand normal (c). Unfortunately, this causes the strand to lift off unnaturally, since the constrained point must stay in the positive halfspace of the constraint.

If we apply conformal remeshing, as shown in Fig. 2.3(d-h), then there are now two normals to choose from the two neighboring strand elements (f). Using these two strand normals still results in an over-constrained configuration, since it prevents the colliding node to go below the top of the box. A Lagrangian simulator must still use averaged normals (e) or (g), since otherwise the strand becomes over-constrained. Unfortunately, with (e) or (g), the cloth is now under-constrained, as these constraints may allow the cloth to penetrate the box. To summarize, conformal remeshing helps a Lagrangian simulator when the cloth is static, but it does not resolve the problem stemming from sliding motion.

2.5.2 Constraints for EOL Cloth

With the EOL approach, we get around this problem by using the Eulerian DOFs to move the cloth around the sharp features.

Before we describe how we construct the Lagrangian and Eulerian constraints, we first mention an important consideration when dealing with EOL methods: the inertia matrix in Eq. 2.8 can become singular depending on the configuration of the cloth. This is because for some Eulerian velocities, there may be a corresponding Lagrangian velocity that exactly cancels out the motion of the cloth material. As an illustration, let us assume that an undeformed cloth is laid flat on the X-Y plane, and that there is an EOL vertex in the middle of the cloth. In this configuration, we can move the Lagrangian DOFs of the vertex (*i.e.,* vertex position) in one direction and the Eulerian DOFs (*i.e.,* vertex texture coordinates) in the other direction[†] so that the actual cloth does not move in world space. Thus, this combination of Lagrangian and Eulerian velocities lies in the nullspace of the inertia matrix. One potential approach for dealing with this singularity is to use

[†]Note the negative sign in Eq. 2.2 (derivation in §A.1).

least squares to solve for the largest Lagrangian velocities first [18, 78]. In our setting of cloth simulation, however, we can exactly account for the singularity by using the local geometry of the cloth and the box. In the rest of this section, we describe a simple set of rules for constructing these constraints on the Lagrangian and Eulerian velocities of an EOL vertex.

There are two cases we must consider: contact with box corner and contact with box edge, which are discussed in the following two paragraphs. In both cases, we use the box normal cone (3D analogues of Fig. 2.3(d)). Alg. 2.2 summarizes the procedure for constructing these constraints on the EOL vertices.

Corner: The Lagrangian constraint for a corner EOL vertex is constructed from an orthonormal frame at the corner of the box. Let the box normals be denoted $\boldsymbol{n}_1$, $\boldsymbol{n}_2$, and $\boldsymbol{n}_3$. The Lagrangian velocity constraint is then $\boldsymbol{n}_1^T \dot{\boldsymbol{x}} \geq 0$, $\boldsymbol{n}_2^T \dot{\boldsymbol{x}} \geq 0$, and $\boldsymbol{n}_3^T \dot{\boldsymbol{x}} \geq 0$. We do not need any constraints on the Eulerian velocity for a corner EOL vertex, since we use these Eulerian DOFs to allow the cloth to slide freely around the corner.

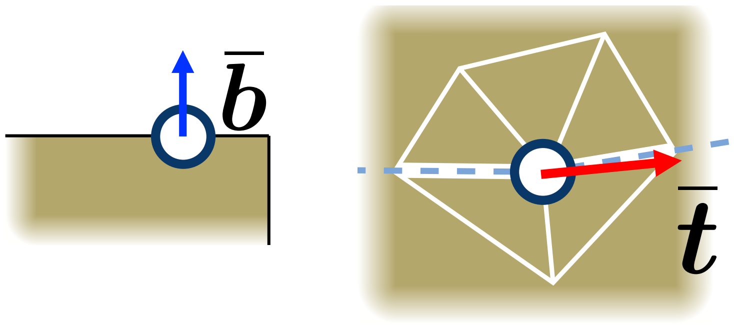


Figure 2.4: (Left) Vector orthogonal to the cloth border in material space. (Right) Vector along the average material space tangent constructed from the aligned and colliding edges. Reprinted from [10].

Edge: The Lagrangian constraint for an edge EOL vertex is constructed from the box normals, $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$. We want the vertex to be able to move freely (in the Lagrangian sense) along the box tangent but be able to lift off if necessary. The Lagrangian velocity constraint is then $\boldsymbol{n}_1^T \dot{\boldsymbol{x}} \geq 0$ and

Algorithm 2.2 EOL Constraint Generation

```
1: for each EOL vertex $v$ do
2:     if $v$ colliding with box corner then
3:         Lagrangian Constraint: $\boldsymbol{n}_1^T\dot{\boldsymbol{x}} \geq 0,\ \boldsymbol{n}_2^T\dot{\boldsymbol{x}} \geq 0,\ \boldsymbol{n}_3^T\dot{\boldsymbol{x}} \geq 0$
4:         Eulerian Constraint: none
5:     else if $v$ colliding with box edge then
6:         Lagrangian Constraint: $\boldsymbol{n}_1^T\dot{\boldsymbol{x}} \geq 0,\ \boldsymbol{n}_2^T\dot{\boldsymbol{x}} \geq 0$
7:         if $v$ on cloth border then
8:             Eulerian Constraint: $\bar{\boldsymbol{b}}^T\dot{\boldsymbol{X}} = 0$
9:         else
10:            Eulerian Constraint: $\bar{\boldsymbol{t}}^T\dot{\boldsymbol{X}} = 0$
11:        end if
12:    end if
13: end for
```

$\boldsymbol{n}_2^T\dot{\boldsymbol{x}} \geq 0$. The Eulerian velocity constraint depends on whether the vertex is on the cloth border or not. For EOL vertices on the cloth border, the Eulerian constraint is $\bar{\boldsymbol{b}}^T\dot{\boldsymbol{X}} = 0$, where $\bar{\boldsymbol{b}}$ is the vector orthogonal the cloth border in material space, Fig. 2.4a. This constraint ensures that the cloth material remains affixed to the border. For internal EOL vertices, the Eulerian constraint is $\bar{\boldsymbol{t}}^T\dot{\boldsymbol{X}} = 0$, where $\bar{\boldsymbol{t}}$ is the averaged material space tangent constructed from the two edges of the colliding vertex that are lying on the box edge. (In Fig. 2.4b, the box collision in material space is shown in dotted blue, and the two edges are shown in thick white.) This constraint ensures that any motion of the cloth along the box edge is realized by the Lagrangian DOF rather than the Eulerian DOF.

All of the local constraints described in this section are collected into global matrices so that the constraints can be written as $\mathsf{A}_{\text{eq}}\dot{\mathsf{q}} = 0$ and $\mathsf{A}_{\text{ineq}}\dot{\mathsf{q}} \geq 0$ where $\dot{\mathsf{q}}$ is the concatenation of all nodal Lagrangian and Eulerian velocities.

2.6 Conformal Remeshing

To remesh the cloth, we use ARCSim (v0.3.1) [20, 56, 58], which has curvature based metrics to help avoid visually unappealing changes to the triangle mesh. In Fig. 2.5, we show a typical remeshing scenario when the cloth first make contact with the box. The cloth mesh is shown

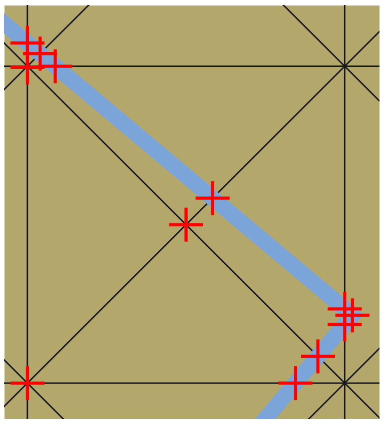


Figure 2.5: Typical remeshing scenario. Reprinted from [10].

in black, the box mesh is shown in blue, and the collisions are shown in red. Our goal is to remesh the cloth so that it conforms to the box. Even though ARCSim has the ability to "preserve" certain edges during triangulation, it does not work out-of-the-box for conformal remeshing for two reasons:

- Our preserved edges move around in the material domain, potentially creating extremely thin triangles.
- Collisions often occur very close to each other. Simply marking all collisions as preserved (*e.g.,* all red crosses touching the blue box mesh in Fig. 2.5) does not allow ARCSim enough freedom to remesh properly.

Therefore, we preprocess the mesh before we send the mesh to ARCSim. This preprocessing procedure is applicable to all conformal remeshing, either with standard Lagrangian or with our EOL framework.

2.6.1 Preprocessing

Before remeshing with ARCSim, we first scan a list of collision features (*e.g.,* box edges and box corners), along with a list of existing conformal vertices. (*Conformal* vertices and edges refer to cloth vertices and edges that are in contact with sharp features. In our EOL framework, conformal vertices are our EOL vertices.) We insert new conformal vertices into our mesh either by splitting faces or edges of the mesh at untracked features. We then sort and connect the conformal vertices to form conformal edges. Then we repeat the following steps, iterating through all triangles incident to at least one conformal vertex, until no more changes are made. In these steps, when we collapse an edge between a conformal vertex and a non-conformal vertex, we always collapse toward the conformal vertex.

- Collapse non-conformal edges that are below a threshold. (For our examples, we used 1% of the cloth length as our threshold.)
- Collapse conformal edges that are below a threshold. The edge can be collapsed in either direction unless one of the conformal vertices is a cloth border/corner, which must be preserved.
- For an ill-conditioned triangle, we split one of its edges to insert a new vertex, which will then be removed during the next iteration of the preprocessing loop.
  - If it has one conformal vertex, split the edge opposite to it.
  - If it has two conformal vertices, split the conformal edge.
  - If it has three conformal vertices, split the non-conformal edge. We assume that a triangle cannot have three conformal edges—*i.e.,* the cloth triangles are sufficiently small compared to the sharp features.

This preprocessing scheme has worked well for our examples, but we do not have a convergence proof. In some cases, it may not be possible to remove all ill-conditioned triangles while preserving conformal features. In such cases, the time step must be slowed down accordingly. In our experience, EOL simulations produce much fewer ill-conditioned triangles than conformal Lagrangian simulations, making it much more robust.

We do not allow conformal remeshing near the border of the cloth. Specifically, before we insert a conformal edge that touches the cloth border, we check to make sure that the angle between the edge and the border is above a threshold. This prevents the formation of thin triangles incident to cloth borders. In the absence of collisions with sharp features, our algorithm reverts to a standard non-conformal Lagrangian cloth simulation using ARCSim as the remesher.

2.6.2 Velocity Transfer

Whenever the cloth is remeshed, the velocities must be interpolated at the new vertex positions. There are four ways in which a new vertex can be introduced, and we use the following scheme to compute the new velocity of the newly inserted vertex.

- New Lagrangian vertex inside a Lagrangian triangle. This is the standard case, and we simply use barycentric averaging to compute the vertex's Lagrangian velocity.
- New Lagrangian vertex inside an EOL triangle. If the newly inserted vertex happens to be inside a triangle with one or more EOL vertices, we must first compute the world velocity at these EOL vertices using Eq. 2.2. Then we use barycentric averaging to compute the inserted vertex's Lagrangian velocity.
- New EOL vertex from an EOL edge split. If the remesher inserts a new EOL vertex by splitting an edge between two EOL vertices, we simply interpolate both the Lagrangian and Eulerian velocities of the two EOL vertices.
- New EOL vertex from collision. Whenever we insert an new EOL vertex as a result of a collision, we first compute the world velocity, $\dot{\boldsymbol{x}}^w$, at the vertex by barycentric averaging. This world velocity is composed of the Lagrangian component, $\dot{\boldsymbol{x}}$, and Eulerian component, $\dot{\boldsymbol{X}}$, and can be expressed as $\dot{\boldsymbol{x}}^w = \dot{\boldsymbol{x}} - F\dot{\boldsymbol{X}}$ (cf. Eq. 2.2). We put as much of this world velocity into the Eulerian velocity as possible by solving a small constrained optimization problem for $\dot{\boldsymbol{X}}$: min. $\frac{1}{2}\|\dot{\boldsymbol{x}}^w + F\dot{\boldsymbol{X}}\|^2$ s.t. $A_{\text{eq}}\dot{\boldsymbol{X}} = 0$. In other words, we minimize the Lagrangian velocity subject to equality constraints from §2.5. This turns into a 3×3 linear system:

$$\begin{pmatrix} F^T F & A_{\text{eq}}^T \\ A_{\text{eq}} & 0 \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{X}} \\ \lambda \end{pmatrix} = \begin{pmatrix} -F^T \dot{\boldsymbol{x}}^w \\ 0 \end{pmatrix}. \tag{2.13}$$

  The Lagrangian velocity is then computed as $\dot{\boldsymbol{x}} = \dot{\boldsymbol{x}}^w + F\dot{\boldsymbol{X}}$.

If an EOL vertex lifts off, it becomes a Lagrangian vertex. In this case, the new Lagrangian velocity must take into account the Eulerian velocity from the last step, using Eq. 2.2.

2.7 Results

We implemented our system in C++ and ran the simulations on a consumer desktop with an Intel Core i7-7700 CPU @ 3.6 Ghz and 16 GB of RAM. The code is single-threaded and uses Eigen for linear algebra and Mosek for quadratic programs. In Figs. 2.6 and 2.8, we show some still frames from the simulations. (Please also see the accompanying video.) In Table 2.1, we show the performance numbers.

Cloth sliding over box EDGE

In this example, shown in Fig. 2.6, the cloth is pulled over an edge of a box. For comparison, we also show how Lagrangian cloth simulations behave under the same scenario. Unless otherwise stated, all Lagrangian simulations use averaged constraints (Fig. 2.3(e)) at sharp features, which approximates the underlying box geometry as smooth.

- With static regular discretization, the cloth is able to form a sharp bend because the box edge happens to be aligned with the cloth mesh. However, bending artifacts become obvious as soon as we pull the cloth.
- With static irregular discretization, the cloth is unable to form a sharp bend.
- With non-conformal remeshing (*i.e.,* vanilla ARCSim), the cloth is able to bend *semi*-sharply only if we allow ARCSim to generate many triangles. Moreover, when we pull the cloth, the cloth lifts unnaturally because it is not able to bend exactly at the box edge, even at high-resolution. This artificial bending energy is completely independent of the material bending stiffness.
- With naïve conformal remeshing (*i.e.,* vanilla ARCSim with "preserved" edges), the simulation abruptly halts when the cloth hits the box because the collision detector generates too many contact points, which creates too many small "preserved" edges to be passed to ARCSim. See Fig. 2.5.
- With conformal remeshing (*i.e.,* our preprocessing + ARCSim), the cloth is able to bend sharply. When we pull the cloth, however, averaged normal constraints (Fig. 2.3(e)) cause the cloth to

lift unnaturally, at any resolution. Again, this artificial bending energy is independent of the material bending stiffness. Also, because of the Lagrangian vertex motion very close to the sharp features, the collision detector parameters must be highly tuned to detect all the collisions correctly. Furthermore, when the cloth is bent sharply, this liftoff *always* causes the cloth to penetrate the box, which must be projected back. (As an aside, this implies that continuous collision detection cannot be used.)

- With conformal remeshing *and* with proper cone constraints (Fig. 2.3(d)), the cloth is again able to bend sharply, but as soon as we pull the cloth, it locks catastrophically. (This result is not included in Table 2.1.)
- With our EOL discretization, the cloth smoothly slides around the edge with a perfectly sharp bend. Because the conformal vertices stay on the box edge, no penetrations are introduced. Also, we are able to use proper cone constraints (Fig. 2.3(d)) without having to approximating the underlying surface as smooth.

Cloth sliding over a WIRE

Even though conformal Lagrangian simulation works for the EDGE example, when the bending angle becomes more extreme, *it is no longer able to slide smoothly* because with the averaged constraint (Fig. 2.3(e)), the conformal vertices *can only move horizontally*. Even a small amount of horizontal movement causes large penetrations in the cloth, making subsequent conformal remeshing challenging. Please see Fig. 2.7 and the video for some of these Lagrangian failure cases. Our EOL cloth, on the other hand, works as expected.

Cloth sliding over NAILS

The EOL framework works just as well with sharp points. In this example, we pull the cloth over a bed of nails. This is an artificially unrealistic scenario, because we would expect the cloth to snag due to the individual cloth fibers getting caught by the nail tips. However, it is important to note that Lagrangian simulations snag due to its discretization rather than by proper physics. With our EOL approach, it would be possible to simulate this snagging behavior properly by including additional external forces.

Cloth sliding through JAWS

We further demonstrate the robustness of our approach by pulling the cloth through "jaws of death." As before, no artificial snagging behavior is observed.

Scripted box PUSH

In this example, we show the cloth being pushed by a scripted box to illustrate how effectively the cloth is able to slide over the box. This example also highlights the proper lift-off of EOL vertices due to our inequality constraints.

Scripted box THROW

This example shows a more dynamic cloth making contact with, sliding over, and then lifting over a scripted box. Many EOL edges and points are created on the fly, as the cloth comes in contact with various edges and corners of the box. Again, no snagging behavior is observed.

Coarse preview with LO-RES cloth

We show that the EOL framework works very well for generating coarse previews of simulations involving sharp features. With a static Lagrangian simulation, we lose the details around the sharp features. With EOL, we are able to retain the sharp features even when the cloth starts to slide off. There are some obvious popping artifacts caused by remeshing along the sharp edges, but this is a side effect of any method that aligns cloth geometry with object geometry, *i.e.,* conformal LAG and EOL, and also occurs at higher resolutions but is less visible. With EOL, we do not get any snagging behavior even at low-res, which, in scenarios where overall quality of motion is paramount, is more important than visual fidelity.

Cloth with FRICTION

In this example, the cloth is dropped between four boxes, hitting their corners. With EOL, the cloth smoothly slides over the corners and edges of the boxes, and without friction, the cloth eventually falls naturally off of the boxes with no bending artifacts or locking. When we repeat the simulation with friction by applying the impulse-based friction formulation of Bridson et al. [65], the cloth stops rather than falling. For each vertex, we first compute the scalar friction multiplication factor using the world velocity of the vertex (Eq. 2.2). For Lagrangian vertices, we apply this factor to the tangential component of the velocity as usual. For EOL vertices on a box corner, we apply the factor to just the Eulerian velocity, since the tangential motion is encoded fully by the Eulerian velocity. For EOL vertices on a box edge, we apply the factor to the Eulerian velocity and the tangential component of the Lagrangian velocity (along the box edge).

2.8 Conclusion

We introduced a novel Eulerian-on-Lagrangian cloth simulation framework that can robustly simulate the cloth sliding over sharp features, a scenario that cannot be simulated by other methods due to the fundamental limitation of purely Lagrangian simulators. In our framework, we use both Eulerian and Lagrangian DOFs for vertices at the sharp features. We derive the equations of motion for elements that involve these special vertices. We define a simple set of geometric rules for constraining these vertices to remove the redundancy that exists between the Eulerian and Lagrangian DOFs. We extend a state-of-the-art remesher (ARCSim) for conformal remeshing around sharp features. Finally, we show various examples of how our framework is able to handle difficult scenarios involving sliding over sharp edges and corners.

2.8.1 Future Work

Our work is the first work to use an Eulerian discretization for cloth, and so we hope that it opens many avenues of future work. We have released our source code to encourage future research in Eulerian-on-Lagrangian cloth simulation,[‡] some directions of which we discuss next. A limitation common to all adaptive cloth simulators is that the remesher does not take into account the sharp features in the environment. Collision-aware remeshing that avoids interpenetrations would benefit not only our work but all other work on adaptive cloth. Another limitation of our

[‡]`https://github.com/sueda/eol-cloth`

current implementation is that we are computing the per-vertex deformation gradient by element-wise averaging of the incident triangle deformation gradients. We expect to see better energy behavior if we compute this average using Lie algebra. Furthermore, allowing EOL style contact handling for cloth-cloth, cloth-fluid and cloth-deformable body interactions would allow for more seamless simulations of such phenomena. In the current implementation, we simply default to purely Lagrangian handling for any contact which is not between cloth and a static rigid body. Another interesting avenue of future work is to remove the restriction that the border vertices must be Lagrangian (corner nodes cannot be Eulerian; edge nodes can only be Eulerian along the edge tangent). With this modification, we expect to see better transition of EOL to Lagrangian vertices near the border of the cloth. Next, while our work focuses on finite element simulation of cloth, we believe it can be extended to other simulation techniques such as Projective [79] and Position-Based Dynamics [80]. Finally, even though our approach is the first to enable smooth sliding cloth, it still requires remeshing around sharp features. Removing this dependence on remeshing would improve efficiency and robustness of both conformal Lagrangian and EOL cloth simulations.

Table 2.1: Performance numbers for the examples. #F: Maximum number of faces. %E: Maximum EOL vertex percentage. %CD: Percentage spent in collision detection. %RM: Percentage spent in remesh. %MF: Percentage spent in matrix fill. %VI: Percentage spent in velocity integration (QP). T: Total time per step (ms). Some scenes are run multiple times with different settings. Regular: Lagrangian simulation with a static regular mesh. Irregular: Lagrangian simulation with a static irregular mesh. Non-conformal: Lagrangian simulation with non-conformal remeshing (ARCSim). Conformal: Lagrangian simulation with conformal remeshing (ARCSim with our preprocessing). EOL: Our EOL simulation. Reprinted from [10]

Scene	#F	%E	%CD	%RM	%MF	%VI	T (ms)
EDGE (reg.)	2116	-	1.2	-	6.6	92.1	436.2
EDGE (irreg.)	2000	-	0.9	-	5.0	94.0	589.2
EDGE (non-conf.)	2521	-	0.9	4.7	5.4	88.9	638.9
EDGE (conf.)	2622	-	1.8	5.2	4.5	88.6	729.6
EDGE (EOL)	2971	2.4	0.9	2.5	7.7	88.8	1101.0
WIRE (reg.)	1936	-	0.3	-	3.4	96.3	827.2
WIRE (irreg.)	2055	-	0.2	-	3.3	96.5	896.5
WIRE (non-conf.)	2048	-	2.6	4.1	2.8	96.9	1028.7
WIRE (conf.)	2022	-	0.6	4.0	3.5	91.9	728.3
WIRE (EOL)	2048	2.4	0.2	3.0	7.7	89.1	1058.7
NAILS (EOL)	2012	1.9	0.6	2.9	7.9	88.6	804.2
JAWS (EOL)	3380	0.3	0.3	9.2	21.8	68.7	503.9
PUSH (EOL)	2110	3.1	1.4	6.2	16.7	75.6	454.9
THROW (EOL)	1913	3.9	0.8	3.9	9.0	86.2	568.4
LO-RES (reg.)	256	-	3.2	-	15.8	81.0	18.1
LO-RES (EOL)	133	6.0	16.8	6.2	23.1	53.9	16.3
FRICTION (EOL)	2927	2.9	6.7	8.4	23.6	61.3	451.5

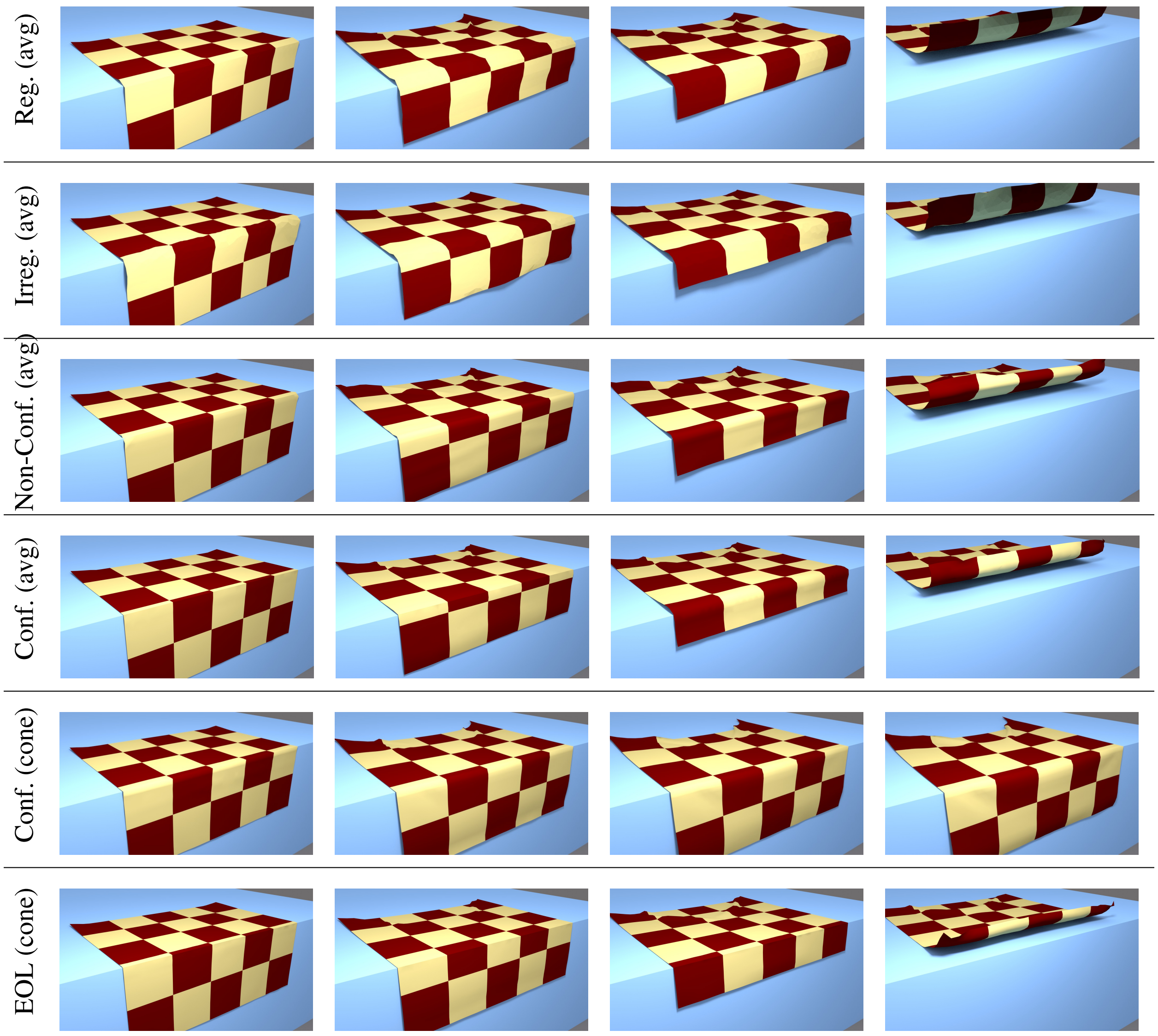


Figure 2.6: Still shots from EDGE simulations. From top to bottom: LAG static regular with averaged constraints, LAG static irregular with averaged constraints, LAG non-conformal with averaged constraints, LAG conformal with averaged constraints, LAG conformal with cone constraints, EOL with cone constraints. Reprinted from [10].

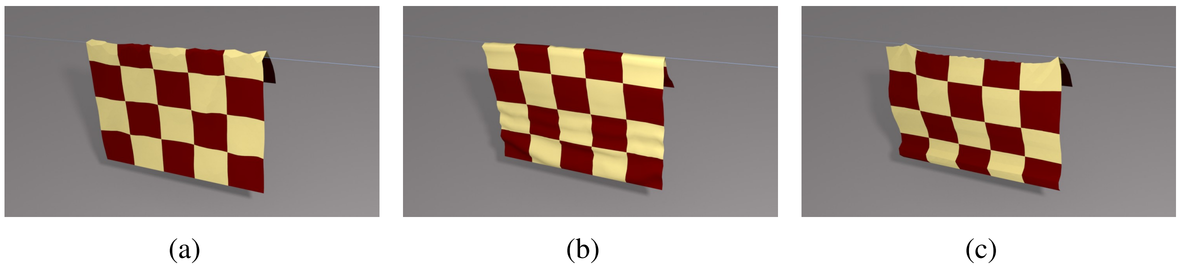


Figure 2.7: Lagrangian failure cases for WIRE. (a) Cloth cannot bend sharply. (b) Cloth bunches up and cannot slide over the wire. (c) Cloth falls off the wire. Reprinted from [10].

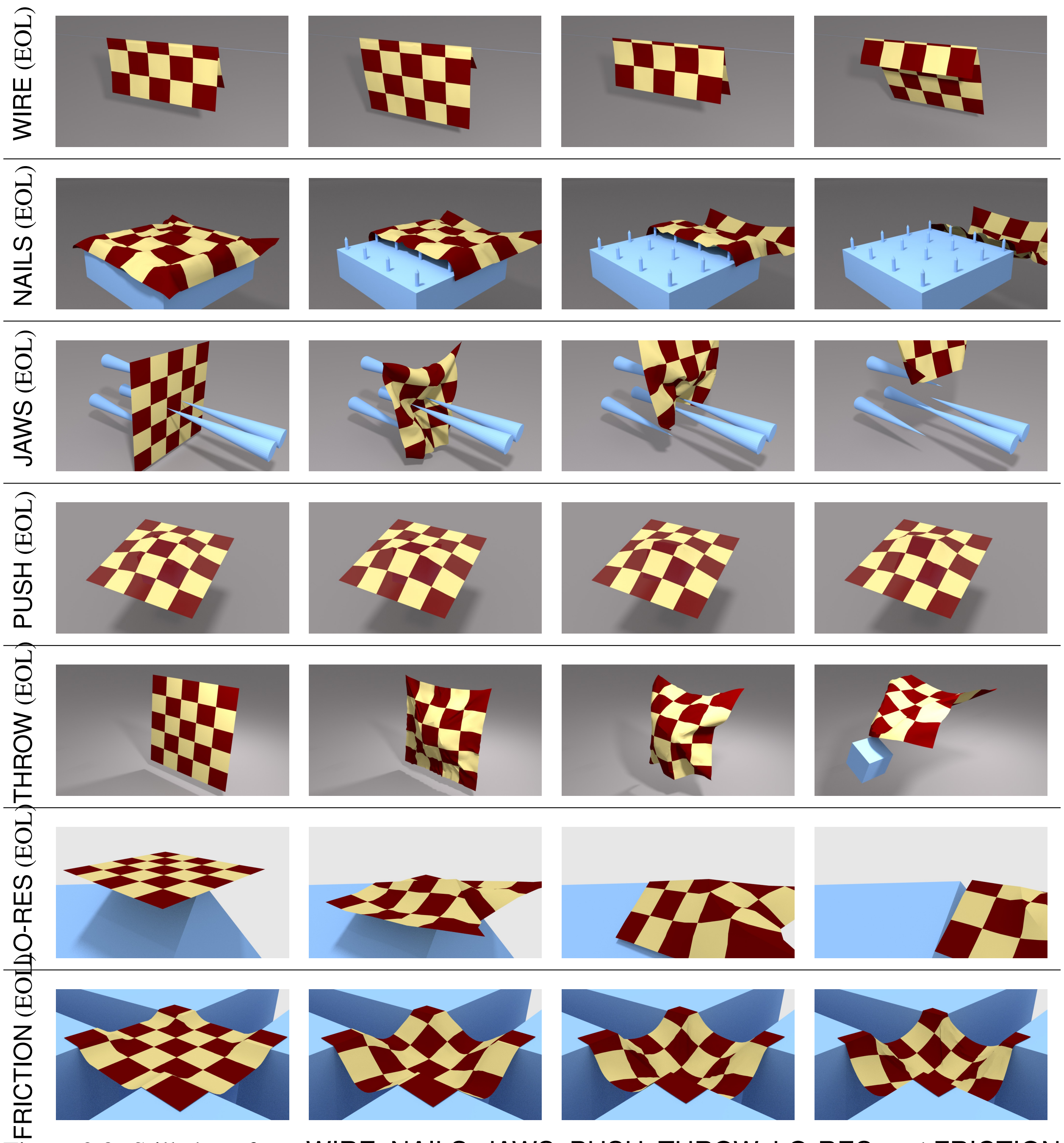


Figure 2.8: Still shots from WIRE, NAILS, JAWS, PUSH, THROW, LO-RES, and FRICTION simulations. Reprinted from [10].

3. BILATERAL STAGGERED PROJECTIONS FOR JOINTS[*]

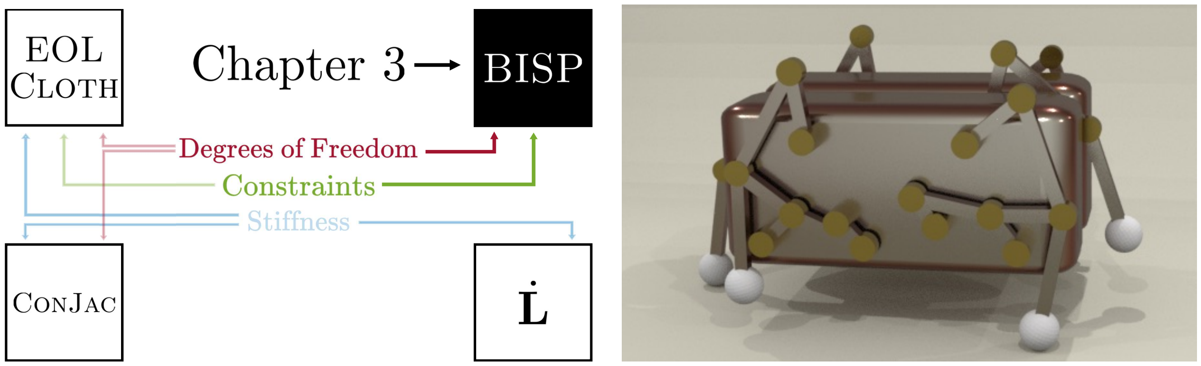


Figure 3.1: (Left) Bilateral Staggered Projections for Joints explores the constrained DOFs of articulated bodies in a reduced state. (Right) Still frame of a Klann walker linkage simulation that utilizes BISP to compute joint frictional forces. Reprinted from [11].

It is well known that the dynamics of articulated rigid bodies can be solved in $O(n)$ time using a recursive method, where n is the number of joints. However, when elasticity is added between the bodies (*e.g.,* damped springs), with linearly implicit integration, the stiffness matrix in the equations of motion breaks the tree topology of the system, making the recursive $O(n)$ method inapplicable. In such cases, the only alternative has been to form and solve the system matrix, which takes $O(n^3)$ time. We propose a new approach that is capable of solving the linearly implicit equations of motion in near linear time. Our method, which we call REDMAX, is built using a combined reduced/maximal coordinate formulation. This hybrid model enables direct flexibility to apply arbitrary combinations of constraints and contact modeling in both reduced and maximal coordinates, as well as mixtures of implicit and explicit forces in either coordinate representation.

[*]Reprinted with permission from "Redmax: Efficient & flexible approach for articulated dynamics" by Y. Wang, N. J. Weidner, M . A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, *ACM Trans. Graph.*, vol 38, no 4, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3306346.3322952. Copyright 2019 by Association for Computing Machinery.

We highlight REDMAX's flexibility with seamless integration of deformable objects with two-way coupling, at a standard additional cost. We further highlight its flexibility by constructing an efficient internal (joint) and external (environment) frictional contact solver that can leverage bilateral joint constraints for rapid evaluation of frictional articulated dynamics.

3.1 Introduction

Articulated rigid body dynamics has many applications in various disciplines, including biomechanics, robotics, aerospace, and computer graphics. It has been extensively studied starting in the 1960s (*e.g.,* [81]), but it was not until the 1980s that an $O(n)$ algorithm, where n is the number of joints or bodies, became widely known [82]. This algorithm and its variants are based on a recursive formulation, where various quantities are computed recursively based on the tree structure of the mechanism. These algorithms take advantage of dynamics represented using "reduced" coordinates, where a minimal set of degrees of freedom (DOFs), such as joint angles for revolute joints and relative translations for prismatic joints, are used to represent the state of the system. An alternate approach that uses "maximal" coordinates has also been studied.[†] For example, an $O(n)$ method for maximal coordinates was discovered by Baraff [83]. However, constraints need to be applied to model joints, and these constraints must be stabilized to avoid drift [84, 85]. On the other hand, reduced coordinates do not require any stabilization, since reduced coordinates only allow configurations that satisfy the joint constraints. Loops are handled with constraints in either approach, but in practice, stabilizing a few loop constraints is much easier than stabilizing the whole structure. Furthermore, reduced coordinates are in general faster, because the number of DOFs is much smaller (*e.g.,* 1/6 the size), and no constraints are required. Also, Baraff [83] notes that there is anecdotal evidence that larger time steps are possible using reduced coordinates.

One of the advantages of maximal coordinates is that it is more intuitive—it is easier to add various implicit/explicit forces and to combine with other deformable objects. To address this point, Wang et al. [11] introduced a formulation of dynamics called REDMAX. Any combina-

[†]Maximal coords are also called absolute coords or Cartesian coords; reduced coords are also called generalized coords or minimal coords.

tion of reduced/maximal forces can be added implicitly or explicitly, and any combination of reduced/maximal constraints can be handled. Furthermore, it becomes trivial to get full two-way coupling between a deformable object (such as an FEM) and the articulated rigid bodies.

REDMAX introduced a few key contributions to articulated dynamics:

- A near linear approach for articulated dynamics, even in the presence of the maximal stiffness matrix, based on our novel matrix-free Projected Block Jacobi Preconditioner.

- A formulation that exposes both maximal and reduced degrees of freedom, allowing any combination of implicit and explicit forces and constraints in either coordinates. It also handles full, implicit two-way coupling between articulated and deformable bodies.

As a further demonstration of the flexibility of REDMAX, we show that we can also incorporate frictional contact within the joints in an efficient manner, by taking advantage of the reduced coordinate representation of the joints. Our approach works well when augmented with both bilateral (*e.g.,* loop closure) and unilateral (*e.g.,* external contact) constraints.

3.2 Related Work

Articulated rigid body dynamics has been an active research area for many decades, especially in the field of robotics, where high-performance algorithms were required for low-power systems.

To model joint friction for articulated bodies formulated in reduced coordinates, a common approach is to treat the frictional force solely as a function of joint velocities, rather than using the geometry of the joint [86, 87]. With our approach, we use the geometry of the joint in our friction algorithm, and we show that changing the geometry parameters affects the resulting motion. Finally, some recent research has shown the effectiveness of using a non-discretized friction cone [88, 89]. However, we note that the joints in most mechanisms have only 1 DOF, and so the friction "cone" can be trivially modeled by a box constraint.

3.3 Frictional Joints

In this section, we highlight the flexibility of the REDMAX formulation with an efficient algorithm for resolving frictional contact *within* joints. This has many applications including: computing the energy required for robotics; and modeling arthritic joints for biomechanics or animation. In this section, we show how REDMAX can be combined with the Staggered Projections (SP) algorithm to take into account the bilateral nature of the joint constraints.

3.3.1 Review of Staggered Projections

The original Staggered Projections algorithm was developed for solids undergoing unilateral contact constraints with friction [12]. SP is shown in Alg. 3.1, slightly modified to match our notation. Since SP was designed for maximal rigid bodies, we remove the m and r (maximal & reduced) subscripts for clarity. There are two quadratic programs (QP) that are solved iteratively: contact and friction. Let $\dot{\mathsf{q}}^{\text{unc}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}$ be the unconstrained velocity, where $\dot{\mathsf{q}}^{\text{prev}}$ is the velocity from the last time step. The contact QP can then be written as:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & \frac{1}{2}\alpha^\top \mathsf{N}\mathsf{M}^{-1}\mathsf{N}^\top\alpha - \alpha^\top\mathsf{N}(\dot{\mathsf{q}}^{\text{unc}} + h\mathsf{M}^{-1}\mathsf{f}_\beta) \\ \text{subject to} \quad & \alpha \geq 0, \end{aligned} \tag{3.1}$$

where α is the contact impulse, N is the contact normal matrix, M is the maximal mass matrix, f is the maximal force, and f_β is the frictional force, which is initially zero. After solving for α, we compute the contact force as $\mathsf{f}_\alpha = -\mathsf{N}^\top\alpha/h$. The frictional QP is:

$$\begin{aligned} \underset{\beta}{\text{minimize}} \quad & \frac{1}{2}\beta^\top \mathsf{T}\mathsf{M}^{-1}\mathsf{T}^\top\beta - \beta^\top\mathsf{T}(\dot{\mathsf{q}}^{\text{unc}} + h\mathsf{M}^{-1}\mathsf{f}_\alpha) \\ \text{subject to} \quad & -\mu\alpha \leq \beta \leq \mu\alpha, \end{aligned} \tag{3.2}$$

where β is the frictional impulse, T is the contact tangent matrix, and $\mu \geq 0$ is the coefficient of friction. The box constraints can only accommodate a four-sided friction cone—if needed, we can rewrite this constraint to give us a polyhedral cone [90] or a continuous cone [88, 89], but we note that for 1 DOF joints, the cone constraint degenerates into a box constraint. After solving for β, we compute the frictional force as $\mathsf{f}_\beta = -\mathsf{T}^\top\beta/h$. These two QPs are solved iteratively until convergence. The convergence rate can be improved by caching the frictional force, f_β, and warm-starting with this cached value at every time step [12].

Algorithm 3.1 Staggered Projections

```
 1: Fill $\mathsf{M}$                                         ▷ mass matrix
 2: $\mathsf{f}_\beta = 0$
 3: while simulating do
 4:     Fill $\mathsf{f}$, $\mathsf{N}$, $\mathsf{T}$          ▷ force vector, normal and tangent matrices
 5:     $\mathsf{f}^0_\alpha = 0$
 6:     $\dot{\mathsf{q}}^{\text{unc}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}$
 7:     while true do
 8:         // CONTACT
 9:         Solve contact QP 3.1 for $\alpha$
10:         $\mathsf{f}_\alpha = -\mathsf{N}^\top \alpha / h$
11:         // CONVERGENCE CHECK
12:         if $\|\mathsf{f}_\alpha - \mathsf{f}^0_\alpha\|_{\mathsf{M}^{-1}} \leq \epsilon$ or max iterations then
13:             break
14:         end if
15:         $\mathsf{f}^0_\alpha = \mathsf{f}_\alpha$
16:         // FRICTION
17:         Solve friction QP 3.2 for $\beta$
18:         $\mathsf{f}_\beta = -\mathsf{T}^\top \beta / h$
19:     end while
20:     $\dot{\mathsf{q}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}(\mathsf{f} + \mathsf{f}_\alpha + \mathsf{f}_\beta)$
21: end while
```

3.3.2 Bilateral Staggered Projections

We extend SP by taking advantage of the bilateral constraints present in articulated rigid body dynamics. The resulting algorithm, which we call Bilateral Staggered Projections (BISP), is much more efficient than SP and can also be combined with SP for handling external frictional contacts, such as between a body and the environment.

BISP has several advantages over SP. First, we do not need collision detection. With BISP, for each joint type (*e.g.,* revolute, spherical, prismatic), we use a small number of implicit contacts at pre-determined positions around the joint. For example, for a revolute joint, we assume that the joint geometry is a cylinder, and we populate the two ends of the cylinder with a sparse set of contact points (see inset figure). By changing the parameters of this cylinder, we get different frictional effects. Second, the size of the friction QP decreases significantly, because the friction cone can be represented exactly using box constraints for 1 DOF (revolute and prismatic) joints,

Algorithm 3.2 Bilateral Staggered Projections

```
1: Fill $\mathsf{M}$                                        ▷ mass matrix
2: $\mathsf{f}_\beta = 0$
3: while simulating do
4:     Fill $\mathsf{f}$, $\mathsf{N}$, $\mathsf{T}$       ▷ force vector, normal and tangent matrices
5:     $\mathsf{f}_\alpha^0 = 0$
6:     while true do
7:         // CONTACT
8:         Evaluate 3.5 for $\mathsf{f}_\alpha$             ▷ §3.3.2.1
9:         while backward traversal do
10:            Distribute $\mathsf{f}_\alpha$ to joint       ▷ §3.3.2.2
11:        end while
12:        while parallel traversal do
13:            Locally solve 3.7 for $\alpha$                ▷ §3.3.2.3
14:        end while
15:        // CONVERGENCE CHECK
16:        if $\|\mathsf{f}_\alpha - \mathsf{f}_\alpha^0\|_{\mathsf{M}^{-1}} \leq \epsilon$ or max iterations then
17:            break
18:        end if
19:        $\mathsf{f}_\alpha^0 = \mathsf{f}_\alpha$
20:        // FRICTION
21:        Solve friction QP 3.2 for $\beta$
22:        $\mathsf{f}_\beta = -\mathsf{T}^\top \beta / h$
23:    end while
24:    $\dot{\mathsf{q}}_r = \dot{\mathsf{q}}_r^{\text{prev}} + h\,\mathsf{M}_r^{-1}\mathsf{J}_{mr}^\top(\tilde{\mathsf{f}}_m + \mathsf{f}_\alpha + \mathsf{f}_\beta)$
25: end while
```

which are often the most used joints. Third, the contact QP can be eliminated, since in reduced coordinates, the contact constraints are satisfied automatically. Finally, we obtain faster convergence, since the contacts are more temporally coherent in a bilaterally constrained system.

In the following subsections, we describe how we extend SP to obtain BISP. As stated above, BISP eliminates the need for the contact QP by taking advantage of reduced coordinates, However, we still need the contact Lagrange multipliers, α, when we solve for the frictional impulses, because α is used as the limits on the friction forces. We compute α in three steps:

- §3.3.2.1: Compute the joint reaction forces.

- §3.3.2.2: Distribute this global force into the joints.

- §3.3.2.3: Compute α locally within each joint.

3.3.2.1 Compute joint reaction force

We first compute the joint reaction (*i.e.,* constraint) force that would produce the same constrained motion as the one generated using reduced coordinates. We do this by comparing the velocity generated by the reduced solve against the velocity generated by an unconstrained maximal solve. As an illustration, suppose we are running the standard SP algorithm with maximal coordinates. Let $\dot{\mathsf{q}}^{\text{unc}} = \dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}\mathsf{f}$, and the corresponding constrained velocity from Eq. 3.1 be $\dot{\mathsf{q}}^{\text{con}}$. We can rearrange the constrained equations of motion to solve for the constraint forces:

$$\begin{aligned} \mathsf{M}\dot{\mathsf{q}} + \mathsf{N}^\top\alpha &= \mathsf{M}\dot{\mathsf{q}}^{\text{prev}} + h(\mathsf{f} + \mathsf{f}_\beta) \\ \mathsf{M}^{-1}\mathsf{N}^\top\alpha &= \underbrace{\dot{\mathsf{q}}^{\text{prev}} + h\mathsf{M}^{-1}(\mathsf{f} + \mathsf{f}_\beta)}_{\dot{\mathsf{q}}^{\text{unc}}} - \underbrace{\dot{\mathsf{q}}}_{\dot{\mathsf{q}}^{\text{con}}} \end{aligned} \tag{3.3}$$

where $\dot{\mathsf{q}}^{\text{prev}}$ is the velocity from the last time step. We can rearrange further to obtain the expression for the constraint force, $\mathsf{f}_\alpha = -\mathsf{N}^\top\alpha/h$:

$$\mathsf{f}_\alpha = \frac{1}{h}\mathsf{M}\left(\dot{\mathsf{q}}^{\text{con}} - \dot{\mathsf{q}}^{\text{unc}}\right). \tag{3.4}$$

What this equation implies is that we can compute the constraint force, f_α, by subtracting the unconstrained velocity from the constrained velocity.

Now we show how BISP uses a similar approach to eliminate the contact QP. As in SP, the current friction force must be taken into account when computing the constrained and unconstrained velocities. In the following equations, since we must now compute both reduced and maximal coordinates, we add back the subscripts m and r. (The contact and friction forces, f_α and f_β, are maximal quantities.) As before, we subtract the unconstrained velocity from the constrained velocity, but now the constrained velocity is computed in reduced coordinates instead of using Eq. 3.1:

$$\mathsf{f}_\alpha = \frac{1}{h}\mathsf{M}_m\left(\mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\text{con}} - \dot{\mathsf{q}}_m^{\text{unc}}\right) \tag{3.5a}$$

$$\dot{\mathsf{q}}_m^{\text{unc}} = \mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\text{prev}} + h\mathsf{M}_m^{-1}\left(\mathsf{f}_m + \mathsf{f}_\beta\right) \tag{3.5b}$$

$$\dot{\mathsf{q}}_r^{\text{con}} = \dot{\mathsf{q}}_r^{\text{prev}} + h\mathsf{M}_r^{-1}\mathsf{J}_{mr}^\top\left(\tilde{\mathsf{f}}_m + \mathsf{f}_\beta\right). \tag{3.5c}$$

To lighten the notation, we use $\tilde{\mathsf{f}}_m$ in Eq. 3.5c to include the quadratic velocity vector from the RHS of Eq. 3.6, derived in the body of the larger REDMAX project in Wang et al. [11]

$$\begin{gathered}\left(\mathsf{J}_{mr}^\top\left(\mathsf{M}_m + h\mathsf{D}_m - h^2\mathsf{K}_m\right)\mathsf{J}_{mr} + h\mathsf{D}_r - h^2\mathsf{K}_r\right)\dot{\mathsf{q}}_r^{(k+1)} = \\ \left(\mathsf{J}_{mr}^\top\mathsf{M}_m\,\mathsf{J}_{mr}\right)\dot{\mathsf{q}}_r^{(k)} + h\left(\mathsf{f}_r + \mathsf{J}_{mr}^\top\left(\mathsf{f}_m - \mathsf{M}_m\,\dot{\mathsf{J}}_{mr}\,\dot{\mathsf{q}}_r^{(k)}\right)\right),\end{gathered} \tag{3.6}$$

3.3.2.2 Distribute contact force to joints

The computed constraint force, f_α, is a global maximal force vector that accounts for all joint reaction forces. Therefore, if we extract a portion of f_α corresponding to a single body, we obtain the *sum* of all the joint reaction forces acting on that body. To compute the Lagrange multipliers for a particular joint, we first need to isolate the joint reaction force from this sum. Fortunately, this can be done in a linear fashion by traversing the joints backward from leaf to root. For a leaf body, there is only one joint force acting on it, and so its portion of f_α is exactly the required joint reaction force. Since this joint reaction force exerts an equal and opposite force on the parent of the leaf, we subtract this force from the parent's portion of f_α and continue the backward traversal.

3.3.2.3 Compute contact Lagrange multipliers

Once we have the joint reaction forces distributed to each joint, we can compute the contact Lagrange multipliers that generate that joint reaction force. This can be done in parallel, since these are local operations performed for each joint independently of each other. For each joint, we search for a least-squares solution to $\left(\mathsf{N}_i\mathsf{M}_i^{-1}\mathsf{N}_i^\top\right)\alpha_i = h\mathsf{N}_i\mathsf{M}_i^{-1}\mathsf{f}_{\alpha i}$, where the subscript i indicates the blocks corresponding to the i^{th} body. We do not require α_i to be positive, since these "contact" constraints are bilateral—they cannot come apart. To deal with contact indeterminacy [91], we add a regularization term, which is critical because otherwise the joint can become arbitrarily tight. For instance, setting $\alpha_i = 1000$ for all contacts will generate the same effective constraint on the joint as setting $\alpha_i = 0.1$. Adding this regularization term, the local linear system becomes:

$$\alpha_i = h \left(\mathsf{N}_i \mathsf{M}_i^{-1} \mathsf{N}_i^\top + \epsilon I \right)^{-1} \left(\mathsf{N}_i \mathsf{M}_i^{-1} \mathsf{f}_{\alpha i} \right) . \tag{3.7}$$

In our experiments, we set $\epsilon = 1\text{e-}6$.

After the contact impulses for all the joints, α, are computed, the convergence check and the friction solve are the same as in SP. The step-by-step algorithm is shown in Alg. 3.2.

3.3.3 Adding External Constraints

BISP can also take into account external constraints, such as loop-closing (bilateral) constraints or frictional contact (unilateral) constraints with the environment. Alg. 3.2 is modified as follows to take into account these additional constraints:

• Line 10: To compute the contact force, both the unconstrained and constrained velocities must take into account the additional external constraints. The unconstrained velocity is obtained by solving a maximal system with only the external constraints, ignoring the implicit constraints exerted by the joints. The corresponding constrained velocity is obtained by solving a reduced system with external bilateral constraints, G, and unilateral constraints, C. The difference between these velocities will give us the joint reaction forces. Thus, instead of Eq. 3.5, we evaluate the following:

$$\mathsf{f}_\alpha = \frac{1}{h}\mathsf{M}_m\left(\mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\text{con}} - \dot{\mathsf{q}}_m^{\text{unc}}\right) \tag{3.8a}$$

$$\begin{aligned}\min_{\dot{\mathsf{q}}_m}.\quad & \frac{1}{2}\dot{\mathsf{q}}_m^\top \mathsf{M}_m \dot{\mathsf{q}}_m - \dot{\mathsf{q}}_m^\top\left(\mathsf{M}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r^{\text{prev}} + h\left(\mathsf{f}_m + \mathsf{f}_\beta\right)\right)\\ \text{s. t.}\quad & \mathsf{G}_m\dot{\mathsf{q}}_m = 0, \quad \mathsf{C}_m\dot{\mathsf{q}}_m \geq 0\end{aligned} \tag{3.8b}$$

$$\begin{aligned}\min_{\dot{\mathsf{q}}_r}.\quad & \frac{1}{2}\dot{\mathsf{q}}_r^\top \mathsf{M}_r \dot{\mathsf{q}}_r - \dot{\mathsf{q}}_r^\top\left(\mathsf{M}_r\dot{\mathsf{q}}_r^{\text{prev}} + h\,\mathsf{J}_{mr}^\top\left(\tilde{\mathsf{f}}_m + \mathsf{f}_\beta\right)\right)\\ \text{s. t.}\quad & \mathsf{G}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r = 0, \quad \mathsf{C}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r \geq 0,\end{aligned} \tag{3.8c}$$

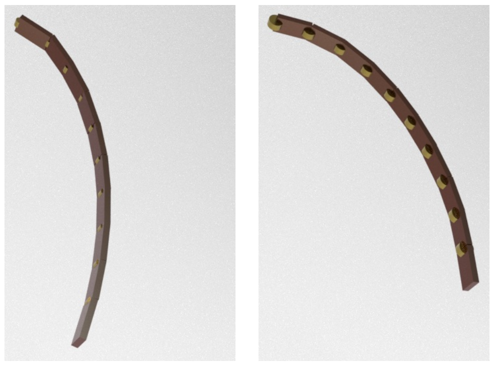
where $\dot{\mathsf{q}}_m^{\text{unc}}$ and $\dot{\mathsf{q}}_r^{\text{con}}$ are the solutions of the two QPs (3.8b & 3.8c). The loop-closing constraint reaction forces are computed as $\mathsf{f}_\lambda = -\mathsf{J}_{mr}^\top\mathsf{G}_m^\top\lambda/h$, where λ is the vector of Lagrange multipliers corresponding to the loop-closing constraints, $\mathsf{G}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r = 0$, from the minimization for $\dot{\mathsf{q}}_r^{\text{con}}$. The *maximal* QP in Eq. 3.8b may seem expensive to solve (all other QPs are in *reduced* coordinates), but usually, the number of external constraints is much smaller than the number of joint constraints. Therefore, we can solve this maximal QP in its dual form instead, which is much smaller. For example, for the KLANN mechanism with 6 legs (Fig. 3.3a), the dual QP is of size at most 30 with only box constraints.

• Line 15: We need to compute the contact forces due to the loop-closing joint constraints, by again solving a small linear system 3.7. These small linear systems are solved for each joint *and* for each loop-closing joint constraint.

• Line 26: To compute the final velocity, we solve a quadratic program that takes into account the external constraints:

$$\begin{aligned}\min_{\dot{\mathsf{q}}_r}.\quad & \frac{1}{2}\dot{\mathsf{q}}_r^\top \mathsf{M}_r \dot{\mathsf{q}}_r - \dot{\mathsf{q}}_r^\top\left(\mathsf{M}_r\dot{\mathsf{q}}_r^{\text{prev}} + h\,\mathsf{J}_{mr}^\top\left(\tilde{\mathsf{f}}_m + \mathsf{f}_\alpha + \mathsf{f}_\beta\right)\right)\\ \text{s. t.}\quad & \mathsf{G}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r = 0, \quad \mathsf{C}_m\mathsf{J}_{mr}\dot{\mathsf{q}}_r \geq 0.\end{aligned} \tag{3.9}$$

With these three changes, any combination of external bilateral and unilateral constraints can be incorporated into BISP.



(a) CHAIN (R) (b) CHAIN ($2R$)

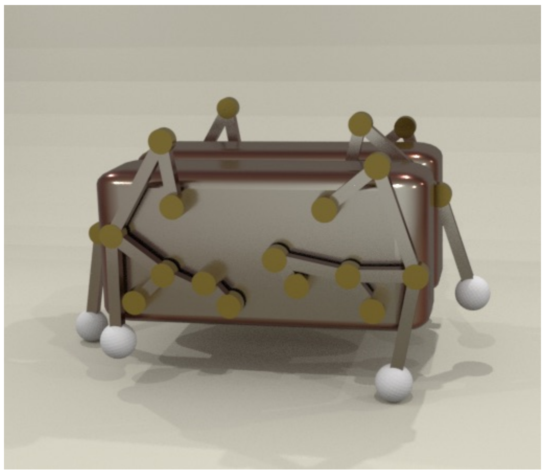
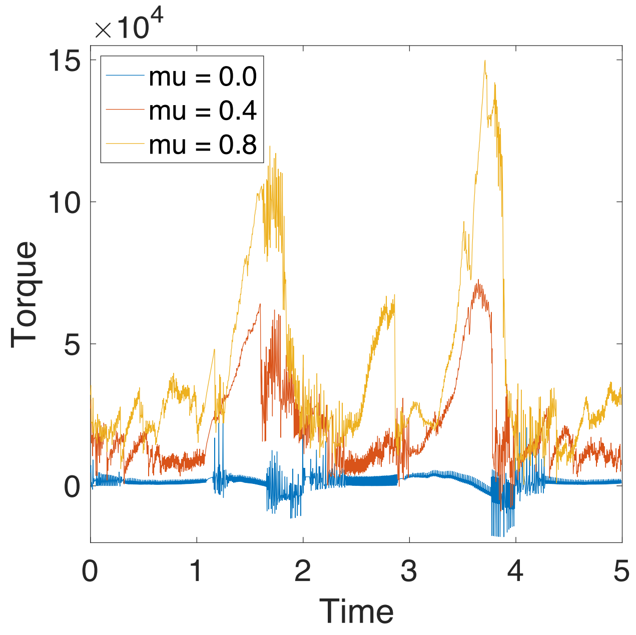
Figure 3.2: (a) CHAIN composed of revolute joints. (b) With a larger radius, the chain stops earlier. Reprinted from [11].

3.4 Results

We implemented our system in C++ and ran the simulations on a consumer desktop with an Intel Core i7-7700 CPU @ 3.6 Ghz and 16 GB of RAM. We use Eigen for dense linear algebra, Pardiso for sparse linear solves, and Mosek for quadratic programs.

• CHAIN (Figs. 3.2a & 3.2b): This scene shows the frictional effect due to changing the geometry of the joint. We initialize the scene so that the chain starts horizontally and falls under gravity, with the axis of rotation oriented $45°$ from the direction of gravity. Since more weight must be supported by bodies closer to the root, the contact force, and thus the force of friction, is stronger at the root compared to the tip. This makes the chain stop rotating starting from the root rather than at the tip. When we increase the joint radius, even with the same coefficient of friction, the force of friction increases since the joint is able to apply more torque. When Staggered Projections is used, the simulation takes an order of magnitude longer to complete because: (1) with maximal coordinates, position stabilization is required, and this can have an adverse effect on the iteration count; and (2) SP requires two global QPs, whereas BISP requires only one.

• KLANN (Figs. 3.3a & 3.3b): Our friction solver can handle loop-closure and contact constraints. We build a walking machine with a Klann linkage for each of the 6 limbs. Each limb

(a) (b)

Figure 3.3: (a) Klann walker with frictional joints. (b) Graph of torque vs. time, with various joint friction coefficients. Reprinted from [11].

has 5 revolute joints and 2 loop-closure constraints. Since each loop-closure constraint removes 2 DOFs, each limb has $5 - 2 \cdot 2 = 1$ effective DOF. In total, there are 26 internal joint DOFs with 24 bilateral constraints and (up to) 6 unilateral constraints. The 2 remaining DOFs are driven with inverse dynamics to have a constant rotational speed. We use $\mu = 0.8$ for floor friction. We run several simulations, increasing μ for loop closure and joint constraints from 0.0 to 0.8. Fig. 3.3b shows the amount of torque required to drive the mechanism as we increase this frictional coefficient. As we expect, more torque is required when there is more friction within the joints. Interestingly, when the $\mu = 0$, the motor does some negative work—the limbs try to push the motor forward, but the motor pushes back.

3.5 Conclusion

We introduced an efficient and flexible approach for computing the dynamics of articulated rigid bodies. This approach can be efficiently integrated into a friction solver that can incorporate friction inside the joints with loop closure as well as external contact constraints. Our algorithm bypasses the expensive bottleneck of a contact QP while still computing the same result of a traditional full staggered projections contacts with friction simulation. We take advantage of the reduced system to create complex scenes with a significantly reduced number of DOFs and avoid the position stabilization that comes along with maximal setups. The algorithm is an extension of the Staggered Projections algorithm [12], which iteratively solves a pair of coupled quadratic programs to resolve the frictional force. It would be interesting to also extend the popular approximate frictional contact model by Anitescu and Hart [92], which uses only a single QP.

4. CONDENSATION JACOBIAN WITH ADAPTIVITY[*]

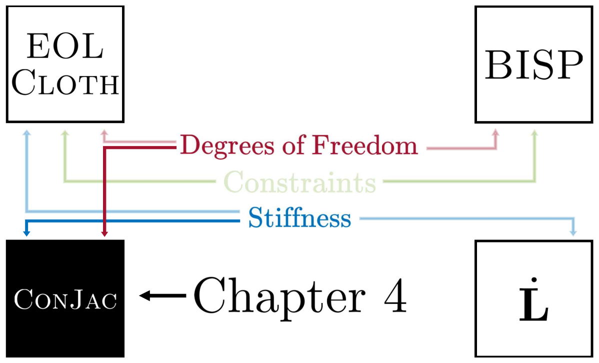


Figure 4.1: (Left) Condensation Jacobian with Adaptivity looks at using reduction methods to improve the efficiency of soft body deformations. (Right) Still frame of a simulated soft body dragon deforming from pulling forces and generating lively motion using only a small number of dynamic nodes. Reprinted from [13].

We present a new approach that allows large time steps in dynamic simulations. Our approach, CONJAC, is based on condensation, a technique for eliminating many degrees of freedom (DOFs) by expressing them in terms of the remaining degrees of freedom. In this work, we choose a subset of nodes to be *dynamic* nodes, and apply condensation at the velocity level by defining a linear mapping from the velocities of these chosen dynamic DOFs to the velocities of the remaining *quasistatic* DOFs. We then use this mapping to derive reduced equations of motion involving only the dynamic DOFs. We also derive a novel stabilization term that enables us to use complex nonlinear material models. CONJAC remains stable at large time steps, exhibits highly dynamic motion, and displays minimal numerical damping. In marked contrast to subspace approaches, CONJAC gives exactly the same configuration as the full space approach once the static state is

[*]Reprinted with permission from "Conjac: Large steps in dynamic simulation" by N. J. Weidner, T. Kim, and S. Sueda, *Motion, Interaction and Games*, ser. MIG '20. New York, NY, USA: Association for Computing Machinery, 2020 [Online]. Available: https://doi.org/10.1145/3424636.3426901. Copyright 2020 by Association for Computing Machinery.

reached. Furthermore, ConJac can automatically choose which parts of the object are to be simulated dynamically or quasistatically. Finally, ConJac works with a wide range of moderate to stiff materials, supports anisotropy and heterogeneity, handles topology changes, and can be combined with existing solvers including rigid body dynamics.

4.1 Introduction

Physics-based simulation of dynamic deformable objects has a long history in computer graphics. Starting with the work by Terzopoulos et al. [1], algorithms for physics-based animation have steadily become an integral part of the visual effects pipeline. Over the years, various improvements have been made, including: novel energy formulations [93], inversion recovery/safety [94, 95], novel Eulerian/Largrangian formulations [96, 97], and completely new time stepping schemes [80, 79].

Computational efficiency is one of the most important aspects of simulation. Real-time applications such as games and virtual surgery have strict computational budgets for physics, while offline applications such as movies need efficiency so that artists can quickly iterate on designs. However, efficiency comes at a price. Various works have made dynamic simulation of deformable objects extremely efficient, but they inescapably introduce limitations. To tackle this issue, we introduce a novel, reduced coordinate approach that has the following desirable properties:

- Reproduces *exactly the same static configuration* as the standard finite element (FE) approach.
- Supports complex nonlinear materials, including heterogeneity, anisotropy, and biomechanical soft tissues.
- Does not require any precomputation.
- Supports topology changes.
- Retains dynamic motion at large time steps, without suffering from excessive numerical damping.

- Can be combined with existing frameworks, including rigid body dynamics, into a fully two-way coupled simulation.

Existing works fail with respect to at least one of these properties. The virtual surgery simulator of Bro-Nielsen and Cotin [98] is highly efficient and produces the same static configuration as the full FE method, which is useful for predicting the behavior of a virtual organ. However, it only supports relatively small deformations, because only linear materials can be factorized as a precomputation. In one of the seminal works on cloth simulation, Baraff and Witkin [2] greatly increased the efficiency of dynamics simulations by introducing a linearly implicit integration method that allowed large time steps. However, this approach fails to retain dynamics under large time steps due to numerical damping. One of the most important approaches to improving efficiency is subspace dynamics [99, 100, 101, 102, 103, 104]. These methods achieve massive speed ups, but sacrifice local detail because the subspace dimension must be kept at a minimum. They also require precomputation, and cannot reproduce the same solution as FE unless a prohibitively large subspace is used.

Our approach is based on condensation [14], a technique for eliminating many degrees of freedom (DOFs) by expressing them in terms of the remaining DOFs. With CONJAC, short for *Con*densation *Jac*obian, we apply condensation at the velocity level—a significant departure from previous work [105, 14, 98, 15, 104]. We select a "dynamic" subset of nodes as the true DOFs of the system, and the remaining "quasistatic" nodes are assumed to follow the dynamic nodes in a quasistatic fashion.[†] More specifically, CONJAC expresses the velocities of the quasistatic nodes as a linear function of the dynamic nodes by leveraging the condition that the net force acting on each quasistatic node vanishes. We also derive a novel stabilization term that allows CONJAC to be used with an arbitrary material model. Previous work was limited to linear materials.

We show that most of the important dynamics of an object are captured by simulating just a few key dynamic nodes, and the remainder can be handled quasistatically. We simulate a bar stretching, compressing, bending, and twisting with only a few (1-4) dynamic nodes placed along the central

[†]Previous works have called these "external/internal" or "master/slave" nodes.

axis. We are also able to simulate the dynamics of a dragon being pulled in various locations, and a bunny being dropped on the floor, each with only 8 dynamic nodes. The CONJAC approach remains stable with large time steps because the quasistatic nodes cannot move independently, which effectively removes the small vibrations that can destabilize standard FE simulators. With CONJAC, a strong force suddenly applied to a node is instantaneously propagated to the dynamic nodes, eliminating the numerical wave that would force a full FE simulator to take small time steps.

CONJAC is a method for reducing the DOFs of a system, and so it is not tied to a specific time integrator. In this work, we showcase the strengths of CONJAC using the popular linearly implicit integration scheme [2]. We show that with a linearly implicit scheme, CONJAC is computationally inexpensive, requiring only one linear solve per time step, but does not suffer excessively from numerical damping and retains all of the advantages listed earlier in the introduction.

4.2 Related Work

Simulation of deformable objects is a well-studied subject in computer animation, and we refer the reader to excellent existing surveys and tutorials [106, 5].

Our method is based on condensation, a technique from structural engineering [107, 108, 105]. Originally developed for static vibrational analysis, condensation has been extended to include dynamics [14]. With these classical condensation approaches, a global generalized eigenvalue problem is solved for the reduced modes of the structure. In our work, we use condensation to derive a linear mapping of the velocities rather than to compute the modes.

Several previous works in computer graphics are motivated by condensation. These methods use the stiffness matrix to couple specially chosen dynamic DOFs to the remaining quasistatic nodes. Our work is closely related to the work by Gao et al. [15] on Steklov-Poincaré skinning. They achieve impressive volumetric effects for skinning using only the surface degrees of freedom, but is limited to quasistatics and corotational elasticity. The same authors later developed a "macroblock" solver for grid-based discretizations, also using a stiffness matrix reduction [109]. By solving the macroblocks in parallel and efficiently aggregating, they quickly compute a deformation that matches the output of a standard FE solver. However, they again rely on linear (corotational) material that can be precomputed. Furthermore, stiff springs are used to couple deformable objects to rigid bodies, which may reduce the time step or introduce unwanted numerical damping.

One of the most important and popular approaches to improving efficiency is subspace dynamics [99, 100, 101, 102, 110, 103]. Rather than simulating the full space of vertex DOFs, dynamics are performed over a reduced set of DOFs. To address artifacts that arise from the global support of subspace basis functions, researchers have explored domain decompositions where subspaces are computed per domain. To stitch these domains together, Barbič and Zhao citeBarbic2011 used locally aligned rigid frames, while Kim and James [111] used penalty forces. These methods can achieve massive speed ups, but sacrifice local detail because the subspace dimension must be kept at a minimum. They also require precomputations such as modal analysis and cubature optimization, so changing object topologies are challenging. Finally, they generally do not reproduce the full FE solution unless the subspace is prohibitively large.

Condensation has also been combined with subspace dynamics. Traditionally, only linear materials could be used, but Teng et al.[104] efficiently performed subspace condensation at runtime, allowing nonlinear materials to also be used. However, the overall limitations remain. The subspace must be carefully constructed, and while the condensation allows objectionable artifacts to be avoided, the final deformation does not match the full FE solution.

Recently, Xian et al. [112] introduced a multigrid-based method to solve for deformation dynamics in the full space, and achieved over 40 FPS on a mesh with over 60k vertices. However, they inherit common limitations of multigrid methods. Without significant extensions, it is not possible to support topological changes, complex materials (heterogeneity and anisotropy), and two-way coupling with rigid body dynamics.

Finally, a number of efficient time stepping schemes have been introduced by graphics researchers. Recently, Li et al. [113] introduced a domain-decomposed optimization method for implicit numerical time integration. In the past two decades, Position-Based Dynamics [80], Projective Dynamics [79], and ADMM [114, 115] have become popular, efficient alternatives to the standard time stepping schemes. Although initially quite limited in terms of available materials and constraints, these methods have become quite general and flexible. These time stepping schemes work well, but are monolithic, and would require a complete rewrite of existing formulations to make them work together. Our work is instead based on a simple mapping of velocities, which can be incorporated into a wide range of existing explicit and implicit integrators.

4.3 CONJAC Dynamics

We begin with a high-level, didactic description of CONJAC in action. Imagine a vertical string discretized as a sequence of 1D nodes (*i.e.,* they can only move vertically). We fix the top node and pick the bottom node to be the *dynamic* node. The remaining nodes in the middle are labeled as *quasistatic* nodes. If we know the material properties of the string (*e.g.,* zero rest-length springs), then by assuming that the net force on each quasistatic node remains zero, we can calculate the position and velocities of all these quasistatic nodes from the position and velocity of the single dynamic node at the bottom of the string.

In this section, we will formalize this approach by deriving the linear mapping between the quasistatic and dynamic nodes of a volumetric solid composed of an *arbitrary nonlinear material*. We will then derive equations of motion that allow us to simulate the object using only the dynamic DOFs. The remaining nodes are simulated quasistatically, so the final resting configuration exactly matches the result of a full, non-reduced FE simulator.

4.3.1 CONJAC Mapping

Once again, we select a set of *dynamic* nodes that are the exposed degrees of freedom of the system. The remaining *quasistatic* nodes move so that their net force always resolves to zero. The CONJAC framework uses the linear mapping that enforces this condition between the dynamic and quasistatic nodal velocities:

$$\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d, \tag{4.1}$$

where $\mathbf{J}_{qd}$ is the Jacobian term that we will derive in the rest of this section. Given any velocities of the dynamic nodes, $\mathbf{v}_d$, this mapping allows us to compute the velocities of the quasistatic nodes,

$\mathbf{v}_q$.

The derivation of $\mathbf{J}_{qd}$ in Eq. 4.1 starts with a linearization of the forces, popularized by Baraff and Witkin [2] and extensively used by other researchers [106]. We approximate the implicit force at the next time step as:

$$\mathbf{f} = \mathbf{f}^0 + \mathbf{K}^0(\mathbf{x} - \mathbf{x}^0), \tag{4.2}$$

where the superscript 0 denotes the quantities at the current time step, and $\mathbf{K} = \partial\mathbf{f}/\partial\mathbf{x}$ is the tangent stiffness matrix. Substituting the next velocity as $\mathbf{v} = (\mathbf{x} - \mathbf{x}^0)/h$, where h is the step size, we obtain:

$$\mathbf{f} = \mathbf{f}^0 + \mathbf{K}^0 h\mathbf{v}. \tag{4.3}$$

We follow previous condensation work [105, 14, 98, 15, 104] and partition each of the terms into dynamic and quasistatic quantities:

$$\begin{pmatrix} \mathbf{f}_d \\ \mathbf{f}_q \end{pmatrix} = \begin{pmatrix} \mathbf{f}_d^0 \\ \mathbf{f}_q^0 \end{pmatrix} + h \begin{pmatrix} \mathbf{K}_{dd}^0 & \mathbf{K}_{dq}^0 \\ \mathbf{K}_{qd}^0 & \mathbf{K}_{qq}^0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_d \\ \mathbf{v}_q \end{pmatrix}. \tag{4.4}$$

Since we are interested in applying the zero net-force condition on the quasistatic nodes, we extract the bottom row of Eq. 4.4. After moving $\mathbf{f}_q^0$ and h to the left hand side (LHS), we have:

$$\frac{1}{h}\left(\mathbf{f}_q - \mathbf{f}_q^0\right) = \mathbf{K}_{qd}^0\mathbf{v}_d + \mathbf{K}_{qq}^0\mathbf{v}_q. \tag{4.5}$$

Our goal is to obtain zero net-force on the quasistatic nodes, so we set the force vectors to zero. (We will return to this point in §4.3.3.) Rearranging Eq. 4.5 in the form of Eq. 4.1, $\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d$, we obtain our *con*densation *Jac*obian (CONJAC):

$$\mathbf{J}_{qd} = -(\mathbf{K}_{qq}^0)^{-1}\mathbf{K}_{qd}^0. \tag{4.6}$$

Moving forward, we will drop the superscript 0 from $\mathbf{K}$, with the understanding that these quantities are evaluated at the current time step.

4.3.2 Equations of Motion

Armed with the ConJac mapping in Eq. 4.6, we are now ready to derive the equations of motion. First, we define an expanded mapping that includes both quasistatic and dynamic nodes:

$$\mathbf{v} = \mathbf{J}\mathbf{v}_d, \quad \mathbf{v} = \begin{pmatrix} \mathbf{v}_d \\ \mathbf{v}_q \end{pmatrix}, \quad \mathbf{J} = \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_{qd} \end{pmatrix}, \tag{4.7}$$

where $\mathbf{I}$ is the identity matrix. This mapping passes the dynamic velocities through untouched, while applying the ConJac mapping defined by Eq. 4.6 to the quasistatic velocities. Taking the time derivative of Eq. 4.7, we have:

$$\dot{\mathbf{v}} = \mathbf{J}\dot{\mathbf{v}}_d + \dot{\mathbf{J}}\mathbf{v}_d. \tag{4.8}$$

Plugging $\dot{\mathbf{v}}$ into Newton's second law, $\mathbf{M}\dot{\mathbf{v}} = \mathbf{f}$, rearranging the terms, and left multiplying by $\mathbf{J}^\top$, we get:

$$\mathbf{J}^\top\mathbf{M}\mathbf{J}\dot{\mathbf{v}}_d = \mathbf{J}^\top\left(\mathbf{f} - \mathbf{M}\dot{\mathbf{J}}\mathbf{v}_d\right). \tag{4.9}$$

The LHS matrix, $\mathbf{J}^\top\mathbf{M}\mathbf{J}$, is the effective inertia tensor acting on the dynamic nodes. This generalized inertia includes not only the self inertia of the dynamic nodes but also the inertia of the quasistatic nodes, since any motion of the dynamic nodes automatically causes the quasistatic nodes to move. The right hand side (RHS) vector is pre-multiplied by the Jacobian transpose, $\mathbf{J}^\top$. Since $\mathbf{J}^\top = \begin{pmatrix} \mathbf{I} & \mathbf{J}_{qd}^\top \end{pmatrix}$, the forces acting on quasistatic nodes are left-multiplied by $\mathbf{J}_{qd}^\top$ to project away the null-space. Finally, since the goal of our approach is to approximate dynamics while preserving quasistatics, we ignore the quadratic velocity vector on the RHS involving $\dot{\mathbf{J}}$, which disappears when $\mathbf{v}$ is zero [116]. In our examples, the lack of the quadratic velocity vector did not cause any visual artifacts.

The ConJac mapping can be used with a variety of time stepping schemes. In this work, we use the popular linearly implicit (which we call "Vanilla") formulation [2, 106, 117, 118]. This integration scheme is easy to implement, requiring only a single linear solve per time step.

$$\left(\mathbf{M} - \beta h^2 \mathbf{K}\right) \mathbf{v} = \mathbf{M}\mathbf{v}^0 + h\mathbf{f}. \tag{4.10}$$

Here, the tangent stiffness matrix, $\mathbf{K}$, is evaluated at the current time step, but we have dropped the superscript for brevity. In addition to the h^2 factor in the stiffness term in Eq. 4.10, we also apply a positive factor β to control the amount of damping [101, 119]. If we increase β, the simulation becomes more stable but at the cost of added numerical damping.

We obtain *our final* CONJAC *equations of motion* by projecting VANILLA with the Jacobian:

$$\mathbf{J}^\top \left(\mathbf{M} - \beta h^2 \mathbf{K}\right) \mathbf{J}\mathbf{v}_d = \mathbf{J}^\top \left(\mathbf{M}\mathbf{v}^0 + h\mathbf{f}\right). \tag{4.11}$$

We solve this linear system at every time step for the new dynamic velocities, $\mathbf{v}_d$. Once the dynamic velocities are computed, we compute the quasistatic velocities as $\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d$. Then, as explained in the next section, we apply stabilization to the positions at the end of the time step.

4.3.3 Stabilization

The Jacobian, $\mathbf{J}_{qd}$, defined in Eq. 4.6 can cause large errors for nonlinear materials, due to the linear approximation introduced in Eq. 4.2. Since we are applying condensation at the velocity level, after taking a time step, the quasistatic forces inevitably contain small non-zero values, which implies that the LHS of Eq. 4.5 is not always zero. In particular, the *current* force acting on the quasistatic nodes, $\mathbf{f}_q^0$, is not exactly balanced, and contains small non-zeros. (On the other hand, the implicit force at the *next* time step, $\mathbf{f}_q$, is what we want to eliminate, so it is set to zero.)

This observation allows us to compute the "residual" velocity that drives the quasistatic nodes back to the zero net-force state. If we do not throw away $\mathbf{f}_q^0$ from Eq. 4.5, we obtain:

$$\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d + \mathbf{b}_q, \quad \mathbf{b}_q = -\frac{1}{h}(\mathbf{K}_{qq}^0)^{-1}\mathbf{f}_q^0. \tag{4.12}$$

This Baumgarte-like stabilization term, $\mathbf{b}_q$, is the key term that makes our approach work, even

Algorithm 4.1 CONJAC pseudocode

```
1: (Optional) Initialize the quasistatic positions
2: Compute $\mathbf{M}$
3: while simulating do
4:     Compute $\mathbf{f}, \mathbf{K}$
5:     Compute $\mathbf{J}$, $\mathbf{b}$
6:     Solve for $\mathbf{v}_d$ (Eq. 4.11)
7:     Compute $\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d$
8:     Update $\mathbf{x}$ (Eq. 4.13)
9: end while
```

in the presence of linearization artifacts [84]. Rather than modifying the velocities, we apply this stabilization term when we update the positions. We multiply this factor by a scalar parameter γ that controls the strength of the stabilization. The position updates for dynamic and quasistatic nodes are then:

$$\begin{aligned} \mathbf{x}_d &= \mathbf{x}_d^0 + h\mathbf{v}_d \\ \mathbf{x}_q &= \mathbf{x}_q^0 + h(\mathbf{v}_q + \gamma\mathbf{b}_q). \end{aligned} \tag{4.13}$$

When applied to the position, this stabilization term becomes exactly a Newton correction term: $\Delta\mathbf{x} = -\gamma\mathbf{K}_{qq}^{-1}\mathbf{f}_q$. In other words, we apply one scaled Newton step at the position level after taking a velocity step, with $\gamma = 1$ corresponding to a full Newton step. In practice, we found that a full Newton step can sometimes cause instabilities. The best value can be obtained with a line search, but we found that simply setting $\gamma = 1/3$ worked well for our examples (unless otherwise stated).

Without the stabilization term $\mathbf{b}_q$, the object becomes visibly distorted due to the accumulation of error, and can eventually blow up. This term had not been derived in previous approaches because linearization does not cause any drift in linear materials. This stabilization approach is both effective and efficient. An alternative approach based on pre- or post-stabilization may work as well [85, 120], but we speculate that they will be less efficient and more difficult to implement.

Algorithm 4.2 VANILLA pseudocode

```
1: Compute $\mathbf{M}$
2: while simulating do
3:     Compute $\mathbf{f}, \mathbf{K}$
4:     Solve for $\mathbf{v}$ (Eq. 4.10)
5:     Update $\mathbf{x} = \mathbf{x}^0 + h\mathbf{v}$
6: end while
```

4.3.4 Time Stepping

The overall simulation pseudocode for CONJAC using linearly implicit Euler [2] is shown in Alg. 4.1. For comparison, we also show the VANILLA pseudocode, also using linearly implicit integration, in Alg. 4.2.

With VANILLA, the performance bottleneck is the linear solve for the new velocities (line 4). On the other hand, with CONJAC, solving for the new dynamic velocities is not the bottleneck because Eq. 4.11 is small. Instead, the bottleneck is in forming the Jacobian (line 5), which involves a series of solves by $\mathbf{K}_{qq}$, which cannot be prefactored for nonlinear materials.

With our current implementation, each time step of CONJAC (lines 4-8 in Alg. 4.1) is about 20% slower than a time step of VANILLA (lines 3-5 in Alg. 4.2) with 4 dynamic nodes, and 40% slower with 10 dynamic nodes (see Fig. 4.3). However, we more than make up for this difference because CONJAC allows much bigger time steps for the same amount of dynamic behavior.

The initial nonlinear solve for the quasistatic positions in CONJAC (line 1 in Alg. 4.1) can be costly, but it only needs to be performed once at the beginning of the simulation. We do not need to run this expensive nonlinear optimization within the simulation loop because of the stabilization term from §4.3.3. In fact, it is even possible to skip the initial nonlinear solve, since the stabilization term eventually eliminates the drift and drives quasistatic nodes to their zero net-force state over time.

4.4 Adaptivity

The liveliness of a ConJac simulation is tied to the number of dynamic nodes in the scene. We can choose to place dynamic nodes only in regions where dynamics are desired to avoid unnecessarily increasing the bottleneck. To generalize objects so that they are still lively and optimized in novel deformations and environments, we introduce a concept of adaptivity—we turn on/off the dynamic nodes *at runtime*. We assume that we know *a priori* a subset of mesh nodes that can become dynamic, which we call the "representative" nodes. As we show in §4.5, this number does not need to be very high to get rich deformations. For example, in the Armadillo mesh shown in Fig. 4.10, this subset consists of 5 representative nodes, placed in the extremities of the four limbs and in the center of the torso. During runtime, we automatically decide which of these representative nodes should be dynamic or quasistatic, depending on our novel "liveliness" metric. This cuts down on unnecessary solves which speeds up simulations, and improves the robustness of scenes. In the rest of this section, we will describe our liveliness metric (§4.4.1) and then discuss the necessary changes to the ConJac algorithm to minimize expensive matrix resizing and slicing operations that occur when dynamic nodes are turned on and off at runtime (§4.4.2).

4.4.1 Adaptivity Metric

To quantify the liveliness of a node, we want a metric that captures how a local region of the mesh is deforming differently from its neighborhood regions. We are interested in capturing the differences in the rate of change of deformation. Therefore, rather than using the deformation gradient $\mathbf{F}$, we use the time derivative of the deformation gradient $\dot{\mathbf{F}}$. In particular, we look at the average change in stretching speed over a local group of tetrahedral elements. Stretch is a very insightful local measurement into how much our object is actually deformed rather than undergoing rigid motion, and change in stretch captures activity instead of a deformed settled state.

To derive this change in stretch over time, or $\dot{\mathbf{S}}$, we look at the deformation gradient $\mathbf{F}$ based on our material matrix $\mathbf{D}_m$ and spacial matrix $\mathbf{D}_s$:

$$\begin{aligned}\mathbf{D}_m &= \left(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0 | \overline{\mathbf{x}}_2 - \overline{\mathbf{x}}_0 | \overline{\mathbf{x}}_3 - \overline{\mathbf{x}}_0\right) \\ \mathbf{D}_s &= \left(\mathbf{x}_1 - \mathbf{x}_0 | \mathbf{x}_2 - \mathbf{x}_0 | \mathbf{x}_3 - \mathbf{x}_0\right) \\ \mathbf{F} &= \mathbf{D}_s \mathbf{D}_m^{-1}.\end{aligned} \tag{4.14}$$

These matrices are a formulation of our nodal material positions in world space, $\overline{\mathbf{x}}$, and our current time step's deformed nodal positions in world space, $\mathbf{x}$, respectively. $\mathbf{F}$ can also be decomposed into rotation and stretch components using the polar decomposition:

$$\mathbf{F} = \mathbf{R}\mathbf{S}. \tag{4.15}$$

Substituting our deformed positions for velocities allows us to instead formulate a velocity gradient. Using the chain rule, this velocity gradient can be similarly decomposed just like $\mathbf{F}$, using nodal velocities $\mathbf{v}_i$ instead of positions $\mathbf{x}_i$ [121]:

$$\begin{aligned}\dot{\mathbf{D}}_s &= \left(\mathbf{v}_1 - \mathbf{v}_0 | \mathbf{v}_2 - \mathbf{v}_0 | \mathbf{v}_3 - \mathbf{v}_0\right) \\ \dot{\mathbf{F}} &= \dot{\mathbf{D}}_s \mathbf{D}_m^{-1} \\ \dot{\mathbf{F}} &= \dot{\mathbf{R}}\mathbf{S} + \mathbf{R}\dot{\mathbf{S}}.\end{aligned} \tag{4.16}$$

From this decomposition we can rearrange and solve for $\dot{\mathbf{S}}$:

$$\dot{\mathbf{S}} = \mathbf{R}^\top \left(\dot{\mathbf{F}} - \dot{\mathbf{R}}\mathbf{S}\right). \tag{4.17}$$

A singular value decomposition of $\mathbf{F}$ gives us definitions for $\mathbf{R}$ and $\mathbf{S}$:

$$\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^\top \quad \mathbf{R} = \mathbf{U}\mathbf{V}^\top \quad \mathbf{S} = \mathbf{V}\Sigma\mathbf{V}^\top. \tag{4.18}$$

The slightly more complicated piece we still need is $\dot{\mathbf{R}}$, which we can decompose as follows:

$$\dot{\mathbf{R}} = \frac{\partial \mathbf{R}}{\partial \mathbf{F}} : \dot{\mathbf{F}}. \tag{4.19}$$

We compute $\partial\mathbf{R}/\partial\mathbf{F}$ in closed form by following the work of Smith et al. [6]. (The pseudocode is given in §B.1.1.) Computationally speaking, when using a material model such as the Stable Neo-Hookean material [93], the expensive SVD component of these operations is already required so the only additional work needed for this new metric is the relatively inexpensive $\dot{\mathbf{R}}$ value.

Once this metric is defined per tetrahedron, we want to quantify this measurement for each representative node so we know whether a particular representative node should be dynamic or quasistatic at a given time step. $\dot{\mathbf{S}}$ is a a matrix whose coefficients represent the speed of change of the deformation, so by using the absolute value of these coefficients to ignore direction and the average of them to alleviate outliers, we arrive at a scalar value giving us a good idea how much deformation is taking place. In other words, the liveliness measure of the j^{th} representative node is:

$$\text{metric}_j = \text{mean}\left(\begin{bmatrix}\text{vec}(|\dot{\mathbf{S}}_1|)^\top & \cdots & \text{vec}(|\dot{\mathbf{S}}_m|)^\top\end{bmatrix}\right), \tag{4.20}$$

where m is the number of tetrahedra in the region owned by the j^{th} representative node. (The pseudocode is given in §B.1.2; in our actual implementation, we use a weighted average using the volume of each element.) To further account for potential noise in the metric, we expand this by averaging these metrics across a window of past time steps. Our final scalar value is compared against a threshold of desirable motion and the end result is a dynamic node that can gracefully revert to a quasistatic state when the local deformations around it are not worth spending the increased number of solves to capture. In the extreme case, when the dynamic motion has mostly died down, the simulation is driven entirely from the stabilization term, with all nodes moving in a quasistatic fashion.

Algorithm 4.3 CONJAC with adaptivity

```
1: (Optional) Initialize the quasistatic positions
2: Compute $\mathbf{M}$
3: while simulating do
4:     Compute $\mathbf{f}, \mathbf{K}, \dot{\mathbf{S}}$
5:     for representative nodes do
6:         Compute metric
7:         Set as dynamic or quasistatic
8:     end for
9:     Adjust sparse values of $\mathbf{K}$ and $\mathbf{K}_{qd}$
10:    Compute $\mathbf{J}$, $\mathbf{b}$
11:    Solve for $\mathbf{v}_d$ (Eq. 4.11)
12:    Compute $\mathbf{v}_q = \mathbf{J}_{qd}\mathbf{v}_d$
13:    Update $\mathbf{x}$ (Eq. 4.13)
14: end while
```

4.4.2 Sparse Matrix Handling

Introducing this mechanic for flipping the state of a subset of our nodes forces us to take another look at our CONJAC algorithm. The construction of our Jacobian is reliant on both the $\mathbf{K}_{qq}$ and $\mathbf{K}_{qd}$ matrices, which can now vary in size *at runtime* depending on the number of dynamic and quasistatic nodes. Because the construction of our Jacobian is our bottleneck, we want to ensure that we are not introducing new overhead on top of the existing CONJAC algorithm that used fixed-sized matrices. More specifically, some operations such as large matrix allocations and sparsity pattern analyses (row/column permutations and symbolic analysis) that usually take place in the simulation setup phase now must happen each time step that a representative node changes from dynamic to quasistatic and vice versa.

Predefining the subset of representative nodes that can flip between states has a major advantage in combatting this issue. Without adaptivity, we used Eq. 4.6, which required the entire stiffness matrix, $\mathbf{K}$, to be pre-partitioned into $\mathbf{K}_{qq}$ and $\mathbf{K}_{qd}$. (Since the number of quasistatic nodes is much larger than the number of dynamic nodes, $\mathbf{K}_{qq}$ is almost the same size as $\mathbf{K}$, but $\mathbf{K}_{qd}$ is a tall and skinny matrix.) With adaptivity, rather than partitioning $\mathbf{K}$ into $\mathbf{K}_{qq}$ and $\mathbf{K}_{qd}$ at runtime, we instead adjust the non-zeros of $\mathbf{K}$ and $\mathbf{K}_{qd}$ on the fly to account for the changing number of dynamic nodes.

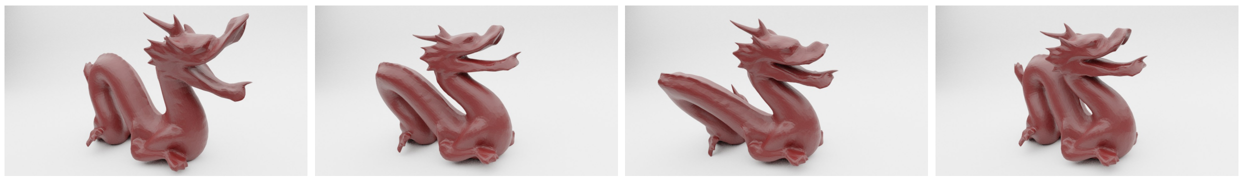


Figure 4.2: DRAGON. Pulling portions of the mesh generates lively motion using only a small number of dynamic nodes. Reprinted from [13].

Specifically: (1) we zero out the row and column of $\mathbf{K}$ corresponding to each of the dynamic nodes and place a negative one on the diagonal; and (2) we replace the rows of $\mathbf{K}_{qd}$ corresponding to the dynamic nodes with the identity matrix. In this context, to zero-out refers to explicitly setting a sparse value to zero rather than to change the sparsity. (The pseudocode for these operations is provided in §B.1.3.) Calling these adjusted matrices $\mathbf{K}_A$ and $\mathbf{K}_{qdA}$, the Jacobian can be computed as:

$$\mathbf{J} = -\mathbf{K}_A^{-1}\mathbf{K}_{qdA}, \tag{4.21}$$

instead of Eqs. 4.6 and 4.7. This allows us to take advantage of the reduced number of solves without shifting around, reallocating, or reanalyzing unnecessary data in the sparse matrices.

4.5 Results

We implemented our system in MATLAB and ran the simulations on a consumer laptop with an Intel Core i9-9880H CPU @ 2.3 GHz and 16 GB of RAM. We use MEX for filling the force vector and the stiffness matrix, and CHOLMOD for sparse linear factorizations and solves [9]. The scene parameters are listed in Table 4.1. All of the objects are table-top sized—roughly 5-15 cm across, weighing a few hundred grams. For all results, we use the Stable Neo-Hookean (SNH) base material [93]. This material is stable under inversion, but like any non-linear material, it can still need a Newton solve plus line search to maintain stability under large deformations. We found that when used with a linearly implicit scheme, it must be heavily damped when using a large time step, especially when Poisson's ratio, ν, is close to 0.5.

DRAGON: We start with a 10k node dragon, shown in Fig. 4.2. This example shows that

CONJAC presents an attractive option for efficiently producing lively simulations. We grab the jaws and the body of the dragon and pull them in different directions. After some time has passed, we let go, instantaneously releasing the built-up energy. We compare the results using CONJAC and VANILLA, both with time step $h = 5$E-3 for this 1 second simulation. For the damping factor, we use $\beta = 0.5$ for CONJAC and $\beta = 3.7$ for VANILLA (Eq. 4.10 and Eq. 4.11). These values were chosen by manually searching for the smallest β values in 0.1 increments that produced stable simulations. As can be seen in the supplemental video, the discrepancy in the β values are visibly significant. *Using the same h,* CONJAC *produces highly dynamic results, whereas* VANILLA *produces heavily damped results.* Since we are using the linearly implicit integrator, more dynamic results can be generated with VANILLA by reducing h, but this adds computational cost. CONJAC, on the other hand, allows large time steps while retaining interesting dynamics. If we reduce the time step to $h = 2$E-3 with VANILLA, the qualitative behavior of the dragon becomes nearly as lively as CONJAC, but the wall-clock simulation time increases to more than double the time of CONJAC with 6 dynamic nodes. For didactic purposes, we also include a CONJAC simulation with 0 dynamic nodes, which produces a quasistatic simulation driven solely by the stabilization term, $\mathbf{b}_q$ from Eq. 4.12. For this example, we used the stabilization factor $\gamma = 1/5$, since the Newton displacements immediately after releasing the jaws and the body are extremely large. Once we add dynamic nodes, the behavior becomes very lively, even with only 2

Table 4.1: List of scene parameters. #vert: number of total vertices. #dyn: number of dynamic vertices. #elem: number of elements. mat: material model. Y: Young's modulus (Pa). ν: Poisson's ratio. Reprinted from [13].

Scene	#vert	#dyn	#elem	mat	Y	ν
DRAGON	10456	0-10	37565	SNH	3E4	0.49
TWIST	1029	1-32	4320	SNH	1E4	0.40
HETERO	5915	1	29376	SNH	1E4	0.40
ANISO	6591	1	32832	+aSTVK	1E4	0.40
MUSCLE	262	5	438	+aFUNG	3E4	0.49
BARCUT	6050	2	29400	SNH	1E4	0.40
BUNNY	5988	8	27695	SNH	4E4	0.45
ARMADILLO	5159	5	18448	SNH	6E4	0.49

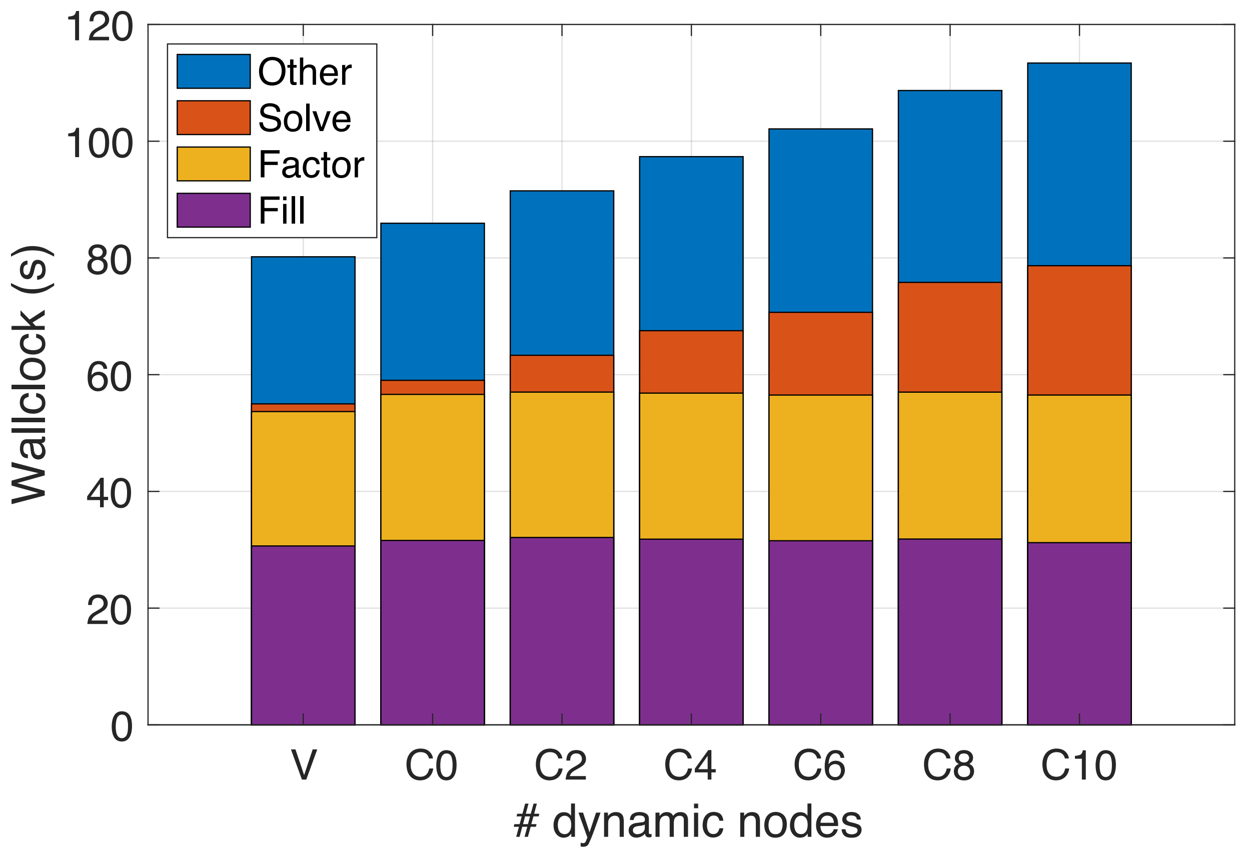


Figure 4.3: Wall-clock times for DRAGON. Starting from the left: VANILLA, CONJAC with 0, 2, 4, 6, 8, and 10 dynamic nodes. Each bar is broken down into Fill ($\mathbf{f}$ and $\mathbf{K}$), Factor, Solve, and Other. As the number of dynamic nodes increases, the Solve cost goes up linearly. Reprinted from [13].

nodes. The wall-clock times of CONJAC is compared to VANILLA in Fig. 4.3. Virtually all of the added cost is in the triangular solves—since we require $3n_d + 1$ solves, where n_d is the number of dynamic nodes, the cost increases linearly in n_d. (The +1 is for computing the stabilization term, b_q.) For most objects, 4 to 8 dynamic nodes are enough to produce convincingly dynamic results. We discuss potential ways to improve performance in §4.6.1.

TWIST: Here, we show the deformation behavior of CONJAC as we increase n_d, the number of dynamic nodes. For this scene, we use CONJAC to simulate a bar with one of its ends moved kinematically to compress, stretch, bend, and twist the bar as shown in Fig. 4.4. For $n_d = \{1, 2, 4\}$, we place the dynamic nodes at equal intervals along the central horizontal axis. For $n_d = \{8, 16, 32\}$, we slice the bar orthogonal to the central axis at equal intervals and place 4 dynamic nodes at the corners of each of these vertical slices. Interestingly, it becomes difficult to visually distinguish between these cases—even with 1 dynamic node, the dynamic motion is convincing. When the dynamic nodes are placed along the central axis ($n_d = \{1, 2, 4\}$), we get the added "feature": the twisting waves are propagated instantaneously along the bar, increasing the stability of the system. If the dynamic nodes are placed along vertical slices ($n_d = \{8, 16, 32\}$), we recover the twisting dynamics.

Hetero: We show that ConJac efficiently and effectively handles heterogeneous materials. In this example, we use ConJac to simulate a vertical bar with alternating layers of stiffnesses. Fig. 4.5 shows that even with only one dynamic node, we can capture the bulging of the soft layers. Because of gravity, the lower soft layer bulges out more than the upper soft layer, even though they have the same stiffness. Once the object reaches its static state, the final shape is exactly the same as the one generated by Vanilla.

Aniso: We show the effect of anisotropic materials. On top of the base SNH material, we add an anisotropic Saint Venant–Kirchhoff material (aSTVK) [95]. In this example, we use ConJac with one dynamic node to simulate a vertical bar with different anisotropic directions: vertical, horizontal, diagonal, and helical. Fig. 4.6 shows that when gravity compresses the bar, it deforms differently depending on the fiber directions. Interestingly, the helical fibers induce a twisting motion.

Muscle: ConJac can easily be combined with existing rigid body dynamics to model a musculoskeletal system (Fig. 4.7). In this 2D example, we combine ConJac with a reduced coor-

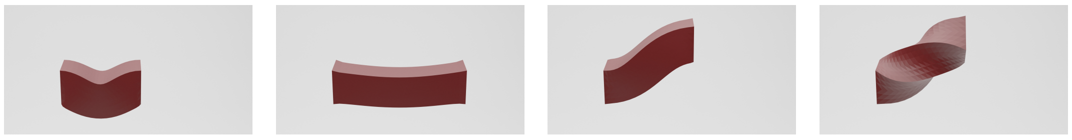
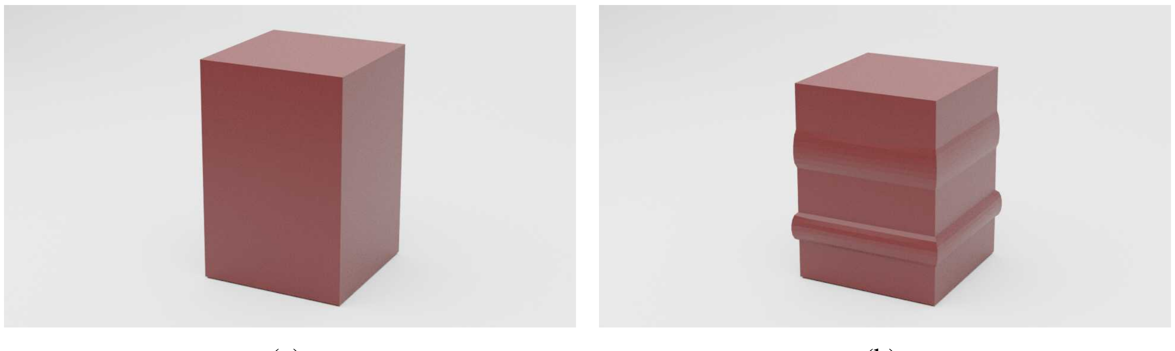


Figure 4.4: Twist scene: A bar is (a) compressed, (b) stretched, (c) bent, (d) and twisted through kinematic motion. The bar in these figures only have a single dynamic node. Reprinted from [13].

dinate articulated rigid body framework [11]. To attach the origin and insertion nodes to the bones, we use a Jacobian mapping that expresses the velocity of these nodes as a function of the velocities of the joints. This allows us to solve for the velocities of the muscles and joints simultaneously to give us full two-way coupling between muscles and bones, which is important because the muscle weighs more than the bones. We use SNH for the background isotropic material, and anisotropic Fung (aFUNG) for the muscle fiber material [122]. We also take advantage of CONJAC's support for heterogeneity—the stiffness of the background SNH material is modulated so that it is stiffer in the tendon regions than in the muscle region. In the resulting simulation, the dynamics of the muscle is fully accounted for by a single, central dynamic node. In total, the system is only 4-dimensional: 2 DOFs for the joints and 2 for the muscle. Unlike quasistatic muscle simulators that assume both bones and muscles are quasistatic, with CONJAC, we can keep the bones fully dynamic and choose how dynamic we want the muscles to be.

BARCUT: In this example, we show that CONJAC supports topology changes. We start with a horizontal bar fixed at its two ends, and we cut the bar in two locations (see Fig. 4.8). We place two dynamic nodes on either side of the initial cut. Because CONJAC requires no precomputation, the cut can be placed anywhere. After the second cut, the right-most piece loses all dynamic nodes and gracefully degrades into a purely quasistatic model.

BUNNY: In this example, we show that CONJAC can be extended to handle frictional contact.


(a) (b)
Figure 4.5: HETERO scene that divides a vertical bar mesh into layers of alternating material stiffness. Reprinted from [13].

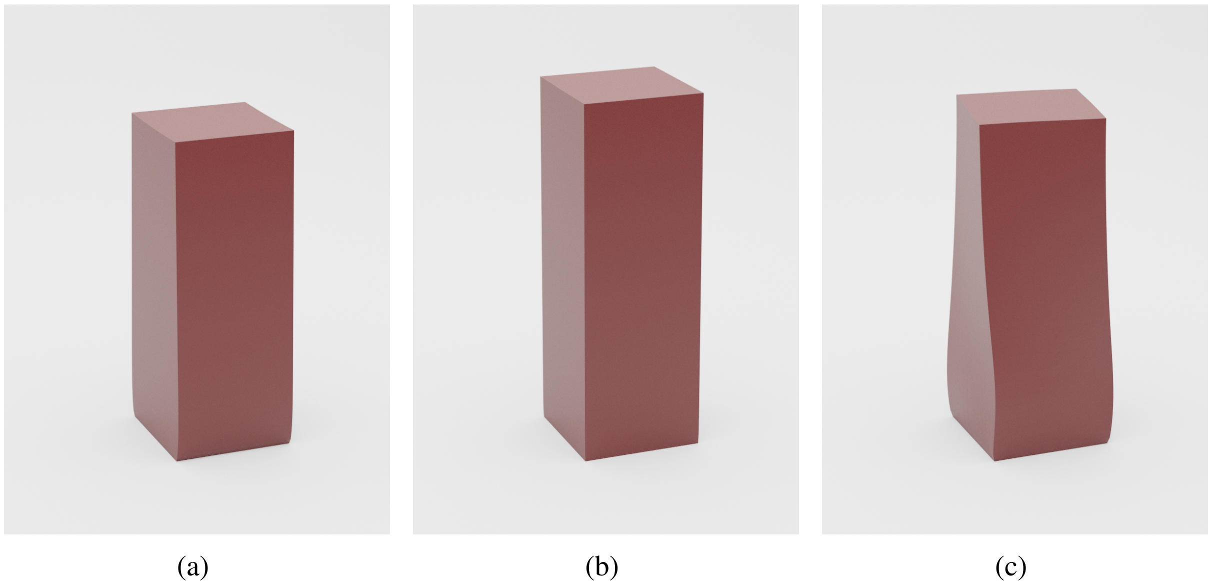


Figure 4.6: ANISO bar with (a) horizontal, (b) vertical, (c) helical directional fibers. Reprinted from [13].

We drop a bunny with 8 dynamic nodes onto the floor with various starting orientations. We follow the formulation by McAdams et al. [123] for the contact penalty force: $\mathbf{f} = K\left((1-\alpha)\mathbf{n}\mathbf{n}^\top + \alpha\mathbf{I}\right)(\mathbf{x}-\mathbf{x}_s)$, where K is a stiffness constant, $\mathbf{n}$ is the collision normal, and $\mathbf{x}_s$ is the closest point on the collision surface. When $\alpha = 0$, the spring acts only along the normal direction, and when $\alpha = 1$, the spring acts isotropically. In our experiments, we use $\alpha = 0.1$. For friction, we use the velocity filter approach by Bridson et al. [65] to compute the post-friction velocity, $\mathbf{v}^f$, of all nodes. For the coefficient of friction, we use a global value of $\mu = 0.3$. We then use weighted least squares to compute our new dynamic velocity: $\mathbf{v}_d^* = \text{argmin}\|\mathbf{v}^f - \mathbf{J}\mathbf{v}_d\|^2_{\tilde{\mathbf{M}}}$, where $\tilde{\mathbf{M}} = \mathbf{M} - \beta h^2 \mathbf{K}$ with $\beta = 0.5$ as in other examples. This solve is inexpensive, since we solve only for the dynamic nodes of domains in contact. When collisions occur with quasistatic nodes, the contact information is added to the global stiffness matrix, making CONJAC be collision-aware. CONJAC intelligently transfers the masses of the quasistatic nodes to the dynamic nodes, giving us a small (24×24 in this case since there are 8 dynamic nodes) and stable system to solve at each time step. Even with only 8 dynamic nodes, CONJAC gives remarkably rich deformations. For example, although the front feet and the two ears only have one dynamic node each, they undergo significant local nonlinear

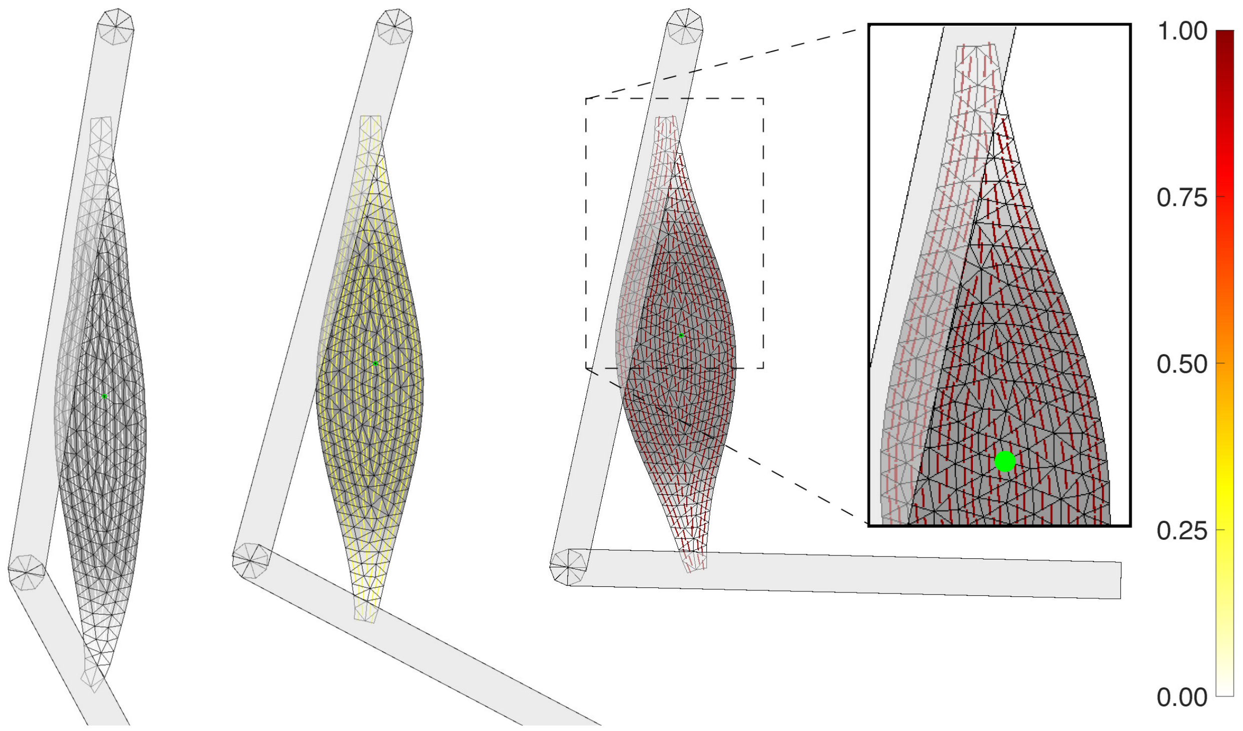


Figure 4.7: MUSCLE scene that combines CONJAC with rigid body dynamics. There is one dynamic node in the middle of the muscle, colored green in the inset subfigure. The colored lines in the muscle foreground show the fiber directions of the anisotropic Fung material, activated from white to yellow to red. The muscle background is color coded in gray with the stiffness of the SNH material. Reprinted from [13].

deformations upon contact, as shown in Fig. 5.13a and the supplemental video.

ARMADILLO: In our last example, we showcase CONJAC with adaptivity in place. An armadillo is fixed in place by a subset of internal nodes in the center of its body. 5 representative nodes are placed in the center of 5 tetrahedral regions shown by the different colors on the mesh in Fig. 4.10. The simulation begins with all representative nodes in a deactivated (quasistatic) state before gravity introduces the initial dynamic motion that is then followed by a series of pulling forces. As the limbs are pulled and released, the representative nodes activate and deactivate, based on the relative motion in the region. While the body is not pulled itself, it picks many of the shockwave motions that activate it briefly multiple times. This can be seen clearly by the motion in the tail and nose. Even though these regions are not pulled themselves, they react realistically in a quasistatic fashion when nearby limbs are pulled. Fig. 4.11 shows the plots of our liveliness

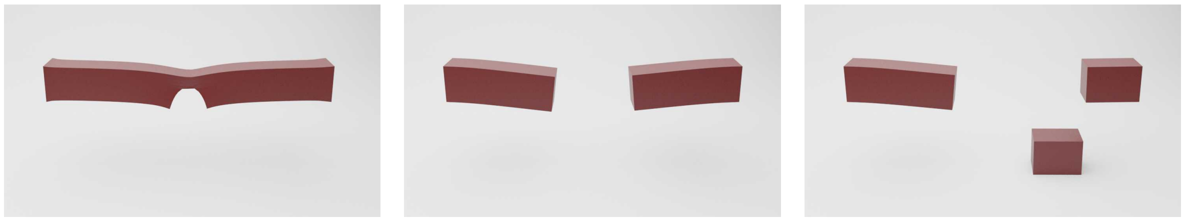

(a) (b) (c)

Figure 4.8: BARCUT scene where a bar is stretched apart, (a) cut, and (b) fully separated into two halves. (c) The right piece is further cut into two pieces. The left and middle pieces are dynamic, while the right piece becomes quasistatic. Reprinted from [13].

(a) (b)

Figure 4.9: BUNNY falling, colliding with the floor, (a) deforming from the impact, (b) bouncing back up. Reprinted from [13].

metrics. The top figure shows the plot of the metric over time with a threshold of $0.02/s$, and the bottom with a threshold of $0.1/s$. In other words, a representative node is dynamic as long as the average stretching speed is greater than 2% or 10% per second. The inset figures show close-ups of these plots. It can be clearly seen that with the lower threshold, the dynamics is retained longer, since the representative nodes remain active for longer.

4.6 Conclusion

CONJAC is a new reduced coordinate approach based on condensation. Unlike previous work, we apply condensation at the velocity level by defining a mapping that expresses the velocities of quasistatic DOFs as a linear function of the dynamic DOFs. Compared to VANILLA (the standard, full FE solution), CONJAC remains stable at large time steps and exhibits highly dynamic

Figure 4.10: ARMADILLO (a) resting under gravity and (b) having individual limbs pulled.

motion with less numerical damping. Furthermore, CONJAC gives the exact same configuration as VANILLA once the static state is reached. To demonstrate CONJAC's versatility, we have shown examples involving: a wide range of materials, anisotropy and heterogeneity, topology changes, integration with rigid body dynamics, and adaptivity.

4.6.1 Limitations & Future Work

For CONJAC to maintain its advantages over VANILLA, the dynamic nodes must not be too close to each other. In our DRAGON and BUNNY examples, we manually placed the first few dynamic nodes in strategic locations (*e.g.,* dragon jaws, bunny ears), and the rest were generated randomly. If two dynamic nodes were generated too close to each other, we reran the random generator with a different seed.

Although the stabilization term, $\mathbf{b}_q$ in Eq. 4.12, works well to fight the drift due to the linearization artifacts of the Jacobian, it still cannot maintain the zero net-force state on the quasistatic nodes during motion, causing visual artifacts especially when the motion is large. Rather than taking a single Newton step, taking multiple steps would produce better results when time steps are large. A quasi-Newton approach, where only the force vector, and not the stiffness matrix, is updated every step may yield a good balance between convergence and performance.

In our current implementation, we explicitly form $\mathbf{J}_{qd}$, which requires $3n_d$ solves with $\mathbf{K}_{qq}$, where n_d is the number of dynamic nodes. When n_d is small, the bottleneck is the factorization of $\mathbf{K}_{qq}$, making CONJAC and VANILLA nearly equivalent in terms of computational cost. As we

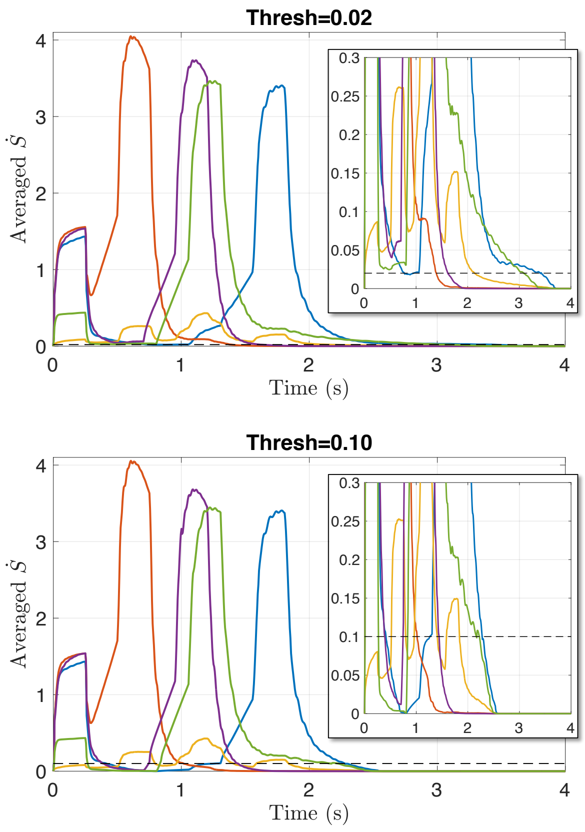


Figure 4.11: Average $\dot{\mathrm{S}}$ values of the representative nodes, color coded to correspond to the ARMADILLO mesh. The top plot is with the threshold set to 0.02, and the bottom with 0.10. Zoomed versions are shown as inset figures. With a higher threshold, the representative nodes become quasistatic earlier.

increase n_d, the solves start to become the bottleneck, making CONJAC more and more expensive compared to VANILLA. However, as shown in §4.5, CONJAC retains important dynamics even with few dynamic nodes. An exciting avenue of future work is to follow the work of Mitchell et al. [109] to decompose the object into domains, which would allow CONJAC to scale up to a very large mesh, since then the factorizations of $\mathbf{K}_{qq}$ can be computed per-domain. However, obtaining good multi-threaded performance would still be a major challenge, requiring careful tuning of domain sizes and topology.

Scaling CONJAC to very large meshes would require an iterative approach, since the factorization of $\mathbf{K}_{qq}$ may not fit into memory. This is non-trivial for the same reason above—the number of

RHS vectors is $3n_d$ where n_d is the number of dynamic nodes. One approach to resolve this issue is the block Krylov method [124], which allows the solver to share information across multiple RHS. However, we would still need to limit n_d to be relatively small to remain competitive.

An important limitation is that frictional impulses acting on quasistatic nodes cannot be accurately handled, since these nodes are not DOFs, and so their frictional impulses can only be satisfied in a least squares sense. This is, however, a limitation common to all reduced coordinate approaches. Therefore, our approach is most suitable when the effects of friction are not too large.

We have found experimentally that CONJAC does not work well for very soft objects, due to severe linearization artifacts. For similar reasons, CONJAC cannot handle extremely fast rotational motion. For these types of simulations, we may need to run Newton's method to convergence, rather than using the linearly implicit Euler scheme.

CONJAC can suffer from locking artifacts with hard constraints if these constraints are applied to quasistatic nodes. In such cases, an averaged or softened constraint will need to be applied, or new dynamic nodes must be inserted [125, 126, 127].

Finally, we are interested in exploring adaptive time step integrators, such as Runge-Kutta-Fehlberg or MATLAB's `ode45` [128, 129]. Given CONJAC's stability at large time steps even with an explicit integrator, these adaptive methods have the potential to reduce the number of total time steps substantially.

5. LDOT: BOOSTING DEFORMATION PERFORMANCE WITH CHOLESKY EXTRAPOLATION[*]

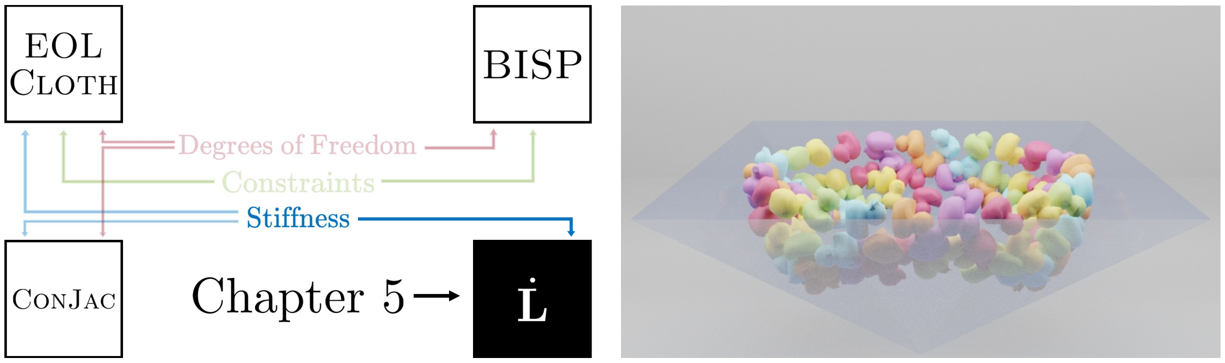


Figure 5.1: (Left) Ldot examines the solvers driving our simulations. (Right) Still frame of a large collection of simulated soft body ducks undergoing collisions and utilizing Cholesky extrapolation in a hybrid solver from Witemeyer et al. [16]. Reprinted from [16].

We propose a method of accelerating nonlinear time integration schemes by extrapolating the Cholesky factorization in time. Unlike previous methods that "freeze" this factorization, we extrapolate it linearly in time and build approximations that are applicable across more nonlinear solver iterations. Our approach supports complex nonlinear materials, including heterogeneity and anisotropy, as well as collisions, including frictional contact and self collisions. The extrapolated Cholesky factor can be used to accelerate both quasi-Newton and Newton solvers. We highlight some shortcomings in this approach, but explore how this extrapolation is an interesting experimental avenue for integration efficiency.

[*]Part this chapter is reprinted with permission from "Qlb: Collision-awware quasi-newton solver with cholesky and l-bfgs for nonlinear time integration" by B. Witemeyer, N. J. Weidner, T. A. Davis, T. Kim, and S. Sueda, *Motion, Interaction and Games*, ser. MIG '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3487983.3488297. Copyright 2021 by Association for Computing Machinery.

5.1 Introduction

Physics-based simulation has become an important tool in computer animation. Starting with the seminal work by Terzopoulos et al. [1], many sophisticated methods have been proposed to improve various aspects of these simulations. An early example of this is the *linearly* implicit Euler integration method of Baraff and Witkin [2], which is still in use today in a production environment due to its favorable blend of efficiency, stability, and visual fidelity [8]. More recently, *nonlinear* integration methods have become increasingly popular in physics-based animation, with examples including BDF1 (1st-order Backward Differentiation Formula) [4], BDF2 (2nd-order Backward Differentiation Formula) [130, 131], TR-BDF2 (Trapezoidal-BDF2) [119], and SDIRK2 (2nd-order Singly-Diagonal Implicit Runge-Kutta) [132]. Quasistatic simulations with time-varying boundary conditions, which require nonlinear solves, have also been extensively studied [93, 133, 134].

The common computational bottleneck of these integration methods is their one or more factorizations and solves. We explore a method for improving the efficiency of these integrators by using an extrapolation of a Cholesky factorization. Unlike previous methods based on "frozen" factorization approaches [135, 136], we look into linearly extrapolating this factor forward in time so that it can be reused later on. Because this extrapolation is not tied to the underlying integrator, it can be applied to any integration method. It also doesn't introduce any limit on the support for important physics based animated features such as complex nonlinear materials, heterogeneity, anisotropy, collisions, and friction.

We propose improving the efficiency of the aforementioned nonlinear methods by using this extrapolation technique. In the process, we examine some algorithmically expensive hurdles that come up in comparison to our "freezing" approach. Direct solvers to sparse linear systems play a major role in this efficiency, so by looking at them on a practical implementation level we see how to leverage them inside the iterative steps of nonlinear solver algorithms.

5.2 Related Work

There are many existing methods for improving the efficiency of simulation-based techniques, but there are always sacrifices made in the process.

Subspace dynamics [99, 101, 100, 137] or condensation [107, 104, 109, 13] can achieve impressive speedups, but they inevitably introduce errors in the resulting motion. We aim to address the efficiency problem from a granularity that does not effect the model of the underlying system. Some of these techniques also require precomputations or basis functions which can restrict robustness of simulations at simulation time.

Unlike these reduction based methods, multigrid methods are capable of producing massive speedups for the full, non-reduced problem [123, 23, 112]. However, significant extensions are necessary to support complex materials involving heterogeneity and anisotropy [138].

Since our method is based on factoring a global matrix and reusing its decomposition, it is similar in flavor to Projective Dynamics, ADMM, and other global-local methods [79, 114, 139, 140, 115, 134]. These methods are highly efficient and can be used for real-time simulations. However, our method is independent of the time-stepping scheme, and can work with any implicit integrator, including higher-order schemes. Furthermore, our method works with an arbitrary material model, including anisotropic and heterogeneous materials.

Position-based Dynamics (PBD) is an attractive option for real-time multi-physics simulation that is capable of modeling everything from rigid and elastic bodies to fluids [80]. Many features have been added over the years, including new material models, new constraints, and more sophisticated integrators [141]. However, PBD is a fundamental departure from classical methods, and cannot be readily deployed without changing the core of an existing simulator.

Similar to our work, Cholesky factorizations have been re-used or updated in geometry processing, but they either apply to local solutions [142, 143], or update based on changing boundary conditions [144]. These methods are efficient for nonlinear solves involving a subset of vertices but are inefficient for nonlinear solves involving all of the vertices of the deformable objects. They further assume that the underlying energy is Laplacian. Several simulation works have also attempted to use updated factorizations, but either only apply to linear [145] or co-rotational materials [146, 109].

5.3 Review of Sparse Cholesky

The stiffness matrix of our simulations is usually indefinite, but previous work has explored analytical techniques for clamping the eigenvalues to be non-zero, making these matrices positive semi-definite [6]. Because the structure of this matrix is based on the connectivity topology of our simulated object, we end up with highly sparse matrices. For dynamic simulations, we sum the diagonal mass matrix to this semi-definite stiffness matrix, giving us a positive definite matrix. Solving sparse positive definite matrices has a long standing history from both the iterative solver and direct solver directions. Iterative solvers are widely popular when the exact solution is not necessarily required or where the size of problem starts to incur memory issues on the machine running the simulation [7]. Direct solvers, on the other hand, ensure an exact solution and are extremely efficient when we are not memory bound [9]. For symmetric positive definite matrices, sparse Cholesky factorization is typically used, and our stiffness matrix structure additionally benefits from supernodal methods. Software packages have taken advantage of these many factors and exploited dense matrix kernels from Basic Linear Algebra Subprograms (BLAS) [147, 148, 149]. We focus our work specifically exploring the CHOLMOD package [150], and extending it towards linear time extrapolation.

Packages such as CHOLMOD divide solving a sparse linear system into three primary parts. These are a symbolic matrix analysis, a numerical factorization, and finally a solve.

The initial analysis is done to determine the structure of the sparse matrix. If the structure is not changed, then this analysis does not need to repeat on subsequent factorization and solving stages. At this stage, it is possible to determine whether or not an up-looking or supernodal approach is best for the given matrix. A maximal supernode is defined as a maximal block of contiguous columns whose diagonal block is fully lower triangular and whose off-block diagonal column sparsity structures are identical [151]. Supernodal methods take advantage of dense blocks within the sparse structure of a matrix in order to achieve efficiency over up-looking methods that work one row at a time. While there is additional overhead introduced with supernodal methods, our stiffness matrix is composed of numerous small dense blocks scattered sparsely and are thus perfectly suited for a supernodal approach.

The factorization step is the bottleneck of sparse Cholesky and aims to decompose our matrix $\mathbf{A}$ into the form $\mathbf{L}\mathbf{L}^\top$. CHOLMOD utilizes a column block pivoting approach in contrast to a row block pivoting approach. In column pivoting, we march through our columns left to right adjusting our columns lower diagonal in regards to previously computed columns and rows, defining a pivot on our diagonal, and updating said column lower diagonal off the pivot. The block version expands these single column operations into block operations that take advantage of BLAS routines to compute them efficiently.

Finally this $\mathbf{L}$ of $\mathbf{A}$ is used to compute the solution to the general problem $\mathbf{A}\mathbf{x} = \mathbf{b}$. Because the decomposition takes the form $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$, the solve takes place in two stages: $\mathbf{L}\mathbf{y} = \mathbf{b}$ and $\mathbf{L}^\top\mathbf{x} = \mathbf{y}$. This solve generates a unique solution which is relatively much less expensive to compute compared to the factorization.

5.4 Cholesky Time Extrapolation

To explore the idea of Cholesky time extrapolation, we will breakdown its formulation and performance factors in order to see how it can be applied to nonlinear solvers.

5.4.1 Formulation

We begin by briefly describing the *frozen* Cholesky approach [135, 136]. This approach improves the efficiency of nonlinear solvers by forming and factoring the system matrix $\mathbf{A}$ only periodically and reusing the frozen factorization $\mathbf{L} = \text{chol}(\mathbf{A})$ multiple times, even when $\mathbf{A}$ changes during the nonlinear solver iterations. This approach was used effectively by Hecht et al. [146] for co-rotational materials, where the Cholesky updates were made not only based on time but also on the locations in the mesh undergoing the largest deformations.

The main problem with the frozen Cholesky approach is that the $\mathbf{L}$ factor can become stale over time—*i.e.,* $\mathbf{L}\mathbf{L}^\top$ no longer approximates $\mathbf{A}$. Therefore, we propose a natural extension: a *linear* extrapolation of $\mathbf{L}$ in time. While more expensive to compute, when this Cholesky is encapsulated as a layer to something grander, such as the iterations of a Preconditioned Conjugate Gradient solver, we can reduce the iteration count and make the extra computation worthwhile.

We augment all functions that compute the force with the time derivative of the corresponding stiffness and damping matrices:

$$(\mathbf{f}, \mathbf{K}, \mathbf{D}, \dot{\mathbf{K}}, \dot{\mathbf{D}}) = \text{evalForce}(\mathbf{x}, \mathbf{v}, \cdots). \tag{5.1}$$

The system matrix $\mathbf{A}$ and its time derivative $\dot{\mathbf{A}}$ can then be computed as

$$\mathbf{A} = \mathbf{M} - h\mathbf{D} - h^2\mathbf{K}, \quad \dot{\mathbf{A}} = -h\dot{\mathbf{D}} - h^2\dot{\mathbf{K}}. \tag{5.2}$$

Similarly, rather than computing just the Cholesky factor $\mathbf{L}$ of $\mathbf{A}$, we also compute its time derivative, $\dot{\mathbf{L}}$:

$$(\mathbf{L}, \dot{\mathbf{L}}) = \text{dchol}(\mathbf{A}, \dot{\mathbf{A}}). \tag{5.3}$$

Once we have $\mathbf{L}$ and $\dot{\mathbf{L}}$, we can linearly extrapolate the Cholesky factor as

$$\tilde{\mathbf{L}} = \mathbf{L} + \Delta t\dot{\mathbf{L}}, \tag{5.4}$$

where Δt is the elapsed time since the factor was computed. During a nonlinear solve for the system state at time step $(k+1)$, instead of using the frozen factor $\mathbf{L}$ from time step (k), we use the extrapolated factor $\tilde{\mathbf{L}}$ as the approximate factor to speed up the convergence.

To help conceptualize this extrapolated Cholesky compared to the frozen Cholesky, we can examine individual matrix values of a simulation factor as they change across time. Fig. 5.2a shows a plot of one such matrix value and takes a time snapshot at the vertical green line to examine the approximations. The red line shows the frozen factor approach, reusing the same factor value until the factor is recalculated. The yellow line shows the Cholesky extrapolation approach, taking a linear step in the direction of the derivative in order to more appropriately approximate the future factors. It is important to note that at the time snapshot of this example, Cholesky approximation is a better fit than the frozen approach, but we can see in 5.2b this does not always hold true. Here the extrapolation overshoots and the frozen approach ends up being a better fit. This fit is also highly dependent on our time step and usage time.

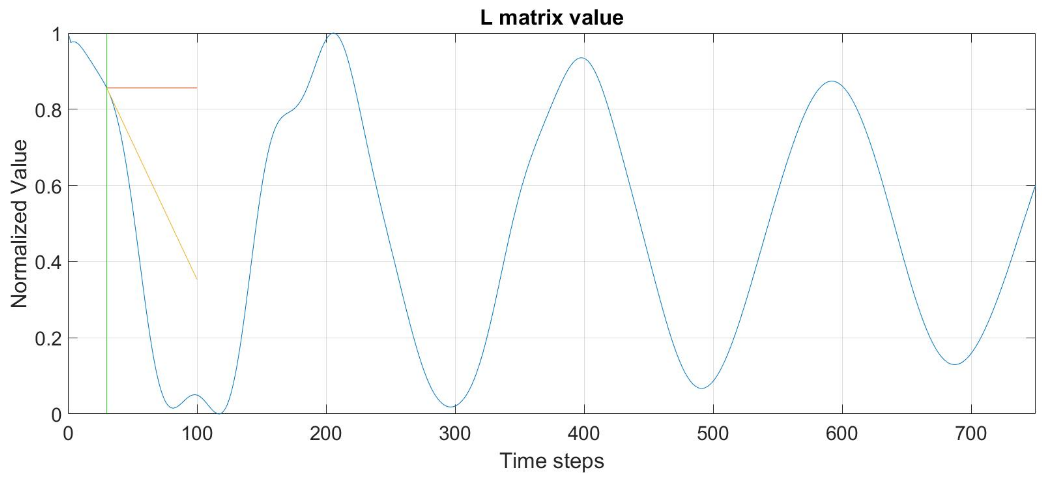
As mentioned prior, we are specifically looking at quasi-Newton and Newton solvers. It is worth noting that in linearly implicit Euler integration methods, the outputted solution is directly influenced by these frozen and extrapolated results. Whenever we forgo refactoring and use a factor approximation instead, we are compounding error into our simulation results. This can create catastrophic errors very early on in simulation time. In contrast, quasi-Newton methods solve for an exact solution and these frozen and extrapolated results only influence the convergence rate of the solve. Here, we do not need to worry about disrupting our simulations with poorly predicted factors and can instead focus on how our iteration counts are impacted with regards to efficiency.

5.4.2 Extrapolation Performance

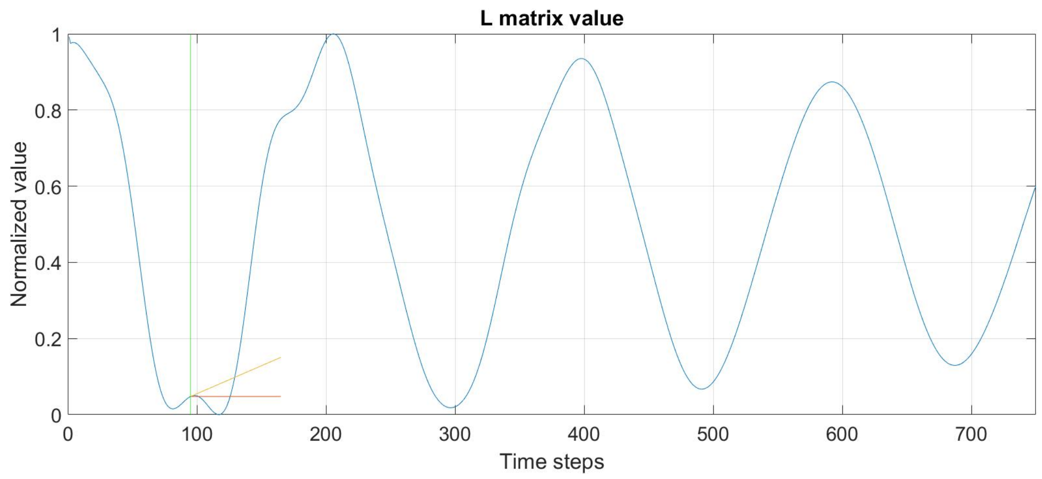
With the aim of improving quasi-Newton solve iterations and not impacting simulation results, Cholesky extrapolation falls fully into the realm of a performance enhancing technique. With this in mind, there are a number of technical components to take into account when considering its impact on the overall system. Working through these, we arrive at a some clear theoretical limits of the Cholesky extrapolation approach alongside some experimental insights.

5.4.2.1 Sparsity Pattern

As mentioned earlier, finite element simulations prove to be very well fit for the sparse supernodal variant of Cholseky factorization. Sparse matrix reordering algorithms such as AMD [152, 153] or METIS [154] can reconfigure our matrices into a collection of very dense blocks perfect for supernodal approaches. At an even finer grain of detail, we can roughly predict the factorization efficiency of a mesh based on its geometric connectivity and what these reordering algorithms will output. At the lowest level, sparse Cholesky is computing blocks of dense Cholesky. A fully dense matrix achieves no efficiency boost from using a sparse approach, and will actually be



(a)



(b)

Figure 5.2: A single value taken from the $\mathbf{L}$ matrix every time step of a simple bar simulation. The peaks and valleys represent dynamic change in the bars movememnt. The green vertical line represents the chosen timestep. The red line represents the frozen value approximation over time. The yellow line represents the linear extrapolation value approximation over time. (a) At this time step, linear extrapolation is much closer to the correct future $\mathbf{L}$ value. (b) At this time step, linear extrapolation overshoots the future $\mathbf{L}$ values. The frozen apprimzation is a much better fit.

slower simply from additional overhead. A sparse matrix which can be reordered to create as many dense blocks of fully non zeros values as possible are our best candidates. We can use various arrangements of connected tetrahedrons as an example. If we take a collective set of independently simulated tetrahedron, we create a best case scenario. Because each tet is completely independent, all matrix values of a tet form a super node without the need for reordering and each block is fully dense. From here we can logical progress to a new connectivity pattern for comparison where each tet is connected by a single node. After reordering we can still achieve very dense super node blocks, but we have introduced zeros which ever so slightly slow our computations in comparison. Now we can accelerate our progression of examples to a worst case scenario. If we create a dense tetrahedral block where the majority of our nodes live inside the mesh and all connect to the maximum amount of neighbors, we create a sparsity pattern that even when reordered is hard to separate into good individualized dense blocks. Using this logic, we can estimate the factor performance of traditional computer graphics simulation meshes and verify them with performance tests. Fig. 5.3 showcases the relative performance of these meshes. These results make sense because visually we can see that the bunny mesh is much blockier and dense compared to the long and skinny dragon mesh. These insights are important to our problem because as we introduce more computationally expensive factoring, we want to know how versatile our methods are.

5.4.2.2 Analytical vs. Finite Differencing

Initially, we derived the analytic forms of all the time derivatives, including the stiffness velocity $\dot{\mathbf{K}}$ of various material types, and the corresponding sparse Cholesky derivative $\dot{\mathbf{L}}$. However, extensive experiments showed that a simple finite differencing scheme is more efficient for both of these calculations, and its accuracy is sufficient.

In the case of $\dot{\mathbf{K}}$, the hope would be that there is a large cross over of work in computing $\mathbf{K}$ and $\dot{\mathbf{K}}$ in order to maximize the efficiency in computing them both together. Unfortunately, while the prerequisite work for $\dot{\mathbf{K}}$ does include the calculations for $\mathbf{K}$, there is the same if not more matrix work done depending on the material model. The scene-wide, global $\mathbf{K}$ in simulation is formed via the compilation of individual tetrahedral $\mathbf{K}$. These tetrahedral computations are independent and favor parallelism, but due to connectivity are not thread safe at the fill step. Experimentation showed that the amount of additional data handling and work involved with computing $\dot{\mathbf{K}}$ was far too expensive, at least within the limit of feasibly large memory bound problems, when compared to a simple finite differencing approach. Since the derivative is with respect to a scalar (time), it takes two evaluations of $\mathbf{K}$ to compute the approximate derivative. The first evaluation uses the

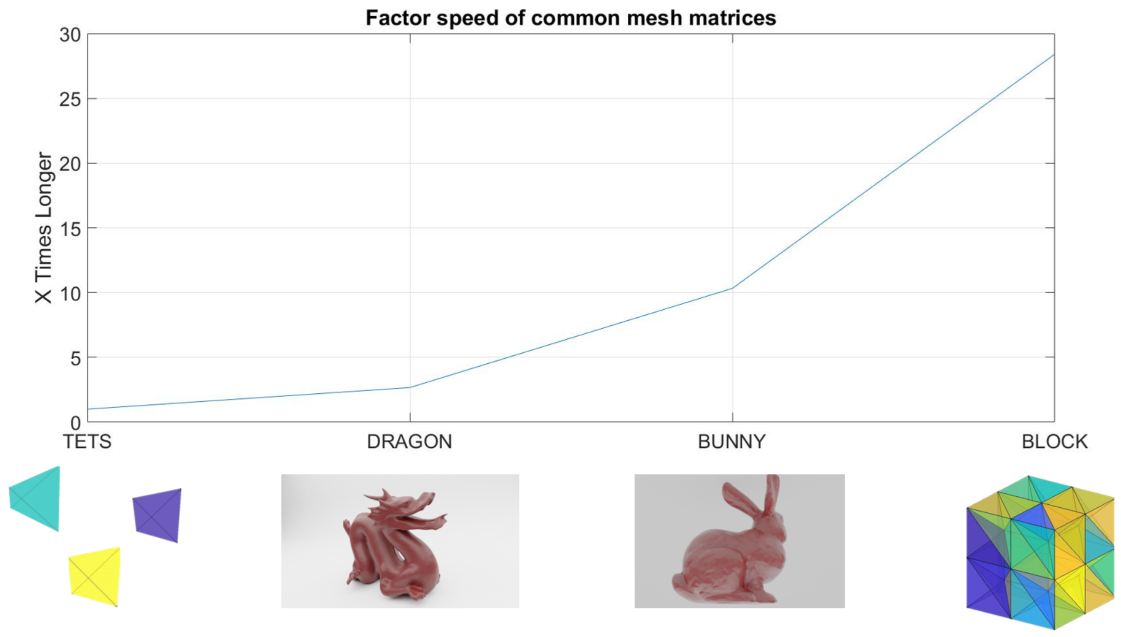


Figure 5.3: Meshes with a high degree of nodal connectivity are slower to factor when compared to sparsely connected meshes. As a best case scenario, a collection of unconnected tetrahdrons creates a perfect collection of dense supernodal blocks without any excess values. In the worst case scenario, a pure block of tetrahedrons with max connectivity internally creates an ill fit supernodal block structure. In between is a long skinny dragon mesh which is relatively fast, and a blocky bunny mesh which is relatively slow. All of these meshes have roughly the same number of nodes.

tetrahedrons nodal positions at the current time.

$$(\mathbf{K}) = \text{evalTetStiffness}(\mathbf{x}). \tag{5.5}$$

We then perturb the positions by the current nodal velocities.

$$(\tilde{\mathbf{K}}) = \text{evalTetStiffness}(\mathbf{x} + \epsilon\mathbf{v}). \tag{5.6}$$

Now the approximated derivative becomes

$$\dot{\mathbf{K}} \approx (\tilde{\mathbf{K}} - \mathbf{K})/\epsilon. \tag{5.7}$$

This proved to be more efficient than the analytical approach, and proved to be well within floating point precision in regards to accuracy. Finite-difference approximation also has the attractive property of automatically incorporating the clamped eigenvalues used to enforce semi-positive definiteness of the stiffness matrix [93]. Clamping these inside the analytical steps increases the computation time even more and, while not a vital step, increases the robustness of simulations.

A similar situation arises in the case of $\dot{\mathbf{L}}$. The sparse Cholesky derivative can be approximated with finite differencing as

$$\dot{\mathbf{L}} \approx (\text{chol}(\mathbf{A} + \epsilon\dot{\mathbf{A}}) - \text{chol}(\mathbf{A}))/\epsilon. \tag{5.8}$$

For the analytical approach to be more efficient, $\mathbf{L}$ and $\dot{\mathbf{L}}$ must be computed in less time than calling `chol()` twice. This is a challenge, because computing $\dot{\mathbf{L}}$ alone takes at least twice the time of computing $\mathbf{L}$ [155]. For experimental validation, we attempted extending CHOLMOD's sparse supernodal Cholesky function to support the computation and return of $\dot{\mathbf{L}}$ alongside $\mathbf{L}$. A nicety of this extension, was that the matrix structure of these two factors would be identical, meaning we could share the ordering, supernodal breakdown, and traversal between the two. To showcase this we can compare the relatively simpler dense `chol()` and `dchol()` column based algorithms. Algorithm 5.1 shows the process of factorizing a symmetric positive definite matrix into the lower triangular matrix satisfying $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$. This algorithm operates across the columns of the matrix in three stages per column. The first stage modifies the working column using the previously calculated columns. The second stage defines the pivot as the current diagonal value. Finally, the third stage uses the pivot to adjusts the values under it. Algorithm 5.2 works through the same process doing the same amount of work with new $\dot{\mathbf{L}}$ work on top of it. The most glaring efficiency complication with this approach comes from the matrix-matrix multiplication that occurs in the row operations stage on line 9. The initial multiplication `A=B*C` undergoes a derivation that turn it into `dA=dB*C+B*dC`, which means that computing `A` *and* `dA` takes about 3x the amount of work of computing just `A`. For pedagogical reasons, Appendix C.1 shows this same `dchol()` algorithm in the sparse block sense to replicate CHOLMOD's supernodal block algorithm. We still run into the same efficiency complications in the sparse algorithm and, at the lowest function

Algorithm 5.1 Column Pivoting Dense Cholesky

```
procedure CHOL($\mathbf{A}$) $\to \mathbf{L}$
    $\mathbf{L} = \texttt{tril}(\mathbf{A})$                         ▷ extract the lower triangular portion of $\mathbf{A}$
    $n$ = rows in $\mathbf{L}$
    for $j = 1 : n$ do
        for $k = 1 : j - 1$ do                                    ▷ row operations
            for $i = j : n$ do
                $\mathbf{L}(i,j) = \mathbf{L}(i,j) - \mathbf{L}(i,k) * \mathbf{L}(j,k)$
            end for
        end for
        $\mathbf{L}(j,j) = \texttt{sqrt}(\mathbf{L}(j,j))$                        ▷ define pivot
        for $j = 1 : n$ do                                        ▷ adjust subdiagonal
            $\mathbf{L}(i,j) = \mathbf{L}(i,j)/\mathbf{L}(j,j)$
        end for
    end for
end procedure
```

level, CHOLMOD takes advantage of BLAS routines which have already been highly optimized. This boils down to the situation of introducing necessary low level matrix multiplication routines that more than double the overall work. The higher level advantages we leverage such as reuse of the sparsity pattern is something that a finite differencing approach can also do. This leaves us with another situation where it simply becomes more efficient and easier to implement by just computing $\dot{\mathbf{L}}$ with equation 5.8. Another potential approach is to use the recently introduced complex-step finite differencing scheme [156]. However, it is still not faster than the standard real-step finite differencing, and incorporating complex steps into a sparse Cholesky derivative is non-trivial.

5.4.2.3 Applications in Nonlinear Integration

While the Cholesky extrapolation application is not restricted to a single nonlinear integrator, we can examine the performance with the relatively easy to understand BDF1 [4]. Let $\mathbf{x}$ and $\mathbf{v}$ be the nodal positions and velocities of a volumetric solid, and $\mathbf{M}$ be the constant mass matrix. Furthermore, let $\mathbf{f}(\mathbf{x}, \mathbf{v})$ be the force vector, and derivatives $\mathbf{D} = d\mathbf{f}/d\mathbf{v}$ and $\mathbf{K} = d\mathbf{f}/d\mathbf{x}$ be the damping and stiffness matrices. We use a trailing superscript with parenthesis to denote the time

Algorithm 5.2 Column Pivoting Dense Cholesky with Derivative

```
procedure DCHOL($\mathbf{A}, \dot{\mathbf{A}}$) → ($\mathbf{L}, \dot{\mathbf{L}}$)
    $\mathbf{L} = \texttt{tril}(\mathbf{A})$                    ▷ extract the lower triangular portion of $\mathbf{A}$
    $\dot{\mathbf{L}} = \texttt{tril}(\dot{\mathbf{A}})$
    $n$ = rows in $\mathbf{L}$
    for $j = 1 : n$ do
        for $k = 1 : j-1$ do                                ▷ row operations
            for $i = j : n$ do
                $\mathbf{L}(i,j) = \mathbf{L}(i,j) - \mathbf{L}(i,k) * \mathbf{L}(j,k)$
                $\dot{\mathbf{L}}(i,j) = \dot{\mathbf{L}}(i,j) - \dot{\mathbf{L}}(i,k) * \mathbf{L}(j,k) - \mathbf{L}(i,k) * \dot{\mathbf{L}}(j,k)$
            end for
        end for
        $\mathbf{L}(j,j) = \texttt{sqrt}(\mathbf{L}(j,j))$            ▷ define pivot
        $\dot{\mathbf{L}}(j,j) = 0.5 * \dot{\mathbf{L}}(j,j)/\mathbf{L}(j,j)$
        for $j = 1 : n$ do                                  ▷ adjust subdiagonal
            $\mathbf{L}(i,j) = \mathbf{L}(i,j)/\mathbf{L}(j,j)$
            $\dot{\mathbf{L}}(i,j) = (\dot{\mathbf{L}}(i,j) - \mathbf{L}(i,j) * \dot{\mathbf{L}}(j,j))/\mathbf{L}(j,j)$
        end for
    end for
end procedure
```

step.

With BDF1 we solve the nonlinear system to advance the state from k to $k+1$:

$$\mathbf{M}\mathbf{v}^{(k+1)} = \mathbf{M}\mathbf{v}^{(k)} + h\mathbf{f}^{(k+1)} \tag{5.9}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + h\mathbf{v}^{(k+1)} \tag{5.10}$$

where h is the time step. We substitute 5.9 into 5.10 to arrive at the following equation:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + h(\mathbf{v}^{(k)} + h\mathbf{M}^{-1}\mathbf{f}^{(k+1)}) \tag{5.11}$$

We now solve for a root $\mathbf{x}^{(k+1)}$ that satisfies the nonlinear system. After some rearranging, the

Newton search direction is then $\Delta \mathbf{x}_i = -\mathbf{H}_i^{-1}\mathbf{g}_i$, with

$$\mathbf{g}_i = \mathbf{M}(\mathbf{x}_i^{(k+1)} - \mathbf{x}^{(k)} - h\mathbf{v}^{(k)}) - h^2\mathbf{f}_i^{(k+1)}, \quad \mathbf{H}_i = \mathbf{M} - h\mathbf{D}_i^{(k+1)} - h^2\mathbf{K}_i^{(k+1)}, \tag{5.12}$$

where i indicates the current iteration of the nonlinear solve.

As previously mentioned, the frozen Cholesky approach improves the efficiency of this integration by forming and factoring the system matrix $\mathbf{H}$ only periodically and reusing the frozen factorization $\mathbf{L} = \text{chol}(\mathbf{H})$ multiple times, even when $\mathbf{H}$ changes during the nonlinear solver iterations. To compute the search direction, only the forward/backward solves are required: $\Delta \mathbf{x}_i = -\mathbf{L}^{-\mathbf{T}}\mathbf{L}^{-1}\mathbf{g}_i$. This $\mathbf{L}$ factor becomes stale over time as $\mathbf{L}\mathbf{L}^{\mathbf{T}}$ is no longer a good approximation of $\mathbf{H}$.

The problem of becoming stale is not void when using Cholesky extrapolation. If our approximation deteriorates slower than the increased cost of its calculation, though, we can improve the overall performance of our nonlinear solver. We explored two main implementations of this idea: a Quasi-Newton and Full-Newton approach shown in algorithms 5.3 and 5.4. In the Quasi-Newton approach, we use the extrapolated $\mathbf{L}$ in order to directly solve $\Delta \mathbf{x}_i$ each iteration. As $\mathbf{L}$ becomes less and less accurate, so does $\Delta \mathbf{x}_i$, which can have adverse effects on the overall iteration count of the solve. In the Full-Newton approach, we replace the direct solve with an Preconditioned Conjugate Gradient (PCG) solve that is preconditioned with the extrapolated $\mathbf{L}$. This approach benefits from accurate $\Delta \mathbf{x}_i$ results in order to minimize the overall Newton iterations, but in turn increases the solve time per iteration as the PCG can never outperform the time of a single direct solve.

5.5 Results

We built our Cholesky extrapolation within the CHOLMOD C++ library and implemented our solvers in C++ running on a consumer laptop with an Intel Core i9-9880H CPU @ 2.3 GHz and 16 GB of RAM. All of our Cholesky extrapolation experiments were used as preliminary tests that evolved into the project QLB: Collision-Aware Quasi-Newton Solver with Cholesky and L_BFGS

Algorithm 5.3 Quasi-Newton solver

```
1: procedure QUASINEWTON($\tilde{\mathbf{L}}, \mathbf{x}_0$) → ($\mathbf{x}$)
2:     $i = 0, \mathbf{x} = \mathbf{x}_0$
3:     while not converged do
4:         Compute $\mathbf{b}(\mathbf{x})$
5:         $\Delta\mathbf{x} = -(\tilde{\mathbf{L}}\tilde{\mathbf{L}}^\top)^{-1}\mathbf{b}$                ▷ direct solve
6:         Line search for $\lambda$
7:         $\mathbf{x} \mathrel{+}= \lambda\Delta\mathbf{x}$
8:         $i$++                                   ▷ break if iteration limit reached
9:     end while
10: end procedure
```

Algorithm 5.4 Newton solver

```
1: procedure NEWTON($\tilde{\mathbf{L}}, \mathbf{x}_0$) → ($\mathbf{x}$)
2:     $i = 0, \mathbf{x} = \mathbf{x}_0$
3:     while not converged do
4:         Compute $\mathbf{A}(\mathbf{x}), \mathbf{b}(\mathbf{x})$
5:         Compute $\Delta\mathbf{x}_0$                         ▷ initial guess for PCG
6:         $\Delta\mathbf{x} = -\mathrm{PCG}(\mathbf{A}, \mathbf{b}, \tilde{\mathbf{L}}, \Delta\mathbf{x}_0)$
7:         Line search for $\lambda$
8:         $\mathbf{x} \mathrel{+}= \lambda\Delta\mathbf{x}$
9:         $i$++                                   ▷ break if iteration limit reached
10:    end while
11: end procedure
```

for nonlinear Time Integration by Witemeyer et al. [16]. Ultimately, the usage of Cholesky extrapolation was dropped due to unsatisfactory performance results. We mentioned in Section 5.4.1, that the linear extrapolation can be a good prediction of future $\mathbf{L}$ values. As Fig. 5.2b showed, there are situations where are linear extrapolation severely overshoots the values we are trying to predict and the frozen factorization ends up yielding closer results. These peaks in values were often associated with large dynamic changes in the simulation such as collisions or vibrations. Many practical experimental simulations undergo changes of varying intensity throughout a simulated object and do not produce uniformity capable of good extrapolation prediction.

Explored to various degrees were combinations of hybrid solvers that toggled the use of New-

Algorithm 5.5 BDF1 solver

```
1: procedure BDF1($\mathbf{x}^{(k)}$) → ($\mathbf{x}^{(k+1)}$)
2:     Compute $\mathbf{A}(\mathbf{x}^{(k)})$
3:     $(\mathbf{L}, \dot{\mathbf{L}}) = \text{dchol}(\mathbf{A}, \dot{\mathbf{A}})$
4:     $\tilde{\mathbf{L}} = \mathbf{L} + h\dot{\mathbf{L}}$                    ▷ approx. factor
5:     $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)} + h\mathbf{v}^{(k)}$                ▷ initial guess
6:     if Quasi then
7:         $\mathbf{x}^{(k+1)} = \text{QuasiNewton}(\tilde{\mathbf{L}}, \mathbf{x}_0^{(k+1)})$
8:     else
9:         $\mathbf{x}^{(k+1)} = \text{Newton}(\tilde{\mathbf{L}}, \mathbf{x}_0^{(k+1)})$
10:    end if
11: end procedure
```

Table 5.1: Test objects. *verts*: total number of vertices. *tets*: total number of tets. *mass*: total mass (kg). Reprinted from [16].

	verts	tets	mass
BAR	10,125	51,744	3.00
ARMADILLO	25,317	98,486	1.37
BUNNY	3,907	14,374	2.25
DUCKS	12,800	36,800	1.31

ton and Quasi-Newton iterations in order to best fit certain scene setups. We found success adaptively switching between these methods during simulation in moments of high and low dynamics. To explore the results, the baseline method is referred to with NEWTON-D, which uses a direct method (CHOLMOD) to solve the linear system. Next, NEWTON-L_0 uses PCG with a frozen Cholesky preconditioner. Finally, our method is HYBRID-L_1 which uses a hybrid approach with a linearly extrapolated Cholesky factor. We use 1E-6 as the relative and absolute tolerance for the Newton and quasi-Newton solvers, and 1E-2 as the PCG tolerance. This relatively high PCG tolerance was chosen experimentally—starting at 1E-6, we increased this tolerance by an order of magnitude until the number of Newton/quasi-Newton iterations started to creep up. Testing was done on a variety of integrators, Young's modulus, Poisson's Ratios, Heterogeneity, Anisotropy, and collisions which can be seen in Table 5.2.

Table 5.2: Scene parameters. *vary*: scene parameter to vary. *material*: material model. E: Young's modulus (Pa). μ: Poisson's ratio. *damping*: damping coefficient relative to E. h: time step (s). *integrator*: time integrator scheme. Reprinted from [16].

	vary	material	E	μ	damping	h	integrator
BAR	E	SNH	6E3, 2E4, 5E4, 1E5	0.49	-	5E-2	quasistatics
BAR	μ	SNH	2E4	0.42, 0.49	-	5E-2	quasistatics
BAR	hetero	SNH	8E3, 2E4	0.49	-	5E-2	quasistatics
BAR	aniso	aSTVK+SNH	4E4+2E4	0.49	-	5E-2	quasistatics
ARMADILLO	damping	SNH	1E7	0.49	0, 1E-6	1E-2	BDF2
ARMADILLO	material	Corot, StVK, SNH	1E7	0.49	0	1E-2	BDF2
ARMADILLO	integrator	SNH	1E7	0.49	0	1E-2	Baraff, BDF1, BDF2
ARMADILLO	h	SNH	1E7	0.49	0	5E-3, 1E-2, 2E-2	BDF2
BUNNY	-	SNH	5E4	0.49	2E-2	3E-3	SDIRK2
DUCKS	-	SNH	5E4	0.49	2E-3	1E-2	SDIRK2

5.5.1 BAR

Our first experiment is a quasistatic bar undergoing time-varying boundary conditions. Over a period of 5 simulation seconds, the bar is compressed, stretched, bent, and twisted by rigidly moving the vertices of the free end of the bar. We vary several different parameters to see the effect on the convergence of the nonlinear solvers.

Fig. 5.4 shows how the speeds of various methods compare to simulate the BAR scene with the default parameters. NEWTON-D and NEWTON-I are the standard Newton Methods with direct and iterative solvers. Since their performance are similar, we use only NEWTON-D as our baseline. NEWTON-L_0 and NEWTON-L_1 are also Newton Methods, but they use the frozen and extrapolated Cholesky factors as preconditioners, respectively. Their performances are similar—with NEWTON-L_1, the added cost of computing $\dot{\mathbf{L}}$ nullifies the advantage of using a better preconditioner. HYBRID-L_0 and HYBRID-L_1 use our hybrid approach where the frozen/extrapolated Cholesky factors are used first for the quasi-Newton solver and then for the Newton solver. Since quasi-Newton is not as robust as Newton, we observe that the frozen Cholesky factor in HYBRID-L_0 often fails, making it waste a lot of computation. On the other hand, quasi-Newton in HYBRID-L_1 converges frequently, resulting in a faster simulation time. For the rest of this section, we will only look at NEWTON-D, NEWTON-L_0, and HYBRID-L_1.

5.5.1.1 *Young's Modulus*

We run the simulation with the Young's modulus values of $6\text{E}3$, $2\text{E}4$, $5\text{E}4$, and $1\text{E}5$. As shown in Fig. 5.5a, our HYBRID-L_1 method works well across a range of stiffness values. With the default Young's modulus value of $2\text{E}4$, our HYBRID-L_1 is 3.3x faster than NEWTON-D and 2.12x faster than NEWTON-L_0. With all three methods, the softer object takes the longest to compute due to the larger deformation changes between the quasistatic time steps. However, we did notice that for very soft objects for which many elements may become indefinite, NEWTON-D is the most robust, probably because the frozen or extrapolated Cholesky factors cannot account for newly clamped elements during the nonlinear solve [93]. When the BAR gets stiffer, HYBRID-L_1 remains fast but slows down relatively, whereas NEWTON-D keeps the same speed. Fig. 5.7 shows the iteration data per time step for the default parameter of $E = 2\text{E}4$. The top plot shows the Newton iterations (NEWTON-D and NEWTON-L_0), and the bottom plot shows the Hybrid iterations (HYBRID-L_1). With the hybrid solver, we switch from quasi-Newton to Newton after 40 iterations. On some time steps, the quasi-Newton solver fails during the line search, in which case we switch to the Newton solver.

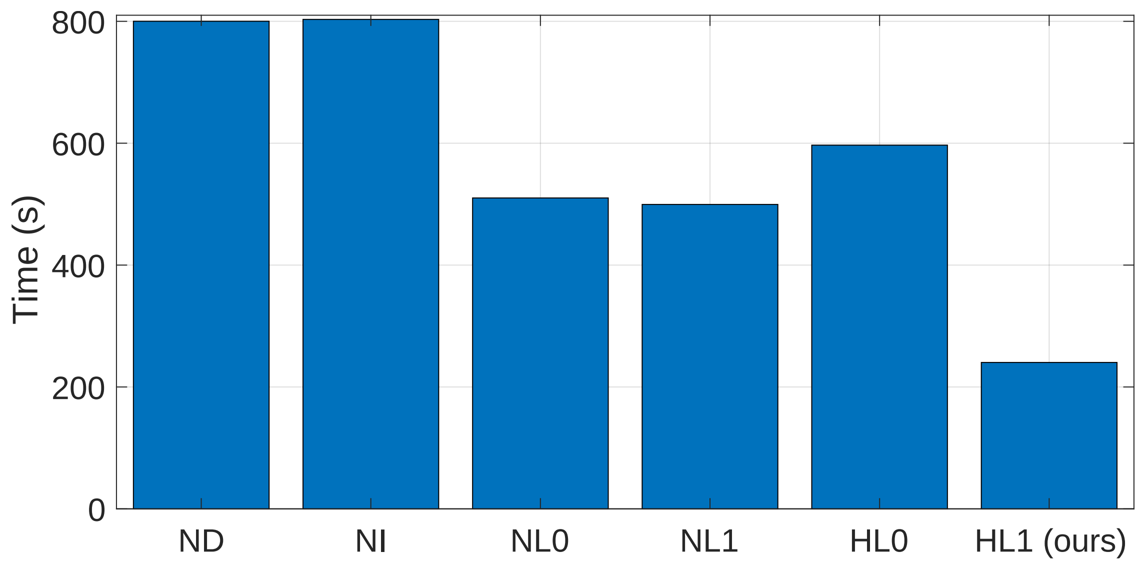


Figure 5.4: Wall-clock time comparison of different methods for the default BAR scene. Reprinted from [16].

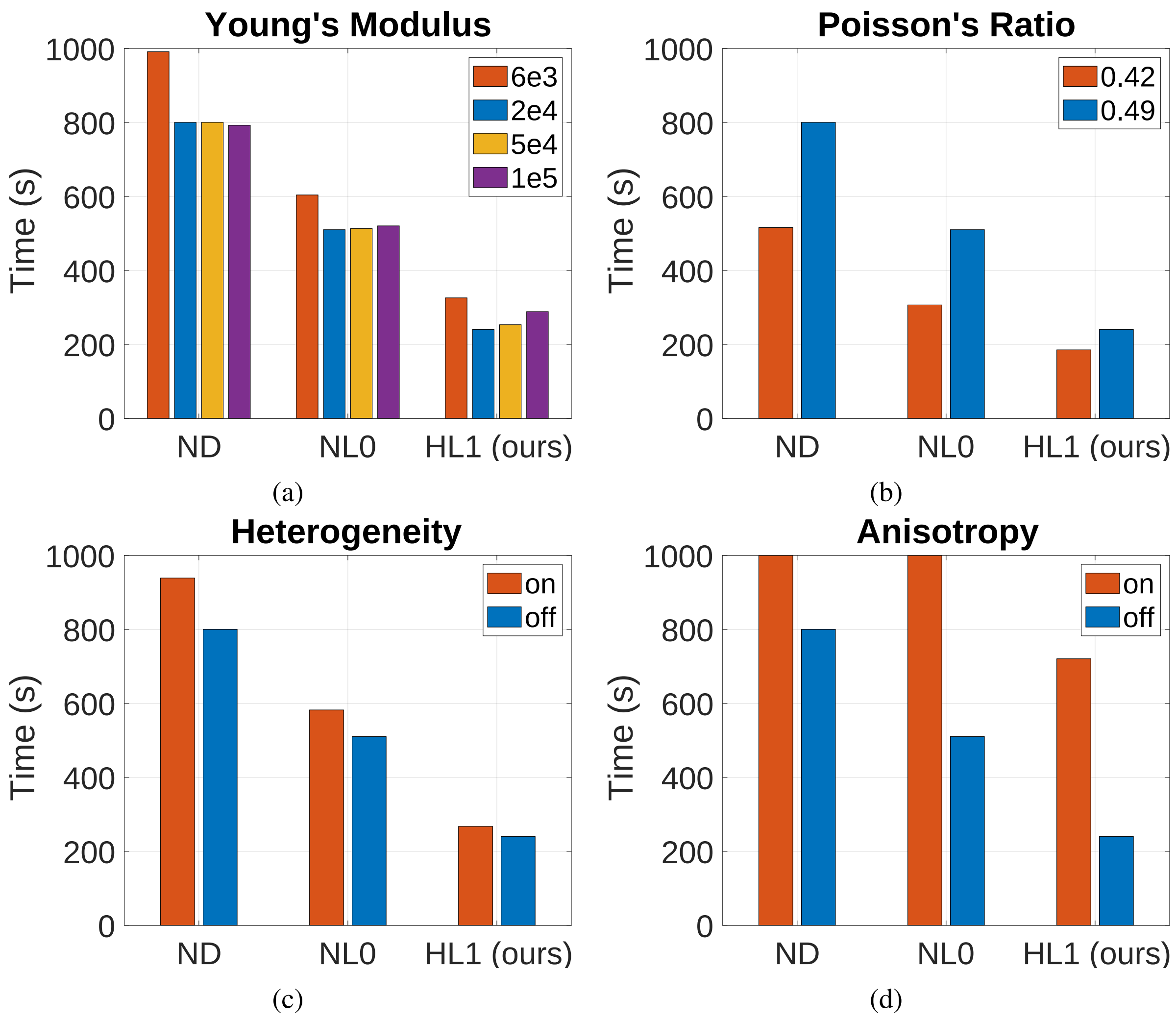


Figure 5.5: Wall-clock times for BAR with: (a) different Young's moduli; (b) different Poisson's ratios; (c) heterogeneity on/off; (d) anisotropy on/off. In all the plots, the BAR with the default parameters are shown in blue. Reprinted from [16].

5.5.1.2 Poisson's Ratio

Next we change the Poisson's ratio to control the amount of volume preservation. With all three nonlinear solvers, the number of iterations is lower when the Poisson's ratio is lower. The relative performance gain by going from the default value of $\mu = 0.49$ down to $\mu = 0.42$ is modest for HYBRID-L_1, but it is still able to beat the baseline methods.

5.5.1.3 Heterogeneity

We repeat the simulation with a heterogeneous material distribution, alternating soft and stiff materials along the length of the bar. When heterogeneity is added, the wall-clock times of all three methods increase due to the added nonlinearity.

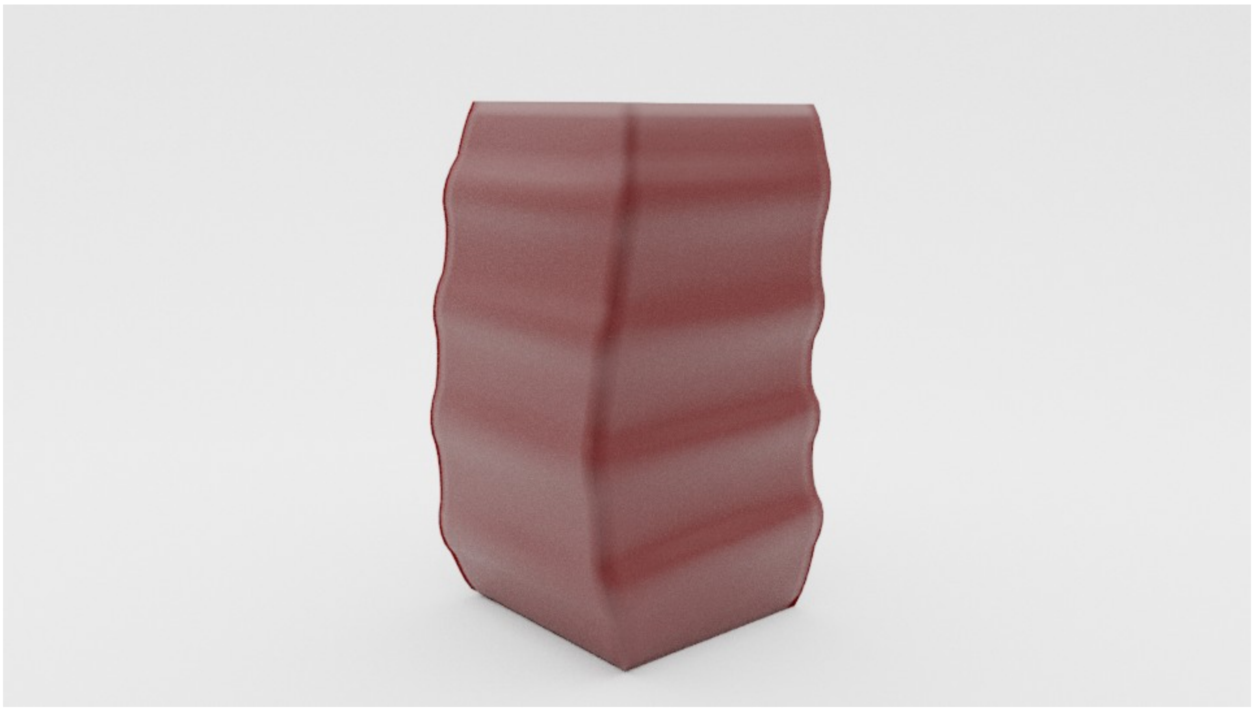


Figure 5.6: Bar undergoing deformation with a heterogeneous material distribution. Reprinted from [16].

5.5.1.4 Anisotropy

Finally, we use anisotropic fibers within the BAR to introduce nonlinearities. On top of the background SNH material, we add the anisotropic StVK material [157] to model the fibers. The added complexity makes the three methods noticeably slower across the board.

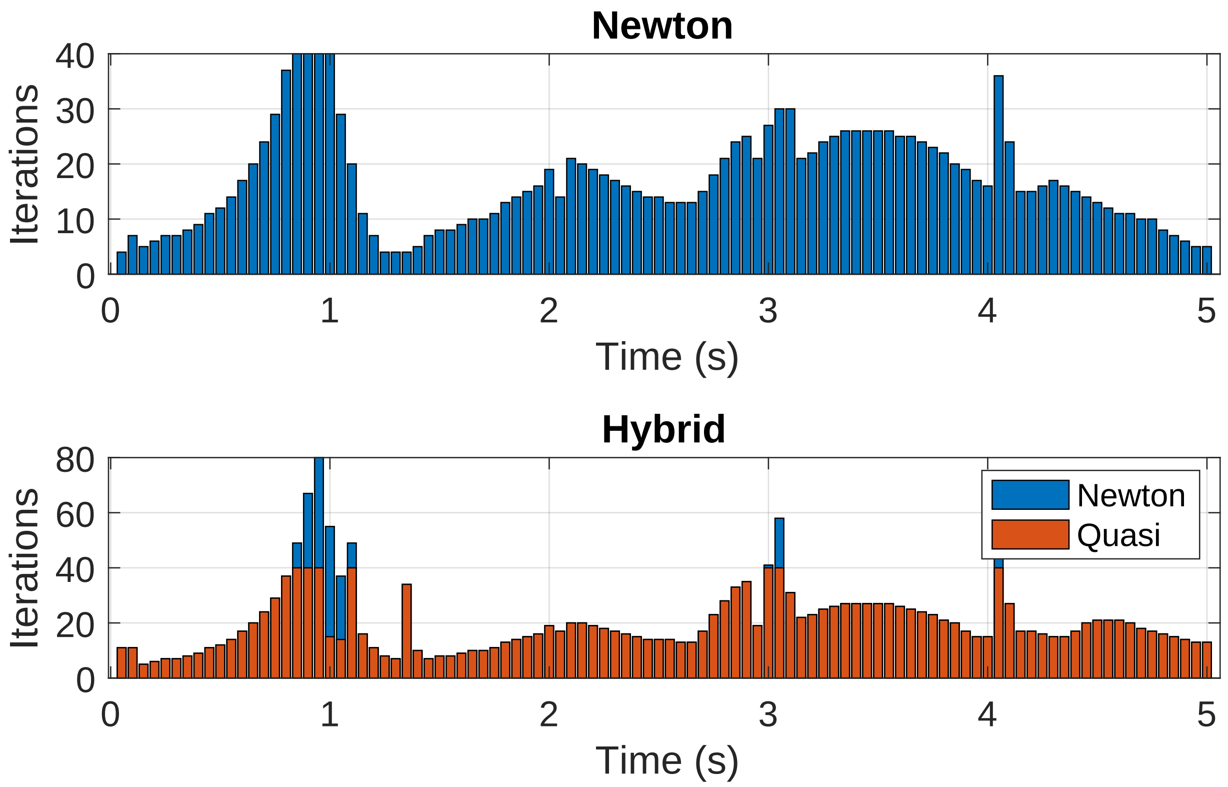


Figure 5.7: Nonlinear iterations for the BAR scene with default parameters. (Top) NEWTON-D and NEWTON-L_0 solvers; (Bottom) HYBRID-L_1 solver. The iterations are the highest during the initial compression phase. With the hybrid solver, most time steps are solved efficiently with the quasi-Newton solver. During some time steps, the quasi-Newton solver fails during the line search and switches to the Newton solver. Reprinted from [16].

5.5.2 ARMADILLO

Next we dynamically simulate the ARMADILLO with the BDF2 integrator. The stiff ARMADILLO is attached to a base by its feet, and the base is moved kinematically to induce motion.

5.5.2.1 Damping

We first add damping. Since we use a higher-order integrator, the added damping is much more controllable and does not depend heavily on the time step (see §5.5.2.4). Fig. 5.9a shows the wall-clock time for the default and damped ARMADILLO. Predictably, with all three methods, the damped simulation finishes more quickly.

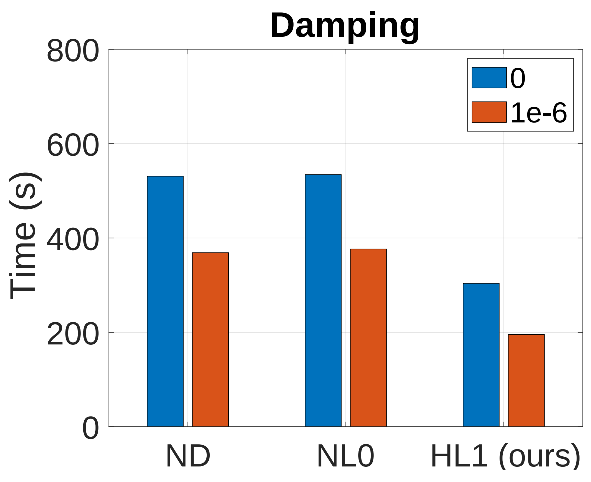
Figure 5.8: Armadillo on a shaken platform to induce deformation. Reprinted from [16].

5.5.2.2 Material

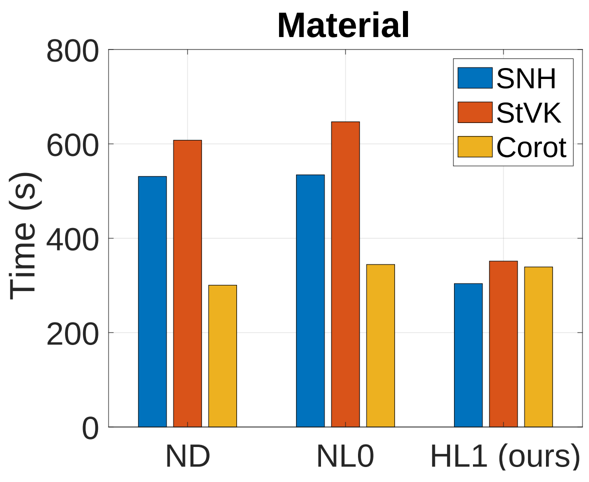
We now change the material model of the object. In addition to the default material of SNH, we use the StVK and co-rotational materials. With StVK, all three methods see some decline in speed. With co-rotation, NEWTON-D and NEWTON-L_0 see dramatic improvement in speed, whereas HYBRID-L_1 sees a minor decline in speed. For simpler, near-linear materials, NEWTON-D and NEWTON-L_0 are the method of choice.

5.5.2.3 Integrator

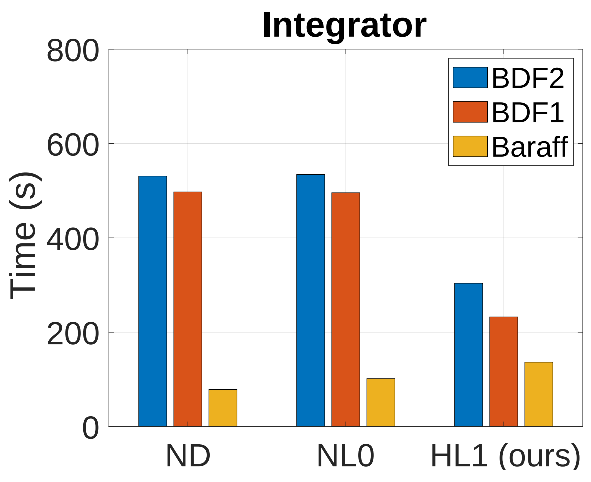
With the same default parameters, we now switch out the time integration scheme. In addition to the BDF2 integrator, we use BDF1 and Baraff, which takes a single step of BDF1 [2]. With BDF1, all three methods become faster. Interestingly, the relative speedup with our HYBRID-L_1 is higher than with the other two. This is most likely due to the excessive artificial damping added by BDF1 and the resulting temporal coherency, which helps keep the extrapolated factor from going stale. For the Baraff integrator, NEWTON-D is clearly the best, since the Cholesky



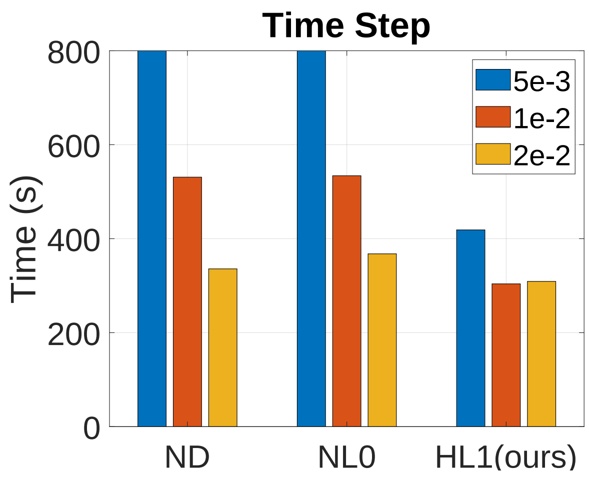
(a)



(b)



(c)



(d)

Figure 5.9: Wall-clock times for ARMADILLO with: (a) different damping coefficients; (b) different materials; (c) different integration schemes; (d) different time steps. In all the plots, the ARMADILLO with the default parameters are shown in blue. Reprinted from [16].

factors cannot be used enough times to make it worthwhile. For NEWTON-L_0 and HYBRID-L_1 in Fig. 5.9c, we used the Cholesky factor for two time steps, so that factorization only happened every other step. Even then, we were not able to beat NEWTON-D.

5.5.2.4 Time step

Using BDF2, we use half and double the default time step of $h = 1\text{E-}2$. At the high time step, the resulting simulation is slightly more damped, due to the small amount of numerical damping in BDF2. If the time step gets very large, the efficacy of the linearly extrapolated Cholesky factor

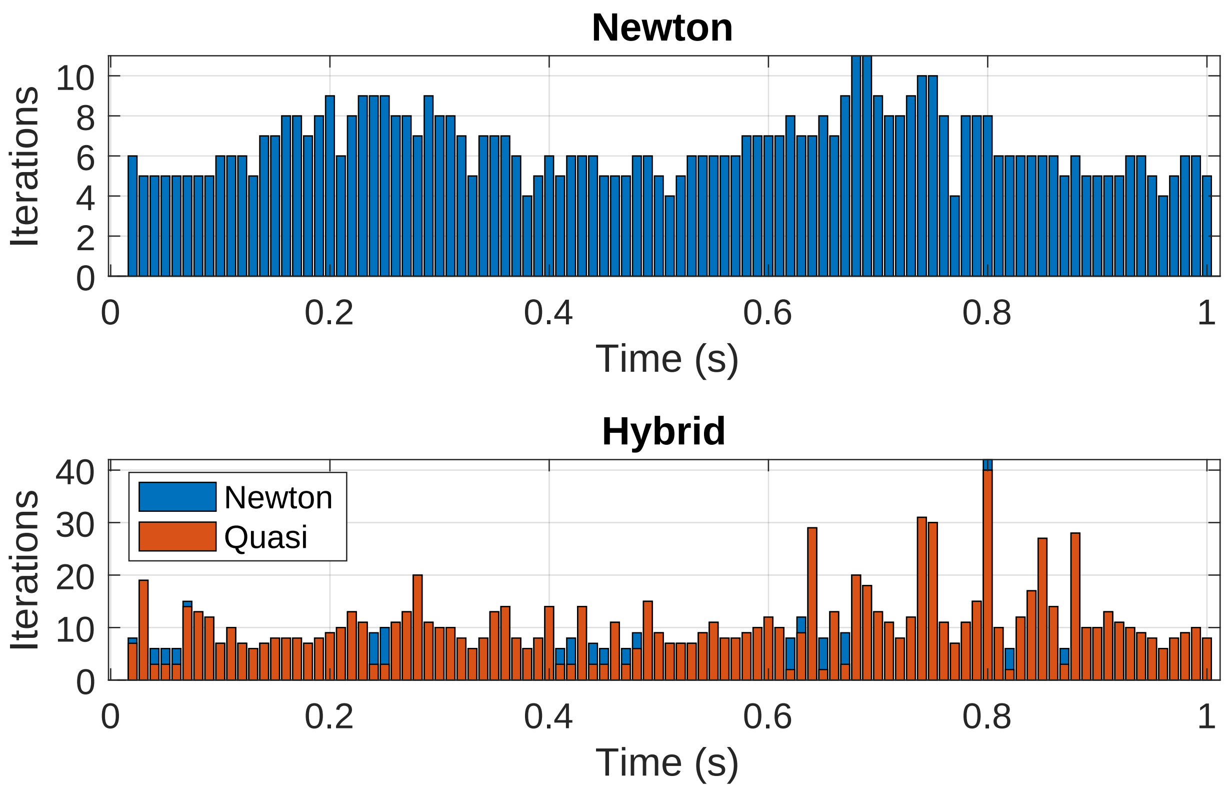


Figure 5.10: Nonlinear iterations for the ARMADILLO scene with default parameters. (Top) NEWTON-D and NEWTON-L_0 solvers; (Bottom) HYBRID-L_1 solver. With the hybrid solver, most time steps are solved efficiently with the quasi-Newton solver. During some time steps, the quasi-Newton solver fails during the line search and switches to the Newton solver. Reprinted from [16].

begins to degrade, making HYBRID-L_1 less attractive. At the lower time step, HYBRID-L_1 retains its advantage over the other methods.

5.5.3 BUNNY

We show self collisions with a BUNNY with the SDIRK2 integrator. We use the contact model by Geilinger et al. [131] for collisions with the floor, and the contact model by McAdams et al. [123] for self collisions. As shown in Appendix C.2.5, the SDIRK2 integrator requires two nonlinear solves per time step. However, unlike BDF2, it is a single-step method and is potentially more suitable for simulations with contact [132] (though Geilinger et al. [131] do use BDF2). We use a relatively small time step of $h = 3\text{E-}3$ to deal with the thin ears. The overall wall-clock times are shown in Fig. 5.13a. For this 3 second simulation, HYBRID-L_1 is 135% and 43% faster than NEWTON-D and NEWTON-L_0, respectively.

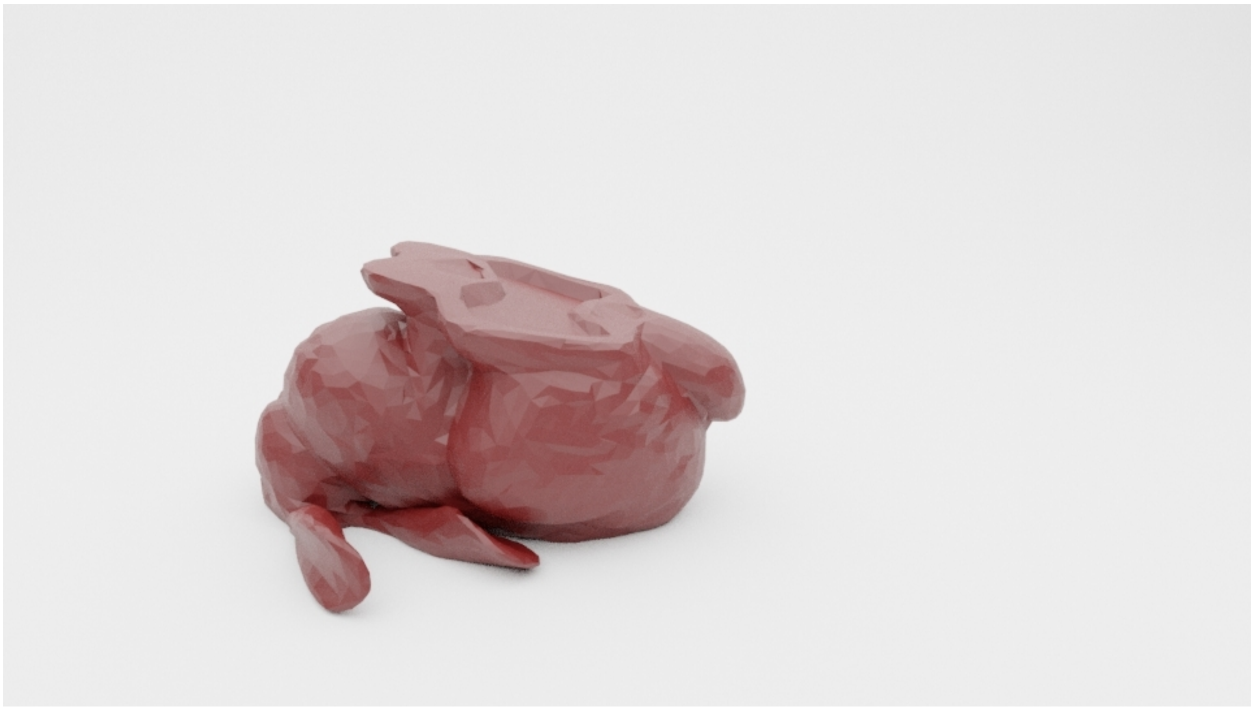


Figure 5.11: Bunny colliding with a floor to induce self collisions. Reprinted from [16].

5.5.4 DUCKS

For the final scene, we simulate 50 and then 100 contacting DUCKS with the SDIRK2 integrator. For added stability, in addition to Green damping (Table 4.1), we also added some velocity-based contact damping forces. Unlike BDF1 or the Baraff integrator, adding damping forces is much more controllable, since changing the time step does not add significant artificial damping. We use NEWTON-D and HYBRID-L_1 for this scene, with a large time step of $h = 1\text{E-}2$. The overall wall-clock times are shown in Fig. 5.13b and the iteration plots are shown in Fig. 5.14. For this 2 second simulation, HYBRID-L_1 is 29% faster than NEWTON-D.

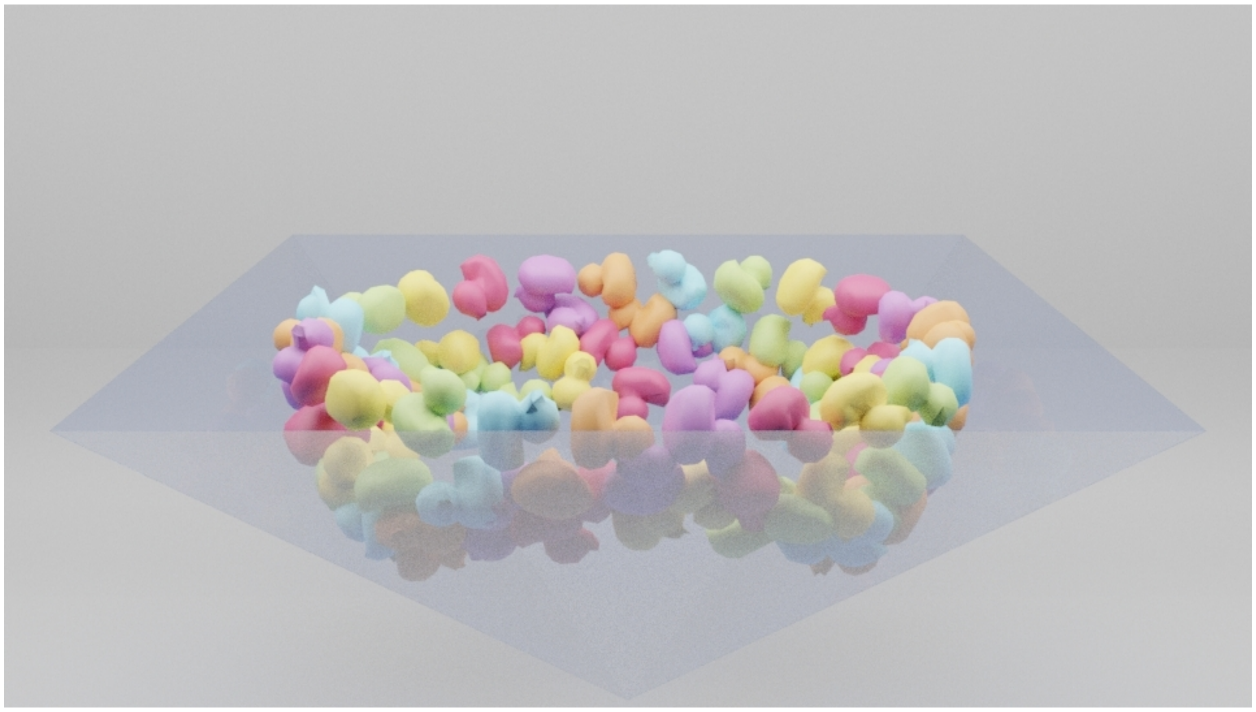
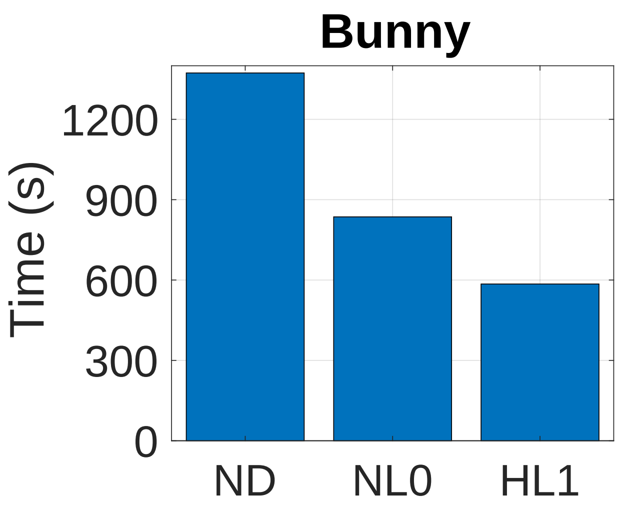
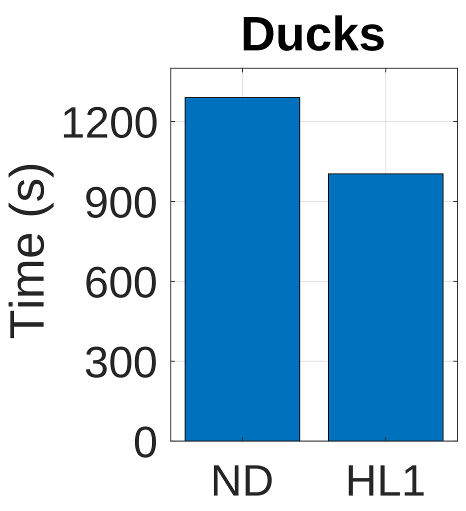


Figure 5.12: Collection of ducks dropped into a bowl and colliding with each other. Reprinted from [16].

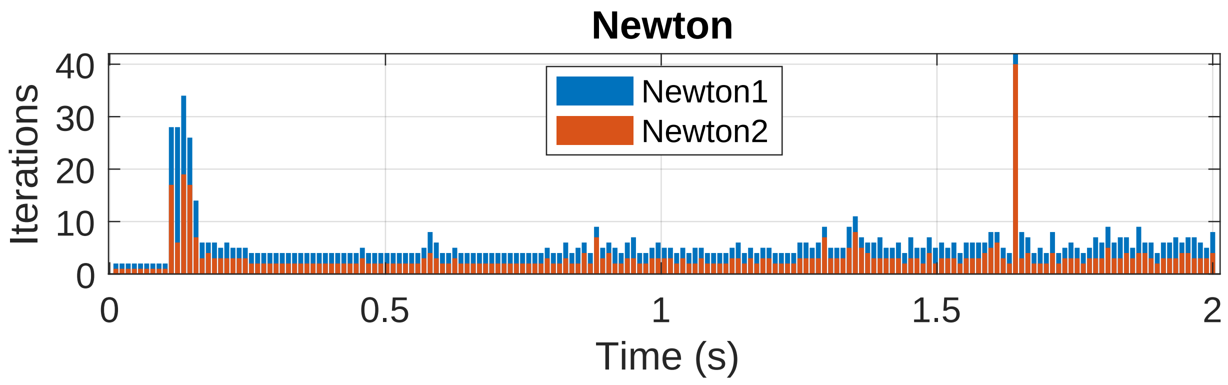


(a)



(b)

Figure 5.13: Wall-clock times for (a) BUNNY and (b) DUCKS. Reprinted from [16].



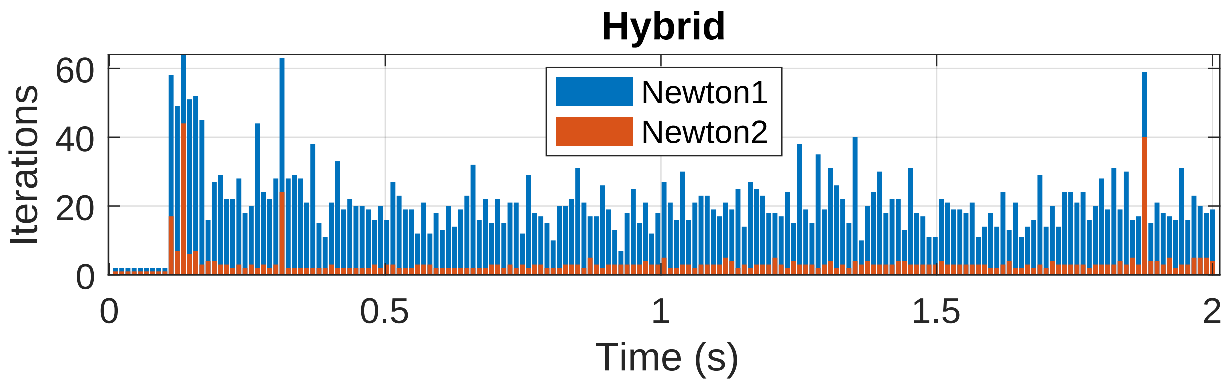


Figure 5.14: Nonlinear iterations for the DUCKS scene. (Top) NEWTON-D: the two nonlinear solves of SDIRK2 are shown. (Bottom) HYBRID-L_1: the two nonlinear solves of SDIRK2 are shown. Each color corresponds to the total number of quasi-Newton and Newton iterations. Reprinted from [16].

5.5.5 Summary of Results

Our approach *works well in a wide range of settings*, but there are some situations in which existing approaches may work better.

- HYBRID-L_1 is most effective with complex nonlinear material models. For simpler near-linear materials, NEWTON-D may be a better choice.
- With nonlinear integrators, HYBRID-L_1 is very efficient, but with the Baraff integrator, NEWTON-D is a better choice since the overhead of factorizations becomes wasteful.
- HYBRID-L_1 works well for a wide range of time steps, but if the time step is very large, NEWTON-D may be a better choice since the extrapolated Cholesky factor becomes stale

more quickly. However, sometimes there are restrictions on the time step due to collisions.

- HYBRID-L_1 can handle a wide range of stiffness values, but for very soft objects for which many elements may become indefinite, NEWTON-D is a better choice, since it can account for newly clamped elements during the nonlinear solve.

Ultimately, these limitations do limit the overall applications of the Cholesky extrapolation approach. Along with that, the overall speed up of this method is not so grand as to make it clearly worth the overall effort of implementation. We do feel these results are interesting to study, but not wholly worth applicable to most simulation environments. With QLB, we see the extension of these ideas and results without Cholesky extrapolation to a much more favorable conclusion.

5.6 Conclusion

Cholesky extrapolation explores the extension of the CHOLMOD library to efficiently compute the time derivative of the factor. Experimentation showed that a finite differencing approach ended up out performing an analytical solution and still retained the required accuracy for dynamic simulation This extrapolation technique is best used for the iterative integration loops in nonlinear solvers where the factor approximation only effects the iteration count and does not destabilize the simulation results. Unfortunately, the non-uniform nature of the $\dot{\mathbf{L}}$ values meant a linear approximation was not always the best approximation for a given scene. Testing showed that the traditional frozen factor approach often performed just as well over the course of a simulation range, and the newly introduced double factor computation time was very rarely worth it.

6. CONCLUSION

Across the broad domain of physics-based animation, I have only touched on a select subset of components and applications. Due to the many knobs to turn when building and tuning simulations, there are always new directions to explore in the name of efficiency, accuracy, and visual fidelity. In this dissertation I focus on three primary components of a simulated system: A) the degrees of freedom of a system; B) the constraints put on that system; C) and the stiffness that derives from force differentiation and in turn enables implicit integration techniques. While these are very much independent components that are only a piece of the overall simulation structure, their scopes are far reaching and intertwined. The degrees of freedom are the foundational components of a configuration state that define any system we look to simulate. At the same time, these degrees of freedom undergo constraints that visually represent many common physical phenomena. Once packaged together, in order to find these constrained degrees of freedom as we step through time, we need some form of time integrator. The popularly used implicit integration techniques are dependent on the force differentiation which are directly defined via the degrees of freedom. Understanding these component's independent contributions and their interactions with each other are the key to advancing the state of the overall simulation problem.

We can look back again on the four different projects that each touched on our three focal categories. *Eulerian-on-Lagrangian Cloth Simulation* [10] uniquely touched on all three by introducing a new simulation framework that robustly simulated cloth sliding over sharp features. This longstanding problem in cloth simulation was tackled by discretizing the cloth into degrees of freedom across the Lagrangian and Eulerian domains. These additional DOFs gave rise to a much more representative set of constraints while still retaining the correct dynamics and supporting traditional implicit integration techniques. REDMAX [11] introduced an efficient and flexible approach for computing the dynamics of articulated rigid bodies. Combining reduced and maximal degrees of freedom and their associated constraints in a novel way allowed for improved efficiency when introducing joint friction with *Bilateral Staggered Projections for Joints*. *Conden-*

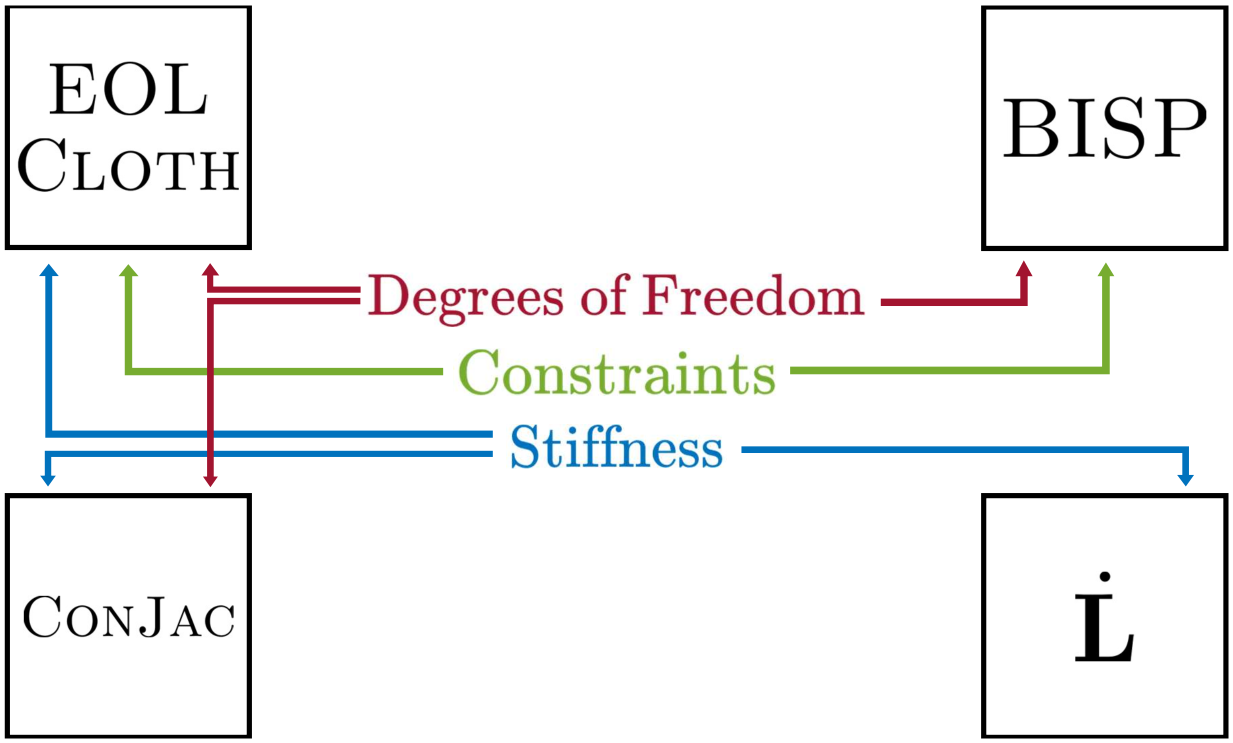


Figure 6.1: A look back on the subcategories of dynamic motion components explored in this work.

sation Jacobian with Adaptivity [13] took a reduced coordinate approach to soft deformable bodies in order to improve the stability of simulations at larger and more desirable time step sizes. By applying condensation techniques at the velocity level, we can adaptively activate DOFs to drive our simulation and encode the neighboring information into a Jacobian matrix. This unique take on the application of the force differentiation means we grow unstable slower without damping out motion. Finally, *Ldot: Boosting Deformation Performance with Cholesky Extrapolation* [16] extended traditional linear solvers in order to cut down the number of highly bottlenecking matrix factorizations. Once we compile our DOF and constraint data into a matrix usable for solving our implicit problem, we take advantage of matrix factor derivations in order to linearly extrapolate a new factor matrix usable for future solves. This cuts down on the iteration counts inside more expensive Newton solvers that could be applied to any number of the previously mentioned applications.

Across all of these projects are new ideas that still hold the possibility for extension. There is no reason why any of these four techniques could not be combined into a single unified solver. A large portion of REDMAX's technical contribution dealt with its hybrid combination of soft and rigid bodies. A natural extension is the combination of CONJAC with REDMAX joints, which was explored partially with CONJAC's muscle simulation. Similarly, EOL cloth could be introduced to the mix with its rigid body interactions and possibility of further cloth-deformable body contacts. EOL cloth itself, while specifically designed for the application of thin sheets, was predated by work in rods and soft deformable bodies. Behind each application is also the integrator, which is always looking to be improved either through direct techniques such as matrix extrapolation or more problem specific approaches yet to be explored. Each individual project has its own potential for future work, and that potential is only expanded upon by finding more unique combinations of physics-based animation components to explore.

Bibliography

[1] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," in *Computer Graphics*, vol. 21, no. 4, 1987, pp. 205–214.

[2] D. Baraff and A. Witkin, "Large steps in cloth simulation," in *Annual Conference Series (Proc. SIGGRAPH)*, 1998, pp. 43–54.

[3] D. House and J. C. Keyser, *Foundations of Physically Based Modeling and Animation*. CRC Press, 2016.

[4] E. Hairer, C. Lubich, and G. Wanner, *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*. Springer Science & Business Media, 2006, vol. 31.

[5] 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*, 2012, pp. 20:1–20:50.

[6] B. Smith, F. D. Goes, and T. Kim, "Analytic eigensystems for isotropic distortion energies," *ACM Trans. Graph.*, vol. 38, no. 1, Feb. 2019.

[7] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., ser. Other Titles in Applied Mathematics. SIAM, 2003. [Online]. Available: http://www-users.cs.umn.edu/~{}saad/IterMethBook_2ndEd.pdf

[8] T. Kim and D. Eberle, "Dynamic deformables: Implementation and production practicalities," in *ACM SIGGRAPH 2020 Courses*, ser. SIGGRAPH '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3388769.3407490

[9] T. A. Davis, *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. USA: Society for Industrial and Applied Mathematics, 2006.

[10] N. J. Weidner, K. Piddington, D. I. W. Levin, and S. Sueda, "Eulerian-on-lagrangian cloth simulation," *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018. [Online]. Available:

https://doi.org/10.1145/3197517.3201281

[11] Y. Wang, N. J. Weidner, M. A. Baxter, Y. Hwang, D. M. Kaufman, and S. Sueda, “Redmax: Efficient & flexible approach for articulated dynamics,” *ACM Trans. Graph.*, vol. 38, no. 4, Jul. 2019. [Online]. Available: https://doi.org/10.1145/3306346.3322952

[12] D. M. Kaufman, S. Sueda, D. L. James, and D. K. Pai, “Staggered projections for frictional contact in multibody systems,” in *ACM SIGGRAPH Asia 2008 Papers*, ser. SIGGRAPH Asia ’08. New York, NY, USA: Association for Computing Machinery, 2008. [Online]. Available: https://doi.org/10.1145/1457515.1409117

[13] N. J. Weidner, T. Kim, and S. Sueda, “Conjac: Large steps in dynamic simulation,” in *Motion, Interaction and Games*, ser. MIG ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3424636.3426901

[14] M. Paz, “Modified dynamic condensation method,” *Journal of Structural Engineering*, vol. 115, no. 1, pp. 234–238, 1989.

[15] M. Gao, N. Mitchell, and E. Sifakis, “Steklov-poincaré skinning,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2014, p. 139–148.

[16] B. Witemeyer, N. J. Weidner, T. A. Davis, T. Kim, and S. Sueda, “Qlb: Collision-aware quasi-newton solver with cholesky and l-bfgs for nonlinear time integration,” in *Motion, Interaction and Games*, ser. MIG ’21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3487983.3488297

[17] S. Sueda, G. L. Jones, D. I. W. Levin, and D. K. Pai, “Large-scale dynamic simulation of highly constrained strands,” *ACM Trans. Graph.*, vol. 30, no. 4, pp. 39:1–39:10, Jul. 2011.

[18] Y. Fan, J. Litven, D. I. Levin, and D. K. Pai, “Eulerian-on-Lagrangian simulation,” *ACM Trans. Graph.*, vol. 32, no. 3, pp. 22:1–22:9, Jul. 2013.

[19] D. Li, S. Sueda, D. R. Neog, and D. K. Pai, “Thin skin elastodynamics,” *ACM Trans. Graph.*, vol. 32, no. 4, pp. 49:1–49:10, Jul. 2013.

[20] R. Narain, A. Samii, and J. F. O’Brien, “Adaptive anisotropic remeshing for cloth simula-

tion," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 152:1–152:10, Nov. 2012.

[21] E. Boxerman and U. Ascher, "Decomposing cloth," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2004, pp. 153–161.

[22] 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 Trans. Graph.*, vol. 33, no. 4, pp. 105:1–105:9, Jul. 2014.

[23] R. Tamstorf, T. Jones, and S. F. McCormick, "Smoothed aggregation multigrid for cloth simulation," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 245:1–245:13, Oct. 2015.

[24] F. Cordier and N. Magnenat-Thalmann, "Real-time animation of dressed virtual humans," in *Computer Graphics Forum*, vol. 21, no. 3, 2002, pp. 327–335.

[25] F. Cordier and N. Magnenat-Thalmann, "A data-driven approach for real-time clothes simulation," in *Computer Graphics Forum*, vol. 24, no. 2, 2005, pp. 173–183.

[26] T.-Y. Kim, N. Chentanez, and M. Müller-Fischer, "Long range attachments - a method to simulate inextensible clothing in computer games," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2012, pp. 305–310.

[27] W. Xu, N. Umentani, Q. Chao, J. Mao, X. Jin, and X. Tong, "Sensitivity-optimized rigging for example-based real-time clothing synthesis," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 107:1–107:11, Jul. 2014.

[28] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behavior," in *Graphics Interface*, 1996, pp. 147–154.

[29] M. Bergou, M. Wardetzky, D. Harmon, D. Zorin, and E. Grinspun, "A quadratic bending model for inextensible surfaces," in *Proc. Eurographics Symp. Geom. Process.*, 2006, pp. 227–230.

[30] B. Thomaszewski, M. Wacker, and W. Straßer, "A consistent bending model for cloth simulation with corotational subdivision finite elements," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2006, pp. 107–116.

[31] M. Wardetzky, M. Bergou, D. Harmon, D. Zorin, and E. Grinspun, "Discrete quadratic

curvature energies," *Comput. Aided Geom. Des.*, vol. 24, no. 8-9, pp. 499–518, Nov. 2007.

[32] B. Thomaszewski, S. Pabst, and W. Strasser, "Continuum-based strain limiting," *Computer Graphics Forum*, vol. 28, no. 2, pp. 569–576, 2009.

[33] P. Volino, N. Magnenat-Thalmann, and F. Faure, "A simple approach to nonlinear tensile stiffness for accurate cloth simulation," *ACM Trans. Graph.*, vol. 28, no. 4, pp. 105:1–105:16, Sep. 2009.

[34] R. Tamstorf and E. Grinspun, "Discrete bending forces and their Jacobians," *Graph. Models*, vol. 75, no. 6, pp. 362–370, Nov. 2013.

[35] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popović, and S. M. Seitz, "Estimating cloth simulation parameters from video," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2003, pp. 37–51.

[36] H. Wang, J. F. O'Brien, and R. Ramamoorthi, "Data-driven elastic models for cloth: Modeling and measurement," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 71:1–71:12, Jul. 2011.

[37] E. Miguel, D. Bradley, B. Thomaszewski, B. Bickel, W. Matusik, M. A. Otaduy, and S. Marschner, "Data-driven estimation of cloth simulation models," in *Computer Graphics Forum*, vol. 31, no. 2pt2, 2012, pp. 519–528.

[38] E. Miguel, R. Tamstorf, D. Bradley, S. C. Schvartzman, B. Thomaszewski, B. Bickel, W. Matusik, S. Marschner, and M. A. Otaduy, "Modeling and estimation of internal friction in cloth," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 212:1–212:10, Nov. 2013.

[39] S. Hadap, E. Bangerter, P. Volino, and N. Magnenat-Thalmann, "Animating wrinkles on clothes," in *Proc. Conference on Visualization*, 1999, pp. 175–182.

[40] R. Bridson, S. Marino, and R. Fedkiw, "Simulation of clothing with folds and wrinkles," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2003, pp. 28–36.

[41] S. Pabst, S. Krzywinski, A. Schenk, and B. Thomaszewski, "Seams and bending in cloth simulation." *VRIPHYS*, vol. 382, no. 1, pp. 24–41, 2008.

[42] D. Rohmer, T. Popa, M.-P. Cani, S. Hahmann, and A. Sheffer, "Animation wrinkling: Augmenting coarse cloth simulations with realistic-looking wrinkles," *ACM Trans. Graph.*,

vol. 29, no. 6, pp. 157:1–157:8, Dec. 2010.

[43] M. Müller and N. Chentanez, “Wrinkle meshes,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2010, pp. 85–92.

[44] H. Wang, F. Hecht, R. Ramamoorthi, and J. F. O’Brien, “Example-based wrinkle synthesis for clothing animation,” *ACM Trans. Graph.*, vol. 29, no. 4, pp. 107:1–107:8, Jul. 2010.

[45] L. Kavan, D. Gerszewski, A. W. Bargteil, and P.-P. Sloan, “Physics-inspired upsampling for cloth simulation in games,” *ACM Trans. Graph.*, vol. 30, no. 4, pp. 93:1–93:10, Jul. 2011.

[46] Z. Chen, R. Feng, and H. Wang, “Modeling friction and air effects between cloth and deformable bodies,” *ACM Trans. Graph.*, vol. 32, no. 4, pp. 88:1–88:8, Jul. 2013.

[47] O. Rémillard and P. G. Kry, “Embedded thin shells for wrinkle simulation,” *ACM Trans. Graph.*, vol. 32, no. 4, pp. 50:1–50:8, Jul. 2013.

[48] R. Gillette, C. Peters, N. Vining, E. Edwards, and A. Sheffer, “Real-time dynamic wrinkling of coarse animated cloth,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2015, pp. 17–26.

[49] N. Umetani, D. M. Kaufman, T. Igarashi, and E. Grinspun, “Sensitive couture for interactive garment modeling and editing,” *ACM Trans. Graph.*, vol. 30, no. 4, pp. 90:1–90:12, Jul. 2011.

[50] D. Kim, W. Koh, R. Narain, K. Fatahalian, A. Treuille, and J. F. O’Brien, “Near-exhaustive precomputation of secondary cloth effects,” *ACM Trans. Graph.*, vol. 32, no. 4, pp. 87:1–87:8, Jul. 2013.

[51] G. Cirio, J. Lopez-Moreno, D. Miraut, and M. A. Otaduy, “Yarn-level simulation of woven cloth,” *ACM Trans. Graph.*, vol. 33, no. 6, pp. 207:1–207:11, Nov. 2014.

[52] G. Cirio, J. Lopez-Moreno, and M. A. Otaduy, “Efficient simulation of knitted cloth using persistent contacts,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2015, pp. 55–61.

[53] J. M. Kaldor, D. L. James, and S. Marschner, “Simulating knitted cloth at the yarn level,” *ACM Trans. Graph.*, vol. 27, no. 3, pp. 65:1–65:9, Aug. 2008.

[54] J. M. Kaldor, D. L. James, and S. Marschner, "Efficient yarn-based cloth with adaptive contact linearization," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 105:1–105:10, Jul. 2010.

[55] J. Villard and H. Borouchaki, "Adaptive meshing for cloth animation," *Engineering with Computers*, vol. 20, no. 4, pp. 333–341, 2005.

[56] R. Narain, T. Pfaff, and J. F. O'Brien, "Folding and crumpling adaptive sheets," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 51:1–51:8, Jul. 2013.

[57] J. Bender and C. Deul, "Adaptive cloth simulation using corotational finite elements," *Computers & Graphics*, vol. 37, no. 7, pp. 820 – 829, 2013.

[58] T. Pfaff, R. Narain, J. M. de Joya, and J. F. O'Brien, "Adaptive tearing and cracking of thin sheets," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 110:1–110:9, Jul. 2014.

[59] W. Koh, R. Narain, and J. F. O'Brien, "View-dependent adaptive cloth simulation," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2014, pp. 159–166.

[60] C. Schreck, D. Rohmer, S. Hahmann, M.-P. Cani, S. Jin, C. C. L. Wang, and J.-F. Bloch, "Nonsmooth developable geometry for interactively animating paper crumpling," *ACM Trans. Graph.*, vol. 35, no. 1, pp. 10:1–10:18, Dec. 2015.

[61] S. Patkar, N. Jin, and R. Fedkiw, "A new sharp-crease bending element for folding and wrinkling surfaces and volumes," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2015, pp. 7–15.

[62] P. Volino, M. Courchesne, and N. Magnenat Thalmann, "Versatile and efficient techniques for simulating cloth and other deformable objects," in *Proc. SIGGRAPH 95, Annual Conference Series*, 1995, pp. 137–144.

[63] P. Volino and N. Magnenat-Thalmann, "Implementing fast cloth simulation with collision response," in *Computer Graphics International*, 2000, pp. 257–266.

[64] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garments," in *Computer Animation and Simulation*. Springer, 1997, pp. 177–189.

[65] R. Bridson, R. Fedkiw, and J. Anderson, "Robust treatment of collisions, contact and friction for cloth animation," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 594–603, Jul. 2002.

[66] D. Baraff, A. Witkin, and M. Kass, “Untangling cloth,” *ACM Trans. Graph.*, vol. 22, no. 3, pp. 862–870, Jul. 2003.

[67] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun, “Efficient simulation of inextensible cloth,” *ACM Trans. Graph.*, vol. 26, no. 3, pp. 49:1–49:7, Jul. 2007.

[68] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun, “Robust treatment of simultaneous collisions,” *ACM Trans. Graph.*, vol. 27, no. 3, pp. 23:1–23:4, Aug. 2008.

[69] E. Sifakis, S. Marino, and J. Teran, “Globally coupled collision handling using volume preserving impulses,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2008, pp. 147–153.

[70] D. Harmon, E. Vouga, B. Smith, R. Tamstorf, and E. Grinspun, “Asynchronous contact mechanics,” *ACM Trans. Graph.*, vol. 28, no. 3, pp. 87:1–87:12, Jul. 2009.

[71] S. Ainsley, E. Vouga, E. Grinspun, and R. Tamstorf, “Speculative parallel asynchronous contact mechanics,” *ACM Trans. Graph.*, vol. 31, no. 6, pp. 151:1–151:8, Nov. 2012.

[72] M. Müller, N. Chentanez, T.-Y. Kim, and M. Macklin, “Air meshes for robust collision handling,” *ACM Trans. Graph.*, vol. 34, no. 4, pp. 133:1–133:9, Jul. 2015.

[73] T. Belytschko, W. K. Liu, B. Moran, and K. Elkhodary, *Nonlinear Finite Elements for Continua and Structures*. John Wiley & Sons, 2013.

[74] J. Sarrate, A. Huerta, and J. Donea, “Arbitrary Lagrangian–Eulerian formulation for fluid–rigid body interaction,” *Comput. Methods. Appl. Mech. Eng.*, vol. 190, no. 24, pp. 3171–3188, 2001.

[75] P. Sachdeva, S. Sueda, S. Bradley, M. Fain, and D. K. Pai, “Biomechanical simulation and control of hands and tendinous systems,” *ACM Trans. Graph.*, vol. 34, no. 4, pp. 42:1–42:10, Jul. 2015.

[76] O. Etzmuß, M. Keckeisen, and W. Straßer, “A fast finite element solution for cloth modelling,” in *Proc. Pac. Conf. Comput. Graph. Appl.*, 2003, pp. 244–251.

[77] C. Lanczos, *The variational principles of mechanics*, 4th ed. Dover, 1986.

[78] R. Malgat, B. Gilles, D. I. W. Levin, M. Nesme, and F. Faure, “Multifarious hierarchies of

mechanical models for artist assigned levels-of-detail," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2015, pp. 27–36.

[79] S. Bouaziz, S. Martin, T. Liu, L. Kavan, and M. Pauly, "Projective dynamics: Fusing constraint projections for fast simulation," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 154:1–154:11, Jul. 2014.

[80] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *J. Vis. Commun. Image Represent.*, vol. 18, no. 2, pp. 109–118, Apr. 2007.

[81] R. E. Roberson, "A dynamical formalism for an arbitrary number of interconnected rigid bodies with reference to the problem of satellite attitude control," *Proc. 3rd Congr. of Int. Fed. Automatic Control*, vol. 1, pp. 46D1–46D8, 1966.

[82] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *INT J ROBOT RES*, vol. 2, no. 1, pp. 13–30, 1983.

[83] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Annual Conference Series (Proc. SIGGRAPH)*, 1996, pp. 137–146.

[84] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Comput. Methods in Appl. Mech. Eng.*, vol. 1, pp. 1–16, Jun 1972.

[85] M. B. Cline and D. K. Pai, "Post-stabilization for rigid body simulation with contact and constraints," in *Proc. IEEE International Conference on Robotics and Automation*, vol. 3, 2003, pp. 3744–3751.

[86] E. Drumwright and D. A. Shell, "Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation," in *Algorithmic Foundations of Robotics IX*. Springer, 2010, pp. 249–266.

[87] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer Science & Business Media, 2012.

[88] V. Acary and B. Brogliato, *Numerical Methods for Nonsmooth Dynamical Systems: Applications in Mechanics and Electronics*. Springer Science & Business Media, 2008.

[89] J. Li, G. Daviet, R. Narain, F. Bertails-Descoubes, M. Overby, G. E. Brown, and

L. Boissieux, “An implicit frictional contact solver for adaptive cloth simulation,” *ACM Trans. Graph.*, vol. 37, no. 4, pp. 52:1–52:15, Jul. 2018.

[90] D. E. Stewart, “Rigid-body dynamics with friction and impact,” *SIAM Rev.*, vol. 42, no. 1, pp. 3–39, Mar. 2000.

[91] H. V. Shin, C. F. Porst, E. Vouga, J. Ochsendorf, and F. Durand, “Reconciling elastic and equilibrium methods for static analysis,” *ACM Trans. Graph.*, vol. 35, no. 2, pp. 13:1–13:16, Feb. 2016.

[92] M. Anitescu and G. D. Hart, “A fixed-point iteration approach for multibody dynamics with contact and small friction,” *MATH. PROG.*, vol. 101, no. 1, pp. 3–32, 2004.

[93] B. Smith, F. D. Goes, and T. Kim, “Stable neo-hookean flesh simulation,” *ACM Trans. Graph.*, vol. 37, no. 2, pp. 12:1–12:15, Mar. 2018.

[94] G. Irving, J. Teran, and R. Fedkiw, “Invertible finite elements for robust simulation of large deformation,” in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2004, pp. 131–140.

[95] T. Kim, F. De Goes, and H. Iben, “Anisotropic elasticity for inversion-safety and element rehabilitation,” *ACM Trans. Graph.*, vol. 38, no. 4, Jul. 2019.

[96] D. I. W. Levin, J. Litven, G. L. Jones, S. Sueda, and D. K. Pai, “Eulerian solid simulation with contact,” *ACM Trans. Graph.*, vol. 30, no. 4, pp. 36:1–36:10, Jul. 2011.

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

[98] 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, no. 3, 1996, pp. 57–66.

[99] A. Pentland and J. Williams, “Good vibrations: Modal dynamics for graphics and animation,” vol. 23, no. 3. New York, NY, USA: ACM, Jul. 1989, p. 207–214.

[100] M. G. Choi and H.-S. Ko, “Modal warping: Real-time simulation of large rotational deformation and manipulation,” *IEEE TVCG*, vol. 11, no. 1, p. 91–101, Jan. 2005.

[101] J. Barbič and D. L. James, "Real-time subspace integration for St. Venant-Kirchhoff deformable models," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 982–990, Jul. 2005.

[102] S. S. An, T. Kim, and D. L. James, "Optimizing cubature for efficient integration of subspace deformations," *ACM Trans. Graph.*, vol. 27, no. 5, Dec. 2008.

[103] Z. Pan, H. Bao, and J. Huang, "Subspace dynamic simulation using rotation-strain coordinates," *ACM Trans. Graph.*, vol. 34, no. 6, Oct. 2015.

[104] Y. Teng, M. Meyer, T. DeRose, and T. Kim, "Subspace condensation: Full space adaptivity for subspace deformations," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 76:1–76:9, Jul. 2015.

[105] E. L. Wilson, "The static condensation algorithm," *International Journal for Numerical Methods in Engineering*, vol. 8, no. 1, pp. 198–203, 1974.

[106] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson, "Physically based deformable models in computer graphics," *Computer Graphics Forum*, vol. 25, no. 4, pp. 809–836, 2006.

[107] R. J. Guyan, "Reduction of stiffness and mass matrices," *AIAA journal*, vol. 3, no. 2, pp. 380–380, 1965.

[108] B. Irons, "Structural eigenvalue problems-elimination of unwanted variables," *AIAA journal*, vol. 3, no. 5, pp. 961–962, 1965.

[109] N. Mitchell, M. Doescher, and E. Sifakis, "A macroblock optimization for grid-based nonlinear elasticity," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.* Goslar, Germany: Eurographics Association, 2016, pp. 11–19.

[110] 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 Trans. Graph.*, vol. 33, no. 4, Jul. 2014.

[111] J. Kim and N. S. Pollard, "Fast simulation of skeleton-driven deformable body characters," *ACM Trans. Graph.*, vol. 30, no. 5, pp. 121:1–121:19, Oct. 2011.

[112] Z. Xian, X. Tong, and T. Liu, "A scalable galerkin multigrid method for real-time simulation of deformable objects," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 162:1–162:13, Nov. 2019.

[113] M. Li, M. Gao, T. Langlois, C. Jiang, and D. M. Kaufman, "Decomposed optimization time integrator for large-step elastodynamics," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 70:1–70:10, Jul. 2019.

[114] R. Narain, M. Overby, and G. E. Brown, "ADMM $\supseteq$ projective dynamics: Fast simulation of general constitutive models," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2016, pp. 21–28.

[115] J. Zhang, Y. Peng, W. Ouyang, and B. Deng, "Accelerating ADMM for efficient simulation and optimization," *ACM Trans. Graph.*, vol. 38, no. 6, pp. 163:1–162:21, Nov. 2019.

[116] A. A. Shabana, *Dynamics of Multibody Systems*. Cambridge University press, 2013.

[117] M. Müller, J. Stam, D. James, and N. Thürey, "Real time physics: class notes," in *ACM SIGGRAPH 2008 classes*. ACM, 2008, p. 88.

[118] J. E. Lloyd, I. Stavness, and S. Fels, "Artisynth: A fast interactive biomechanical modeling toolkit combining multibody and finite element simulation," in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, 2012, pp. 355–394.

[119] H. Xu and J. Barbič, "Example-based damping design," *ACM Trans. Graph.*, vol. 36, no. 4, Jul. 2017.

[120] R. Weinstein, J. Teran, and R. Fedkiw, "Dynamic simulation of articulated rigid bodies with contact and collision," *IEEE TVCG*, vol. 12, no. 3, p. 365–374, May 2006.

[121] R. M. Sánchez-Banderas and M. A. Otaduy, "Strain rate dissipation for elastic deformations," in *Computer Graphics Forum*, vol. 37, no. 8, 2018, pp. 161–170.

[122] Y.-C. Fung, *Biomechanics: mechanical properties of living tissues*. Springer Science & Business Media, 2013.

[123] 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 Trans. Graph.*, vol. 30, no. 4, pp. 37:1–37:12, Jul. 2011.

[124] D. P. O'Leary, "The block conjugate gradient algorithm and related methods," 1980.

[125] M. Bergou, S. Mathur, M. Wardetzky, and E. Grinspun, "Tracks: Toward directable thin

shells," *ACM Trans. Graph.*, vol. 26, no. 3, p. 50–59, Jul. 2007.

[126] M. Tournier, M. Nesme, B. Gilles, and F. Faure, "Stable constrained dynamics," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 132:1–132:10, Jul. 2015.

[127] S. Andrews, M. Teichmann, and P. G. Kry, "Geometric stiffness for real-time constrained multibody dynamics," *Computer Graphics Forum (Proc. Eurographics)*, vol. 36, no. 2, p. 235–246, May 2017.

[128] E. Fehlberg, "Low-order classical runge-kutta formulas with stepsize control and their application to some heat transfer problems," Tech. Rep. NASA-TR-R-315, 1969.

[129] L. F. Shampine and M. W. Reichelt, "The matlab ode suite," *SIAM Journal on Scientific Computing*, vol. 18, no. 1, pp. 1–22, 1997.

[130] E. English and R. Bridson, "Animating developable surfaces using nonconforming elements," *ACM Trans. Graph.*, vol. 27, no. 3, Aug. 2008.

[131] M. Geilinger, D. Hahn, J. Zehnder, M. Bächer, B. Thomaszewski, and S. Coros, "Add: Analytically differentiable dynamics for multi-body systems with frictional contact," *ACM Trans. Graph.*, vol. 39, no. 6, Nov. 2020.

[132] F. Löschner, A. Longva, S. Jeske, T. Kugelstadt, and J. Bender, "Higher-order time integration for deformable solids," in *Proc. ACM SIGGRAPH / Eurographics Symp. Comput. Anim.*, 2020.

[133] V. Modi, L. Fulton, A. Jacobson, S. Sueda, and D. I. W. Levin, "EMU: Efficient muscle simulation in deformation space," *Computer Graphics Forum*, vol. 40, no. 1, pp. 234–248, 2021.

[134] G. E. Brown and R. Narain, "WRAPD: Weighted rotation-aware ADMM for parameterization and deformation," *ACM Trans. Graph.*, vol. 40, no. 4, Jul. 2021.

[135] R. E. Bank and D. J. Rose, "Global approximate newton methods," *Numerische Mathematik*, vol. 37, no. 2, pp. 279–295, 1981.

[136] P. Deuflhard, *Newton methods for nonlinear problems: affine invariance and adaptive algorithms*. Springer Science & Business Media, 2011, vol. 35.

[137] L. Lan, R. Luo, M. Fratarcangeli, W. Xu, H. Wang, X. Guo, J. Yao, and Y. Yang, "Medial elastics: Efficient and collision-ready deformation via medial axis transform," *ACM Trans. Graph.*, vol. 39, no. 3, Apr. 2020.

[138] J. Chen, M. Budninskiy, H. Owhadi, H. Bao, J. Huang, and M. Desbrun, "Material-adapted refinable basis functions for elasticity simulation," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019.

[139] T. Liu, S. Bouaziz, and L. Kavan, "Quasi-newton methods for real-time simulation of hyperelastic materials," *ACM Trans. Graph.*, vol. 36, no. 4, May 2017.

[140] Y. Peng, B. Deng, J. Zhang, F. Geng, W. Qin, and L. Liu, "Anderson acceleration for geometry optimization and physics simulation," *ACM Trans. Graph.*, vol. 37, no. 4, Jul. 2018.

[141] J. Bender, M. Müller, and M. Macklin, "A survey on position based dynamics," in *Proceedings of the Eurographics: Tutorials*, 2017.

[142] P. Herholz, T. A. Davis, and M. Alexa, "Localized solutions of sparse linear systems for geometry processing," *ACM Trans. Graph.*, vol. 36, no. 6, Nov. 2017.

[143] P. Herholz and M. Alexa, "Factor once: Reusing cholesky factorizations on sub-meshes," *ACM Trans. Graph.*, vol. 37, no. 6, Dec. 2018.

[144] P. Herholz and O. Sorkine-Hornung, "Sparse cholesky updates for interactive mesh parameterization," *ACM Trans. Graph.*, vol. 39, no. 6, Nov. 2020.

[145] D. L. James and D. K. Pai, "Artdefo: Accurate real time deformable objects," in *Proceedings of SIGGRAPH*, 1999, p. 65–72.

[146] F. Hecht, Y. J. Lee, J. R. Shewchuk, and J. F. O'Brien, "Updated sparse cholesky factors for corotational elastodynamics," *ACM Trans. Graph.*, vol. 31, no. 5, Sep. 2012.

[147] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 308–323, 1979.

[148] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," *ACM Transactions on Mathematical Software (TOMS)*, vol. 16, no. 1,

pp. 1–17, 1990.

[149] J. J. Dongarra, , S. Hammarling, and J. C. Richard J. Hanson, “Corrigenda:“an extended set of fortran basic linear algebra subprograms”,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 14, no. 4, p. 399, 1988.

[150] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, “Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate,” *ACM Trans. Math. Softw.*, vol. 35, no. 3, Oct. 2008.

[151] J. W. Liu, E. Ng, and B. W. Peyton, “On finding supernodes for sparse matrix computations.” [Online]. Available: https://www.osti.gov/biblio/6756314

[152] P. R. Amestoy, T. A. Davis, and I. S. Duff, “An approximate minimum degree ordering algorithm,” *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[153] P. R. Amestoy, T. A. Davis, and I. S. Duff, “Algorithm 837: Amd, an approximate minimum degree ordering algorithm,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 30, no. 3, pp. 381–388, 2004.

[154] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM J. Sci. Comput.*, vol. 20, no. 1, p. 359–392, Dec. 1998.

[155] S. P. Smith, “Differentiation of the cholesky algorithm,” *Journal of Computational and Graphical Statistics*, vol. 4, no. 2, pp. 134–147, 1995.

[156] R. Luo, W. Xu, T. Shao, H. Xu, and Y. Yang, “Accelerated complex-step finite difference for expedient deformable simulation,” *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019.

[157] T. Kim, F. De Goes, and H. Iben, “Anisotropic elasticity for inversion-safety and element rehabilitation,” *ACM Trans. Graph.*, vol. 38, no. 4, Jul. 2019.

[158] H. Nishikawa, “On large start-up error of bdf2,” *J. Comput. Phys.*, vol. 392, pp. 456 – 461, 2019.

[159] R. Alexander, “Diagonally implicit runge–kutta methods for stiff ode’s,” *SIAM J. Numer. Anal.*, vol. 14, no. 6, pp. 1006–1021, 1977.

APPENDIX A

EULERIAN-ON-LAGRANGIAN CLOTH SIMULATION APPENDIX

A.1 Derivation of World Velocity

Let $\boldsymbol{x}_a$, $\boldsymbol{x}_b$, and $\boldsymbol{x}_c$ be the Lagrangian positions of the three vertices of a triangle, and let $\boldsymbol{X}_a$, $\boldsymbol{X}_b$, and $\boldsymbol{X}_c$ be their corresponding Eulerian positions. Let $\boldsymbol{X}$ be any material point within this triangle. The world position of this point can be expressed as

$$\begin{aligned}\boldsymbol{x}(\boldsymbol{X}) = &\alpha(\boldsymbol{X}_a, \boldsymbol{X}_b, \boldsymbol{X}_c, \boldsymbol{X})\boldsymbol{x}_a + \\ &\beta(\boldsymbol{X}_a, \boldsymbol{X}_b, \boldsymbol{X}_c, \boldsymbol{X})\boldsymbol{x}_b + \\ &\gamma(\boldsymbol{X}_a, \boldsymbol{X}_b, \boldsymbol{X}_c, \boldsymbol{X})\boldsymbol{x}_c.\end{aligned} \tag{A.1}$$

Here we have made it explicit that the barycentric coordinates, (α, β, γ), are all functions of the query material point, $\boldsymbol{X}$, as well as the Eulerian coordinates of the triangle vertices, $\boldsymbol{X}_a$, $\boldsymbol{X}_b$, and $\boldsymbol{X}_c$. Using the standard expression for converting between barycentric and Cartesian coordinates, we have

$$\begin{pmatrix} \beta \\ \gamma \end{pmatrix} = T^{-1}(\boldsymbol{X} - \boldsymbol{X}_a), \quad T = \begin{pmatrix} \boldsymbol{X}_b - \boldsymbol{X}_a & \boldsymbol{X}_c - \boldsymbol{X}_a \end{pmatrix} \in \mathbb{R}^{2\times 2}. \tag{A.2}$$

Since $\alpha = 1 - \beta - \gamma$, we have

$$\begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = DT^{-1}(\boldsymbol{X} - \boldsymbol{X}_a) + d, \quad D = \begin{pmatrix} -1 & -1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad d = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \tag{A.3}$$

We combine Eq. A.1 and Eq. A.3 to get

$$\begin{aligned}\boldsymbol{x}(\boldsymbol{X}) &= \begin{pmatrix}\boldsymbol{x}_a & \boldsymbol{x}_b & \boldsymbol{x}_c\end{pmatrix}\begin{pmatrix}\alpha \\ \beta \\ \gamma\end{pmatrix} \\ &= \begin{pmatrix}\boldsymbol{x}_a & \boldsymbol{x}_b & \boldsymbol{x}_c\end{pmatrix}\left(DT^{-1}(\boldsymbol{X} - \boldsymbol{X}_a) + d\right).\end{aligned} \tag{A.4}$$

Taking the time derivative, we get

$$\dot{\boldsymbol{x}} = \underbrace{\begin{pmatrix}\dot{\boldsymbol{x}}_a & \dot{\boldsymbol{x}}_b & \dot{\boldsymbol{x}}_c\end{pmatrix}\begin{pmatrix}\alpha \\ \beta \\ \gamma\end{pmatrix}}_{\dot{\boldsymbol{x}}^L} + \underbrace{\begin{pmatrix}\boldsymbol{x}_a & \boldsymbol{x}_b & \boldsymbol{x}_c\end{pmatrix} D\frac{d}{dt}\left\{T^{-1}(\boldsymbol{X} - \boldsymbol{X}_a)\right\}}_{\dot{\boldsymbol{x}}^E}. \tag{A.5}$$

The first term is the Lagrangian contribution, and the second term is the Eulerian contribution. We now focus only on the Eulerian term. First, we note that $T = D_X$, the edge matrix of the Eulerian DOFs from Eq. 2.3. Similarly, the multiplication of the Lagrangian DOFs by D in the first portion of $\dot{\boldsymbol{x}}^E$ gives D_x, the edge matrix of the Lagrangian DOFs. Taking the time derivative, we have

$$\begin{aligned}
\dot{\boldsymbol{x}}^E &= D_x\left(\frac{dD_X^{-1}}{dt}(\boldsymbol{X}-\boldsymbol{X}_a) - D_X^{-1}\dot{\boldsymbol{X}}_a\right)\\
&= D_x\left(-D_X^{-1}\left(\frac{dD_X}{dt}\right)D_X^{-1}(\boldsymbol{X}-\boldsymbol{X}_a) - D_X^{-1}\dot{\boldsymbol{X}}_a\right)\\
&= -D_xD_X^{-1}\left(\left(\frac{dD_X}{dt}\right)\begin{pmatrix}\beta\\ \gamma\end{pmatrix} + \dot{\boldsymbol{X}}_a\right)\\
&= -D_xD_X^{-1}\left(\left(\sum_{i=a}^{c}\frac{\partial D_X}{\partial \boldsymbol{X}_i}\otimes\dot{\boldsymbol{X}}_i\right)\begin{pmatrix}\beta\\ \gamma\end{pmatrix} + \dot{\boldsymbol{X}}_a\right)\\
&= -F\left(\left(\begin{pmatrix}-\dot{\boldsymbol{X}}_a & -\dot{\boldsymbol{X}}_a\end{pmatrix} + \begin{pmatrix}\dot{\boldsymbol{X}}_b & 0\end{pmatrix} + \begin{pmatrix}0 & \dot{\boldsymbol{X}}_c\end{pmatrix}\right)\begin{pmatrix}\beta\\ \gamma\end{pmatrix} + \dot{\boldsymbol{X}}_a\right)\\
&= -F\left(-\beta\dot{\boldsymbol{X}}_a - \gamma\dot{\boldsymbol{X}}_a + \beta\dot{\boldsymbol{X}}_b + \gamma\dot{\boldsymbol{X}}_c + \dot{\boldsymbol{X}}_a\right)\\
&= -F\left(\alpha\dot{\boldsymbol{X}}_a + \beta\dot{\boldsymbol{X}}_b + \gamma\dot{\boldsymbol{X}}_c\right),
\end{aligned} \tag{A.6}$$

giving us the final expression for the world velocity:

$$\dot{\boldsymbol{x}} = (\alpha\dot{\boldsymbol{x}}_a + \beta\dot{\boldsymbol{x}}_b + \gamma\dot{\boldsymbol{x}}_c) - F\left(\alpha\dot{\boldsymbol{X}}_a + \beta\dot{\boldsymbol{X}}_b + \gamma\dot{\boldsymbol{X}}_c\right). \tag{A.7}$$

APPENDIX B

CONDENSATION JACOBIAN WITH ADAPTIVITY APPENDIX

B.1 Miscellaneous Pseudocode

B.1.1 Rotation Derivative

Given $\mathbf{F}$ and $\dot{\mathbf{F}}$, we compute $\dot{\mathbf{R}}$ with the following function:

```
function [A] = Rdot(F, Fdot)
  Rgradient = DRDF(F); % Smith et al. 2019
  fdot = reshape(Fdot, [], 1);
  A = reshape(Rgradient * fdot, 3, 3);
end
```

B.1.2 Metric

Let `Sdot_1` through `Sdot_m` be the $\dot{\mathbf{S}}$ matrices of the m tetrahedra of the j^{th} representative node. Then the liveliness metric is computed as:

```
metric_j = mean([ ...
  reshape(abs(Sdot_1), [], 1)
  ...
  reshape(abs(Sdot_m), [], 1)
]);
```

B.1.3 Adjusted Jacobian

Let `iq` and `id` be the indices of the quasistatic and dynamic nodes, respectively, and `nd=length(id)`. Eqs. 4.6 and 4.21 are computed as:

```
% Jacobian without adaptivity
```

```
Kqq = K(iq,iq);

Kqd = K(iq,id);

J(iq,:) = -Kqq \ Kqd; % Eq. 6

J(id,:) = eye(nd); % Eq. 7


% Jacobian with adaptivity

KA = K;

KA(:,id) = 0; % zero out dynamic rows

KA(id,:) = 0; % zero out dynamic columns

KA = KA - sparse(id,id,ones(nd,1),n,n);

KqdA = K(:,id);

KqdA(id,:) = speye(nd);

JA = -KA \ KqdA; % Eq. 21
```

APPENDIX C

LDOT: BOOSTING DEFORMATION PERFORMANCE WITH CHOLESKY EXTRAPOLATION APPENDIX

C.1 Column Block Cholesky Time Derivative

Given $\mathbf{A}$, $\dot{\mathbf{A}}$, and a block pattern we compute $\mathbf{L}$ and $\dot{\mathbf{L}}$ together with the following function:

```
function [L,dL] = dchol_col_block(A,dA,bsize)
  L = tril(A);
  dL = trilblock(dA, bsize);
  bj0 = 0; % starting index for block j
  for j = 1 : length(bsize)
    % (a) block row operation
    bj = bj0 + (1:bsize(j));
    bi = bj0 + (1:sum(bsize(j:end))); % All rows in block j
    bk0 = 0; % Starting index for block k
    for k = 1 : j - 1
      bk = bk0 + (1:bsize(k));
      % Construct update matrix C for block k
      C = L(bi,bk) * L(bj,bk)';
      L(bi,bj) = L(bi,bj) - C;
      % Construct update matrix dC for block k
      dC = dL(bi,bk) * L(bj,bk)';
      dL(bi,bj) = dL(bi,bj) - dC;
      dC = L(bi,bk) * dL(bj,bk)';
      dL(bi,bj) = dL(bi,bj) - dC;
      % Go to next block
      bk0 = bk0 + bsize(k);
    end
    % (b) define pivot
    % Call dchol and return the block L and dL
    %[L(bj,bj),dL(bj,bj)] = cdchol(L(bj,bj),dL(bj,bj));
    % OR
    % Call dchol and only return the L block
    % Calculate the dL block using the dcholphi operation on the L block
    [L(bj,bj),~] = dchol_col(L(bj,bj),dL(bj,bj));
    dL(bj,bj) = dcholphi(L(bj,bj),dL(bj,bj));
    % (c) compute subdiagional block
    bi = bj(end) + (1:sum(bsize(j+1:end))); % All off diagonal rows in bloc
    L(bi,bj) = L(bi,bj) / L(bj,bj)';
    dC = L(bi,bj) * dL(bj,bj)';
    dL(bi,bj) = dL(bi,bj) - dC;
    dL(bi,bj) = dL(bi,bj) / L(bj,bj)';
    % Go to next block
    bj0 = bj0 + bsize(j);
    end
end

%% Block lower triangular part of A
function L = trilblock(A,bsize)
  L = zeros(size(A));
  bk0 = 0; % starting index for block k
  for k = 1 : length(bsize)
    % Block diagonal indices
    bk = bk0 + (1:bsize(k));
    % Fill block row below block diagonal
    L(bk(1):end,bk) = A(bk(1):end,bk);
    % Go to next block
    bk0 = bk0 + bsize(k);
  end
end


%% dcholphi
function dL = dcholphi(L,dA)
  phi = phifun((L \ dA) / L');
  dL = L * phi;
end


%% phifun
function phi = phifun(A)
  phi = tril(A,-1);
  phi = phi + diag(0.5 * diag(A));
end
```

C.2 Integrators

In this section, we describe the various integrators we use to test the performance of our approach. Let $\mathbf{x}$ and $\mathbf{v}$ be the nodal positions and velocities of the volumetric solid, and $\mathbf{M}$ be the constant diagonal mass matrix. Furthermore, let $\mathbf{f}(\mathbf{x}, \mathbf{v})$ be the total forces and derivatives $\mathbf{D} = d\mathbf{f}/d\mathbf{v}$ and $\mathbf{K} = d\mathbf{f}/d\mathbf{x}$ be the damping and stiffness matrices, respectively. We use a trailing superscript with parenthesis to denote the time step. Thus, the goal of each integrator is to compute the nodal positions $\mathbf{x}^{(k+1)}$ given $\mathbf{x}^{(k)}$. After computing $\mathbf{x}^{(k+1)}$, we compute the nodal velocities $\mathbf{v}^{(k+1)}$ using the discretization scheme of the particular integrator.

C.2.1 BDF1

With BDF1 (1st-order Backward Differentiation Formula) [4], we solve a nonlinear system to advance the state from step k to $k+1$:

$$\mathbf{M}\mathbf{v}^{(k+1)} = \mathbf{M}\mathbf{v}^{(k)} + h\mathbf{f}^{(k+1)} \tag{C.1a}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + h\mathbf{v}^{(k+1)}, \tag{C.1b}$$

where h is the time step. After some substitutions and rearranging, the Newton search direction for the nonlinear system becomes $\Delta\mathbf{x}_i = -\mathbf{A}_i^{-1}\mathbf{b}_i$, with:

$$\begin{aligned} \mathbf{b}_i &= \mathbf{M}\left(\mathbf{x}_i^{(k+1)} - \mathbf{x}^{(k)} - h\mathbf{v}^{(k)}\right) - h^2\mathbf{f}_i^{(k+1)} \\ \mathbf{A}_i &= \mathbf{M} - h\mathbf{D}_i^{(k+1)} - h^2\mathbf{K}_i^{(k+1)}, \end{aligned} \tag{C.2}$$

where i indicates the current iteration of the nonlinear solve.

C.2.2 Baraff

A popular integrator in graphics is a variant of BDF1 where only one iteration is taken — *i.e.,* we solve one linear system per time step. In this work, we call this the Baraff integrator [2].

C.2.3 Quasistatic

The quasistatic time stepper solves for the nonlinear system

$$\mathbf{f}^{(k+1)} = 0 \tag{C.3}$$

at each time step, with time-varying boundary conditions or forces. The Newton search direction for the nonlinear system is $\Delta\mathbf{x}_i = -\mathbf{A}_i^{-1}\mathbf{b}_i$, with:

$$\begin{aligned} \mathbf{b}_i &= -\mathbf{f}_i^{(k+1)} \\ \mathbf{A}_i &= -\mathbf{K}_i^{(k+1)}. \end{aligned} \tag{C.4}$$

C.2.4 BDF2

With BDF2 (2nd-order Backward Differentiation Formula) [4], we solve a nonlinear system to compute the state at $k + 1$ using the states at $k - 1$ and k:

$$\mathbf{M}\mathbf{v}^{(k+1)} = \frac{4}{3}\mathbf{M}\mathbf{v}^{(k)} - \frac{1}{3}\mathbf{M}\mathbf{v}^{(k-1)} + \frac{2}{3}h\mathbf{f}^{(k+1)} \tag{C.5a}$$

$$\mathbf{x}^{(k+1)} = \frac{4}{3}\mathbf{x}^{(k)} - \frac{1}{3}\mathbf{x}^{(k-1)} + \frac{2}{3}h\mathbf{v}^{(k+1)}. \tag{C.5b}$$

After some substitutions and rearranging, the Newton search direction for the nonlinear system becomes $\Delta\mathbf{x}_i = -\mathbf{A}_i^{-1}\mathbf{b}_i$, with:

$$\begin{aligned}\mathbf{b}_i &= \mathbf{M}\left(\mathbf{x}_i^{(k+1)} - \frac{4}{3}\mathbf{x}^{(k)} + \frac{1}{3}\mathbf{x}^{(k-1)} - \frac{8}{9}h\mathbf{v}^{(k)} + \frac{2}{9}h\mathbf{v}^{(k-1)}\right) - \frac{4}{9}h^2\mathbf{f}_i^{(k+1)} \\ \mathbf{A}_i &= \mathbf{M} - \frac{2}{3}h\mathbf{D}_i^{(k+1)} - \frac{4}{9}h^2\mathbf{K}_i^{(k+1)}.\end{aligned} \tag{C.6}$$

On the very first time step, we cannot use BDF2 since it requires solutions at two previous time steps. Instead, we use SDIRK2, described below, to start BDF2 [158].

C.2.5 SDIRK2

With SDIRK2 (2nd-order Singly-Diagonal Implicit Runge-Kutta) [159], we solve two nonlinear systems to advance the state from step k to $k + 1$:

$$\mathbf{M}\mathbf{v}^{(k+\alpha)} = \mathbf{M}\mathbf{v}^{(k)} + \alpha h\mathbf{f}^{(k+\alpha)} \tag{C.7a}$$

$$\mathbf{x}^{(k+\alpha)} = \mathbf{x}^{(k)} + \alpha h\mathbf{v}^{(k+\alpha)}, \tag{C.7b}$$

and

$$\mathbf{M}\mathbf{v}^{(k+1)} = \mathbf{M}\mathbf{v}^{(k)} + (1-\alpha)h\mathbf{f}^{(k+\alpha)} + \alpha h\mathbf{f}^{(k+1)} \tag{C.8a}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + (1-\alpha)h\mathbf{v}^{(k+\alpha)} + \alpha h\mathbf{v}^{(k+1)}, \tag{C.8b}$$

where $\alpha = (2 - \sqrt{2})/2$. These two nonlinear systems are solved sequentially: first for $\mathbf{x}^{(k+\alpha)}$ and then for $\mathbf{x}^{(k+1)}$. (The velocities can be computed from the positions using Eqs. C.7b and C.8b.) After some substitutions and rearranging, the Newton search direction for the first nonlinear system becomes $\Delta\mathbf{x}_i = -\mathbf{A}_i^{-1}\mathbf{b}_i$, with:

$$\begin{aligned}
\mathbf{b}_i &= \mathbf{M}\left(\mathbf{x}_i^{(k+\alpha)} - \mathbf{x}^{(k)} - \alpha h \mathbf{v}^{(k)}\right) - (\alpha h)^2 \mathbf{f}_i^{(k+\alpha)} \\
\mathbf{A}_i &= \mathbf{M} - \alpha h \mathbf{D}_i^{(k+\alpha)} - (\alpha h)^2 \mathbf{K}_i^{(k+\alpha)}.
\end{aligned} \tag{C.9}$$

Similarly, for the second nonlinear system, we have:

$$\begin{aligned}
\mathbf{b}_i &= \mathbf{M}\left(\mathbf{x}_i^{(k+1)} - \mathbf{x}^{(k)} - \beta h \mathbf{v}^{(k)} - \gamma h \mathbf{v}^{(k+\alpha)}\right) - (\alpha h)^2 \mathbf{f}_i^{(k+1)} \\
\mathbf{A}_i &= \mathbf{M} - \alpha h \mathbf{D}_i^{(k+1)} - (\alpha h)^2 \mathbf{K}_i^{(k+1)},
\end{aligned} \tag{C.10}$$

where $\beta = 2\alpha - 1$ and $\gamma = 2 - 2\alpha$.