Provably Robust and Accurate Methods for Rigid and Deformable Simulation with Contact

by

Zachary Ferguson

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May, 2023

Professor Daniele Panozzo

Dedication

To my parents and fiancée, with love.

Acknowledgments

This work would not have been possible without a ton of help and support from all of my collaborators, friends, and family. I would like to spend this space to say thank you for everything.

My largest thank you goes out to the entire Geometric Computing Lab, both past and present. I especially owe much of my success to my advisor, Prof. Daniele Panozzo. Thank you for being not only the best advisor I could ask for but also an amazing friend. Thank you Prof. Teseo Schneider for being an awesome collaborator, mentor, teacher, and best friend. Thank you Prof. Denis Zorin for constantly lending your expertise to strengthen my work and to debug the strange numerical behavior of my code. A massive thank you to Dr. Francisca Gil-Ureta who taught me so much about being a productive researcher and a better programmer. You deserve a lot more credit than has been given for helping to lay the foundation of this research. Thank you Dr. Bolun Wang for being a friend and amazing co-author. Without your help, much of this research would not have been as robust as it is.

I extend a special thank you to Prof. Yotam Gingold, Dr. Songrun Liu, and Prof. Alec Jacobson who introduced me to the field of computer graphics as an undergraduate at George Mason University. You introduced me to the wonderful world of computer graphics research, and for that, I am forever grateful.

I had the opportunity to have three amazing internships while a Ph.D. student, two at Adobe and one at Carbon. To Dr. Qingnan (James) Zhou, I say thank you for helping me develop as an early Ph.D. student. To the entire computational geometry team at Carbon and especially Dr.

Weixiong Zheng, Dr. Ruiqi Chen, and Dr. Hardik Kabaria, I thank you for your continued interest and motivation in my research. I learned a ton while interning with you and am thankful for the experience. To Dr. Danny M. Kaufmann, we worked closely together for almost the entirety of my Ph.D. and without your constant depth of knowledge and guidance I would not have been nearly as successful in my studies.

I would also like to acknowledge Prof. Georg Stadler and Prof. Lerrel Pinto who so generously took time out of their schedules to participate in my dissertation committee and provided valuable suggestions.

Thank you to all of my collaborators and co-authors: Prof. Kenny Erleben, Dr. Faezeh Moshfeghifar, Dr. Torkan Gholamalizadeh, Prof. Steve Abramowitch, Prof. Sheldon Andrews, Prof. Marco Attene, Pranav Jain, Dr. Davi Colli Tozoni, Zizhou Huang, Arvi Gjoka, David Belgrad, Dr. Zhongshi Jiang, Liam Martin, Xuan Tang, Duo Zhang, Dr. Minchen Li, Prof. Chenfanfu Jiang, Dr. Timothy Langlois, and Dr. Jérémie Dumas.

Thank you to the NYU IT High-Performance Computing team, especially Shenglong Wong, for their help and the computational resources necessary to conduct my research.

Thank you to New York University for awarding me with the MacCracken Fellowship in 2017, Jacob T. Schwartz Ph.D. fellowship in 2021, and Dean's Dissertation Fellowship in 2022; and Adobe Inc. for awarding me with the Adobe Research Fellowship in 2022, providing me with financial support during my Ph.D.

Last but most importantly, thank you to my friends and family. Especially my mother and father, Linda and Glenn, for your constant love, support, and encouragement; my grandmother (gram), Lanaya Abernathy, for sparking my interest in computers from an early age; and, most of all, my fiancée, Amanda Chen, for your constant love and support over the last six years.

Abstract

Contacts are essential to virtually every aspect of life and play a vital role in many physical phenomena. Because of this, the study of contact mechanics has a deep wealth of knowledge. Surprisingly, however, simulating contact is a challenge with many parameters to carefully adjust. Incorrect parameters can result in numerical explosions, intersections, and other failures. Our research seeks to address these problems by developing robust methods that can handle arbitrary scenarios with guaranteed success.

In this thesis, we introduce the Incremental Potential Contact (IPC) method. IPC is the first simulation algorithm for deformable and rigid bodies that is unconditionally robust, requires minimal parameter tuning, and provides a direct way of controlling the trade-off between running time and accuracy. We further back up these claims by providing a large-scale benchmark of continuous collision detection (CCD) algorithms (a core component of the IPC method) based on their efficiency and correctness. As part of this study, we introduce the first efficient CCD algorithm that is provably conservative. For extended accuracy and efficiency, we show how nonlinear geometry and function spaces can be used within the IPC framework. Finally, we introduce the first physically-based adaptive meshing strategy which produces more accurate discretizations depending on elastic, contact, and frictional forces.

This work and our open-source implementations have quickly garnered attention from the computer graphics, mechanical engineering, and biomechanical engineering communities for their robustness and ability to seamlessly handle scenarios that have long been a challenge. This

marks a large step towards democratizing simulation tools for design, robotics, biomechanical, and visual effects applications, among others.

Contents

List of Figures

List of Tables

List of Abbreviations

ABD Affine Body Dynamics . . . 216

AL augmented Lagrangian . . . 130

AM adaptive meshing . . . 6

AMD approximate minimum degree . . . 42

`AVX2` Advanced Vector Extensions 2 . . . 69

BC boundary condition . . . 225

BDF2 second-order backward differentiation formula . . . 189

BSC Bernstein sign classification . . . 68

BP broad-phase . . . 216

BVH bounding volume hierarchy . . . 109

CCD continuous collision detection . . . 4

C-IPC Codimensional Incremental Potential Contact . . . 154

CO collision objects . . . 225

CSR compressed sparse row . . . 43

CSV comma-separated values . . . 258

DOF degrees of freedom . . . 108

EE edge-edge . . . 39

FCR fixed corotational . . . 43

FE finite element . . . 5

FEM finite element method . . . 5

PN projected newton 26

PP point-point 39

PSD positive semi-definite 26

PT point-triangle 39

QP quadratic programming 19

RBD rigid body dynamics 110

RP root parity 77

RRP rational implementation of Root Parity 77

SDF signed distance fields 18

SMC smooth contacts 142

SPD symmetric positive definite 26

SQP sequential quadratic programming 17

TCCD TightCCD 77

TI Tight Inclusion 216

TOI time of impact 69

UIRF univariate interval root-finder 76

UR uniformly-refined 195

VFX visual effects 60

List of Appendices

1 Introduction

Every day we interact and manipulate the world through touch. This simple act has become so ingrained in our lives that we often take it for granted. However, the physical laws governing this interaction are surprisingly complex. Underlying these interactions are numerous nonlinear contact and frictional forces. Simulating these forces in a virtual environment has been the topic of much research over the past 50 years [Ball 1981; Harmon et al. 2009; Kane et al. 1999; Kikuchi and Oden 1988; Moreau 1973; Ortiz and Stainier 1999]. An ideal simulation algorithm for this should be efficient, accurate, and robust. However, up to this point, no prior work has been able to fully satisfy these criteria seamlessly. Existing methods often make trade-offs between accuracy and efficiency, or require extensive parameter tuning to achieve robustness. In this thesis, we introduce the Incremental Potential Contact (IPC) method, which is the first simulation algorithm for deformable and rigid bodies that is unconditionally robust, requires minimal parameter tuning, and provides a direct way of controlling the trade-off between running time and accuracy.

Anecdotally, this line of work stems from an attempt to use existing simulation software to test the viability of computer-aided designs by performing large sweeps over design parameters (Figure 1.1). However, we quickly ran into difficulties with this approach as each change in design would often lead to intersecting and infeasible results. This is because prior methods require per-scene tuning of simulation parameters (see for example Figure 1.2). So while this area of research was largely thought to be solved [Harmon et al. 2009], we show here that existing methods often require user expertise and care to get viable results. In contrast, the methods proposed here aim

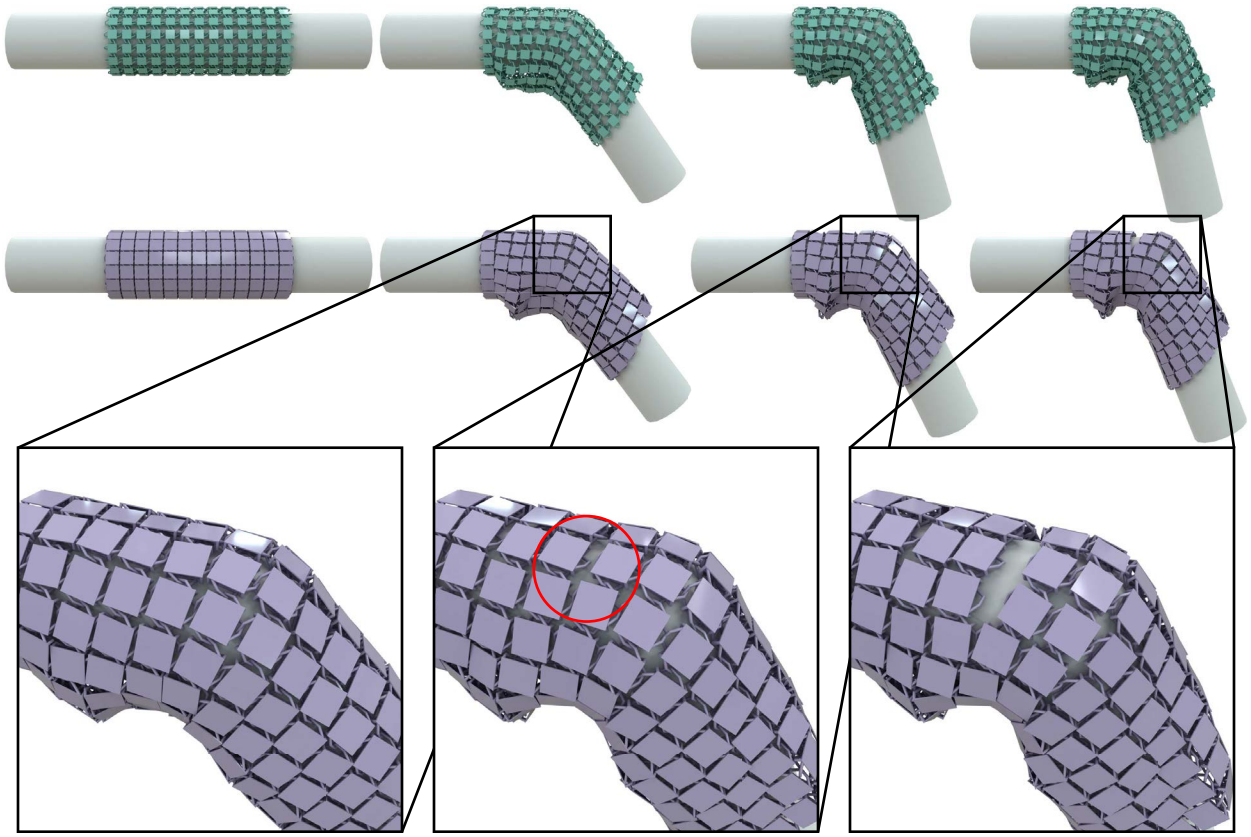
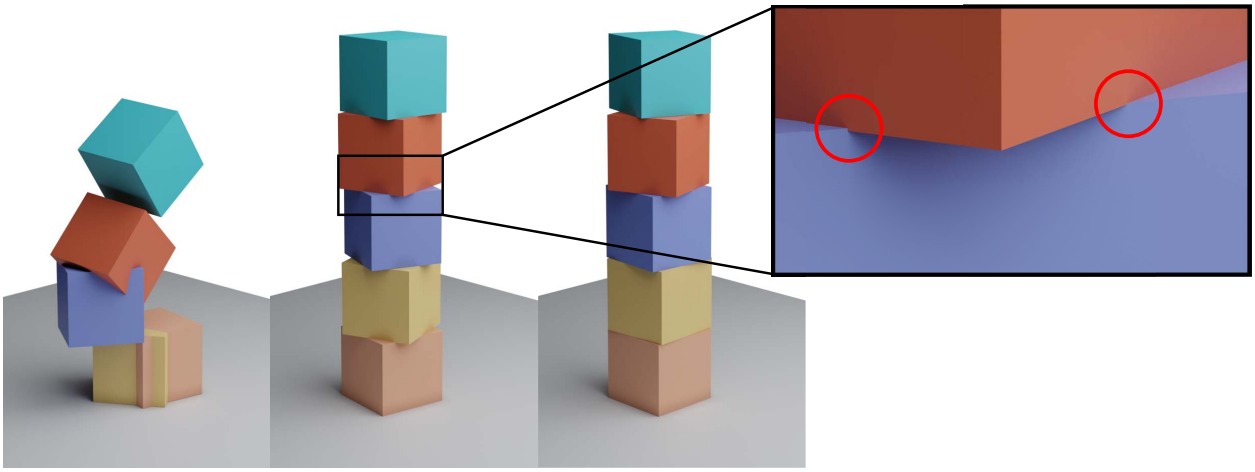


Figure 1.1: Shape optimization failure. Here we want to study the effects of link shape on the bendability of a "chainmail" sleeve. We simulate the sleeve with two different link shapes as a collection of rigid bodies in the Project Chrono physics engine [Tasora et al. 2016]. The simulation succeeds in producing valid results for the first design (top row), but when we increase the size of the links outer plate (middle row) we get intersections and eventually topological discontinuities (circled in red) as the links pass through each other. To resolve these issues a user would have to manually adjust the simulation parameters. This is impractical at large scale or for long-running simulations. In contrast, our work is able to guarantee a feasible result independent of the geometry or loading conditions.

to be as parameter-free as possible, and, more importantly, the feasibility of results is wholly independent of parameter choice.

We start by introducing the IPC algorithm in Chapter 2 in the context of nonlinear elastodynamics. IPC maintains an intersection- and inversion-free trajectory regardless of material parameters, time step sizes, impact velocities, severity of deformation, or boundary conditions enforced. Constructed with a custom nonlinear solver, IPC enables efficient resolution of time-stepping problems with separate, user-exposed accuracy tolerances that allow independent speci-



$h = 0.01$ s $h = 0.001$ s $h = 0.0001$ s

Figure 1.2: Parameter tuning. Incorrect parameters can lead to intersections, jittering, or even numerical explosions. This necessitates per-scene parameter tuning. For example, here we try to stack five cubes in the Project Chrono physics engine [Tasora et al. 2016] using a large timestep size (h), but this results in large intersections. When reducing the timestep size, the stack looks better but if we look closely (right) we still have intersections (circled in red). We are only able to avoid intersections by using an even smaller timestep size. Using a small timestep size works well on this example, but for more complicated examples it can be challenging, if not impossible, to find a single set of parameters that works over all scenes. This makes prior methods difficult to use at a large scale (e.g., reinforcement learning or shape optimization) because each scene variation requires different parameters.

fication of the physical accuracy of the dynamics and the geometric accuracy of surface-to-surface conformation. This enables users to decouple, as needed per application, desired accuracies for a simulation's dynamics and geometry. The resulting time stepper solves contact problems that are intersection-free (and thus robust), inversion-free, efficient (at speeds comparable to or faster than available methods that lack both convergence and feasibility), and accurate (solved to user-specified accuracies). To our knowledge, this is the first implicit time-stepping method, across both the engineering and graphics literature that can consistently enforce these guarantees as we vary simulation parameters.

Additionally, we provide an extensive comparison of available simulation methods, research libraries, and commercial codes in Appendix B. We confirm that available engineering and computer graphics methods, while each succeeding admirably in custom-tuned regimes, often fail with instabilities, egregious constraint violations, and/or inaccurate and implausible solutions,

as we vary input materials, contact numbers, and timestep size. We also exercise IPC across a wide range of existing and new benchmark tests and demonstrate its accurate solution over a broad sweep of reasonable time-step sizes and beyond (up to $h = 2\,\text{s}$) across challenging large-deformation, large-contact stress-test scenarios with meshes composed of up to 2.3M tetrahedra and processing up to 498K contacts per time step. For applications requiring high accuracy, we demonstrate tight convergence on all measures. While, for applications requiring lower accuracies, e.g. animation, we confirm IPC can ensure feasibility and plausibility even when specified tolerances are lowered for efficiency.

As part of Chapter 2, we discuss the role continuous collision detection (CCD) plays in determining a safe step size. Initially investigated by Provot [1997], this topic has been addressed by numerous authors. Some propose to ignore floating-point error entirely in favor of efficiency [Provot 1997; Vouga et al. 2011] while others carefully account for these errors [Redon et al. 2002a; Snyder 1992]. Brochu and Bridson [2009] and Tang et al. [2014], propose custom tailored algorithms designed to "exactly" produce the correct Boolean answer to if objects collide. However, we show in Chapter 3 that despite the widespread use of CCD algorithms, existing algorithms are either: (1) correct but impractically slow, (2) efficient but incorrect, introducing false negatives which will lead to interpenetration, or (3) correct but over-conservative, reporting a large number of false positives which might lead to inaccuracies when integrated into a simulator. We do so by the introduction of a large-scale benchmark with an accompanying dataset of symbolically calculated ground truth to over 60 million queries. We use the benchmark to evaluate the accuracy, correctness, and efficiency of state-of-the-art CCD algorithms, both with and without a minimum separation.

To close the gap between efficient and reliable methods, we introduce a new method that is not only efficient but provably conservative. Additionally, this algorithm allows for an explicit trade-off between runtime efficiency and the number of false positives reported.

We then extend the IPC algorithm to handle rigid bodies in Chapter 4. This marks the first

implicit time-stepping algorithm for rigid body dynamics, with contact and friction, that guarantees intersection-free configurations at every time step. Importantly, our algorithm explicitly models the curved trajectories traced by rigid bodies in both collision detection and response. For collision detection, we propose a conservative narrow-phase collision detection algorithm for curved trajectories, which reduces the problem to a sequence of linear CCD queries with minimum separation (Chapter 3). We demonstrate the effectiveness of this method by introducing a benchmark for rigid body simulation and show that our approach, while less efficient than alternatives, can robustly handle a wide array of complex scenes, which cannot be simulated with competing methods, without requiring per-scene parameter tuning.

With the finite element method (FEM) (utilized in Chapter 2) there are two common approaches for increasing accuracy: p-refinment (i.e., utilizing high-order geometry and/or interpolatory bases) and h-refinement (i.e., refining the mesh spatially). While these approaches are directly applicable and well suited to elastic deformation [Hu et al. 2018; Manteaux et al. 2015; Mitchell and McClain 2014; Schneider et al. 2019a, 2018], their application to contacting elastodynamics and in particular IPC remains an open problem. To address this we introduce two methods.

First, we investigate the use of higher-order geometry (i.e., curved meshes) and functional interpolations (i.e., nonlinear displacements) in Chapter 5. High-order bases provide major advantages over linear ones in terms of *efficiency*, as they provide (for the same physical model) higher accuracy for the same running time, and *reliability*, as they are less affected by locking artifacts and mesh quality. However, detecting and handling contact between curved surfaces is computationally expensive negating any possible performance benefits. Thus, we show it is possible to decouple the mesh used for elasticity from the surface mesh used for contact and friction. Our approach is based on the observation that each IPC optimization step used to minimize the elasticity, contact, and friction potentials leads to linear trajectories even in the presence of nonlinear meshes or nonlinear finite element (FE) bases. It is thus possible to retain the strong

non-penetration guarantees and large time steps of the original formulation while benefiting from the high-order bases and high-order geometry. We accomplish this by mapping displacements and resulting contact forces between a linear collision proxy and the underlying high-order representation.

Second, in Chapter 6 we explore the topic of adaptive meshing (AM) while preserving invariants such as remaining intersection-free. We propose In-Timestep Remeshing, a fully coupled, adaptive meshing algorithm for contacting elastodynamics where remeshing steps are tightly integrated, implicitly, within the timestep solve. Our algorithm refines and coarsens the domain automatically by measuring physical energy changes within each ongoing timestep solve. This provides consistent, degree-of-freedom-efficient, productive remeshing that, by construction, is physics-aware and so avoids the errors, over-refinements, artifacts, per-example hand-tuning, and instabilities commonly encountered when remeshing with timestepping methods. Our in-timestep computation then ensures that each simulation step's output is both a converged stable solution on the updated mesh and a temporally consistent trajectory with respect to the model and solution of the last timestep. At the same time, the output is guaranteed safe (intersection- and inversion-free) across all operations. We demonstrate applications across a wide range of extreme stress tests with challenging contacts, sharp geometries, extreme compressions, large timesteps, and wide material stiffness ranges – all scenarios well-appreciated to challenge existing remeshing methods.

An important contribution in conjunction with the algorithmic designs discussed in this thesis is the continued development of several open-source libraries. Both within computer graphics and beyond (e.g., biomechanics [Gholamalizadeh et al. 2022; Moshfeghifar et al. 2022] and robotics [Kim et al. 2022]) we see the quick adoption and improvement of our work thanks in large part to these libraries. This is despite there being widely used and long-standing commercial (e.g., Ansys^{TM} [Ansys, Inc. 2023], ABAQUS^{TM} [Smith 2023], COMSOL [COMSOL Inc. 2022], and Houdini [SideFX 2023]) and open-source (e.g., FEBio [Maas et al. 2012], FEniCS [Logg et al.

2011], and SOFA [Faure et al. 2012]) alternatives. In turn, this highlights the impact potential of this work.

2 Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics

2.1 Introduction

Contact is ubiquitous and often unavoidable and yet modeling contacting systems continues to stretch the limits of available computational tools. In part, this is due to the unique hurdles posed by contact problems. Several intricately intertwined physical and geometric factors make contact computations hard, especially in the presence of friction and nonlinear elasticity.

Real-world contact and friction forces are effectively discontinuous, immediately making the time-stepping problems very stiff, especially if the contact constraints are enforced exactly. On the other hand, even small violations of exact contact constraints (which are nonconvex) can lead to impossible-to-untangle geometric configurations, with a direct impact on physical accuracy and stability. In addition, stiff contact forces often lead to extreme deformations, resulting in element inversions for mesh-based discretization. Friction modeling then introduces further challenges with asymmetric force coupling and rapid switching between sliding and sticking modes.

In this work, our goal is to achieve very high robustness (by which we mean the absence

Figure 2.1: Squeeze out. Incremental Potential Contact (IPC) enables high-rate time stepping, here with $h = 0.01$s, of extreme nonlinear elastodynamics with contact that is intersection- and inversion-free at all time steps, irrespective of the degree of compression and contact. Here a plate compresses and then forces a collection of complex soft elastic FE models (181K tetrahedra in total, with a neo-Hookean material) through a thin, codimensional obstacle tube. The models are then compressed entirely together forming a tight mush to fit through the gap and then once through they cleanly separate into a stable pile.

of catastrophic failures or stagnation) for contact modeling even for the most challenging elastodynamic contact problems with friction. Robustness should be obtained independent of user-controllable accuracy in time-stepping, spatial discretization and contact resolution, while maintaining efficiency required to solve large-scale problems. At the same time we wish to also ensure that all accuracies – across the board – are efficiently attainable (of course with additional cost) when required.

With these goals in mind, we reexamine the contact problem formulation, discretization and numerical methods from scratch, building on numerous ideas and observations from prior work.

Our Incremental Potential Contact (IPC) solver is constructed for mesh-based discretizations of nonlinear volumetric elastodynamic problems supporting large nonlinear deformations, implicit time-stepping with contact, friction and boundaries of arbitrary codimension (points, curves, surfaces, and volumes). A key principle we follow is that while the physics and shape can be approximated arbitrarily coarsely, the geometric constraints (absence of intersections of the ap-

proximate geometry and inversions of elements) are maintained exactly at all times. We achieve this for essentially arbitrary target time steps and spatial discretization resolution.

The key element of our approach is the formulation of the contact problem and the customized numerical method to solve it. As a starting point, we use an exact contact constraint formulation, described in terms of an *unsigned* distance function, and rate-based maximal dissipation for friction.

For every time step, we solve the discrete nonlinear contact problem with a given tolerance using a smoothed barrier method, ensuring that the solution remains intersection-free at all intermediate steps. We use a comparably smoothed, arbitrarily close, approximation to static friction, also eliminating the need for an explicit Coulomb constraint, and cast friction forces at every time step in a dissipative potential form, using an alternating lagged formulation. All forces can then be solved by unconstrained minimization.

Our barrier formulation for contact has several important properties: 1) it is an almost everywhere C^2 function of the unsigned distances between mesh elements, C^1 continuous for a measure-zero set of configurations; 2) its support is restricted to a small part of the configuration space close to configurations with contact. The former property makes it possible to use rapidly converging Newton-type unconstrained optimization methods to solve the barrier approximation of the problem, the latter ensures that additional contact forces are applied highly locally *and* that only a small set of terms of the barrier function need to be computed explicitly during optimization. Jointly they enable stable, conforming contact between geometries.

To guarantee a collision-free state at every time step, feasibility is maintained throughout all nonlinear solver iterations: the line search in our customized Newton-based solver is augmented with efficient, filtered CCD accelerated by a conservative CFL-type contact bound on line search step sizes. Friction forces are resolved directly in the same solver via our lagged potential with geometric accuracy improved by alternating updates.

2.1.1 Contributions

In summary, IPC solves nonlinear elastodynamic trajectories that are intersection- and inversion-free, efficient and accurate (solved to user-specified accuracies) while resolving collisions with both nonsmooth and codimensional obstacles. To our knowledge, this is the first implicit time-stepping method, across both the engineering and graphics literature, with these properties.

We demonstrate the efficacy of IPC with stress tests containing large deformations, many contact primitive pairs, large friction, tight collisions as well as sharp and codimensional obstacles. Our technical contributions include

- A contact model based on the unsigned distance function;
- An almost everywhere C^2, C^1-continuous barrier formulation, approximating the contact problem with arbitrary accuracy, with barrier support localized in the configuration space, enabling efficient time-stepping;
- Contact-aware line search that continuously guarantees penetration-free descent steps with CCD evaluations accelerated by a conservative-bound contact-specific CFL-inspired filter;
- A new variational friction model with smoothed static friction, formulated as a lagged dissipative potential, robustly resolving challenging frictional contact behaviors; and
- A new benchmark of simulation test sets with careful evaluation of constraint and time stepping formulations along with an extensive evaluation of existing contact solvers.

2.2 Contact Model

We focus on solving numerical time-integration for nonlinear volumetric elastodynamic models with contact. These models can interact with fixed and moving obstacles which can be of

arbitrary dimension (surfaces, curves and points). The simulation domain is discretized with finite elements. Given n nodal positions, x, finite-element mass matrix, M, and a hyper-elastic deformation energy, $\Psi(x)$, the contact problem extremizes the extended-value action

$$S(x) = \int_0^{\mathrm{T}} \left(\frac{1}{2} \dot{x}^\top M \dot{x} - \Psi(x) + x^\top (f_e + f_d) \right) dt.$$

on an *admissible set of trajectories* $\mathcal{A}$, which we discuss below. Here f_e are external forces and f_d are dissipative frictional forces. We assume, for simplicity, that all object geometry is discretized with n-dimensional piecewise-linear elements, $n = 1, 2, 3$.

Admissible trajectories. We construct a new definition of admissibility based on unsigned distance functions that has a number of advantages. Most importantly, in the context of our work, it naturally allows us to formulate exact contact constraints in terms of constraints on collisions between pairs of primitives (triangles, vertices and edges), and can be defined in exactly the same way for objects of any dimensions (points, curves, surfaces and volumes).

Specifically we define trajectories $x(t)$, with $x \in \mathbb{R}^{3n}$ as *intersection-free*, if for all moments t, $x(t)$ ensures that the distance $d(p, q)$ between any distinct points p and q on the boundaries of objects is positive. In the space of trajectories, the set of intersection-free trajectories forms an open set $\mathcal{A}_I$, as it is defined by strict inequalities. Optimization problems may not have solutions in this set; for this reason, we add the limit trajectories to it, which involve contact. Specifically, we define the *set of admissible trajectories* $\mathcal{A}$ as the *closure* of $\mathcal{A}_I$. In other words, a trajectory is admissible, if it is intersection-free, *or* there is an intersection-free trajectory arbitrarily close.

Note that this closure is not equivalent to replacing the constraint $d(p, q) > 0$ with $d(p, q) \geq 0$; the latter is always satisfied for unsigned distances, so that all trajectories would be admissible. This is not the case for our definition. Consider for example, a point moving towards a plane. If its trajectory touches the plane and then turns back, an arbitrarily small perturbation makes it intersection-free, and the trajectory is in $\mathcal{A}$. However, if the trajectory crosses the plane small

perturbations do not make it intersection-free. This highlights the need for our treatment even in the volumetric setting as the boundaries of our mesh upon which we impose constraint are exactly surfaces whose potential collisions include the point-face case above.

We can describe $\mathcal{A}_I$ directly in terms of constraints on unsigned distances d between surface primitives (vertices, edges, and faces in the simulation surface mesh and domain boundaries). We denote this set of mesh primitives $\mathcal{T}$. Equivalently to the more general definition above, a piecewise-linear trajectory $x(t)$ starting in an intersection-free state x_0 is admissible, if for all times t, the configuration $x(t)$ satisfies positive distance constraints $d_{ij}(x(t)) > 0$ for all $\{i, j\} \in \mathcal{B}$, where $\mathcal{B} \subset \mathcal{T} \times \mathcal{T}$ is the set of all non-adjacent and non-incident surface mesh primitive pairs.

We then observe that the distance between any pair of primitives is bounded from below by the distance for triangle-vertex and edge-edge pairs, *if* there are no intersections. For this reason, it is sufficient to enforce constraints $d_k(x(t)) > 0$ *continuously* in time, for all $k \in C \subset \mathcal{B}$ where C contains all *non-incident point-triangle and all non-adjacent edge-edge pairs* in the surface mesh.

Time discretization. Discretizing in time, we can directly construct discrete energies whose stationary points give an unconstrained time step method's update [Ortiz and Stainier 1999]. Concretely, given nodal positions x^t and velocities v^t, at time step t, we formulate the time step update for new positions x^{t+1} as the minimization of an incremental potential (IP) [Kane et al. 2000], $E(x, x^t, v^t)$, over valid $x \in \mathbb{R}^{3n}$. For example the IP for implicit Euler is then simply

$$E(x, x^t, v^t) = \tfrac{1}{2}(x - \hat{x})^\top M(x - \hat{x}) - h^2 x^\top f_d + h^2 \Psi(x), \tag{2.1}$$

where h is the time step size and $\hat{x} = x^t + hv^t + h^2 M^{-1} f_e$. IPs for implicit Newmark (see Section 2.7) and many other integrators follow similarly by simply treating their update rule as stationarity conditions of a potential with respect to variations of x^{t+1}.

The addition of contact constraints restricts minimization of the IP to admissible trajecto-

ries [Kane et al. 1999; Kaufman and Pai 2012] and so yields for our model the following time step problem:

$$x^{t+1} = \underset{x}{\operatorname{argmin}}\ E(x, x^t, v^t),\ x^\tau \in \mathcal{A}, \tag{2.2}$$

where x^τ, $\tau \in [t, t+1]$, is the linear trajectory between x^t and x^{t+1}.

Our goal is to define a numerical method for approximating the solution of this problem in Equation (2.2). Solving it is challenging due to the complex nonlinearity of the admissibility constraint, especially in the context of large deformations.

In turn, when frictional forces in Equation (2.2) include frictional contact, solving the time step problem becomes all the more challenging as f_d is now governed by the Maximal Dissipation Principle [Moreau 1973] and so must satisfy further challenging, asymmetric and strongly nonlinear consistency conditions [Simó and Hughes 1998]. We present a friction formulation in Section 2.5 that is naturally integrated into this formulation via a lagged dissipative potential.

We further define a set of piecewise-linear surfaces as ϵ-separated, if the distance between two boundary points of the set is at least ϵ, unless these are on the same element of the boundary. An ϵ-separated trajectory is then a trajectory for which surfaces stay ϵ-separated. We denote the set of such trajectories $\mathcal{A}_\epsilon$.

To handle contact constraints, in our algorithm, we use the following overall approach: (a) the IP function E is unmodified on $\mathcal{A}_\epsilon$ – the set of trajectories for which any ϵ-separated trajectory extremizes the action are preserved; (b) we introduce a barrier term that vanishes for trajectories in $\mathcal{A}_\epsilon$ and diverges as the distance between any two boundary points vanishes, converting the problem to an unconstrained optimization problem. This barrier, together with continuous collision detection within minimization steps, ensure that trajectories remain in $\mathcal{A}_I$.

This algorithm then guarantees that the trajectories are modified, compared to the exact solution, in an arbitrarily small, user-controlled (by ϵ) region near object boundaries and, at the same time, always remain admissible.

2.2.1 Trajectory Accuracies

A discrete contacting trajectory is accurate if it satisfies 1) admissibility, 2) discrete momentum balance, 3) positivity, 4) injectivity and 5) complementarity.

In the discrete setting, *momentum balance* requires that the gradient of the incremental potential, $\nabla_x E(x)$, balance against the time-integrated contact forces. Its accuracy, is then exactly measured by the residual error in the optimization of the constrained incremental potential. We simply and directly control accuracy of momentum balance by setting stopping tolerance in our nonlinear optimization; see Section 2.4.5.

In turn *positivity* means that contact forces' signed magnitudes, λ_k, per contact pair $k \in C$, are always non-negative and so push surfaces but do not pull. Our method guarantees exact positivity.

Combined with *admissibility*, *injectivity* requires positive volumes for all tetrahedra in the simulation mesh. This invariant is enforced when a non-inverting energy density function (e.g., neo-Hookean) is modeled.[1]

Finally, the classic definition of *complementarity* in contact mechanics [Kikuchi and Oden 1988] is the requirement that contact forces enforcing admissibility can only be exerted between surfaces if they are touching with no distance between them. We do not allow $d_k(x) = 0$, and so define a comparable measure of discrete ϵ-complementarity requiring

$$\lambda_k \max(0, d_k(x) - \epsilon) = 0, \ \forall k \in C \tag{2.3}$$

to measure how well contact accuracy is achieved. Discrete complementarity is then satisfied whenever distances between all contact pairs, defined as surface pairs with nonzero contact forces, are less than the ϵ and converge to complementarity as we reduce ϵ.

[1] When an invertible deformation model (e.g. fixed corotational) is modeled, injectivity need not be preserved in computation. We primarily focus on non-inverting neo-Hookean but will also demonstrate the weaker invertible case with fixed corotational.

2.3 Related Work

Computational contact modeling is a fundamental and long studied task in mechanics well covered from diverse perspectives in engineering, robotics and computer graphics [Brogliato 1999; Kikuchi and Oden 1988; Stewart 2001; Wriggers 1995]. At its core the contact problem combines enforcement of significant and challenging *geometric* non-intersection constraints with the resolution of a deformable solid's momentum balance. The latter task is well-explored, often independent of contact [Belytschko et al. 2000; Stuart and Humphries 1996]. We focus below on related works in defining contact constraints, implicitly time stepping with contact and friction, and barriers.

2.3.1 Constraints and Constraint Proxies

Contact simulation requires a computable model of admissibility and so a choice of contact constraint representation. For volumetric models, admissibility generally begins with description of a signed distance function. This allows a clean formulation of the continuous model. However, when it comes to computing non-intersection on deformable meshes, choices for representing non-intersection must be made and a diversity of constraint representations exist. Contact constraints for deformable meshes, in both engineering [Belytschko et al. 2000; Wriggers 1995] and graphics [Bridson et al. 2002; Daviet et al. 2011; Harmon et al. 2009, 2008; Otaduy et al. 2009; Verschoor and Jalba 2019] are most commonly defined pairwise between matched surface primitives.

Existing methods most often define a local, signed distance evaluation using a diverse array of nonlinear proxy functions as well as their linearizations. These include linear gap functions, linearized constraints built from geometric normals, as well as a number of oriented volume constraints [Kane et al. 1999; Sifakis et al. 2008]. These nonlinear proxies, such as the tetrahedral volumes formed between surface point-face and edge-edge pairs, are only locally valid. They can introduce artificial ghost contact forces when sheared, false positives when rotated (e.g. for

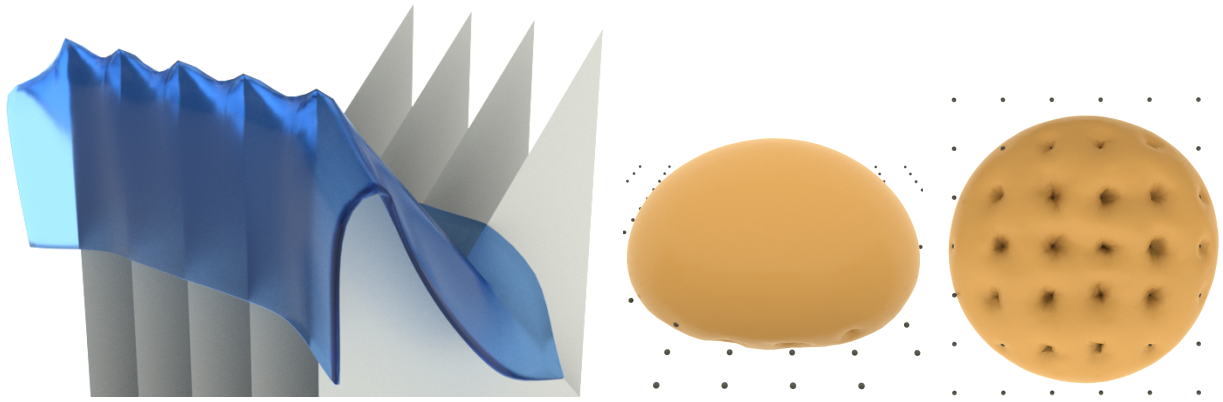


Figure 2.2: Nonsmooth, codimensional collisions. Left: thin volumetric mat falls on codimensional (triangle) obstacles. Right: a soft ball falls on a matrix of point obstacles, front and bottom views.

edge-edge tetrahedra), discontinuities when traversing surface element boundaries, and, in many methods, must still be further linearized and so introduce additional levels of approximation in order to solve a constrained time step.

Alternately gap functions and other related methods approximate signed distance functions for pairs of primitives by locally projecting a linearized distance measure between pairwise surface primitives onto a fixed geometric normal [Otaduy et al. 2009; Wriggers 1995]. As discussed in Erleben [Erleben 2018] these "contact point and normal" based constraint functions can be inconsistent over successive iterations and so are highly sensitive to surface and meshing variations with well known failure modes if care is not taken. Indeed, as we investigate in Section 2.8.3, even with iterative updates of these linear constraints inside sequential quadratic programming (SQP)-type methods, time stepping with gap functions and related representations produces highly varied results whose success or failure is largely dependent on the scene simulated. In turn, all of these challenges are only further exacerbated when simulations encounter the sharp, nonsmooth, and even codimensional collisions imposed by meshed obstacles [Kane et al. 1999]; see e.g. Figure 2.2.

Recent fictitious domain methods [Jiang et al. 2017; Misztal and Bærentzen 2012; Müller et al. 2015; Pagano and Alart 2008; Zhang et al. 2005] offer a promising alternative. In these methods,

motivated by global injectivity conditions [Ball 1981] negative space is separately discretized by a compatible discretization sometimes called an air-mesh [Müller et al. 2015]. Maintaining a non-negative volume on elements of this mesh then guarantees non-inversion. However, as with locally defined proxy volumes, the globally defined mesh introduces increasingly severe errors, e.g., shearing and locking forces, as it distorts with the material mesh. In 2D this can be alleviated by local [Müller et al. 2015] or global [Jiang et al. 2017] remeshing, however this is highly inefficient in 3D, does not provide a continuous constraint representation for optimization, nor, even with remeshing, can it resolve sliding and resting contact where air elements must necessarily be degenerate [Li et al. 2018b].

Alternately, discrete signed distance fields (SDF) representations can be constructed via a number of approximation strategies over spatial meshes [Jones et al. 2006]. However, while state-of-the-art adaptive SDF methods now gain high-resolution accuracy for sampling against a fixed meshes [Koschier et al. 2017], they can not yet be practically updated at rates suitable for deformable time steps, much less at rates suitable for querying deformations at every iterate within a single implicit time step solve [Koschier et al. 2017]. At the same time, discontinuities across element boundaries, while improved in recent works, still preclude smooth optimization methods.

We observe that while approximating signed distance pairwise between surface mesh elements is problematic, unsigned distance is well defined. We then design a new contact model for exact admissibility constraints in terms of unsigned distances between mesh-element pairs. This model of constraint is constructed sufficiently smooth to enable efficient, super-linear Newton-type optimization, maintains exact constraint satisfaction guarantees throughout all steps (time steps and iterations) and requires evaluation of just mesh-surface primitive pairs.

2.3.2 Implicit Time Step Algorithms for Contact

With choice of contact constraint proxy, $g(x) \geq 0$, the solve for the implicit time-step update is then the minimization of the contact-constrained IP [Doyen et al. 2011; Kane et al. 1999; Kaufman

and Pai 2012],

$$\min_x E(x, x^t, v^t) \quad \text{s.t.} \quad g(x) \geq 0. \tag{2.4}$$

The variational problem Equation (2.4), or its approximation is then minimized to compute the configuration at each time step.

This is typically done with off-the-shelf [Nocedal and Wright 2006] or customized constrained optimization techniques. In engineering, commonly used methods include SQP [Kane et al. 1999], augmented Lagrangian and occasionally interior point methods [Belytschko et al. 2000]. All such methods iteratively linearize constraint functions and elasticity. However, both nonlinear constraint functions and their linearizations are generally valid only in local regions of evaluation and so can lead to intersections due to errors at larger time steps, faster speeds and/or larger deformations. For example, Kane et al.'s [1999] volumes are only valid under a strong assumption of the relative position of contact primitive pairs.

In turn linearization of the full constraint set can also introduce additional error, result in infeasible sub-problems, locking and/or constraint drift [Erleben 2018]. This often requires complex and challenging (re-)evaluations of constraints in inter-penetrating states. Even when such obstructions are not present, iterated constraint linearization generally can not guarantee inter-penetration-free state except upon convergence and so often must resort to small time steps and non-physical fail-safes in order to limit damage caused by missed constraint enforcement.

Although SQP- [Kane et al. 1999] and linear complementarity problem (LCP)/quadratic programming (QP)-based contact solvers [Kaufman et al. 2008] support and generally employ a variety of constraint-set culling and active-set update strategies, e.g., incrementally adding newly detected collisions at each iteration [Otaduy et al. 2009; Verschoor and Jalba 2019], they also can become infeasible and generate constraint drift when linearizing and filtering constraints.

Irrespective of how the contact-IP is solved and constraints are enforced, we then remain faced with combinatorial explosion in the number of contact constraints to handle. Determining the

active set, i.e., finding which constraints are necessary for admissibility and so can not be ignored, remains an outstanding computational challenges. At the same time, to take large time steps or handle large deformation, we must resolve strongly nonlinear deformation energies in balance with contact forces. This requires line search. However, for constrained optimization methods, e.g., SQP, efficient line search in the presence of large numbers of active constraints remains an open problem [Bertsekas 2016; Nocedal and Wright 2006]. For this reason, existing methods in graphics currently avoid line search altogether and are, as a consequence, mostly restricted to quadratic energy models per time step [Otaduy et al. 2009; Verschoor and Jalba 2019] and, often, small time step sizes for even moderate material stiffness [Verschoor and Jalba 2019].

2.3.3 Friction

The addition of accurate friction with stiction only increases the computational challenge for time stepping deformation [Wriggers 1995]. Friction forces are tightly coupled to the computation of both deformation and the contact forces that prevent intersection. These side conditions are generally formulated by their own governing variational Maximal Dissipation Principle (MDP) [Goyal et al. 1991; Moreau 1973] and thus introduce additional nonlinear, nonsmooth and asymmetric relationships to dynamics. In transitions between sticking and sliding modes large, nonsmooth jumps in both magnitude and direction are made possible by frictional contact model. Asymmetry, in turn, is a direct consequence of MDP: frictional forces are not uniquely defined by the velocities they oppose, and are also determined by additional consistency conditions and constraints, e.g., Coulomb's law. One critical consequence is that there is no well-defined potential that we can add to an IP to directly produce contact friction via minimization.

To address these challenges, frictional contact is often solved by seeking a joint solution to the optimality conditions of MDP together with the discretized equations of motion (the latter are equivalent to optimality conditions for E). This requires, however, simultaneously solving for primal velocity unknowns together with a large additional number of dual contact and friction

force unknowns. These latter variables then scale in the number of active contacts and their number grows large for even moderately sized simulation meshes.

To solve these systems it is generally standard to apply iterative per-contact, nonlinear Gauss-Seidel-type methods [Alart and Curnier 1991; Bridson et al. 2002; Daviet et al. 2011; Jean and Moreau 1992; Kaufman et al. 2014]. Here elasticity is again often, but not always, linearized per time step, while contact and friction constraints are similarly often approximated per iteration with a range of linear and nonlinear proxies. Alternate iteration strategies [Kaufman et al. 2008; Otaduy et al. 2009] have also been applied. However, as in the frictionless setting, all such splittings remain challenging to solve with guarantees for complex, real-world scenarios. Most recently, the same discrete formulation has been solved with new custom-designed algorithms – both with nonsmooth Newton-type strategies [Bertails-Descoubes et al. 2011; Macklin et al. 2019] and an extension of the Conjugate Residual method [Verschoor and Jalba 2019] with improved accuracy and efficiency.

2.3.4 Barrier Functions

Barrier functions are commonly applied in nonlinear optimization, especially in interior-point methods [Nocedal and Wright 2006]. Here primal-dual interior point methods are generally favored with Lagrange multipliers as additional unknowns for improved convergence. For contact problems, this impractically enlarges system sizes by orders-of-magnitude. Here we focus on a primal solution suited for contact problems. Similarly, the vast majority of the literature focuses on globally supported functions, which are not viable for contact due to the quadratic set (collision primitive pairs) of constraints that must be considered. Recently, a few works have begun exploiting *locally supported* barriers [Harmon et al. 2009; Schüller et al. 2013; Smith and Schaefer 2015]. Harmon et al. [2009] propose a set of layered discrete penalty barriers that grow unbounded as the configuration reaches toward contact. While well-suited for small time-step explicit methods, the incremental construction of the barriers challenge application in implicit time integration

with Newton-type optimization. Most recently methods in geometry processing [Schüller et al. 2013; Smith and Schaefer 2015] propose locally supported barriers in the context of 2D mesh parametrization to prevent element inversion and overlap. Our formulation builds on a similar idea. Here we design smoothed, local barriers custom-constructed for the challenges of resolving contact-response and preventing intersection between 3D mesh-primitives.

2.3.5 Summary

In summary, state-of-the-art methods for contact simulation are often highly effective per example. However, in order to do so they generally require significant hand-tuning per simulation set-up in order to obtain successful simulation output, i.e., stable, nonintersecting, plausible, or predictive output. Currently, many of the tuned parameters, as we discuss in Section 2.8.3, are not physical but rather guided by expected intersection constraint violation errors and stability needs, and so need to be experimentally determined by many simulation test runs. Thus, to date, direct, fully automated simulation has not been available for contact simulation – despite contact's fundamental role in many design, engineering, robotics, learning and animation tasks. Towards a direct, "plug-and-play" contact simulation framework we propose IPC. Across a wide range of mesh resolutions, time step sizes, physical conditions, material parameters and extreme deformations we confirm IPC performs and completes simulations to requested accuracy without algorithm parameter tuning.

2.4 Primal Barrier Contact Mechanics

In this section, we describe how we solve our time step problem (Equation (2.2)) formulated in Section 2.2. We defer consideration of friction to Section 2.5, focusing on handling contact dynamics here. We solve the minimization problem (Equation (2.2)) with primitive-pair admissibility constraints using a carefully designed barrier-augmented incremental potential that can be

Algorithm 2.1 Barrier Aware Projected Newton

procedure BarrierAwareProjectedNewton($x^t, \boldsymbol{\epsilon}$)
  $x \leftarrow x^t$
  $\hat{C} \leftarrow$ComputeConstraintSet$(x, \hat{d})$ ▷ Sections 2.4.6 and 2.6.1
  $E_{\text{prev}} \leftarrow B_t(x, \hat{d}, \hat{C})$
  $x_{\text{prev}} \leftarrow x$
  do
    $H \leftarrow \text{SPDProject}(\nabla^2_x B_t(x, \hat{d}, \hat{C}))$ ▷ Section 2.4.3
    $p \leftarrow -H^{-1}\nabla_x B_t(x, \hat{d}, \hat{C})$
    // CCD line search: ▷ Section 2.4.4
    $\alpha \leftarrow \min(1, \text{StepSizeUpperBound}(x, p, \hat{C}))$
    do
      $x \leftarrow x_{\text{prev}} + \alpha p$
      $\hat{C} \leftarrow$ComputeConstraintSet$(x, \hat{d})$
      $\alpha \leftarrow \alpha/2$
    while $B_t(x, \hat{d}, \hat{C}) > E_{\text{prev}}$
    $E_{\text{prev}} \leftarrow B_t(x, \hat{d}, \hat{C})$
    $x_{\text{prev}} \leftarrow x$
    Update κ, BCs and equality constraints ▷ Appendices A.5 and A.6
  while $\frac{1}{h}\|p\|_\infty > \boldsymbol{\epsilon}_d$
  return x

evaluated efficiently. In turn, to solve this potential we design a custom, contact-aware, Newton-type solver, outlined in Algorithm 2.1, with constraint culling for efficient evaluation of objectives, gradients and Hessians (Section 2.4.3).

2.4.1 Barrier-Augmented Incremental Potential

To enforce distance constraints $d_k(t) > 0$, for all $k \in C$, we construct a continuous barrier energy b (Section 2.4.2), that creates a highly localized repulsion force, affecting motion only when primitives are close to collision, and vanishing if primitives are a small user-specified distance apart. We then augment the time step potential $E(x, x^t, v^t)$ with a sum of these barriers over all

possible pairs in C. The barrier-augmented IP is then

$$B_t(x) = E(x, x^t, v^t) + \kappa \sum_{k \in C} b\big(d_k(x)\big), \tag{2.5}$$

with $\kappa > 0$ an adaptive conditioning parameter automatically controlling the barrier stiffness (see Section 2.4.3 and our Supplemental for details.).

Minimizing Equation (2.5) enables the solution of contact-constrained dynamics with unconstrained optimization. Computing the energy naively, however, would require evaluation of the barrier functions for all $O(|\mathcal{T}|^2)$ pairs. To address similar challenges many simulation methods simply remove constraints corresponding to distant primitives that are hoped to be unnecessary for the current solution. However, this tempting operation is dangerous, as significant errors and instabilities can be introduced when constraint sets are modified and critical collisions can also be missed (see Sections 2.3 and 2.8). Instead, we design smooth barrier functions that allow us to compute the barrier energy exactly and efficiently for all constraints while evaluating distances only for a small subset of pairs of primitives that are close and simultaneously ensuring that the rest smoothly evaluate to zero.

2.4.2 Smoothly Clamped Barriers

We begin by defining a smooth barrier function composed of terms that are *local* for every primitive pair, that is each term is exactly zero if the two primitives are far away, enabling reliable and efficient pruning of pairs in C without change to the solution.

We start by defining a computational distance accuracy target, $\hat{d} > 0$ (corresponding to ϵ in Section 2.2) that specifies the maximum distance at which contact repulsions can act. We then construct a barrier potential that approaches infinity at zero distance, initiates contact forces for pairs closer than the target distance, $\hat{d}$, and applies no repulsion at distances greater than $\hat{d}$.

Considering the smooth log-barrier function commonly applied in optimization [Boyd and

Vandenberghe 2004] gives $\ln(d/\hat{d})$, where d is the unsigned distance evaluation between a primitive pair. However, simply truncating this function produces an unacceptably non-smooth energy which cannot be efficiently optimized and is effectively no better than simply discarding constraints. Some examples of problems this generates in optimization are covered in the supplemental. We thus propose a smoothly clamped barrier to regain superlinear convergence for Newton-type methods

$$b(d,\hat{d}) = \begin{cases} -(d-\hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d}. \end{cases} \tag{2.6}$$

Our barrier function is now C^2 at the clamping threshold, and it is exactly zero for pairs beyond the target accuracy (see Figure 2.3). Now, without harm, at any configuration x, we only need to evaluate barrier terms for the *culled constraint set*

$$\hat{C}(x) = \{k \in C \,:\, d_k(x) \leq \hat{d}\},$$

composed of barriers between close primitives. As we increase accuracy by specifying smaller $\hat{d}$ we then need to evaluate increasingly smaller numbers of contact barriers, albeit with increased cost in nonlinearity.

Next, while the barrier function $b(d,\hat{d})$ itself is now C^2, the distance function it evaluates between primitives will be C^0 for certain unavoidable configurations; i.e., parallel edge-edge collisions – see Figure 2.9. For this reason, we multiply the barrier terms for edge-edge collisions by a mollifier that ensure our distance function is C^1 (and piecewise C^2) for all primitive pair types. Distance evaluation and mollifier are discussed in detail in Section 2.6. Additional important considerations related to numerical stability and roundoff error in distance evaluation are then detailed further in the Supplemental.

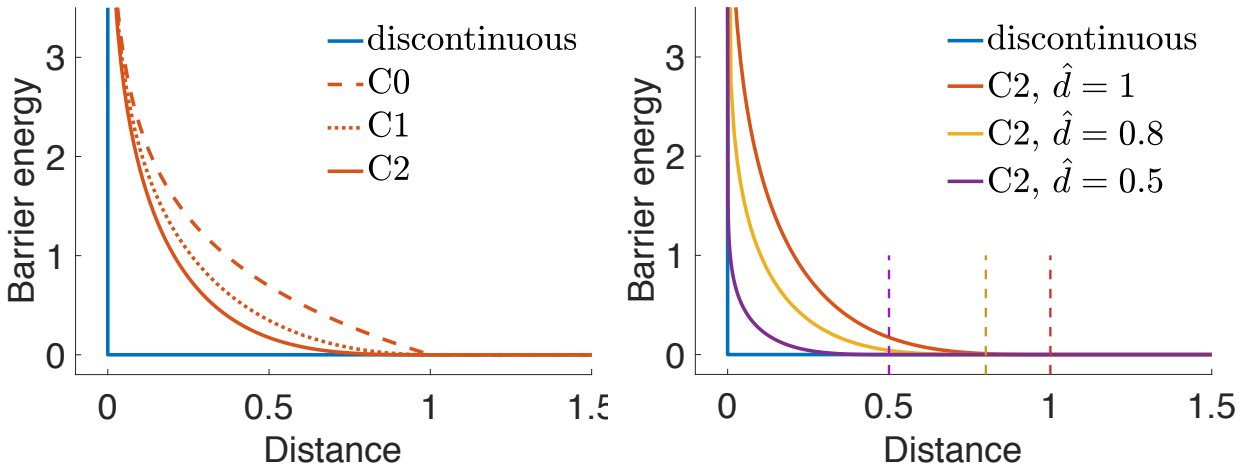


Figure 2.3: Barriers. Left: log barrier function clamped with varying continuity. We can augment the barrier to make clamping arbitrarily smooth (see our Supplemental). We apply our C^2 variant for best tradeoff: smoother clamping improves approximation of the discontinuous function while higher-order continuity introduces more computational work. Right: our C^2 clamped barrier improves approximation to the discontinuous function as we make our geometric accuracy threshold, $\hat{d}$, smaller.

2.4.3 Newton-type Barrier Solver

Projected newton (PN) methods are second-order unconstrained optimization strategies for minimizing nonlinear nonconvex functions where the Hessian may be indefinite. Here we apply and customize PN to the barrier-augmented IP (Equation (2.5)). At each iteration, we project each local energy stencil's Hessian to the cone of symmetric positive semi-definite (PSD) matrices (see `SPDProject` function in Algorithm 2.1) prior to assembly. Specifically, following Teran et al. [2005] we project per-element elasticity Hessians to PSD. We then comparably project the Hessian of each barrier to PSD. Each barrier Hessian has the form

$$\frac{\partial^2 b}{\partial d^2}\nabla_x d(\nabla_x d)^\top + \frac{\partial b}{\partial d}\nabla_x^2 d \tag{2.7}$$

and so can be constructed as a small matrix restricted to the vertices in the stencil of the barrier's primitive pair. The addition of mass matrix terms then ensures that the assembled total IP Hessian is symmetric positive definite (SPD). Originally we also investigated a Gauss-Newton

approximation to the above barrier Hessian, taking only the first, SPD term in the sum. However, we find that resulting search directions are far less efficient than using the full projected barrier Hessian.

Termination. For termination of the solver we check the infinity norm of the Newton search direction scaled by time step (but *unscaled* by line-search step size). Specifically we solve each time step's barrier IP to an accuracy satisfying $\frac{1}{h}\|H^{-1}\nabla B_t(x)\|_\infty < \boldsymbol{\epsilon}_d$. This provides affine invariance and a characteristic measure using the Hessian's natural scaling as metric. Accuracy is then directly defined by $\boldsymbol{\epsilon}_d$ in physical units of velocity (and so is independent of time-step size applied) and consistently measures quadratically approximated distance to local optima across examples with varying scales and conditions.

Barrier stiffness adaptation. We automatically adapt our barrier stiffness to provide repulsive scaling that balances necessary distances against conditioning from the barrier stiffness. Our barrier-augmented potential, B_t, has two key parameters: $\hat{d}$ and κ, that jointly scale the effective stiffness of each contact barrier. The strength of our barriers' contact forces (equivalently Lagrange multipliers) are directly determined during minimization by evaluating distances, d_k, and stiffness, κ. When κ is too small, contact-pair distances must become tiny to exert sufficient repulsion. On the other hand, when κ is too large, contact-pair distances must be below $\hat{d}$ in order to exert non-zero force, but at the same time remain exceedingly close to $\hat{d}$ so as to not exert too large a repulsion. Both cases thus generate unnecessary ill-conditioning and nonsmoothness that challenge efficiency. As we directly control geometric accuracy by setting $\hat{d}$, this frees κ to adaptively condition our Newton-solver to improve convergence. While conceptually one could imagine finding improved scalings of κ by hand, per example, this is unacceptable and inefficient for an automated simulation pipeline. Instead, in our Supplemental, we derive our stiffness update algorithm that automatically adapts barrier stiffness per iterate for improved conditioning.

Relation to homotopy solves. While in IPC we directly set and solve for a desired target accuracy $\hat{d}$, a natural alternative is to solve with a homotopy as is typical in interior point methods. We initially experimented with this approach: solving for larger distances (and so less stiff systems) and then decreasing to the target distance, $\hat{d}$, over successive nonlinear solves. We find, however, that this is unnecessary for elastodynamics where the direct barrier solves we employ are much more efficient. In part, this is because we typically have a good warm start available from the prior time step.

2.4.4 Intersection-aware Line Search

While our barrier energy is infinite for contact, this by itself does not guarantee that constraints $d_k(t) > 0$ are not violated by the solver. Standard line search [Nocedal and Wright 2006], e.g, back-tracking with Wolfe conditions, can find an energy decrease in configurations that have passed through intersection, resulting in a step that takes the configuration out of the admissible set.

Smith and Schaefer's [2015] line-search filter computes the largest step size in 2D per triangle and per boundary point-edge pair that first triggers inversion or overlap, and then take the minimum as a step size upper bound for the current Newton iteration to stay feasible. Taking inspiration from this line-search filter we propose a continuous, intersection-aware line search filter for 3D meshes. In each line search we first apply CCD to conservatively compute a large feasible step size along the descent step. We then apply back-tracking line search from this step size upper bound to obtain energy decrease. CCD then certifies that each step taken is always valid. When we apply barrier-based energy densities (our default) for our elasticity potential, Ψ, i.e., neo-Hookean, we combine the inversion-aware line search filter [Smith and Schaefer 2015] with our intersection-aware filter to obtain descent steps. In combination this guarantees that every step of position update in our solver (and so simulation) maintains an inversion- and intersection-free trajectory.

2.4.5 IPC Solution Accuracy

Revisiting accuracy we confirm *momentum balance* is directly satisfied by IPC after convergence. For example, for implicit Euler we have

$$\nabla_x B_t(x, \hat{d}) = 0 \implies M(\frac{x - \hat{x}}{h^2}) = -\nabla\Psi(x) + \sum_{k \in C} \lambda_k \nabla d_k(x), \tag{2.8}$$

where our contact forces λ_k are given by barrier derivatives

$$\lambda_k = -\frac{\kappa}{h^2} \frac{\partial b}{\partial d_k}. \tag{2.9}$$

Comparable discrete momentum balance follows when we apply alternate time integration methods, e.g. implicit Newmark. *Positivity* is then confirmed directly by Equation (2.9) and observing that our barrier function definition guarantees $\frac{\partial b}{\partial d_k} \leq 0$. In turn our above line-search filters guarantee *admissibility* and, when applicable for barrier-type elasticity energy densities, *injectivity*. Finally, our barrier definition ensures *discrete complimentarity* is always satisfied as contact forces can not be applied at distance more than $\epsilon = \hat{d}$ away.

2.4.6 Constraint Set Update and CCD Acceleration

Every line search, prior to backtracking, performs CCD to guarantee non-intersection, while every evaluation of energies and their derivatives compute distances to update the culled constraint set, $\hat{C}(x)$. To accelerate these computations, we construct a combined spatial hash and distance filtering structure to efficiently reduce the number of primitive-pair distance checks. Then, to further accelerate intersection-free stepping along each Newton iterate's descent direction, p, we derive an efficient conservative bound motivated by CFL conditions [Courant et al. 1967]. As in force evaluations we aim to avoid unnecessary and expensive CCD computation on primitive pairs *not in* $\hat{C}$. We leverage the fact that all contact pairs *not* in $\hat{C}$ are at distances greater

Figure 2.4: Extreme stress test: rod twist for 100s. We simulate the twisting of a bundle of thin volumetric rod models at both ends for 100s. IPC efficiently captures the increasingly conforming contact and expected buckling while maintaining an intersection- and inversion-free simulation throughout. Top: at 5.5s, before buckling. Bottom: at 73.6s, after significant repeated buckling is resolved.

than $\hat{d}$, and use the maximal relative search step in p of each such pair to obtain a conservative upper bound on step size. We then need only perform the CCD tests on primitive pairs in $\hat{C}$. This CCD culling generally provides an average 50 % speed-up for all CCD costs across our simulations, with negligible increase in Newton iterations and an overall impact of 10 % improvement in simulation times. Details on these accelerations and our adaptive application of this bound (to avoid taking overly conservative steps) are detailed in our Supplemental.

2.5 Variational friction forces

Frictional contact adds contact-dependent dissipative forcing to our system. At macroscale these friction forces are modeled by the MDP [Moreau 1973]. MDP posits that frictional forces maximize rate of dissipation in relative motion directions orthogonal to contact constraints up to a maximum magnitude imposed by limit surfaces, e.g. as modeled by Coulomb's constraint [Goyal

et al. 1991].

2.5.1 Discrete friction

To include frictional contact in our time stepping, we add local friction forces F_k for every active surface primitive pair, $k \in \hat{C}(x)$. For each such pair k, at current state x, we construct a consistently oriented sliding basis $T_k(x) \in \mathbb{R}^{3n\times 2}$. Each T_k is built so that $u_k = T_k(x)^\top (x - x^t) \in \mathbb{R}^2$ gives the local relative sliding displacement at contact k, in the frame orthogonal to the distance vector between closest points on the two primitives defining $d_k(x)$. See Section 2.6 and our supplemental document for details on construction of $T_k(x)$. The corresponding sliding velocity is then $v_k = u_k/h \in \mathbb{R}^2$.

Maximizing dissipation rate subject to the Coulomb constraint defines friction forces variationally

$$F_k(x,\lambda) = T_k(x) \operatorname*{argmin}_{\beta\in\mathbb{R}^2} \beta^\top v_k \quad \text{s.t.} \quad \|\beta\| \le \mu\lambda_k \tag{2.10}$$

where λ_k is the contact force magnitude and μ is the local friction coefficient.

Friction forces governed by Equation (2.10) are bimodal. If $\|v_k\| > 0$, there is sliding and the corresponding friction force opposes it with $F_k = -\mu\lambda_k T_k(x)\frac{u_k}{\|u_k\|}$. If $\|v_k\| = 0$, there is sticking and the corresponding static friction force is $F_k = -\mu\lambda_k T_k(x) f$, where the friction direction f can take any value in the closed 2D unit disk.

2.5.2 Challenges to Computation

Friction forces F_k are then challenging to solve for in three interconnected ways. First, F_k is *nonsmooth*. In transitions between sticking and sliding modes, nonsmooth jumps in both magnitude and direction are possible. Second, because of sticking modes, F_k in MDP is not uniquely

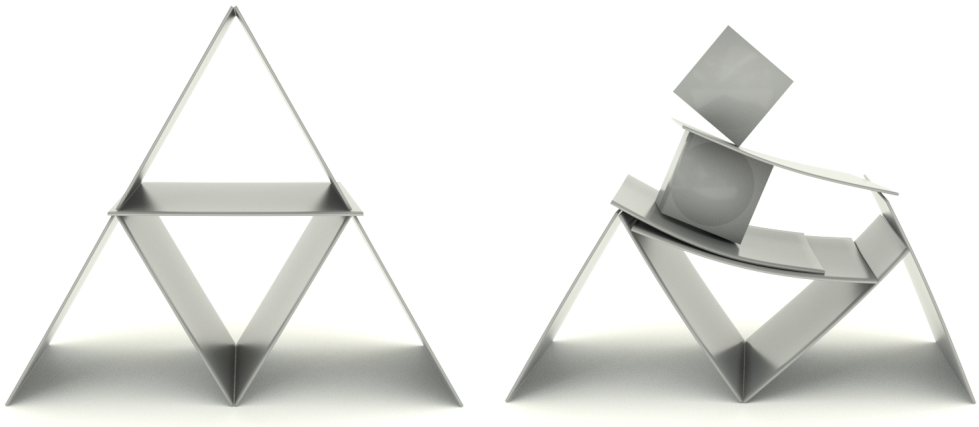


Figure 2.5: Friction benchmark: stiff card house. Left: we simulate a frictionally stable "card" house with $0.5m \times 0.5m \times 4mm$ stiff boards ($E = 0.1GPa$). Right: we impact the house at high-speed from above with two blocks; elasticity is now highlighted as the thin boards rapidly bend and rebound.

defined by displacements until we have found a solution satisfying stationarity:

$$\nabla B_t(x) - h^2 \sum_{k \in C} F_k(x, \lambda) = 0. \tag{2.11}$$

Third, there is no well defined dissipation potential whose spatial gradient will generate friction forces. As a consequence, frictional contact forces do not naturally fit into variational time-stepping frameworks.

To tackle these challenges, we first examine F_k as a nonsmooth function of u_k. Next, as in our barrier treatment of contact, we smooth the friction function with controlled and bounded accuracy. Then, in order to apply friction as an energy potential in our variational solve, we lag updates of the sliding bases T_k and contact forces λ_k over nonlinear solves within each time step (or over time steps). This allows us to define a smooth dissipative potential for friction that can be consistently integrated into our Newton-type solver.

2.5.3 Smoothed Static Friction

During each of our Newton iterations any transitions of sliding displacements to or from sticking conditions will introduce large and sudden jumps in friction forces, F_k. These discontinuities, if left unmollified, would severely slow and even break convergence of gradient-based optimization; see Section 2.7. To enable efficient and stable optimization, we smooth the friction-velocity relation in the transition to static friction.

We start with a useful and equivalent (re-)expression for friction forces:

$$F_k = -\mu\lambda_k T_k(x) f(\|u_k\|)\, s(u_k), \tag{2.12}$$

with $s(u_k) = \frac{u_k}{\|u_k\|}$ when $\|u_k\| > 0$, while $s(u_k)$ takes any 2D unit vector when $\|u_k\| = 0$. The friction magnitude function, f, is then correspondingly nonsmooth with $f(\|u_k\|) = 1$ when $\|u_k\| > 0$, and $f(\|u_k\|) \in [0, 1]$ when $\|u_k\| = 0$.

To smooth f, and so Equation (2.12), with bounded approximation error, we first define a velocity magnitude bound ϵ_v (in units of m/s) below which sliding velocities $v_k = u_k/h$ are treated as static. Then, we define a smoothed approximation of f with f_1. We maintain $f_1(y) = 1$ for all $y > h\epsilon_v$, (sliding) while for $y \in [0, h\epsilon_v]$, we require $f_1(y)$ to smoothly and monotonically transition from 1 to 0 over a finite range. This forms a bijective map from velocity magnitudes to friction magnitudes for velocities below the ϵ_v limit. For smoothing, we experiment with satisfying interpolating polynomials ranging from C^0 to C^2. Increased continuity order introduces greater smoothing and faster error reduction for decreasing ϵ_v, at the cost of introducing greater nonlinearity into the IP solve. In the end, we find that our C^1 interpolant

$$f_1(y) = \begin{cases} -\frac{y^2}{\epsilon_v^2 h^2} + \frac{2y}{\epsilon_v h}, & y \in (0, h\epsilon_v) \\ 1, & y \geq h\epsilon_v, \end{cases} \tag{2.13}$$

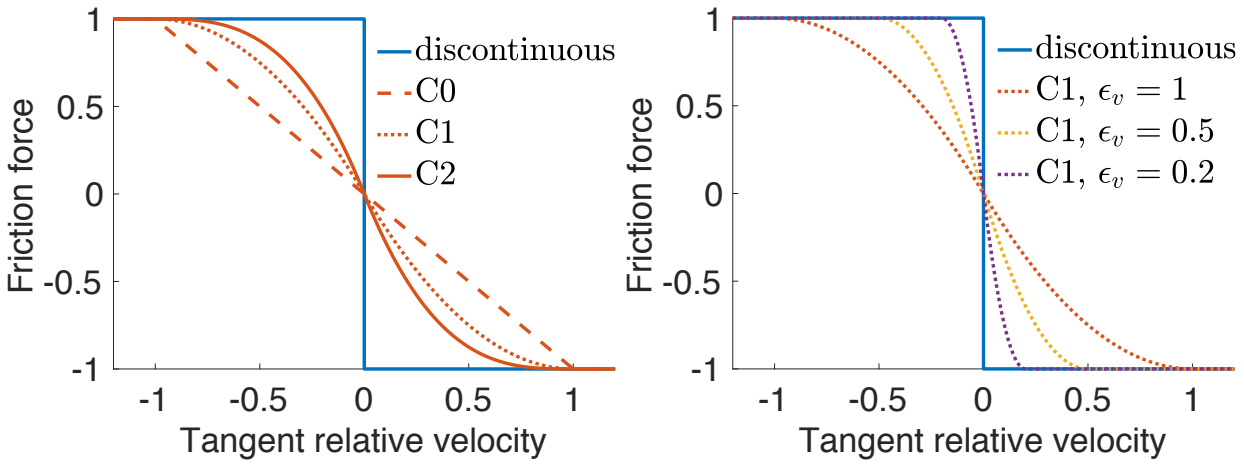


Figure 2.6: Friction smoothing in 1D. Left: increasing orders of our polynomials better approximate the friction-velocity relation with increasing smoothness. Right: Our C^1 construction improves approximation to the exact relation as we make our frictional accuracy threshold, ϵ_v, and so the size of static friction zone, smaller.

provides best balance – yielding a continuous force Jacobian while introducing less nonlinearity and so fewer overall iterations in testing. See Figure 2.6 and our discussion in the Supplemental.

2.5.4 Variationally Approximated Friction

With a smooth and uniquely defined F_k for each u_k, we are now able to define friction forces solely based on nodal displacement unknowns. A next natural step would then be to define a so-called dissipative potential [Kane et al. 2000; Pandolfi et al. 2002] for inclusion in our optimization. An ideal potential would be a scalar function with respect to x whose gradient returns F_k. However, even with our smoothing, no well-defined displacement-based potential for friction exists, and F_k cannot be approximated by a potential force without introducing significant approximation errors. In other words, we do not have a variational form of friction that we can yet minimize.

We start by making dependence of our friction on both $T_k(x)$ and $\lambda_k(x)$ explicit:

$$F_k(x, \lambda_k, T_k) = -\mu \lambda_k T_k f_1(||u_k||) \frac{u_k}{||u_k||}. \tag{2.14}$$

Now, if we set $T_k = T_k(x)$ and $\lambda_k = \lambda_k(x)$ this friction evaluation is exact. However, if we decouple dependence of the evaluated sliding basis and contact force from x and instead lag them to values, λ^n, T^n, from a prior nonlinear solve (or previous time step) n, then all remaining terms in the expression for friction are integrable. The lagged friction force is then $F_k(x, \lambda_k^n, T_k^n)$ and provides a simple and compact friction potential,

$$D_k(x) = \mu \lambda_k^n f_0(||u_k||). \tag{2.15}$$

Here f_0 is given by the relations $f_0' = f_1$ and $f_0(\epsilon_v h) = \epsilon_v h$ so that $F_k(x) = -\nabla_x D_k(x)$. This potential provides easy-to-compute Hessian, $\nabla_x^2 D_k(x)$, and energy contributions to the barrier potential, described in detail in the supplemental document. Our full friction potential is then $D(x) = h^2 \sum_{k \in C} D_k(x)$, and the frictional barrier-IP potential for the time step $t + 1$ is

$$B_t(x, \hat{d}) + D(x). \tag{2.16}$$

Friction Hessian projection. For our Newton method (Section 2.4.3), we again need to project the friction potential Hessian to the space of PSD matrices. The friction Hessian structure is similar to that of elasticity, in that it can be written as a product of the T_k matrices. This allows us to apply the same strategy as used for elasticity Hessians, and so we need only perform a 2×2 PSD projection for each friction term per primitive pair. This is detailed in our Supplemental.

2.5.5 Frictional Contact Accuracy

Accuracy of friction forces generated by each solution of our IP (Equation (2.16)) are defined by the static threshold, sliding basis and contact force magnitudes.

Static friction threshold. As we apply smaller ϵ_v we decrease the range of sliding velocities that we exert static friction upon and correspondingly sharpen the friction forces towards the

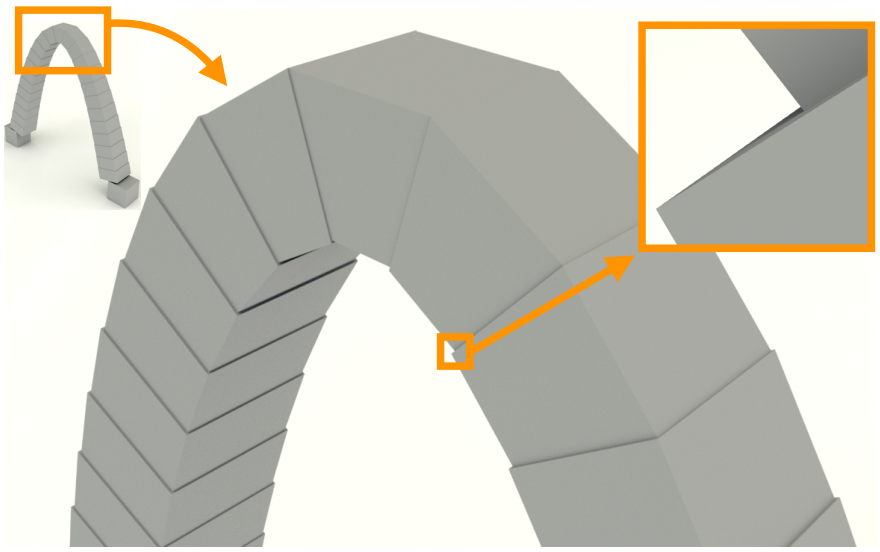


Figure 2.7: Friction benchmark: Masonry arch. IPC captures the static stable equilibrium of a 20 m high cement ($\rho = 2300\,\mathrm{kg/m^3}$, $E = 20\,\mathrm{GPa}$, $\nu = 0.2$) arch with tight geometric, $\hat{d} = 1\,\mu\mathrm{m}$, and friction, $\epsilon_v = 10^{-5}\,\mathrm{m/s}$ accuracy. Decreasing μ then obtains the expected instability and the stable arch does not form (see our supplemental videos). Inset: zoomed 100× (orange) highlights the minimal gaps with a geometric accuracy of small $\hat{d}$.

exact nonsmooth friction relation. Decreasing ϵ_v thus reduces stiction error while increasing compute times as we introduce a sharper nonlinearity in a tighter range; see Figure 2.6. For accurate reproduction of dynamic behaviors with friction and for visually plausible results, we observed that $\epsilon_v = 10^{-3}\ell m/s$, where ℓ is characteristic length (i.e. bounding box size), works well as a default across a wide range of examples with friction coefficients. See e.g., Figure 2.8. As static accuracy becomes important, we then find solutions with $\epsilon_v = 10^{-5}m/s$ work well. We have further confirmed IPC convergence down to $\epsilon_v = 10^{-9}m/s$. See, for example, our reproduction of the stable frictional contact structures in the masonry arch and card house examples in Figures 2.5 and 2.7.

Friction direction and magnitude. We improve accuracy of the direction and magnitude of the friction forces by solving successive minimizations of Equation (2.16) within each time step. For each solve we update the lagged T^n and λ^n (warm-starting from the previous time step) with results from the last nonlinear solve. Convergence of lagged iterations is then achieved when we

reach approximate momentum balance with

$$\|\nabla B_t(x^{t+1}) - h^2 \sum_{k \in C} F_k(x^{t+1}, \lambda^{t+1}, T^{t+1})\| \leq \epsilon_d, \tag{2.17}$$

where ϵ_d is the targeted dynamics accuracy.

We confirm lagged iterations rapidly converge over nonlinear solves with our FE models for the well-known, standard frictional benchmarks, e.g., block-slopes, catenary arches and card houses. See Figures 2.5 and 2.7 and Section 2.7. However, we emphasize that we do not have convergence guarantees for lagging. In particular, we have identified cases with large deformation and/or high speed impacts where we do not reach convergence for T and λ in the friction forces. Thus, in our large-deformation frictional examples we apply just a single lagging iteration. In these cases, sliding directions and contact force magnitudes in the friction force evaluation may not match. However, even in these cases, all other guarantees, including non-intersection, momentum balance (as in the frictionless case) and accurate stiction are maintained. More generally, we observe high-quality, predictive frictional results for large deformation examples independent of the number of lagging iterations applied; see e.g. Figure 2.16. We also emphasize that for frictionless models, IPC continues to guarantee convergence for contact and elasticity with just a single nonlinear solve per time step.

2.6 Distance Computation

Evaluating unsigned distance functions between point-triangle and edge-edge pairs requires care as closed-form distance formulas change with relative position of surface primitives.

2.6.1 Combinatorial Distance Computation

Unsigned distances are given by the closest points on the two primitives evaluated.

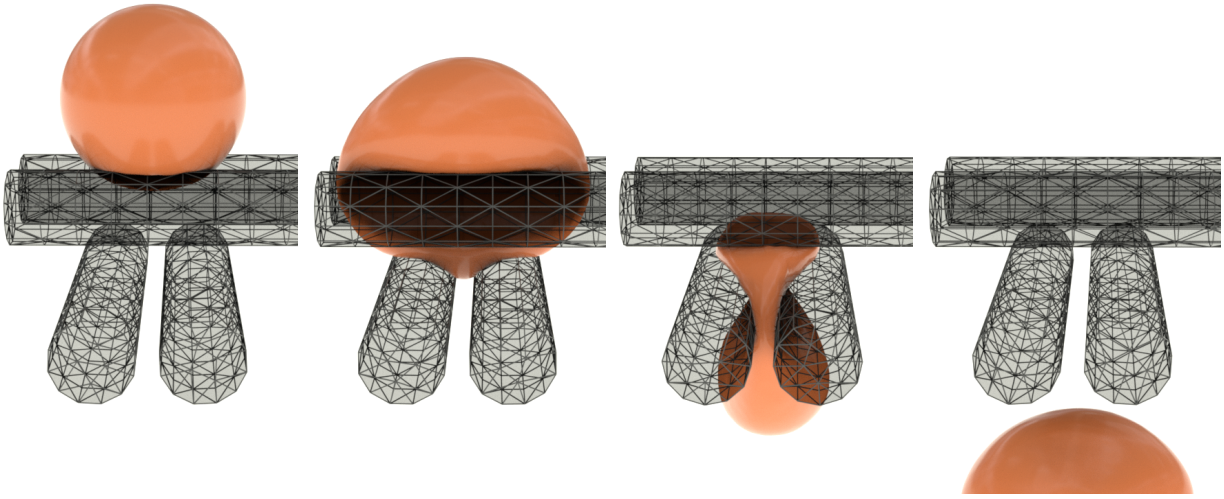


Figure 2.8: Large deformation, frictional contact test. We drop a soft ball ($E = 10^4 Pa$) on a roller (made transparent to highlight friction-driven deformation). Here IPC simulates the ball's pull through the rollers with extreme compression and large friction ($\mu = 0.5$).

Distance between a point v_P and a triangle $T = (v_{T1}, v_{T2}, v_{T3})$ can be formulated as a constrained optimization problem,

$$\begin{aligned} \mathcal{D}^{\text{PT}} = & \min_{\beta_1,\beta_2} ||v_P - (v_{T1} + \beta_1(v_{T2} - v_{T1}) + \beta_2(v_{T3} - v_{T1}))|| \\ & s.t. \quad \beta_1 \geq 0, \quad \beta_2 \geq 0, \quad \beta_1 + \beta_2 \leq 1. \end{aligned} \tag{2.18}$$

Similarly the distance between edges v_{11}-v_{12} and v_{21}-v_{22} is

$$\begin{aligned} \mathcal{D}^{\text{EE}} = & \min_{\gamma_1,\gamma_2} ||v_{11} + \gamma_1(v_{12} - v_{11}) - (v_{21} + \gamma_2(v_{22} - v_{21}))|| \\ & s.t. \quad 0 \leq \gamma_1, \gamma_2 \leq 1. \end{aligned} \tag{2.19}$$

Each possible *active set* of these two minimizations corresponds to a closed-form distance formula. In each, at most two constraints can be active at the same time.

- When *two* constraints are active in either Equation (2.18)) or (Equation (2.19), the distance

between primitives is a point-point distance evaluation:

$$d^{PP} = ||v_a - v_b||. \tag{2.20}$$

Here v_a and v_b correspond to v_P and a v_{Ti} for Equation (2.18)), or to the two endpoints of the edges in the edge-edge pair for Equation (2.19).

- When a *single* constraint is active in either Equation (2.18)) or (Equation (2.19), the distance in both cases becomes a point-edge distance evaluation:

$$d^{PE} = \frac{||(v_a - v_c) \times (v_b - v_c)||}{||v_a - v_b||}. \tag{2.21}$$

Here (v_a, v_b) corresponds to one of the triangle edges of T and $v_c = v_P$ for Equation (2.18)), or else (v_a, v_b) corresponds to one of the edges in the edge-edge pair and v_c corresponds to an endpoint of the other edge for Equation (2.19).

- When *no* constraints are active in either Equation (2.18)) or Equation (2.19), distance computations are simply parallel-plane distance evaluations. For the point-triangle pairing in Equation (2.18) this is

$$d^{PT} = |(v_P - v_{T1}) \cdot \frac{(v_{T2} - v_{T1}) \times (v_{T3} - v_{T1})}{||(v_{T2} - v_{T1}) \times (v_{T3} - v_{T1})||}|, \tag{2.22}$$

while for the edge-edge pairing in Equation (2.19) it is

$$d^{EE} = |(v_{11} - v_{21}) \cdot \frac{(v_{12} - v_{11}) \times (v_{22} - v_{21})}{||(v_{12} - v_{11}) \times (v_{22} - v_{21})||}|. \tag{2.23}$$

For evaluations of d, ∇d, and $\nabla^2 d$, we apply the currently valid, closed-form distance formula (either point-point (PP), point-edge (PE), point-triangle (PT), or edge-edge (EE) above) and its

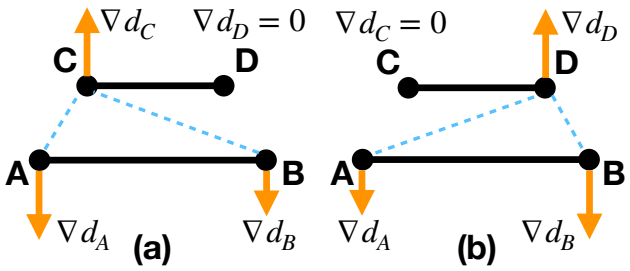


Figure 2.9: Nonsmoothness of parallel edge-edge distance. When edge AB and CD are parallel, the distance computation can be reduced to either (a) $C - AB$ point-edge or (b) $D - AB$ point-edge. Then for the trajectory of C moving down from above D, the distance gradient is not continuous at the parallel point even though the distance is always continuously varying.

analytic derivatives. The formula to apply, at each evaluation of a surface pair, is determined by the active constraint subset defined by the current relative positions of the pair's primitives. This information is computed and stored together with our culled constraint set $\hat{C}$ data, and so is then available for direct use whenever computing barrier energies and derivatives. This treatment is analogous to storing and reusing singular value decompositions of deformation gradients for elasticity computations. As in elasticity, our distance state and evaluations can efficiently be reused for all energy and derivative evaluations at the same nodal positions. Correspondingly, having now reduced general point-triangle and edge-edge distance evaluations to the above closed-form formulas, we can directly compute and store our sliding bases, $T_k(x)$, for friction computation with respect to each case; please see our Supplemental for details.

2.6.2 Differentiabilty of d

In collision-resolution methods, close-to-parallel edge-edge contacts are notorious failure modes – to the extent that existing methods often ignore this case by throwing out all corresponding constraints [Harmon et al. 2008]. However, despite the challenges imposed, these constraint cases cannot be removed, as doing so would lead to intersection. The reason for the difficulty in these cases is the (lack of) differentiability of the distance function for some configurations. Each above analytic formula for distances corresponds to a subset of the relative configuration space

of a primitive pair. For example, for vertex-triangle pairs, relative configurations are completely characterized by fixing the triangle and varying v_P positions. If the projection of v_P to T is in the triangle interior, no constraints are active, while if the projection lies on the interior of a triangle edge then one constraint is active. Otherwise, two constraints are active.

Each of these geometric criteria defines a subset of $\mathbf{R}^3$, where one of the three analytic formulas is valid. The distance function is C^∞ inside each such domain, and, in general, is C^1 at the boundaries between domains. However, the critical exception is in parallel edge-edge configurations: at these points, the distance function is not differentiable (see Figure 2.9). Configurations close to these parallel edge-edge conditions, when reached, lead to unacceptably slow convergence of Newton iterations or even convergence failures altogether. Numerically, the issue is similar to the C^0-continuous friction problem we faced in Section 2.5.2. To resolve this issue, we once again apply a local smoothing solution to mollify the barrier corresponding to nearly parallel edge-edge contact conditions.

We smooth by multiplying all edge-edge barrier terms by a piecewise-polynomial mollifier closely analogous to our static-friction smoother; recall Figure 2.6. Here, for each edge-edge contact pair k, we define $e_k(x)$ to vanish when edges $(v_{11}v_{12} - v_{21}v_{22})$ are parallel and to smoothly grow to 1 as the edge-pair become far from parallel,

$$e_k(x) = \begin{cases} -\frac{1}{\epsilon_\times^2}c^2 + \frac{2}{\epsilon_\times}c & c < \epsilon_\times, \\ 1 & c \geq \epsilon_\times, \end{cases} \tag{2.24}$$

where $c = ||(v_{12} - v_{11}) \times (v_{22} - v_{21})||^2$ and $\epsilon_\times = 10^{-3}||v'_{12} - v'_{11}||^2||v'_{22} - v'_{21}||^2$ is defined with respect to edge-edge vertex-pair rest positions v'.

Our mollified edge-edge barriers are then $e_k(x)b\big(d_k(x)\big)$ and so now extend our barrier potentials to a piecewise C^∞, *everywhere* C^1-continuous (for nonintersecting configurations) barrier formulation. At the same time our barriers now remain sufficient to guarantee that no collisions

are missed: there are always point-triangle contact pairs at distance no more than the parallel edge-edge distance; see our Supplemental for details on this. In turn, our construction of the parallel-edge mollifier then minimizes its impact on edge-edge pair barriers as they move away from degeneracy. While in principle increasing smoothness to C^1 is sufficient to avoid most dramatic degeneracy failures, there are additional numerical stability issues to be addressed related to nearly parallel edges. Please see our Supplemental for details.

Now, with this third and last smoothing in place we have an overall time-stepping potential for contact and friction that can leverage superlinear convergence and robustness of Newton-type stepping. As we analyze in Sections 2.7 and 2.8 below (see Section 2.7.1) this gains robust simulation against failure – even when simulating challenging conditions with unavoidable numbers of degenerate evaluations.

2.7 Evaluation

Our IPC code is implemented in C++, parallelizing assembly and evaluations with Intel TBB, applying CHOLMOD [Chen et al. 2008] with approximate minimum degree (AMD) reordering for linear system solves in all examples (except for the squishy ball example – see below) and Eigen [Guennebaud et al. 2010a] for linear algebra routines. We run most experiments on a 4-core 3.5 GHz Intel Core i7, a 4-core 2.9 GHz Intel Core i7, and a 8-core 3.0 GHz Intel Xeon machine. Machine use per example is summarized along with performance statistics and problem parameters in Table 2.2 and in our Supplemental. The reference implementation, scripts used to generate these results and our benchmarks are released as an open-source project.

Linear system computations and solves. We compute elasticity and barrier Hessians (with PSD projections) in parallel, and have designed and implemented a custom multi-threaded, sparse matrix data structure construction routine that, given the connectivity graph of nodes, efficiently

builds the compressed sparse row (CSR) format with index entries ready. While we utilize efficient symbolic factorization and parallel numerical factorization routines in CHOLMOD [Chen et al. 2008] compiled with Intel Math Kernel Library (MKL) LAPACK and BLAS, we also tested IPC with AMGCL [Demidov 2019] – a multigrid preconditioned solver. Here, we found behavior is as might be expected, less memory overhead and faster linear solves by avoiding direct factorization. However, for majority of examples the large deformations and many contacts generate poorly conditioned systems. We then found AMGCL requires extensive parameter tuning to perform well and still can not compete, in general, with the parallel direct solver. All examples in the following then apply CHOLMOD for linear solves, with the exception of our largest, squishy ball example (Figure 2.22), where we apply AMGCL.

Models and practical considerations. We primarily employ the non-inverting, neo-Hookean (NH) elasticity model and implicit Euler time stepping. In the following examples we also apply and evaluate implicit Newmark time stepping, as well as the invertible fixed corotational (FCR) elasticity model. While for clarity in the preceding we derive IPC with unmodified distance evaluations, for numerical accuracy and efficiency our implementation applies squared distances for evaluations of the barrier, we use $b(d^2, \hat{d}^2)$, and related computations, thus avoiding squared roots. In turn expressions for contact forces, λ_k, and related terms must be modified, from our direct exposition and derivations above. To do so we rescale for consistent dimensions and units in our implementation; see our Supplemental for details. Finally and importantly we note that IPC's barrier formulation requires nonzero separation distances to be strictly satisfied at initialization and then guarantees it throughout simulation. Exact initialization at zero distance is neither possible (as the barrier of course diverges) nor for that matter physically meaningful. Contact, including resting contact, instead occurs around the specified geometric distance accuracy given by the user. Here we demonstrate simulated configurations with distances down to 10^{-8} m reached in simulation (e.g., arch in Figure 2.7) or initialized by users.

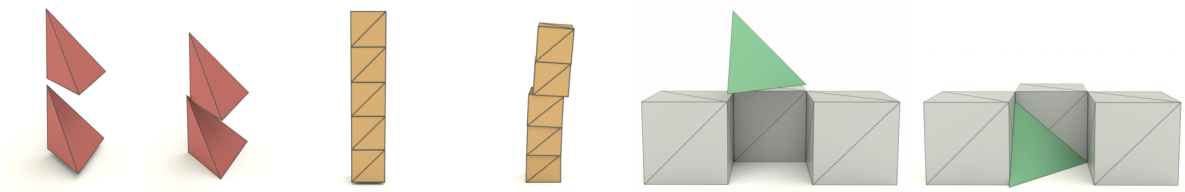


Figure 2.10: Aligned, close and nonsmooth contact tests. Pairs of before and after frames of deformable geometries initialized with exact alignment of corners and/or asperities; dropped under gravity. We confirm both nonsmooth and conforming collisions are accurately and stably resolved.

Evaluation and tests. Below we first introduce a set of unit tests for seemingly simple yet challenging scenarios with nonsmooth, aligned and close contacts (Section 2.7.1), stress tests involving large deformation and high velocities (Section 2.7.2), and friction (Section 2.7.3). We next study IPC's scaling, run time, and accuracy behavior as we vary simulation problem parameters (Section 2.7.4). Finally, we present an extensive, quantitative comparison with previous works in Section 2.8.

2.7.1 Unit Tests

Aligned, close and nonsmooth contact. We apply a set of unit tests exercising closely aligned, conforming and nonsmooth contact known to stress contact algorithms. We build them with two simple models: a single tetrahedron and an 8-node unit cube; see Figures 2.10 and 2.11. For contact handling, these seemingly simple tests are designed to trigger degenerate edge cases that often cause failure in existing methods (see Section 2.8). IPC resolves all cases including those in which we exercise *exact* parallel edge-edge (e.g., Figure 2.10 middle) and point-point (e.g., Figure 2.10, left) collisions. For unit tests like Figure 2.10 right we drop objects into slotted obstacles so that they fit tightly with tiny gaps; here IPC retrieves a tight conforming fit into a 1 µm gap.

Erleben's fundamental cases. Erleben [2018] proposes unit tests (see Figure 2.11 top row) for contact constraint failure testing. Here these tests are again simple but designed to challenge

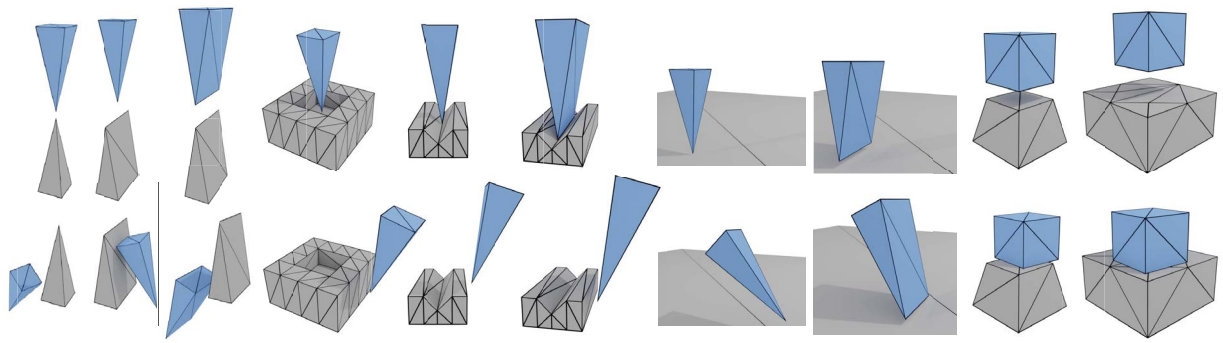


Figure 2.11: Erleben's tests. Top: fundamental test cases to challenge mesh-based collision handling algorithms proposed by Erleben [2018]. Bottom: IPC robustly passes all these tests even when stepped at frame-rate size time steps.

mesh-based collision-handling algorithms. IPC again resolves all tests robustly (see Figure 2.11, bottom row), even when stepped at frame-rate size time steps.

Tunneling. Tunneling through obstacles when simulating high-speed velocities is a common failure mode in dynamic contact modeling. We thus add an example to our unit tests: we fire an elastic ball (diameter 0.1 m) at a fixed 0.02 m thin board at successive speeds of 10, 100, and 1000 m/s stepped at $h = 0.02$ s. IPC accurately rebounds at large time step without tunneling in all cases.

Large mass and stiffness ratio tests. Contact resolution between objects with largely varying scale, mass, and/or stiffness ratios has long-challenged time stepping methods due to ill-conditioning. In Figure 2.12, we simulate IPC dropping of a range of objects upon each other with widely varying weight and stiffness. Here we apply $E = 0.1$ GPa for the sphere, board, and large cube, $E = 1$ MPa for the small cube and the mat holding the sphere, and $E = 10$ kPa for the mat dropped on boards. For the stiff ball and large cube, we set their respective densities to 2× and 10× that of softer objects (1000 kg/m^3) to add large mass ratios to the challenge. Regardless of these different ill-conditioned settings, IPC simulates all scenes robustly and efficiently without any artifacts; see also our supplemental videos.

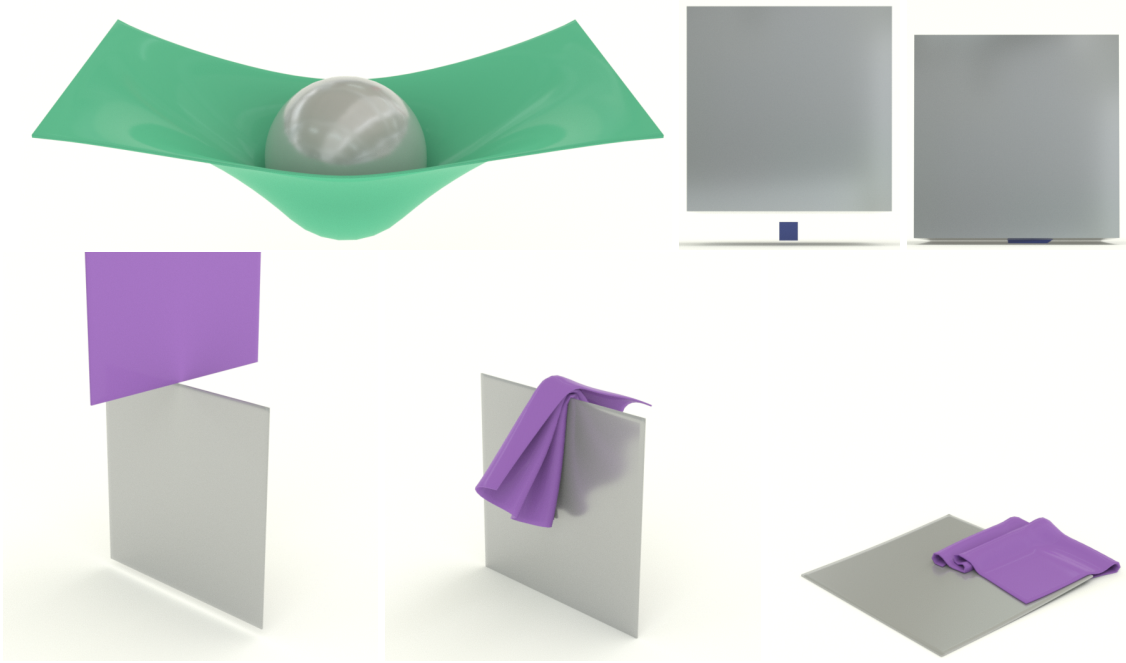


Figure 2.12: Large Mass and Stiffness ratios.

Chains. While resolving transient collisions exercises stability, large numbers of persistent, coupled contacts, as in a long chain of elastic links, exercises contact constraint accuracy. A small amount of constraint error integrated over time will cause such chains to break. We simulate chains of 100 elastic links under gravity, observe stable oscillations and shock-propagation while shorter chains stably bounce – all preserve constraints; see our supplemental videos.

2.7.2 Stress Tests

We next consider IPC's ability to resolve a range of extreme stress-test examples motivated by well-known pre-existing challenges and previously proposed benchmarks.

Funnel. To confirm contact resolution under strong boundary conditions, extreme compression, and elongation, we pull a stiff NH material dolphin model through a codimensional funnel mesh obstacle. We step IPC at large time steps of $h = 0.04$s with up to 32.3K contacts per step. The resulting simulation is intersection- and inversion-free throughout with the model regaining

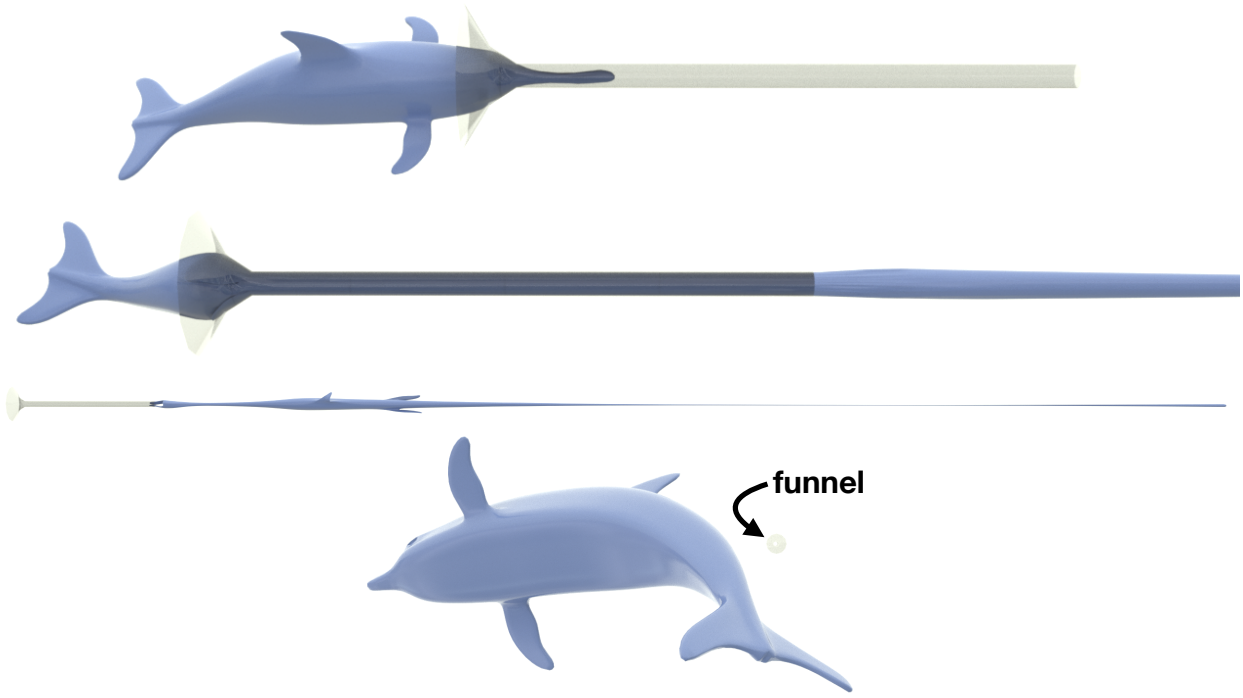


Figure 2.13: Funnel test. Top: the tip of a stiff neo-Hookean dolphin model is dragged through a long, tight funnel (a codimensional mesh obstacle). Middle top: due to material stiffness and tightness of fit the tip of the model is elongated well before the tail pulls through. Middle bottom: extremity of the deformation is highlighted as we zoom out immediately after the model passes through the funnel. Bottom: finally, soon after pulling through, the dolphin safely recovers its rest shape. We confirm that the simulation is both intersection- and inversion-free throughout all time steps.

its rest shape once pulled through (Figure 2.13).

Thin volumetric meshes. Thin geometries notoriously stress contact simulations. Likewise, as more simulation nodes are involved in collision stencils, simulation challenges grow. Here we test IPC's handling of extreme cases with both challenges, by simulating single layer meshes of tetrahedra. Here IPC robustly handles the contacts with accurate solutions at all time steps across a range of large deformation contact examples (Figures 2.5, 2.12, and 2.14).

Extreme and extended twisting. As large deformation high-contact examples, we twist thin mats (Figure 2.14), rods (Figure 2.4), and Armadillos (Figure 2.21, bottom) with rotating speeds of $72\,^\circ/\mathrm{s}$ at both ends. We simulate the twist of both the rods and mats for 100s – efficiently

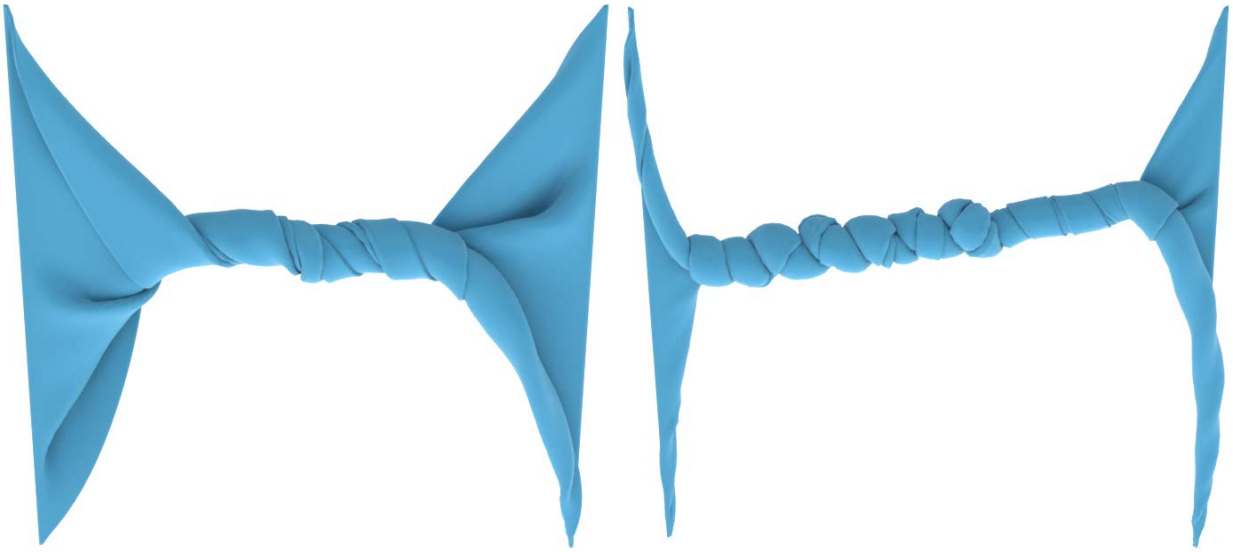


Figure 2.14: Stress test: extreme twisting of a volumetric mat for 100s. Left: IPC simulation at 10s after 2 rounds of twisting at both ends. Right: at 40s after 8 rounds of twisting. This model, designed to stress IPC, has all of its 45K simulation nodes lying on the mesh surface.

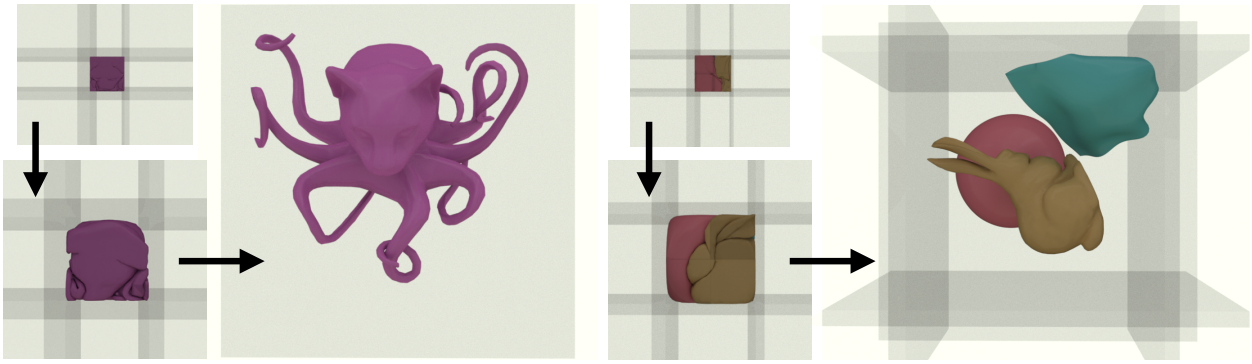


Figure 2.15: Trash Compactor. An octocat (left) and a collection of models (right) are compressed to a small cube by 6 moving walls and then released. Here, under extreme compression IPC remains able to preserve intersection- and inversion-free trajectories solved to requested accuracies.

capturing increasingly tight conforming contact and expected buckling in all simulations.

Compactor test. In Figure 2.15, we test the "trash" compactor-type examples from Harmon et al. [2009]. After releasing the compactor from the extreme compression point we clearly see that the tentacles of the octocat model and correspondingly the sphere, mat, and bunnies models are all cleanly separated.

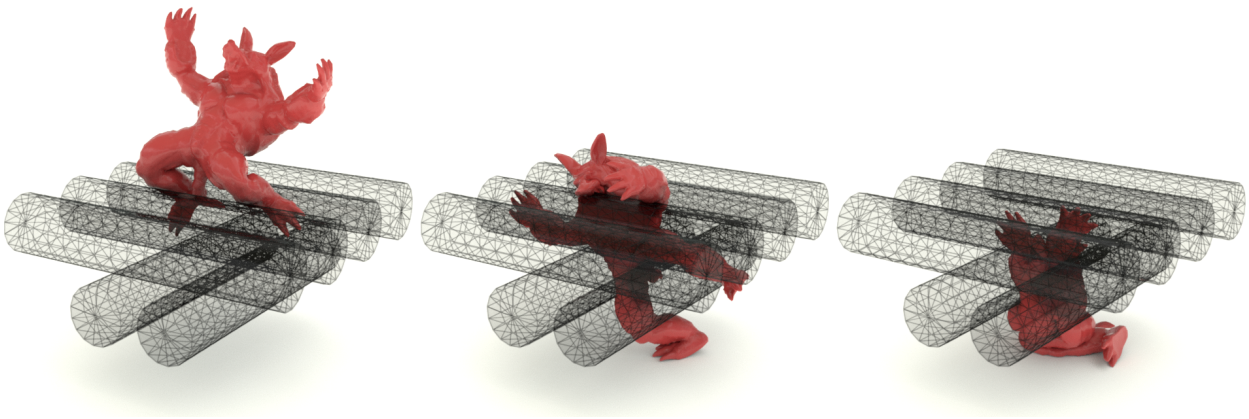


Figure 2.16: Roller tests. Simulating the Armadillo roller from Verschoor and Jalba [2019] (same material parameters) in IPC now captures the expected stick-slip behavior for the high-friction, moderate stiffness conditions.

Rollers compression and stick-slip instability. To combine extreme deformation with friction, we match the set-up of the kinematic roller test from Verschoor and Jalba [2019] with the same originally applied, high friction coefficient $\mu = 0.5$ (Figure 2.16). This scene is highly challenging due to the competing large magnitude of the friction and the large compression induced by the rollers. Here, with a moderately stiff material ($E = 5 \times 10^5$ Pa Young's modulus) we observe that IPC with our friction model obtains the expected stick-slip instability effects that such competition should generate. In simulation we observe deformation grows in opposition to static friction in the rollers until stress overcomes static friction and we observe slip – this process is then repeated. This stick-slip effect is captured by our Armadillo with moderate stiffness when tested with both the NH and FCR elasticity models (see our supplemental videos for the motion). We also note, as expected, when we subsequently test with softer material, i.e., $E = 5 \times 10^5$ Pa, we get smooth rolling behavior for the Armadillo, as expected, without stick slip.

Codimensional collision obstacles. Collision obstacles, especially in animation and gaming, are often most easily expressed in their default form as triangle meshes or even unorganized triangle soups. While highly desirable in applications, codimensional collision types are not generally supported by available simulation methods, which often suffer tunneling, snagging, and

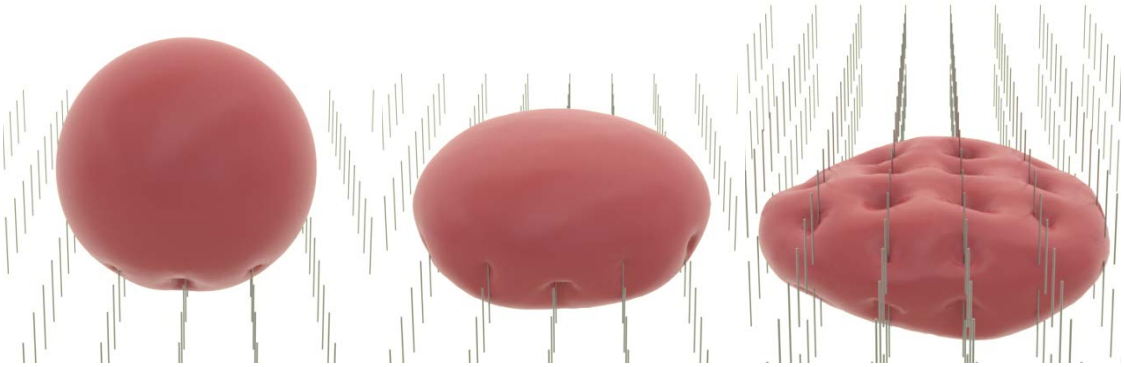


Figure 2.17: Codimensional collision objects: pin-cushions. We drop a soft ball onto pins composed of codimensional line-segments and then torture it further by pressing down with another set of codimensional pins to compress from above. IPC robustly simulates the ball to a "safe", stable resting state under compression against the pins.

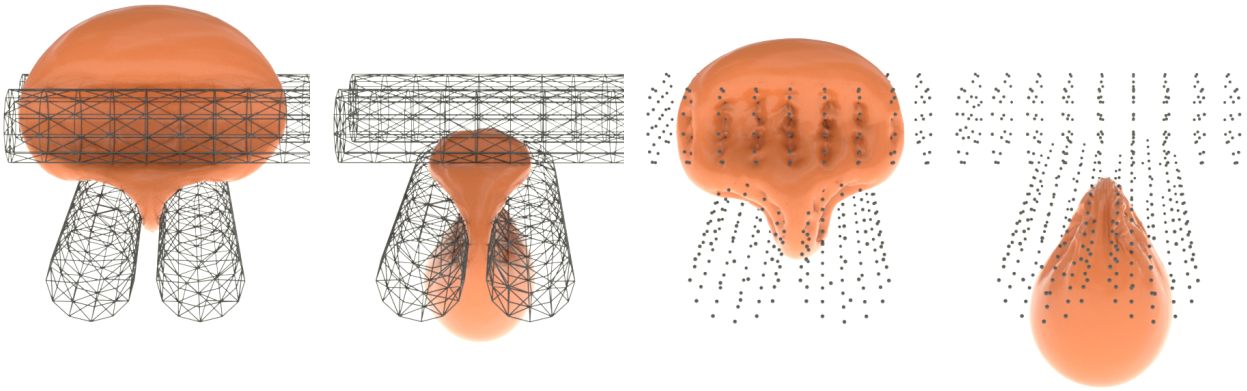


Figure 2.18: Codimensional collision objects: rollers. We modify the ball roller example from Figure 2.8 by using only the edge segments (left) or even just vertices (right) for the moving roller obstacle. For these extremely challenging tests IPC continues robust simulation exhibiting tight compliant shapes in contact regions pressed by the sharp obstacles.

resulting instabilities when exposed to them. To our knowledge IPC is the first algorithm to stably and accurately resolve collisions between volumes and codimensional collision objects. We perform a set of tests dropping different objects on planes, segments, and points, see e.g., Figures 2.2, 2.17, and 2.18. Collisions are stably resolved and we see tight compliance to the sharp poking obstacles in contacting regions.

Codimensional rollers. What if we modify the roller test in Figure 2.8, leaving only the edges or even only the points for the roller obstacles? This leads to our codimensional roller tests

(Figure 2.18). Here, with solely codimensional wire (just edges) and points (just vertices) rollers, the big ball still pulls inwards, forming tightly pressed geometries in the contact regions as it is compressed and pulled against and then out of the codimensional rollers (Figure 2.18). For sharp point rollers we require negligible friction (for points we apply $\mu = 10^{-3}$) to pull the ball inwards as the sharp points directly grab the deforming surface.

Squeeze out stress test. A plate compresses and then forces a collection of complex soft material models into a tight conforming mush through a thin co-dimensional tube obstacle. Once through they cleanly separate (Figure 2.1).

High speed impact test. To examine IPC's fidelity in capturing high-speed dynamics we match the reported material properties and firing speed of an experiment of foam practice ball fired at high-speed towards a fixed steel wall. In Figure 2.19 top, we show key frames of a high-speed capture of the event. Middle: we visualize velocity magnitudes simulated by IPC, stepped with implicit Newmark and the NH material, at the same corresponding times in the simulation, and bottom the IPC-simulated geometry. Here we observe both the expected shockwave propagating through the sphere during the finite-time collision as well as the overall matching dynamics and shape across the simulation. Please see our supplemental video for complete simulation moving through the phases of inelastic collision impact: compression (first shockwave), restoration (second shockwave), and release.

2.7.3 Frictional Contact Tests

To examine IPC's frictional model we simulate a set of increasingly challenging frictional benchmark tests. All utilize a tight accuracy of $\epsilon_v = 10^{-5}$ m/s and apply lagged iterations to update sliding bases and normal forces until the system is confirmed as fully converged by satisfying Equation (2.11).

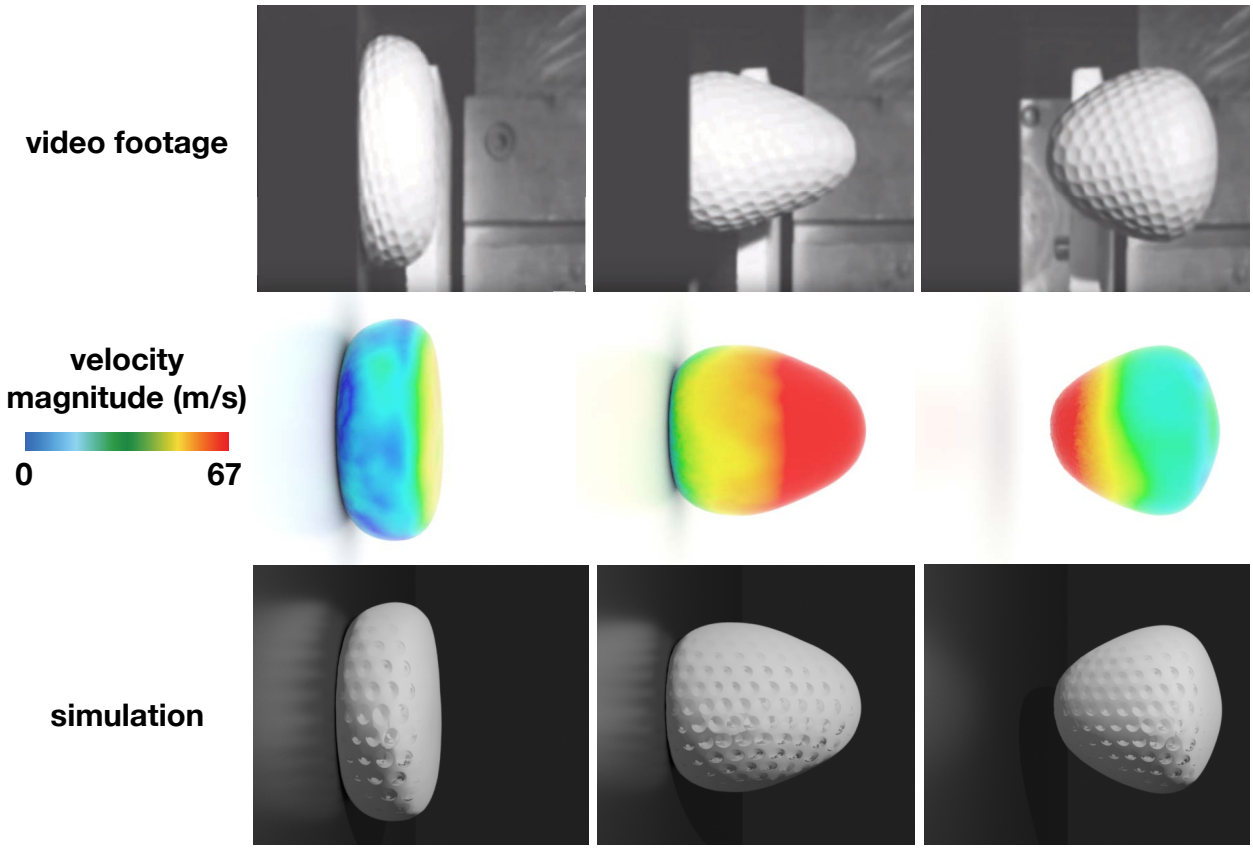


Figure 2.19: High-speed impact test. Top: we show key frames from a high-speed video capture of a foam practice ball fired at a fixed plate. Matching reported material properties (0.04m diameter, $E = 10^7$Pa, $\nu = 0.45$, $\rho = 1150$kg/m^3) and firing speed ($v_0 = 67$m/s), we apply IPC to simulate the set-up with Newmark time stepping at $h = 2 \times 10^{-5}$s to capture the high-frequency behaviors. Middle and bottom: IPC-simulated frames at times corresponding to the video frames showing respectively, visualization of the simulated velocity magnitudes (middle) and geometry (bottom).

Block tests. We start by placing stiff elastic blocks on a slope with tangent at 0.5. Here for $\mu = 0.5$, IPC generates the expected result of frictional equilibrium – the block does not slide. Switching to $\mu = 0.49$, IPC then immediately sets the block sliding, again matching the analytic solution.

Frictionally dependent structures. We test IPC on the challenging, frictionally dependent stable structure tests from Kaufman et al. [2008]. We model both the card house (Figure 2.5) and masonry arch (Figure 2.7) with stiff deformable materials. We further extend the challenge of the

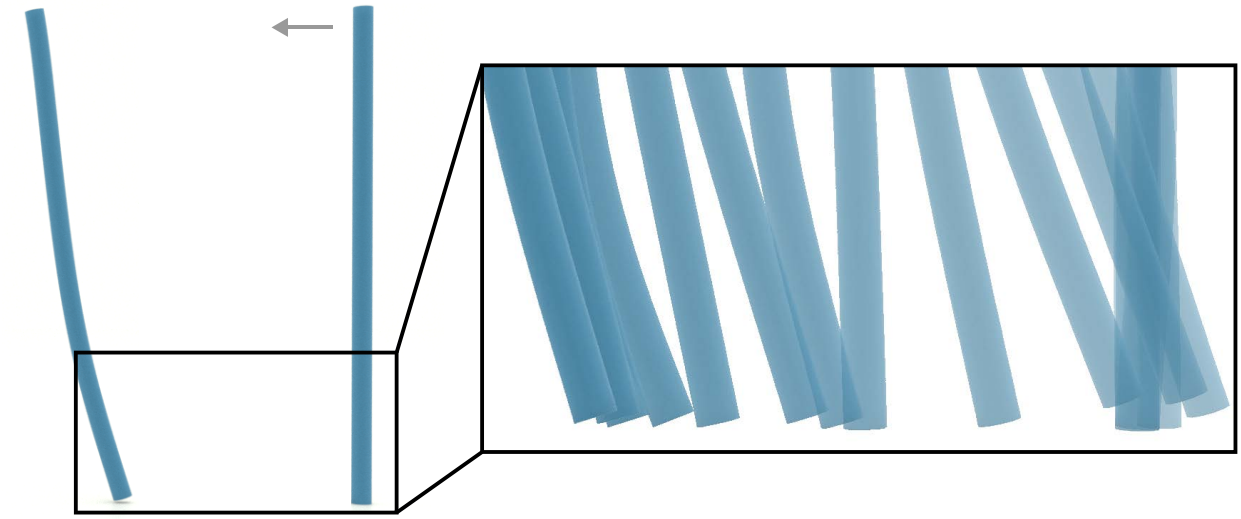


Figure 2.20: Stick-slip. oscillations with friction simulated with IPC by dragging an elastic rod along a surface.

arch with a precarious base balanced on sharp edges. We obtain long-term stable structures with $\mu = 0.5$ and $\mu = 0.2$ respectively and confirm that they fall apart as we reduce to $\mu = 0.2$ and $\mu = 0.1$ respectively (see our supplementals for statistics and videos).

Stick-slip instability. Finally, we script the motion of the top of a thin, volumetric elastic rod pushed slightly down towards, and then along a surface ($\mu = 0.35$) to test stick-slip oscillations. As in the Armadillo roller example, large static friction creates a buildup of elastic energy in the rod which is released when the friction force, opposing sliding contact, is exceeded by the tangential stiffness at the contact. This interaction between the friction forces and the sliding velocities becomes periodic, and so induces self-excited oscillations that buildup and dissipate energy; see Figure 2.20 and our supplemental video.

2.7.4 Scaling, Performance, and Accuracy

Varying time step sizes. Existing contact-resolution methods generally rely on small time step sizes for simulation success. As demonstrated above, IPC is able to simulate across a wide

Table 2.1: Increasing time step sizes to frame-rate and beyond. Here we demonstrate tradeoffs in varying time step sizes for the same tight twisted rods example (2.5K nodes, 6.9K tets, 4.6K faces, $E = 10^4 Pa$) with a 4-core 2.9GHz Intel Core i7 CPU, 16GB memory. # iters is the number of Newton iterations *per time step* or in *total* for the simulation sequences.

h (s)	# constraints	per time step		total	
	avg (max)	t (s)	# iters	t (s)	# iters
0.002	137 (430)	0.29	2.12	862	6351
0.005	194 (584)	0.36	2.37	435	2843
0.01	269 (707)	0.38	2.65	229	1591
0.025	435 (1.0K)	0.38	2.69	91	645
0.05	551 (1.2K)	0.46	3.06	56	368
0.1	597 (1.3K)	0.73	4.75	44	285
0.2	607 (1.2K)	1.79	14.37	54	431
0.5	653 (1.4K)	11.39	100.58	137	1207
1	708 (1.3K)	18.41	188.17	110	1129
2	843 (1.3K)	52.02	522.00	156	1566

range of time step sizes h and so can capture a range of different frequency effects. Choice of time step size for IPC is then simply a question of accuracy required per application as balanced against efficiency needed, rather than a predicate required for success. To investigate the effect of varying time step size, h, in IPC we simulate the tight twisted rods example (Figure 2.4) for 6 s. We range h from 0.002 to 2 s. In Table 2.1 we observe that transitioning from large to small time step sizes, our method improves its *per time-step* performance – but not by orders of magnitude. This is because the costs of intersection-free time stepping, distance computation and CCD do not change much. Since we do not miss any contacts, the number of constraints we process decrease only sublinearly as we decrease time step sizes. This is a key computational feature to ensure feasibility and robustness. On the other hand, we happily observe that our method is robust even well beyond standard time step sizes. While, in general, such excessively large step sizes beyond frame-rate are not useful for dynamics, this offers a robust opportunity for quasi-statically computing equilibria subject to challenging contact conditions. When we deploy IPC with implicit Euler IP (taking advantage of numerical dissipation), these very-large time steps rapidly compute equilibria with extreme contact conditions in just a few steps.

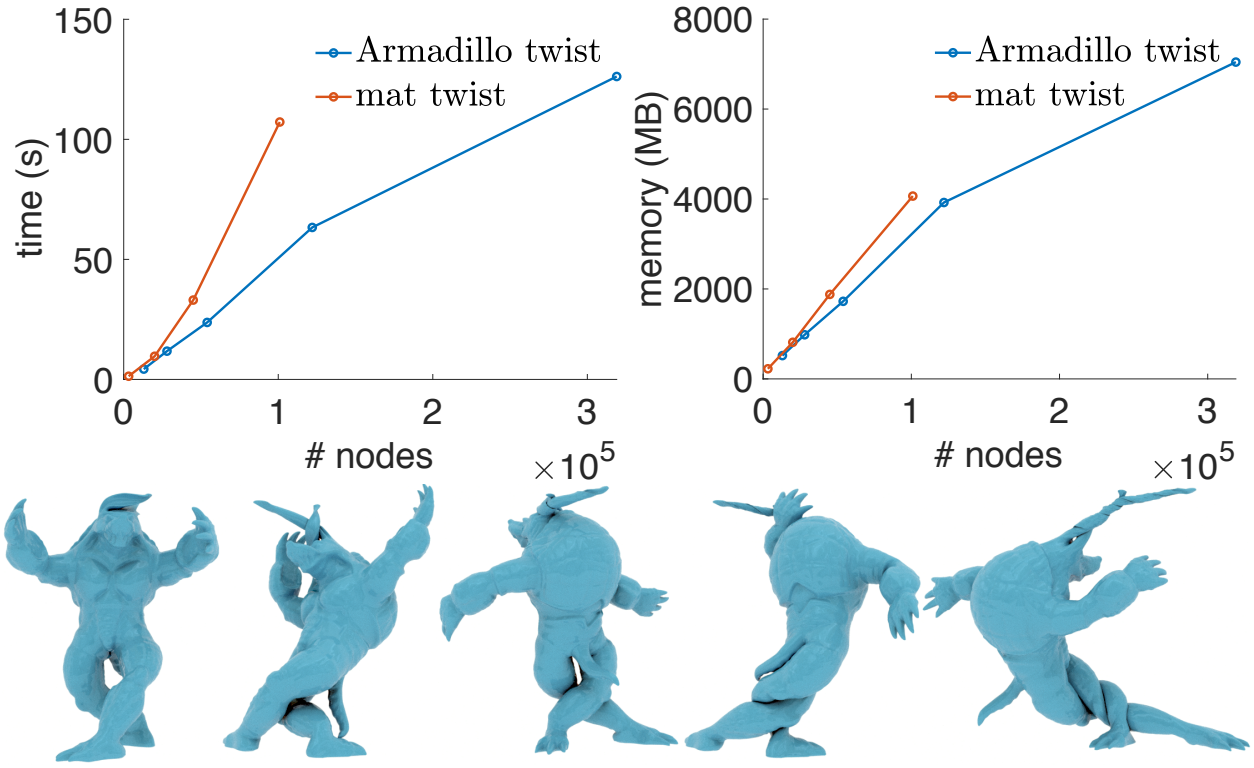


Figure 2.21: Scaling tests. Top: applying increasing resolution meshes ranging from 3K to 219K nodes we examine the time (left) and memory (right) scaling behavior of IPC on a range of resolutions of the twisting Armadillo and twisting mat (Figure 2.14) examples. Bottom: frames from the highest-resolution twisting Armadillo example (219K nodes, 928K tets).

Scaling. In Figure 2.21 (top), we study scaling behavior of IPC, with twisting mat (Figure 2.14) and twisting Armadillo (Figure 2.21, bottom) simulations of increasing-resolution meshes ranging from 3K to 219K nodes. Armadillo is a representative volumetric model while the single-layer mat is an extreme example designed to especially stress IPC. The mat meshes importantly have all simulation nodes on their surfaces and so, as contacts tighten in the twisting mat, they can form arbitrarily dense Hessians. For the mat we observe iteration count, memory and contact counts increase linearly with resolution, while timing increases in a slight superlinear trend. For the more standard volumetric Armadillo model we observe iteration count remains flat as we increase resolution, while timing and memory increase linearly.

In addition, when mesh sizes and contacts grow large, available memory can potentially pre-

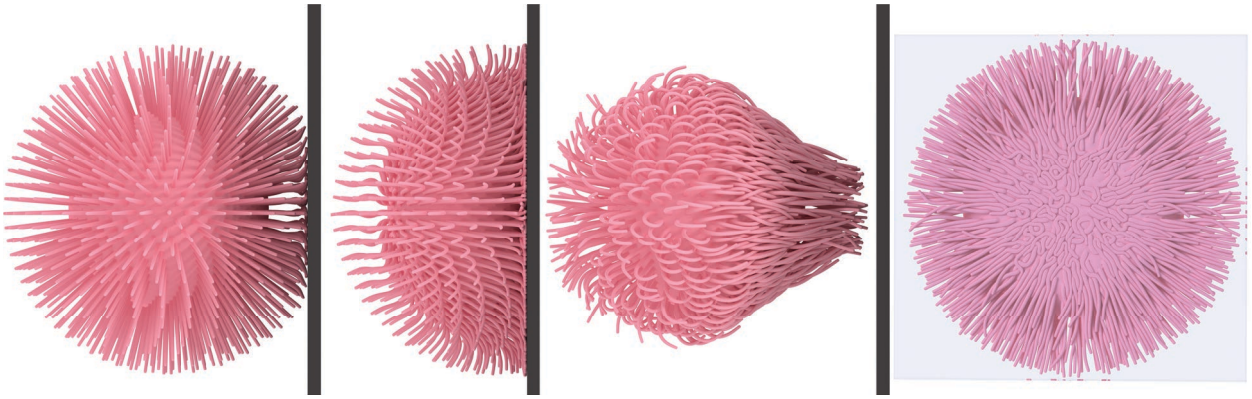


Figure 2.22: Squishy ball. Simulated by IPC an elastic squishy ball toy model (688K nodes, 2.3M tets) is thrown at a glass wall. The left three frames show side views before, at, and after the moment of maximal compression during impact. The right-most frame then shows the view behind the glass during the moment of maximal compression, highlighting how all of the toy's intricately intertwined tendrils remain intersection free.

clude application of direct linear solvers. To confirm IPC applicability in these settings we simulate the firing of a 688K node, 2.3M tetrahedra, squishy ball model from Zheng and James [2012] at a glass board using AMGCL's [Demidov 2019] multigrid-preconditioned iterative linear solver. Here both the large element count and the large numbers of collisions enabled by the toy's many colliding tendrils introduce very large system solves during the most contact-rich steps colliding against the glass (Figure 2.22).

Performance. Comprehensive statistics on all simulations, models, parameters and performance are reported in Table 2.2 and in our Supplemental. For reference dynamics please see our supplemental videos.

Accuracy. User-facing parameters in IPC have three accuracies that can be specified: 1) dynamics accuracy (ϵ_d), defining how well dynamics are resolved; 2) geometric accuracy ($\hat{d}$), defining how close objects can come to touching; and 3) stiction accuracy (ϵ_v), defining how well static friction is resolved. All three provide users direct and intuitive control (with meaningful physical units) of the trade-off between accuracy and compute cost.

In our extensive testing, IPC converges to satisfy these requested accuracies while always maintaining an intersection- and inversion-free state. These guarantees (non-intersection, non-inversion) hold even as we radically increase speed of collision at large time step, apply extreme deformations, and model highly stiff materials. We have tested this across a wide range of test examples with material stiffnesses up to $E = 2 \times 10^{11}$ Pa and have confirmed our IPC implementation's ability to converge to tight tolerances for all these measures when requested with ϵ_d down to 10^{-7} m/s, ϵ_v down to 10^{-8} m/s, and $\hat{d}$ down to 1 μm.

As we discuss and demonstrate in Sections 2.3 and 2.8, all previously available methods introduce computational error for these accuracy measures; to our knowledge, IPC is the first to provide and expose direct and separable control of them. Our singular exception, as detailed in Section 2.5 above, is the number of frictional lagging iterations applied. When accurate friction is required, e.g., our arch, stick-slip and card house experiments, we set no upper bound on this parameter. Then as discussed above, in these examples IPC fully converges and is entirely parameter-free. However, (as detailed above) we do not have convergence guarantees for lagging, and in our large-deformation frictional examples we apply a single lagging iteration. In these cases, as discussed in Section 2.5.5, sliding directions and contact forces in the friction may not match. However, even in such cases all other guarantees, including non-intersection are maintained. We observe high-quality results regardless of number of lagging iterations applied or accuracies specified.

Finally, on the other end of the spectrum in many applications, e.g., animation, it can be desirable to trade accuracy for efficiency. We confirm robust, plausible behavior for IPC when we set very large, loose tolerances on all the above parameters, e.g., with $\epsilon_d = 10^{-1}$ m/s, while still maintaining feasible (non-inverting, non-intersecting) trajectory guarantees.

Exact CCD admissibility check. IPC's collision aware line search ensures intersection-free trajectories. Our implementation applies standard floating-point CCD[2] combined with the conservative advancement strategies detailed in Section 2.4 and our Supplemental to ensure efficient, intersection-free stepping. Exact CCD then offers the possibility for aggressive advancement of intersection-free steps and so improved efficiency. To this end we tested the robust CCD methods from both Bridson et al. [2002] and Tang et al. [2014] but found the reference implementations for each missed critical intersections in degeneracies. We then reimplemented Bridson et al. [2002] with rationals. While this version now guarantees exactness, it is much slower (~30×) than our floating-point implementation. Currently we apply this exact CCD just for re-analysis as a post step check after every Newton iterate to test three of our challenging contact stress tests: octocat on codimensional "knives", ball roller and mat twist. We confirm that every step taken in every time step was intersection free in these examples.

Varying material model. A general expectation from unconstrained simulation is that modeling with non-invertible materials like NH should be more costly than comparable set-ups with invertible materials like FCR. However, when studying our large deformation examples with contact we find that the picture is more complex. Here the larger bottleneck is generally resolving contact barrier terms. In many examples, we then observe that simulations with NH and FCR have comparable costs. In a number of other simulations with extreme contact conditions (e.g., pin-cushion and mat twist) element degeneracies allowed by FCR actually increase the overall cost of simulation well over the same simulations with the NH material. Finally, in other cases where stress is most extreme (e.g., armadillo roller and dolphin funnel), NH entails more cost than the comparable simulation with FCR.

[2] https://github.com/evouga/collisiondetection

2.8 Comparisons

We perform extensive quantitative comparisons with existing algorithms and commercial codes used in both computer graphics (Section 2.8.1) and mechanical engineering (Section 2.8.2). Then, to more fairly compare across a large class of previous contacts algorithms based on SQP-type methods, we implement their core contact resolution procedures in a single framework, and perform a large scale comparison on our benchmark test set (Section 2.8.3). While our implementations are not finely tuned as for the first two sets of comparisons, this approach allows us to compare the core algorithmic components in a common, objective and unbiased context.

2.8.1 Computer Graphics Comparisons

Contact algorithms in graphics often target performance with small compute budgets and so admirably face many efficiency challenges in balancing fidelity against speed. We investigate what happens if we push these methods' settings to be most accurate without regard to speed, e.g., max iteration caps of 1M per step and time steps down to 10^{-5} s. Here, nevertheless, we still document failures, e.g., tunneling, non-convergence, instabilities and ghost forces, even on very simple test examples.

Verschoor and Jalba [2019]. We apply the reference implementation of Verschoor and Jalba [2019] to reproduce available scenes with their default and reported input parameters. Here we observe that small adjustments to time step sizes and material parameters lead to divergent simulations. Specifically, the Armadillo roller example does not converge when applying the implementation's default time step of $h = 10^{-3}$ s for a range of stiffnesses of $E = 5 \times 10^4,\ 5 \times 10^5$, and 5×10^6 Pa, nor when applying the default material setting $E = 5 \times 10^5$ Pa for a range of time step sizes of $h = 10^{-3},\ 2 \times 10^{-3},\ 4 \times 10^{-3}$, and 10^{-2} s. In all these cases the implementation maxes out at its default max-iteration cap of 1M.

We extract the Armadillo mesh, roller models and replicate the same example in IPC with identical scene settings. Here it is noteworthy that IPC applies fully nonlinear NH and FCR models with variational friction while the reference code (matching paper) linearizes elasticity once per time step. As covered in Section 2.7.2, IPC obtains the stick-slip oscillations expected in this setting (see also our video), when rolling the Armadillo. This does not match the Verschoor and Jalba reference code nor paper video. Artificial material softening due to the per-step linearization of Verschoor and Jalba's elasticity likely explains the difference. We confirm this in Section 2.7.2 where IPC's fully nonlinear simulation of the Armadillo roller, with a softer $E = 10^5$ Pa (5× softer) does not, as expected, stick-slip.

SOFA. SOFA [Faure et al. 2012] is an open-source simulation framework featuring a range of physical models. These include deformable models via FEM. We modify a SOFA demo scene to simulate the five-link chain example with the top link fixed and four free FE links. We use the linear elasticity model (most robust) and found SOFA to provide a stable solution for the chain with large time steps up to $h = 10^{-2}$ s. We extend the chain to ten links and are unable to find a converging time step size (tested down to $h = 10^{-4}$ s). Please see our Supplemental for the full SOFA simulation settings.

Houdini. Houdini [SideFX 2023] is a widely used visual effects (VFX) tool that provides two performant simulation methods for deformable volumes: 1) a FE solver with co-rotated linear and neo-Hookean materials, and 2) Vellum, a state-of-the-art Position Based Dynamics (PBD) solver. While capable of producing impressive effects – especially for rapid collision denting and bouncing, we find that both solvers suffer in different ways when enforcing contact constraints accurately is critical. As a simple demonstration we again apply the chain example.

Trying a simple, lower-stiffness, 5-link chain we aim Houdini's FE solver towards robustness over speed by finely tetrahedralizing the link rings (~8000 tets per ring), applying small time steps (we tried increasing solver substeps to $h \approx 1$ ms), and increasing collision passes (up to 16). Up to

and including these maximum settings we observe rings tunneling through. We verify the same tunneling with both FE solvers provided in Houdini 18 (GNL,GSL), with both available materials. With similar stretchy material, IPC is able to accurately resolve the chain collisions even with a much coarser mesh (~500 tets per ring), and frame-rate size time steps, e.g., $h = 0.04$ s.

For the same 5-link scene, Houdini's Vellum PBD system does better, avoiding tunneling. However, as we increase numbers of links different tradeoffs (expected of PBD) are exposed. For example, a 35-link chain, requires collision passes and/or substeps to be increased quite high to prevent tunneling. However, this unavoidably changes the material (stiffer) and introduces biasing, in this case with sideways ghost forces. Careful experimentation with substep, smoothing, and constraint iteration parameters do not help alleviate these issues. For long chains (e.g., 100 links) we confirm IPC produces stable results, with accurate physical effects (e.g., shockwaves). See our supplemental videos.

2.8.2 Comparison with Engineering Codes

We compare IPC with two commercial engineering codes, COMSOL [COMSOL Inc. 2022] and ANSYS [Ansys, Inc. 2023], and one open-source engineering simulation framework [Krause and Zulian 2016]. For all three codes we set up exceedingly simple scenes involving small numbers of objects. All three methods generate intersection during simulation and exhibit instabilities highly dependent on parameters and tuning choices. In stark contrast to these three engineering solutions, IPC resolves a range of contact problems, demonstrates robust output across parameters, and ensures feasible trajectories. Please see our Supplemental for details on this comparison set.

2.8.3 Large Scale Benchmark Testing with SQP-type Methods

We focus on frictionless contact to compare a wide range of recently developed, implicit time-stepping algorithms. Removing the various and diverse treatments for friction allows us to carefully consider behavior with contact for a broad set of recent methods [Daviet et al. 2011; Harmon et al. 2008; Jean and Moreau 1992; Kane et al. 1999; Kaufman et al. 2008, 2014; Macklin et al. 2019; Otaduy et al. 2009; Verschoor and Jalba 2019] in a common test-harness framework. This is because all these methods, once friction is removed, follow a common iterated, Newton-type process to solve each time step as follows: 1) To help reduce constraint violation heuristic distance offsets/thickenings are applied to constraints; 2) at the start of each time step collision detection is performed to update a running estimate of active constraints; 3) The currently determined active (and possibly offset) constraint set and the IP energy are respectively approximated by first and second-order expansions; 4) The resulting quadratic energy is minimized subject to the linearized inequality constraints. This is a QP problem and so a bottleneck. A wide range of algorithms thus focus particularly on the efficient solution of this QP with custom approaches including QP, CR, LCP and nonsmooth-Newton strategies. Given the common sequential QP structure, we will jointly refer to them going forward as SQP-type. 5) A resulting displacement is then found and applied to the current iterate. This entire process is then repeated until a termination criteria is reached.

The above methods then differ in amount of offset, choice of constraint function, active set update strategy, IP approximations – most in graphics use just a fixed quadratic energy approximation (and so linearized elasticity) per time step, and choice of QP solver.

Here we focus on the ability of these methods to achieve convergent and accurate solves on a benchmark composed of our unit tests from Section 2.7.1 and a few additional low-resolution examples. To eliminate uncertainty of errors from the wide range of QP methods, we use the same state-of-the-art, albeit slow, QP solver Gurobi [Gurobi Optimization, LLC 2019] for all methods

and test each simulation method across a grid of variations on an high-performance computing (HPC) cluster.

We implement three common constraint types: the projected gap function, see e.g., Harmon et al. [2008]; the volume based proxy of Kane et al. [1999]; and the CCD-based gap function, see e.g., Otaduy et al. [2009] and Verschoor and Jalba [2019]. For each constraint type we test on a 3D sweep of (a) time steps (10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5} s), (b) constraint offsets (10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}), and (c) both fully nonlinear SQP and the graphics-standard of per time-step fixed quadratic approximation of the elastic energy with nonlinear constraints.

A general pattern appears in our results (entire output is provided in the Supplemental): for simulations to succeed all methods require small time step and/or large constraint offset. With large time steps accuracy of the constraint linearization diminishes, thus larger constraint offsets are necessary to compensate for constraint violations. A too large constraint offset leads to failures as the local QP may become infeasible. Additionally, with large constraint offset, a constraint pair may initially violate the constraints (a common-case for self-collision due to arbitrarily small distances between elements). While it is possible to recover from such initial constraint violations, this rarely happens in our experiments. In contrast, we (re-)confirm IPC is unconditionally robust across all test cases and time steps in the benchmark.

2.9 Discussion

In summary, IPC provides an exceedingly flexible, efficient, and unconditionally feasible solution for volumetric, mesh-based nonlinear elasticity simulations with self or external, volumetric or codimensional contacts. Guaranteeing intersection- and inversion-free output, IPC allows both computer graphics and engineering applications to run simulations by directly specifying just physically and geometrically meaningful parameters and tolerances as required per application.

At the same time much more remains to be done. While we have enabled a first of its

Table 2.2: Simulation statistics. for IPC on a subset of our benchmark examples. Complete benchmark statistics are summarized in our supplemental documents. For each simulation we report geometry, time step, materials, accuracies solved to ($\hat{d}$, ϵ_d and ϵ_v are generally set w.r.t. to bounding box diagonal length ℓ), number of contacts processed per time step, machine, memory, as well as average timing and number of Newton iterations per time step solve. When applicable, for friction we additionally report number of lagged iterations, with number of iterations set to ∞ indicating lagged iterations are applied until convergence until Equation (2.17) is satisfied. We apply implicit Euler time stepping and the neo-Hookean material by default unless specified in example name; i.e., "NM" for implicit Newmark time stepping and "FCR" for the fixed-corotational material model.

Example	nodes, tets, faces	h (s)	ρ (kg/m^3), E (Pa), ν	$\hat{d}$ (m)	μ, ϵ_v (m/s), max friction iterations	ϵ_d (m/s)	contacts avg. (max.) (per timestep)	machine	memory (MB)	timing (s), iterations (per timestep)
Ball on points	7K, 28K, 10K	0.04	1000, 10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	126 (182)	4-core 2.9 GHz Intel Core i7, 16 GB memory	229	2.8, 6.6
Mat on knives	3.2K, 9.1K, 6.4K	0.04	1000, 2×10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	291 (472)	4-core 2.9 GHz Intel Core i7, 16 GB memory	147	1.4, 5.5
100 chains	20K, 49K, 40K	0.04	500, 10^7, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	40K (53K)	4-core 2.9 GHz Intel Core i7, 16 GB memory	450	4.0, 2.4
Dolphin funnel	8K, 36K, 10K	0.04	1000, 10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	7K (31K)	4-core 2.9 GHz Intel Core i7, 16 GB memory	357	27.9, 39.7
Pin-cushion compress	9K, 28K, 10K	0.04	1000, 10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	317 (496)	4-core 2.9 GHz Intel Core i7, 16 GB memory	233	3.7, 9.5
Golf ball (NM)	29K, 118K, 38K	2×10^{-5}	1150, 10^7, 0.45	$10^{-3}\ell$	N/a	$10^{-2}\ell$	1K (4K)	4-core 2.9 GHz Intel Core i7, 16 GB memory	861	12.1, 9.3
Mat twist (100s)	45K, 133K, 90K	0.04	1000, 2×10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	264K (439K)	8-core 3.0 GHz Intel Xeon, 32 GB memory	4,546	776.2, 34.5
Rods twist (100s)	$53K$, $202K$, $80K$	0.025	1000, 10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	243K (498 K)	8-core 3.0 GHz Intel Xeon, 32 GB memory	2,638	141.5, 14.1
Trash compactor: ball, mat, and bunny	15K, 56K, 22K	0.01	1000, 10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	6K (132K)	8-core 3.0 GHz Intel Xeon, 32 GB memory	638	61.9, 29.4
Squeeze out	45K, 181K, 60K	0.01	1000, 5×10^4, 0.4	$10^{-3}\ell$	N/a	$10^{-2}\ell$	37K (277K)	8-core 3.0 GHz Intel Xeon, 32 GB memory	1,700	252, 42.5
Ball mesh roller	7K, 28K, 11K	0.01	1000, 10^4, 0.4	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	2.3K (5.6K)	4-core 2.9 GHz Intel Core i7, 16 GB memory	215	63.3, 58.6
Hit board house	6K, 15K, 11K	0.025	1000, 10^8, 0.4	$10^{-4}\ell$	1.0, $10^{-5}\ell$, 2	$10^{-2}\ell$	7K (13K)	4-core 2.9 GHz Intel Core i7, 16 GB memory	186	10.0, 16.6
Cement Arch	216, 150, 324	0.01	2300, 2×10^{10}, 0.2	10^{-6}	0.5, $10^{-5}\ell$, ∞	$10^{-4}\ell$	101 (118)	4-core 3.6 GHz Intel Core i7, 32 GB memory	54	0.05, 5.7
Stick-slip Armadillo roller (FCR)	67K, 386K, 24K	0.025	1000, 5×10^5, 0.2	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	8K (33K)	4-core 3.6 GHz Intel Core i7, 32 GB memory	3,651	346, 66.8
Squishy ball (AMGCL)	688K, 2314K, 1064K	10^{-3}	1000, 7×10^4, 0.4	$10^{-4}\ell$	N/a	$4\times10^{-2}\ell$	3.6K (105K)	8-core 3.6 GHz Intel Core i9, 64 GB memory	19,463	328.3, 12.2

kind "plug-and-play" contact simulation framework that provides convergent, intersection- and inversion-free simulation, clearly costs rise as scene complexity (both in contacts enforced and mesh resolutions) increase. There are thus many promising directions for future improvement that are exciting directions for exploration including further customized Newton-type methods, practical speed exact CCD, extensions to higher-order elements and improved convergence for frictional contact. We emphasize that we have no guarantee for convergence of lagged friction for λ and T (although we do for stiction) and so another meaningful avenue of future development is better exploration and understanding of its behavior.

Our hope is to enable engineers, designers, and artists to utilize predicative, expressive, and

differentiable simulation, free from having to perform extra per-scene algorithmic tuning or deviation from real-world physical parameters. We look forward to enabling design, machine learning, robotics, and other processes reliant on automated and reliable simulation output across parameter sweeps and iterations and hope to better enable artists to use real-world materials and settings as useful design tools for creative exploration.

3 A Large Scale Benchmark and an Inclusion-Based Algorithm for Continuous Collision Detection

3.1 Introduction

Collision detection and response are two separate, yet interconnected, problems in computer graphics and scientific computing. Collision detection specializes in finding when and if two objects collide, while collision response uses this information to deform the objects following physical laws. A large research effort has been invested in the latter problem, assuming that col-

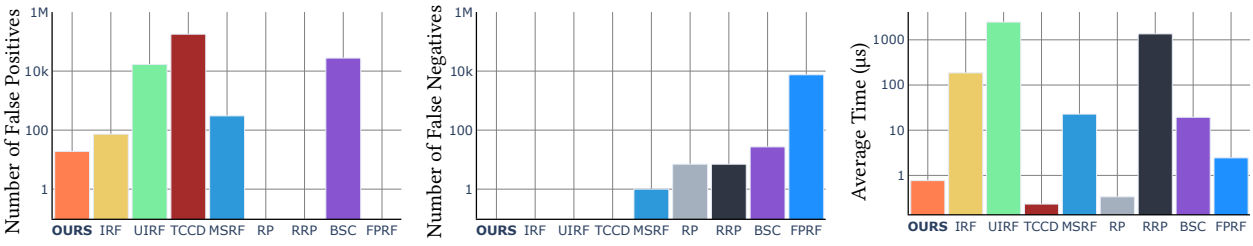


Figure 3.1: CCD benchmark overview. An overview of the results of our study of different CCD methods run on 60 million queries (both vertex-face and edge-edge). For each method, we show the number of false positives (i.e., the method detects a collision where there is none), the number of false negatives (i.e., the method misses a collision), and the average run time. Each plot reports results in a logarithmic scale. False positives and negatives are computed with respect to the ground truth computed using Mathematica [Wolfram Research Inc. 2020]. Acronyms are defined in Section 3.4.2.

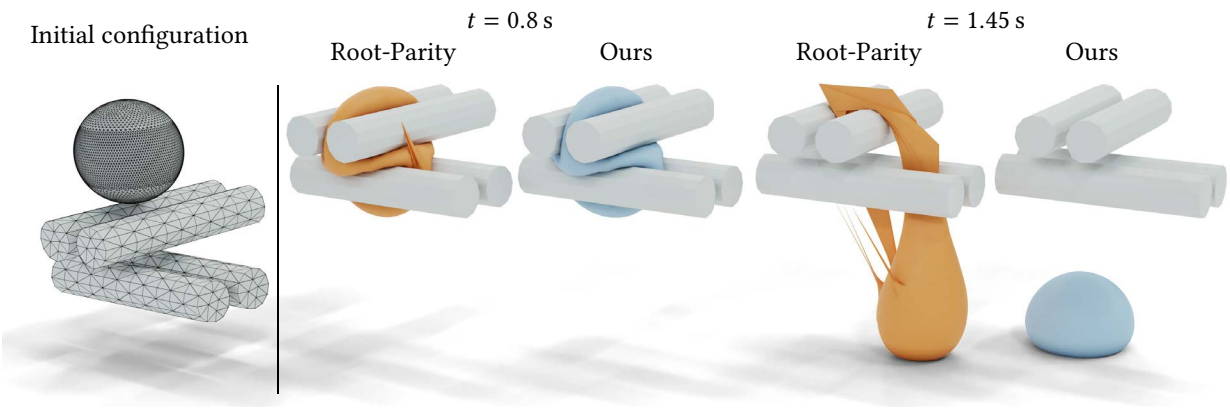


Figure 3.2: CCD failure. Inaccurate collision detection can lead to unnatural "sticking" and eventual failure when integrated into simulators (shown here using IPC [Li et al. 2020]) because part of the geometry gets stuck inside. Here we show a false negative reported by the Root-Parity method (orange) of Brochu et al. [2012] causes the ball to get stuck inside the rollers. Our conservative CCD (blue) never misses collisions and so the ball can pass through the rollers without problems.

lision detection can be solved reliably and efficiently. In this study we focus on the former, using an experimental approach based on large-scale testing. We use existing collision response methods to generate collision detection queries to investigate the pros and cons of existing collision detection algorithms.

Static collision detection is popular in interactive applications due to its efficiency, but its inability to detect collisions between fast-moving objects passing through each other (tunneling) hinders its applicability. To address this limitation, continuous collision detection (CCD) methods have been introduced: by solving a more computationally intensive problem, usually involving finding roots of a low-degree polynomial, these algorithms can detect any collision happening in a time step, often assuming linear trajectories.

The added robustness makes this family of algorithms popular, but they can still fail due to floating-point rounding errors. Floating point failures are of two types: false negatives (FNs), i.e., missed collisions, which lead to interpenetration, and false positives (FPs), i.e., detecting collisions when there are none.

Most collision response algorithms can tolerate minor imperfections, using heuristics to re-

cover from physically invalid states (in reality, objects cannot inter-penetrate). However, these heuristics have parameters that needs to be tuned for every scene to ensure stability and faithfulness in the simulation (as shown in Chapter 2). In Chapter 2, the collision response problem has been reformulated to avoid the use of heuristics, and the corresponding parameter tuning, by disallowing physically invalid configurations. For instance, in Figure 3.2, the method in Chapter 2 cannot recover from interpenetration after the CCD misses a collision leading to an unnatural "sticking" and eventual failure of the simulation. This comes with a heavier burden on the CCD algorithm used, which should never report FNs.

We introduce a large benchmark of CCD queries with ground truth computed using the *exact, symbolic solver* of Mathematica [Wolfram Research Inc. 2020], and evaluate the correctness (lack of FNs), conservativeness (FP count), and runtime efficiency of existing state-of-the-art algorithms. The benchmark is composed of both manually designed queries to identify degenerate cases (building upon [Erleben 2018]) and a large collection of real-world queries extracted from simulation sequences. On the algorithmic side, we select representative algorithms from the three main approaches existing in the literature for CCD root-finding: *inclusion-based bisection methods* [Redon et al. 2002a; Snyder et al. 1993], *numerical methods* [Vouga et al. 2011; Wang et al. 2015], and *exact methods* [Brochu et al. 2012; Tang et al. 2014]. Thanks to our benchmark, we identified missing cases that were not handled by previous methods, and we did a best effort to fix the corresponding algorithms and implementations to account for these cases.

The surprising conclusion of this study (Section 3.4.2) is that the majority of the existing CCD algorithms produce FNs, except three: (1) symbolic solution of the system and evaluation with exact arithmetic computed using Mathematica [Wolfram Research Inc. 2020], (2) Bernstein sign classification (BSC) with conservative error analysis [Wang et al. 2015], and (3) inclusion-based bisection root finding [Redon et al. 2002a; Snyder et al. 1993]. (1) is extremely expensive and, while it can be used for generating the ground truth, it is impractical in simulation applications. (2) is efficient but generates many FPs and the number of FPs depends on the geometric configuration

and velocities involved. (3) is one of the oldest methods proposed for CCD. It is slow compared to state-of-the-art algorithms, but it is correct and allows precise control of the trade-off between FPs and computational cost.

This extensive analysis and benchmark inspired us to introduce a specialization of the classical inclusion-based bisection algorithm proposed in [Snyder 1992] to the specific case of CCD for triangular meshes (Section 3.5). The major changes are: a novel inclusion function, an efficient strategy to perform bisection, and the ability to find CCD roots with a minimum separation (Section 3.6). Our novel inclusion function:

1. is tighter leading to smaller boxes on average thus making our method more accurate (i.e., less FPs);

2. reduces the root-finding problem into the iterative evaluation of a Boolean function, which allows replacing explicit interval arithmetic with a more efficient floating point filtering;

3. can be vectorized with Advanced Vector Extensions 2 (`AVX2`) instructions.

With these modifications, our inclusion-based bisection algorithm is only 3× slower on average than the fastest inaccurate CCD algorithm. At the same time it is provably conservative, provides a controllable ratio of FPs (within reasonable numerical limits), supports a minimum separation, and reports the time of impact (TOI). We also discuss how to integrate a minimum separation CCD in algorithms employing a line search to ensure the lack of intersections, which are common in locally injective mesh parametrization and have been as introduced in Chapter 2 for physical simulation.

Our dataset is available at the NYU Faculty Digital Archive[1], while the implementation of all the algorithms compared in the benchmark, a reference implementation of our novel inclusion-based bisection algorithm, and scripts to reproduce all results (Section 3.4) are available on our

[1]NYU Faculty Digital Archive: `https://archive.nyu.edu/handle/2451/61518`

project web page[2]. We believe this dataset will be an important element to support research in efficient and correct CCD algorithms, while our novel inclusion-based bisection algorithm is a practical solution that will allow researchers and practitioners to robustly check for collisions in applications where a 3× slowdown in the CCD (which is usually only one of the expensive steps of a simulation pipeline) will be preferable over the risk of FNs or the need to tune CCD parameters.

3.2 Related Work

We present a brief overview of the previous works on continuous collision detection for triangle meshes. Our work focuses only on CCD for deformable triangle meshes and we thus exclude discussing methods approximating collisions using proxies (e.g., Hubbard [1995]; Mirtich [1996]).

Inclusion-Based Root-Finding. The generic algorithm in the seminal work of Snyder [1992] on interval arithmetic for computer graphics is a conservative way to find collisions [Redon et al. 2002a; Snyder et al. 1993; Von Herzen et al. 1990]. This approach uses inclusion functions to certify the existence of roots within a domain, using a bisection partitioning strategy. Surprisingly, this approach is not used in recent algorithms despite being provably conservative and simple. Our algorithm is based on this approach, but with two major extensions to improve its efficiency (Section 3.5).

Numerical Root-Finding. The majority of CCD research focuses on efficient and accurate ways of computing roots of special cubic polynomials. Among these, a most popular cubic solver approach is introduced by Provot [1997], in which a cubic equation is solved to check for coplanarity, and then the overlapping occurrence is validated to determine whether a collision truely occurs. Refined constructions based on this idea have been introduced for rigid [Kim and

[2] our project web page: https://continuous-collision-detection.github.io/

Rossignac 2003; Redon et al. 2002a] and deformable [Hutter and Fuhrmann 2007; Tang et al. 2011b] bodies. However, all of these algorithms are based on floating-point arithmetic, requiring numerical thresholds to account for the unavoidable rounding errors in the iterative root-finding procedure. In fact, even if the cubic polynomial is represented exactly, its roots are generally irrational and thus not representable with floating-point numbers. Unfortunately, the numerical thresholds make these algorithms robust only for specific scenarios, and they can in general introduce false negatives. Our approach has a moderately higher runtime than these algorithms, but it is guaranteed to avoid false negatives without parameter tuning. We benchmark Provot [1997] using the implementation of Vouga et al. [2011] in Section 3.4.

For most applications, false positives are less problematic than false negatives since a false negative will miss a collision, leading to interpenetration and potentially breaking the simulation. Tang et al. [2010] propose a simple and effective filter which can reduce both the number of false positives and the elementary tests between the primitives. Wang [2014] and Wang et al. [2015] improve its reliability by introducing forward error analysis, in which error bounds for floating-point computation are used to eliminate false positives. We benchmark the representative method of Wang et al. [2015] in Section 3.4.

Exact Root-Finding. Brochu et al. [2012] and Tang et al. [2014] introduce algorithms relying on exact arithmetic to provide exact continuous collision detection. However, after experimenting with their implementations and carefully studying their algorithms, we discovered that they cannot always provide the exact answer (Section 3.4). Brochu et al. [2012] rephrase the collision problem as counting the number of intersections between a ray and the boundary of a subset of $\mathbb{R}^3$ bounded by bilinear faces. The ray casting and polygonal construction can be done using rational numbers (or more efficiently with floating point expansions) to avoid floating-point rounding errors. In [Tang et al. 2014] the CCD queries are reduced to the evaluation of the signs of Bernstein polynomials and algebraic expressions, using a custom root finding algorithm. Our algorithm

uses the geometric formulation proposed in [Brochu et al. 2012], but uses a bisection strategy instead of ray casting to find the roots. We benchmark both [Brochu et al. 2012] and [Tang et al. 2014] in Section 3.4.

Minimum Separation. Minimum separation CCD (MSCCD) [Harmon et al. 2011; Lu et al. 2019; Provot 1997; Stam 2009] reports collisions when two objects are at a (usually small) user-specified distance. These approaches have two main applications: (1) a minimum separation is useful in fabrication settings to ensure that the fabrication errors will not lead to penetrations, and (2) a minimum separation can ensure that, after floating-point rounding, two objects are still not intersecting, an invariant which must be preserved by certain simulation codes [Harmon et al. 2011; Li et al. 2020]. We benchmark [Harmon et al. 2011] in Section 3.6.2. Our algorithm supports a novel version of minimum separation, where we use the L^∞ norm instead of L^2 (Section 3.6.1).

Collision Culling. An orthogonal problem is efficient high-level collision culling to quickly filter out primitive pairs that do not collide in a time step. Since in this case it is tolerable to have many false positives, it is easy to find conservative approaches that are guaranteed to not discard potentially intersecting pairs [Curtis et al. 2008; Govindaraju et al. 2005; Mezger et al. 2003; Pabst et al. 2010; Provot 1997; Schvartzman et al. 2010; Tang et al. 2009a, 2008; Volino and Thalmann 1994; Wong and Baciu 2006; Zhang et al. 2007c; Zheng and James 2012]. Any of these approaches can be used as a preprocessing step to any of the CCD methods considered in this study to improve performance.

Generalized Trajectories. The linearization of trajectories commonly used in collision detection is a well-established, practical approximation, ubiquitous in existing codes. There are, however, methods that can directly detect collisions between objects following polynomial trajectories [Pan et al. 2012] or rigid motions [Canny 1986; Redon et al. 2002a; Tang et al. 2009b; Zhang et al. 2007c], and avoid the approximation errors due to the linearization. Our algorithm

currently does not support curved trajectories and we believe this is an important direction for future work.

3.3 Preliminaries and Notation

Assuming that the objects are represented using triangular meshes and that every vertex moves in a linear trajectory in each time step, the first collision between moving triangles can happen either when a vertex hits a triangle, or when an edge hits another edge.

Thus a continuous collision detection algorithm is a procedure that, given a vertex-face or edge-edge pair, equipped with their *linear trajectories*, determines if and when they will touch. Formally, for the vertex-face CCD, given a vertex p and a face with vertices v_1, v_2, v_3 at two distinct time steps t^0 and t^1 (we use the superscript notation to denote the time, i.e., p^0 is the position of p at t^0), the goal is to determine if at any point in time between t^0 and t^1 the vertex is contained in the moving face. Similarly, for the edge-edge CCD, the algorithm aims to find if there exists a $t \in [t^0, t^1]$ where the two moving edges (p_1^t, p_2^t) and (p_3^t, p_4^t) intersect. We will briefly overview and discuss the pros and cons of the two major formulations present in the literature to address the CCD problem: *multi-variate* and *univariate*.

Multivariate CCD Formulation. The most direct way of solving this problem is to parametrize the trajectories with a parameter $t \in [0, 1]$ (i.e., $p_i(t) = (1-t)p_i^0 + tp_i^1$ and $v_i(t) = (1-t)v_i^0 + tv_i^1$) and write a multivariate polynomial whose roots correspond to intersections. That is finding the roots of

$$F_{\text{vf}} \colon \Omega_{\text{vf}} = [0, 1] \times \{u, v \geq 0 | u + v \leq 1\} \rightarrow \mathbb{R}^3$$

with

$$F_{\text{vf}}(t, u, v) = p(t) - \big((1 - u - v)v_1(t) + uv_2(t) + vv_3(t)\big), \tag{3.1}$$

for the vertex-face case. Similarly, for the edge-edge case, the goal is to find the roots of

$$F_{\text{ee}} \colon \Omega_{\text{ee}} = [0,1] \times [0,1]^2 \longrightarrow \mathbb{R}^3$$

with

$$F_{\text{ee}}(t,u,v) = \big((1-u)p_1(t) + up_2(t)\big) - \big((1-v)p_3(t) + vp_4(t)\big). \tag{3.2}$$

In other words, the CCD problem reduces to determining if F has a root in Ω (i.e., there is a combination of valid t, u, v for which the vector between the point and the triangle is zero) [Brochu et al. 2012]. The main advantage of this formulation is that it is direct and purely algebraic: there are no degenerate or corner cases to handle. The intersection point is parameterized in time and local coordinates and the CCD problem reduces to multivariate root-finding. However, finding roots of a system of quadratic polynomials is difficult and expensive, which led to the introduction of the univariate formulation.

Univariate CCD Formulation. An alternative way of addressing the CCD problem is to rely on a *geometric* observation: two primitives intersect if the four points (i.e., one vertex and the three triangle's vertices or the two pairs of edge's endpoints) are coplanar [Provot 1997]. This observation has the major advantage of only depending on time, thus the problem becomes finding roots in a univariate cubic polynomial:

$$f(t) = \langle n(t), q(t) \rangle = 0, \tag{3.3}$$

with

$$n(t) = \big(v_2(t) - v_1(t)\big) \times \big(v_3(t) - v_1(t)\big) \text{ and } q(t) = p(t) - v_1(t)$$

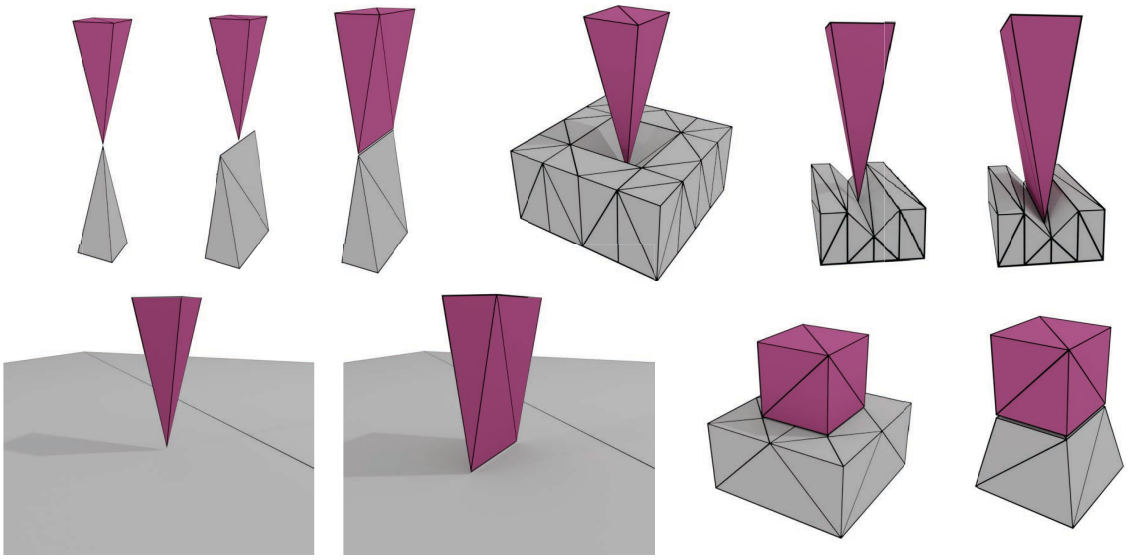


Figure 3.3: Scenes from Erleben [2018] that are used to generate a large part of the handcrafted dataset.

for the vertex-face case and

$$n(t) = \big(p_2(t) - p_1(t)\big) \times \big(p_4(t) - p_3(t)\big) \text{ and } q(t) = p_3(t) - p_1(t)$$

for the edge-edge case. Once the roots $t^\star$ of f are identified, they need to be filtered, as not all roots correspond to actual collisions. While filtering is straightforward when the roots are finite, special care is needed when there is an infinite number of roots, such as when the two primitives are moving on the same plane. Handling these cases, especially while accounting for floating point rounding, is very challenging.

3.4 Benchmark

3.4.1 Dataset

We crafted two datasets to compare the performance and correctness of CCD algorithms: (1) a *handcrafted* dataset that contains over 12 thousand point-triangle and 15 thousand edge-edge

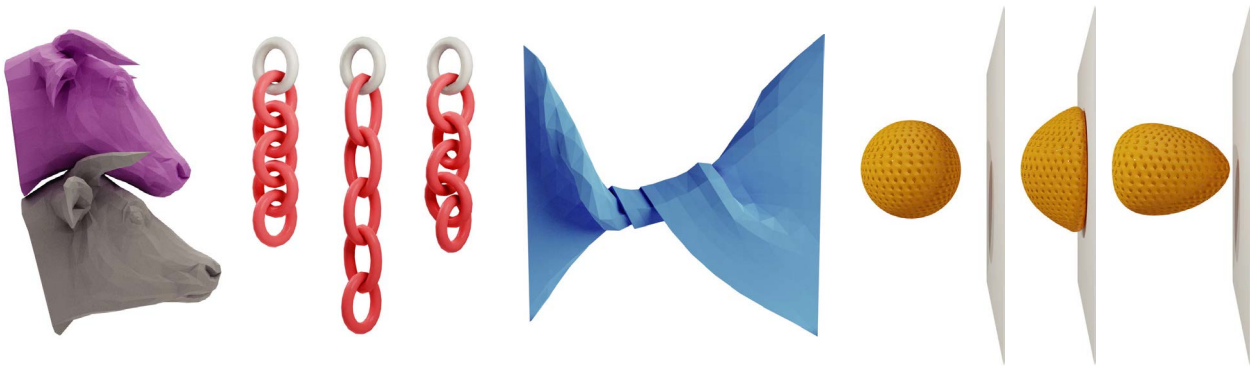


Figure 3.4: The scenes used to generate the simulation dataset of queries. We use two simulation methods: (top) a SQP method with constraints and active set update from Verschoor and Jalba [2019] and (bottom) the method proposed in Chapter 2.

queries, and (2) a *simulation* dataset that contains over 18 million point-triangle and 41 million edge-edge queries. To foster replicability, we describe the format of the dataset in Appendix E.1.

The handcrafted queries are the union of queries simulated with IPC (Chapter 2) from the scenes in [Erleben 2018] (Figure 3.3) and a set of handcrafted pairs for degenerate geometric configurations. These include: point-point degeneracies, near collisions (within a floating-point epsilon from collision), coplanar vertex-face and edge-edge motion (where the function f (Equation (3.3)) has infinite roots), degenerated function F_{vf} and F_{ee}, and CCD queries with two or three roots.

The simulation queries were generated by running four nonlinear elasticity simulations. The first two simulations (Figure 3.4 top row) use the constraints of [Verschoor and Jalba 2019] to simulate two cow heads colliding and a chain of rings falling. The second two simulations (Figure 3.4 bottom row) use the IPC method (Chapter 2) to simulate a coarse mat twisting and the high speed impact of a golf ball hitting a planar wall.

3.4.2 Comparison

We compare seven state-of-the-art methods: (1) the interval root-finder (IRF) [Snyder 1992], (2) the univariate interval root-finder (UIRF) (a special case of the rigid-body CCD from [Redon

Table 3.1: CCD benchmark. Summary of the average runtime in µs (t), number of false positive (FP), and number of false negative (FN) for the six competing methods.

Handcrafted Dataset (12K) – Vertex-Face CCD

	IRF	UIRF	FPRF	TCCD	RP	RRP	BSC	MSRF	Ours
t	14942.40	124242.00	2.18	0.38	1.41	928.08	176.17	12.90	1532.54
FP	87	146	9	903	3	0	11	16	108
FN	0	0	70	0	5	5	13	386	0

Handcrafted Dataset (15K) – Edge-Edge CCD

	IRF	UIRF	FPRF	TCCD	RP	RRP	BSC	MSRF	Ours
t	12452.60	18755.80	0.48	0.33	2.33	1271.32	121.80	2.72	3029.83
FP	141	268	5	404	3	0	28	14	214
FN	0	0	147	0	8	8	47	335	0

Simulation Dataset (18M) – Vertex-Face CCD

	IRF	UIRF	FPRF	TCCD	RP	RRP	BSC	MSRF	Ours
t	115.89	6191.98	7.53	0.24	0.25	1085.13	34.21	51.07	0.74
FP	2	18	0	95638	0	0	23015	75	2
FN	0	0	5184	0	0	0	0	0	0

Simulation Dataset (41M) – Edge-Edge CCD

	IRF	UIRF	FPRF	TCCD	RP	RRP	BSC	MSRF	Ours
t	215.80	846.57	0.23	0.23	0.37	1468.70	12.87	10.39	0.78
FP	71	16781	0	82277	0	0	4593	228	17
FN	0	0	2317	0	7	7	27	1	0

et al. 2002a]), (3) the floating-point time-of-impact root finder (FPRF) [Provot 1997] implemented in [Vouga et al. 2011], (4) TightCCD (TCCD) [Wang et al. 2015], (5) root parity (RP) [Brochu et al. 2012], (6) a rational implementation of Root Parity (RRP) with the degenerate cases properly handled, and (7) Bernstein sign classification (BSC) [Tang et al. 2014]. For each method we collect the average query time, the number of false positives (i.e., there is no collision but the method detects one), and the number of false negatives (i.e., there is a collision but the method misses it). To obtain the ground truth we solve the multivariate CCD formulation (Equations (3.1) and (3.2)) symbolically using Mathematica [Wolfram Research Inc. 2020] which takes multiple seconds per query. Table 3.1 summarizes the results. Note that "Ours" corresponds to our new method that will be introduced and discussed in Section 3.5 and minimum separation floating-point time-of-

impact root finder (MSRF) is a minimum separation CCD discussed in Section 3.6.2.

IRF. The inclusion-based root-finding described in [Snyder 1992] can be applied to both the multivariate and univariate CCD. For the multivariate case we can simply initialize the parameters of F (i.e., t, u, v) with the size of the domain Ω, evaluate F and check if the origin is contained in the output interval [Snyder et al. 1993]. If it is, we sequentially subdivide the parameters (thus shrinking the size of the intervals of F) until a user-tolerance δ is reached. In our comparison we use $\delta = 10^{-6}$. The major advantage of this approach is that it is guaranteed to be conservative: it is impossible to shrink the interval of F to zero. A second advantage is that a user can easily trade accuracy (number of false positives) for efficiency by simply increasing the tolerance δ (Appendix E.4). The main drawback is that bisecting Ω in the three dimensions makes the algorithm slow, and the use of interval arithmetic further increases the computational cost and prevents the use of certain compiler optimization techniques (such as instruction reordering). We implement this approach using the numerical type provided by the Boost interval library [Schling 2011].

UIRF. [Snyder 1992] can also be applied to the univariate function in Equation (3.3) by using the same subdivision technique on the single variable t (as in [Redon et al. 2002a] but for linear trajectories). The result of this step is an interval containing the earliest root in t which is then plugged inside a geometric predicate to check if the primitives intersect in that interval. While finding the roots with this approach might, at a first glance, seem easier than in the multivariate case and thus more efficient, this is not the case in our experiments. If the polynomial has infinite roots, this algorithm will have to refine the entire domain to the maximal allowed resolution, and check the validity of each interval, making it correct but very slow on degenerate cases (Appendix E.4). This results in a longer average runtime than its multivariate counterpart. Additionally, it is impossible to control the accuracy of the other two parameters (i.e., u, v), thus introducing more false positives.

FPRF. Vouga et al. [2011] aim to solve the univariate CCD problem using only floating-point computation. To mitigate false negatives, the method uses a numerical tolerance η (Appendix E.5) shows how η affects running time, the false positive, and negative). The major limitations are that the number of false positives cannot be directly controlled as it depends on the relative position of the input primitives and that false negatives can appear if the parameter is not tuned accordingly to the objects velocity and scale. Additionally, the reference implementation does not handle the edge-edge CCD when the two edges are parallel. This method is one of the fastest, which makes it a very popular choice in many simulation codes.

TCCD. TightCCD is a conservative floating-based implementation of Tang et al. [2014]. It uses the univariate formulation coupled with three inequality constraints (two for the edge-edge case) to ensure that the univariate root is a CCD root. The algorithm expresses the cubic polynomial f as a product and sum of three low order polynomials in Bernstein form. With this reformulation the CCD problem becomes checking if univariate Bernstein polynomials are positive, which can be done by checking some specific points. This algorithm is extremely fast but introduces many false positives which are impossible to control. In our benchmark, this is the only non-interval method without false negatives. The major limitation of this algorithm is that it *always* detects collision if the primitives are moving in the same plane, independently from their relative position.

RP AND RRP. These two methods use the multivariate formulation F (Equations (3.1) and (3.2)). The main idea is that the *parity* of the roots of F can be reduced to a ray casting problem. Let $\partial\Omega$ be the boundary of Ω, the algorithm shoots a ray from the origin and counts the parity of the intersection between the ray and $F(\partial\Omega)$ which corresponds to the parity of the roots of F. Parity is however insufficient for CCD: these algorithms cannot differentiate between zero roots (no collision) and two roots (collision), since they have the same parity. We note that this is a rare case happening only with sufficiently large time-steps and/or velocities: we found 13 (handcrafted

dataset) and 7 (simulation dataset) queries where these methods report a false negative.

We note that the algorithm described in [Brochu et al. 2012] (and its reference implementation) does not handle some degenerate cases leading to both false negatives and positives. For instance, in Appendix E.2, we show an example of a "hourglass" configuration where RP misses the collision, generating a false negative. To overcome this limitations and provide a fair comparison to these techniques, we implemented a naïve version of this algorithm that handles all the degenerate cases using rational numbers to simplify the coding (see the additional materials). We opted for this rational implementation since properly handling the degeneracies using floating-point requires designing custom higher precision predicates for all cases. The main advantage of this method is that it is exact (when the degenerate cases are handled) as it does not contain any tolerance and thus has zero false positives. We note that the runtime of our rational implementation is extremely high and not representative of the runtime of a proper floating point implementation of this algorithm.

BSC. This efficient and exact method uses the univariate formulation coupled with inequality constraints to ensure that the coplanar primitives intersects. The coplanarity problem reduces to checking if f in Bernstein form has a root. Tang et al. [2014] explain how this can be done *exactly* by classifying the signs of the four coefficients of the cubic Bernstein polynomial. The classification holds only if the cubic polynomial has monotone curvature; which can be achieved by splitting the curve at the inflection point. This splitting, however, cannot be computed exactly as it requires divisions (Appendix E.3). In our comparison, we modified the reference implementation to fix a minor typo in the code and to handle f with inflection points by *conservatively* reporting collision. This change introduces potential false positives, and we refer to the additional material for more details and for the patch we applied to the code.

Discussion and Conclusions. From our extensive benchmark of CCD algorithms, we observe that most algorithms using the univariate formulation have false negatives. While the reduction

to univariate root findings provides a performance boost, filtering the roots (without introducing false positives) is a challenging problem for which a robust solution is still elusive.

Surprisingly, only the oldest method, IRF, is at the same time reasonably efficient (e.g., it does not take multiple seconds per query as Mathematica), correct (i.e., no false negatives), and returns a small number of false positives (which can be controlled by changing the tolerance δ). It is however slower than other state-of-the-art methods, which is likely the reason why it is currently not widely used. In the next section we show that it is possible to change the inclusion function used by this algorithm to keep its favorable properties, while decreasing its runtime by ~250×, making its performance competitive with state-of-the-art methods.

3.5 Method

We describe the seminal bisection root-finding algorithm introduced in [Snyder 1992] (Section 3.5.1) and then introduce our novel Boolean inclusion function and how to evaluate it exactly and efficiently using floating point filters (Section 3.5.2).

3.5.1 Solve Algorithm [Snyder 1992]

An interval $i = [a, b]$ is defined as

$$i = [a, b] = \{x \mid a \leq x \leq b, x, a, b \in \mathbb{R}\},$$

and, similarly, an n-dimensional interval is defined as

$$I = i_1 \times \cdots \times i_n,$$

where i_k are intervals. We use $\mathcal{L}(i)$ and $\mathcal{R}(i)$ to refer to the left and right parts of an unidimensional interval i. The width of an interval, written as $w(i) = w([\mathcal{L}(i), \mathcal{R}(i)])$, is defined by

$$w(i) = \mathcal{R}(i) - \mathcal{L}(i)$$

and similarly, the width of an n-dimensional interval

$$w(I) = \max_{k=\{1,\ldots,n\}} w(i_k).$$

An interval can be used to define an inclusion function. Formally, given an m-dimensional interval D and a continuous function $g\colon \mathbb{R}^m \to \mathbb{R}^n$, an inclusion function for g, written $\square g$, is a function such that

$$\forall x \in D \quad g(x) \in \square g(D).$$

In other words, $\square g(D)$ is a n-dimensional interval bounding the range of g evaluated over an m-dimensional interval D bounding its domain. We call the inclusion function $\square g$ of a continuous function g convergent if for an interval X

$$w(X) \to 0 \implies w\big(\square g(X)\big) \to 0.$$

A *convergent* inclusion function can be used to find a root of a function g over a domain bounded by the interval $I_0 = [\mathcal{L}(x_1), \mathcal{R}(x_1)] \times \cdots \times [\mathcal{L}(x_m), \mathcal{R}(x_m)]$. To find the roots of g, we sequentially bisect the initial m-dimensional interval I_0, until it becomes sufficiently small (Algorithm 3.1). Figure 3.5 shows a 1D example (i.e., $g\colon \mathbb{R} \to \mathbb{R}$) of a bisection algorithm. The algorithm starts by initializing a stack S of intervals to be checked with I_0 (Line 3). At every level ℓ (Line 5), the algorithm retrieves an interval I from S and evaluates the inclusion function to obtain the interval I_g (Line 7). Then it checks if the root is included in I_g (Line 8). If not I can

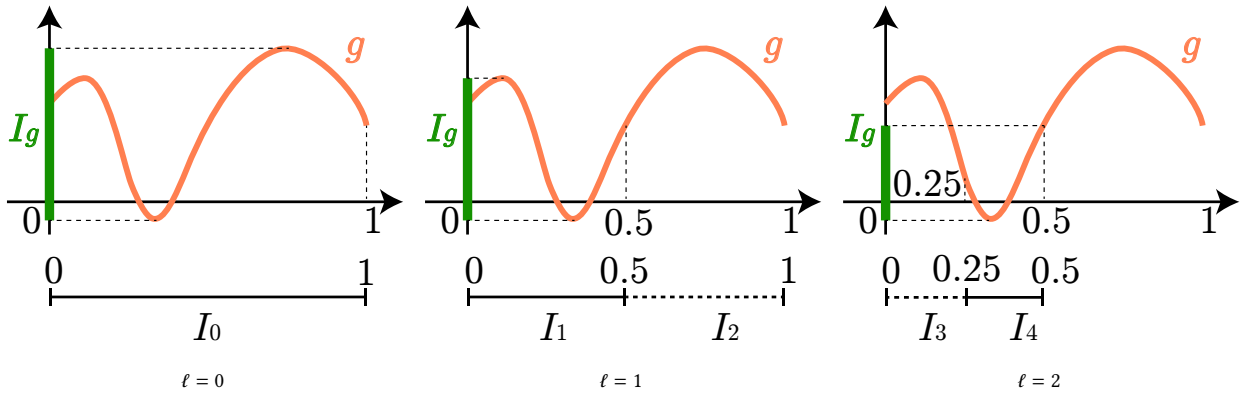


Figure 3.5: 1D illustration of the first three levels of the inclusion-based root-finder in [Snyder 1992].

Algorithm 3.1 Inclusion-based root-finder

```
1:  function SOLVE(I_0, g, δ)
2:      res ← ∅
3:      S ← {I_0}
4:      ℓ ← 0
5:      while S ≠ ∅ do
6:          I ← POP(S)
7:          I_g ← □g(I)                          ▷ Compute the inclusion function
8:          if 0 ∈ I_g then
9:              if w(I) < δ then                 ▷ I is small enough
10:                 res ← res ∪ I
11:             else
12:                 I_1, I_2 ← SPLIT(I)
13:                 S ← S ∪ {I_1, I_2}
14:         ℓ ← ℓ + 1
15:     return res
```

be safely discarded since I_g bounds the range of g over the domain bounded by I. Otherwise ($0 \in I_g$), it checks if $w(I)$ is smaller than a user-defined threshold δ. If so it appends I to the result (Line 10). If I is too large, the algorithm splits one of its dimensions (e.g., $[\mathcal{L}(x_1), \mathcal{R}(x_1)]$ is split in $[\mathcal{L}(x_1), \tilde{x}_1]$ and $[\tilde{x}_1, \mathcal{R}(x_1)]$ with $\tilde{x}_1 = (\mathcal{L}(x_1) + \mathcal{R}(x_1))/2$) and appends the two new intervals I_1, I_2 to the stack S (Line 13).

Generic Construction of Inclusion Functions. Snyder [1992] proposes the use of interval arithmetic as a universal and automatic way to build inclusion functions for arbitrary expressions.

However, interval arithmetic adds a performance overhead to the computation. For example, the product between two intervals is

$$[a, b] \cdot [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)],$$

which requires four multiplications and two min/max instead of one multiplication. In addition, the compiler cannot optimize composite expressions, since the rounding modes need to be correctly set up and the operation needs to be executed in order to avoid rounding errors [Schling 2011].

3.5.2 Predicate-Based Bisection Root Finding

Instead of using interval arithmetic to construct the inclusion function $\square F$ for the interval $I_\Omega = I_t \times I_u \times I_v = [0, 1] \times [0, 1] \times [0, 1]$ around the domain Ω, we propose to define an inclusion function tailored for F (both for Equations (3.1) and (3.2)) as the box

$$B_F(I_\Omega) = [m^x, M^x] \times [m^y, M^y] \times [m^z, M^z] \tag{3.4}$$

with

$$m^c = \min_{i=1,\dots,8} (v_i^c), \quad M^c = \max_{i=1,\dots,8} (v_i^c), \quad c = \{x, y, z\}$$

$$v_i = F(t_m, u_n, v_l), \quad t_m, u_n, v_l \in \{0, 1\}, \text{ and } \quad m, n, l \in \{1, 2\}.$$

Proposition 3.1. *The inclusion function B_F defined in Equation* (3.4) *is the* tightest axis-aligned inclusion function *of F.*

Proof. We note that for any given $\tilde{u}$ the function $F(t, \tilde{u}, v)$ is bilinear; we call this function function $F_{\tilde{u}}(t, v)$. Thus, F can be regarded as a bilinear function whose four control points move along linear trajectories $\mathcal{T}(u)_i, i = 1, 2, 3, 4$. The range of $F_{\tilde{u}}$ is a bilinear surface which is bounded by

the tetrahedron constructed by the four vertices forming the bilinear surface, which are moving on $\mathcal{T}_i$. Thus, F is bounded by every tetrahedron formed by $\mathcal{T}(u)_i$, implying that F is bounded by the convex hull of the trajectories' vertices, which are the vertices $v_i, i = 1, \cdots, 8$ defining F. Finally, since B_F is the axis-aligned bounding box of the convex-hull of $v_i, i = 1, \cdots, 8$, B_F is an inclusion function for F.

Since the vertices of the convex hull belong to F and the convex hull is the tightest convex hull, the bounding box B_F of the convex hull is the tightest inclusion function. □

Theorem 3.2. *The inclusion function B_F defined in Equation* (3.4) *is convergent.*

Proof. We first note that F is trivially continuous, second that the standard interval-based inclusion function $\square F$ constructed with intervals is axis-aligned. Therefore, from Proposition 3.1, it follows that $B_F(I) \subseteq \square F(I)$ for any interval I. Finally, since $\square F$ is convergent [Snyder 1992], then also B_F is. □

The inclusion function B_F turns out to be ideal for constructing a predicate: to use this inclusion function in the SOLVE algorithm (Algorithm 3.1), we only need to check if, for a given interval I, $B_F(I)$ contains the origin (Line 8). Such a Boolean predicate can be conservatively evaluated using floating point filtering.

CONSERVATIVE PREDICATE EVALUATION. Checking if the origin is contained in an axis-aligned box is trivial and it reduces to checking if the zero is contained in the three intervals defining the sides of the box. In our case, this requires us to evaluate the sign of F at the eight box corners. However, the vertices of the co-domain are computed using floating point arithmetic and can thus be inaccurate. We use forward error analysis to conservatively account for these errors as follows.

Without loss of generality, we focus only on the x-axis. Let $\{v_i^x\}, i = 1, \ldots, 8$ be the set of x-coordinates of the 8 vertices of the box represented in double precision floating-point numbers.

The error bound for F (on the x-axis) is

$$\begin{aligned} \varepsilon_{\text{ee}}^x &= 6.217248937900877 \times 10^{-15} \gamma_x^3 \\ \varepsilon_{\text{vf}}^x &= 6.661338147750939 \times 10^{-15} \gamma_x^3 \end{aligned} \tag{3.5}$$

with

$$\gamma_x = \max(x_{\max}, 1) \quad \text{and} \quad x_{\max} = \max_{i=1,\ldots,8} (|v_i^x|).$$

That is, the sign of F_{ee}^x computed using floating-point arithmetic is guaranteed to be correct if $|F_{\text{ee}}^x| > \varepsilon_{\text{ee}}^x$, and similarly for the vertex face case. If this condition does not hold, we conservatively assume that the zero is contained in the interval, thus leading to a possible false positive. The two constants $\varepsilon_{\text{ee}}^x$ and $\varepsilon_{\text{vf}}^x$ are floating point filters for F_{ee}^x and F_{vf}^x respectively, and were derived using [Attene 2020].

Efficient Evaluation. The x, y, z predicates defined above depend only on a subset of the coordinates of the eight corners of $B_F(I)$. We can optimally vectorize the evaluation of the eight corners using AVX2 instructions (~4× improvement in performance), since it needs to be evaluated on eight points and all the computation is standard floating-point arithmetic. Note that we used AVX2 instructions because newer versions still have spotty support on current processors. After the eight points are evaluated in parallel, applying the floating-point filter involves only a few comparisons. To further reduce computation, we check one axis at a time and immediately return if any of the intervals do not contain the origin.

Algorithm. We describe our complete algorithm in pseudocode in Algorithm 3.2. The input to our algorithm are the eight points representing two primitives (either vertex-face or edge-edge), a user-controlled numerical tolerance $\delta > 0$ (if not specified otherwise, in the experiment we use the default value $\delta = 10^{-6}$), and the maximum number of checks $m_I > 0$ (we use the default value $m_I = 10^6$). These choice are based on our empirical results (Figures 3.8 and 3.9). The output is

Algorithm 3.2 Complete overview of our CCD algorithm (Part 1 of 2).

```
1:  function SOLVE(F, δ, m_I)
2:      n ← 0                                          ▷ Number of check counter
3:      Q ← {{[0,1]^3, 0}}                             ▷ Push first interval and level 0 in Q
4:      ℓ_p ← −1                                       ▷ Previous checked level is -1
5:      while Q ≠ ∅ do
6:          I, ℓ ← POP(Q)                              ▷ Retrieve level and interval
7:          B ← B_F(I)                                 ▷ Compute the box inclusion function
8:          n ← n + 1                                  ▷ Increase check number
9:          if B ∩ C_ε ≠ ∅ then
10:             if ℓ ≠ ℓ_p then                        ▷ I is the first colliding interval of ℓ
11:                 I_f ← I_t                          ▷ Save t-component of I
12:             if n ≥ m_I then                        ▷ Reached max number of checks
13:                 return L(I_f), w(I_t)              ▷ Return left side of I_f
14:
15:             if w(B) < δ or B ⊆ C_ε then
16:                 if ℓ ≠ ℓ_p then
17:                     return L(I_f), w(I_t)          ▷ Root found
18:             else
19:                 I_1, I_2 ← SPLIT(I)
20:                 Q ← Q ∪ {{I_1, ℓ + 1}, {I_2, ℓ + 1}}
21:                 SORT(Q, ORDER)
22:             ℓ_p = ℓ                                ▷ Update the previous colliding level
23:     return ∞, 0                                    ▷ Q is empty and no roots were found
```

a conservative estimate of the earliest TOI or infinity if the two primitives do not collide in the time intervals coupled with the reached tolerance.

Our algorithm iteratively checks the box $B = B_F(I)$, with $I = I_t \times I_u \times I_v = [t_1, t_2] \times [u_1, u_2] \times [v_1, v_2] \subset I_\Omega$ (initialized with $[0,1]^3$). To guarantee a uniform box size while allowing early termination of the algorithm, we explore the space in a breadth-first manner and record the current explored level ℓ (Line 6). Since our algorithm is designed to find the earliest TOI, we sort the visiting queue Q with respect to time (Line 21).

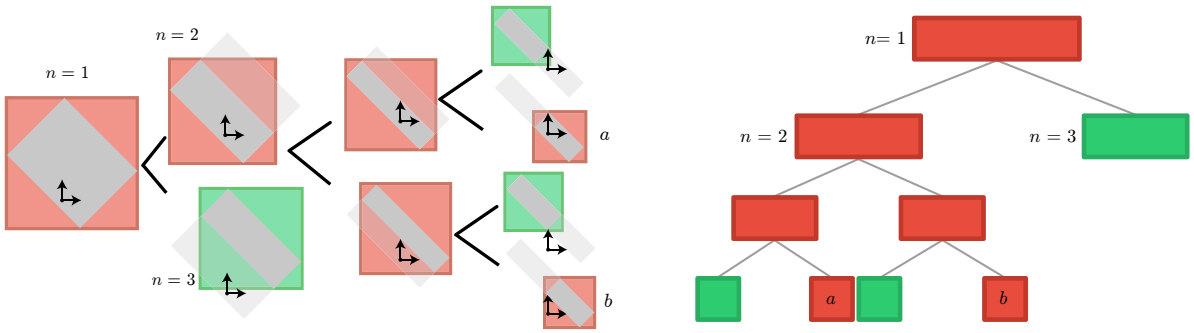
At every iteration we check if B intersects the cube $C_\varepsilon = [-\varepsilon^x, \varepsilon^x] \times [-\varepsilon^y, \varepsilon^y] \times [-\varepsilon^z, \varepsilon^z]$ (Line 9); if it does not, we can safely ignore I since there are no collisions.

Algorithm 3.3 Complete overview of our CCD algorithm (Part 2 of 2).

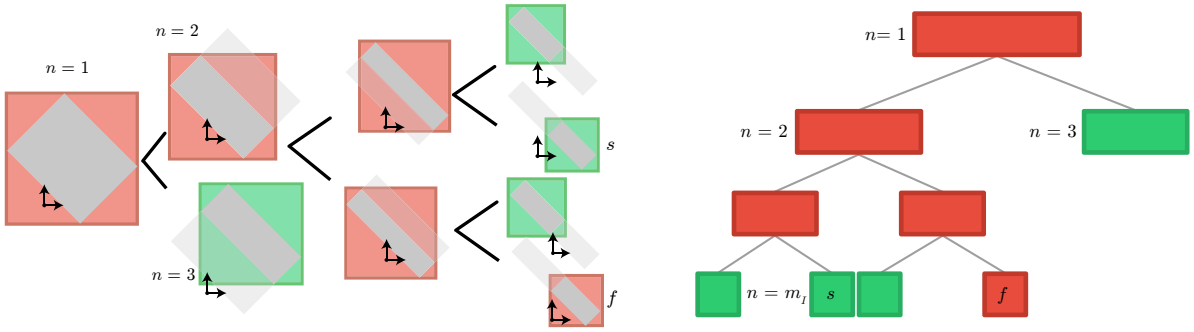
24: **function** SPLIT($I = I_t \times I_u \times I_v$)
25: Compute $\kappa_t, \kappa_u, \kappa_v$ according to Equation (3.7)
26: $c_t \leftarrow w(I_t)\kappa_t, \quad c_u \leftarrow w(I_u)\kappa_u, \quad c_v \leftarrow w(I_v)\kappa_v$
27: $c \leftarrow \max(c_t, c_u, c_v)$
28: **if** $c_t = c$ **then** ▷ c_t is the largest
29: $I_1 \leftarrow [\mathcal{L}(I_t), (\mathcal{L}(I_t) + \mathcal{R}(I_t))/2] \times I_u \times I_v$,
30: $I_2 \leftarrow [(\mathcal{L}(I_t) + \mathcal{R}(I_t))/2, \mathcal{R}(I_t)] \times I_u \times I_v$
31: **else if** $c_u = c$ **then** ▷ c_u is the largest
32: $I_1 \leftarrow I_t \times [\mathcal{L}(I_u), (\mathcal{L}(I_u) + \mathcal{R}(I_u))/2] \times I_v$,
33: $I_2 \leftarrow I_t \times [(\mathcal{L}(I_u) + \mathcal{R}(I_u))/2, \mathcal{R}(I_u)] \times I_v$
34: **else** ▷ c_v is the largest
35: $I_1 \leftarrow I_t \times I_u \times [\mathcal{L}(I_v), (\mathcal{L}(I_v) + \mathcal{R}(I_v))/2]$,
36: $I_2 \leftarrow I_t \times I_u \times [(\mathcal{L}(I_v) + \mathcal{R}(I_v))/2, \mathcal{R}(I_v)]$
37: **return** I_1, I_2
38:
39: **function** ORDER($\{I_1, \ell_1\}, \{I_2, \ell_2\}$)
40: **if** $\ell_1 = \ell_2$ **then**
41: **return** $I_1^t < I_2^t$
42: **else**
43: **return** $\ell_1 < \ell_2$

If $B \cap C_\varepsilon \neq \emptyset$, we first check if $w(B) < \delta$ or if B is contained inside the ε-box (Line 15). In this case, it is unnecessary to refine the interval I more since it is either already small enough (if $w(B) < \delta$) or any refinement will lead to collisions (if $B \subseteq C_\varepsilon$). We return I_t^l (i.e., the left-hand-side of the t interval of I) only if I was the first intersecting interval of this current level (Line 16). If I is not the first intersecting in the current level, there is an intersecting box (which is larger than δ) with an earlier time since the queue is sorted according to time (Figure 3.6(a)).

If B is too big we split the interval I in two sub-intervals and push them to the priority queue Q (Line 19). Note that, differently from Algorithm 3.1, we use a *priority queue* Q instead of the stack S. For the vertex-triangle CCD, the domain Ω is a prism, thus, after spitting the interval (Line 19), we append I_1, I_2 to Q only if they intersect with Ω. To ensure that B shrinks uniformly (since the termination criteria, Line 15, is $w(B) < \delta$) we *conservatively* estimate the width of B (in the codomain) from the widths of the domain's (i.e., where the algorithm is acting) intervals



(a) A small colliding (red) box b is not the earliest, since another box a exists at the same level (a did not trigger the termination of the algorithm since it is too big).



(b) Our algorithm stops when the number of checks n reaches m_I after checking the box s, which is a non-colliding box (green). The algorithm will return the first colliding box (f) of the same level, right.

Figure 3.6: A 2D example of root finding (left) and its corresponding diagram (right).

I_t, I_u, I_v:

$$\alpha > 0, w(I_t) < \frac{\alpha}{\kappa_t}, w(I_u) < \frac{\alpha}{\kappa_u}, w(I_v) < \frac{\alpha}{\kappa_v} \implies w(B_F(I)) < \alpha \tag{3.6}$$

with α a given constant and

$$\begin{aligned}
\kappa_t &= 3 \max_{i,j=1,2} \|F(0, u_i, v_j) - F(1, u_i, v_j)\|_\infty, \\
\kappa_u &= 3 \max_{i,j=1,2} \|F(t_i, 0, v_j) - F(t_i, 1, v_j)\|_\infty, \\
\kappa_v &= 3 \max_{i,j=1,2} \|F(t_i, u_j, 0) - F(t_i, u_j, 1)\|_\infty.
\end{aligned} \tag{3.7}$$

Proposition 3.3. *Equation* (3.6) *holds for any positive constant* α.

Proof. While $B_F(I)$ is an interval, for the purpose of the proof we equivalently define it as an

axis-aligned bounding box whose eight vertices are b_i. We will use the super-script notation to refer to the x, y, z component of a 3D point (e.g., b_i^x is the x-component of b_i) and define the set $\mathcal{I} = \{1, \ldots, 8\}$. By using the box definition the width of $B_F(I)$ can be written as

$$w(B_F(I)) = \|b_M - b_m\|_\infty$$

with

$$b_M^k = \max_{i \in \mathcal{I}}(b_i^k) \quad \text{and} \quad b_m^k = \min_{i \in \mathcal{I}}(b_i^k).$$

Since $B_F(I)$ is the tightest axis-aligned inclusion function (Proposition 3.1)

$$b_M^k \leq \max_{i \in \mathcal{I}} v_i^k, \quad b_m^k \leq \min_{i \in \mathcal{I}} v_i^k,$$

where $v_i = F(I_t^j, I_u^k, I_v^l)$, with $j, k, l \in \{l, r\}$, thus for any coordinate k we bound

$$b_M^k - b_m^k = \max_{i,j \in \mathcal{I}}(v_i^k - v_j^k) \leq \max_{i,j \in \mathcal{I}} \|v_i - v_j\|_\infty.$$

For any pair of v_i and v_j we have

$$v_i - v_j = s_1 \alpha_{l,m} + s_2 \beta_{n,p} + s_3 \gamma_{p,q},$$

for some indices $l, m, n, o, p, q \in \{1, 2\}$ and constant $s_1, s_2, s_3 \in \{-1, 0, 1\}$ with

$$\alpha_{i,j} = w(I_t)\big(F(0, u_i, v_j) - F(1, u_i, v_j)\big),$$

$$\beta_{i,j} = w(I_u)\big(F(t_i, 0, v_j) - F(t_i, 1, v_j)\big),$$

$$\gamma_{i,j} = w(I_v)\big(F(t_i, u_j, 0) - F(t_i, u_j, 1)\big),$$

since F is linear on the edges. We note that $\alpha_{i,j}, \beta_{i,j}$, and $\gamma_{i,j}$ are the 12 edges of the box B_F. We now define

$$e_t^k = \max_{i,j\in\{1,2\}} |\alpha_{i,j}^k|, \quad e_u^k = \max_{i,j\in\{1,2\}} |\beta_{i,j}^k|, \quad e_v^k = \max_{i,j\in\{1,2\}} |\gamma_{i,j}^k|$$

which allows us to bound

$$\max_{i,j\in\mathcal{I}} \|v_i - v_j\|_\infty \leq \|e_t + e_u + e_v\|_\infty \leq \|e_t\|_\infty + \|e_u\|_\infty + \|e_v\|_\infty.$$

Since

$$\|e_t\|_\infty \leq w(I_t) \max_{i,j=1,2} \|F(t_1, u_i, v_j) - F(t_2, u_i, v_j)\|_\infty = w(I_t)\kappa_t/3,$$

and similarly $\|e_u\|_\infty \leq \kappa_u/3, \|e_v\|_\infty \leq \kappa_v/3$, we have

$$\|e_t\|_\infty + \|e_u\|_\infty + \|e_v\|_\infty \leq \frac{w(I_t)\kappa_t + w(I_u)\kappa_u + w(I_v)\kappa_v}{3}$$

Finally, from the assumption in Equation (3.6) it follows that

$$w(B_F(I)) \leq \max_{i,j\in\mathcal{I}} \|v_i - v_j\|_\infty \leq \|e_t\|_\infty + \|e_u\|_\infty + \|e_v\|_\infty < \alpha.$$

□

Using the estimate of the width of I_t, I_u, I_v we split the dimension that leads to the largest estimated dimension in the range of F (Line 27).

Fixed Runtime or Fixed Accuracy. To ensure a bounded runtime, which is very useful in many simulation applications, we stop the algorithm after an user-controlled number of checks m_I. To ensure that our algorithm always returns a *conservative* TOI we record the first colliding interval I_f of every level (Line 11). When the maximum number of check is reached we can safely return the latest recorded interval I_f (Line 13) (Figure 3.6(b)). We note that our algorithm will not

respect the user specified accuracy when it terminates early: if a constant accuracy is required by applications, this additional termination criteria could be disabled, obtaining an algorithm with guaranteed accuracy but sacrificing the bound on the maximal running time. Note that without the termination criteria m_I, it is possible (while rare in our experiments) that the algorithm will take a long time to terminate, or run out of memory due to storing the potentially large list of candidate intervals L.

3.5.3 Results

Our algorithm is implemented in C++ and uses Eigen [Guennebaud et al. 2010a] for the linear algebra routines (with the `-avx2` `g++` flag). We run our experiments on a 2.35 GHz AMD EPYC™ 7452. We attach the reference implementation and the data used for our experiments, which will be released publicly.

The running time of our method is comparable to the floating-point methods, while being provably correct, for any choice of parameters. For this comparison we use a default tolerance $\delta = 10^{-6}$ and default number of iterations $m_I = 10^6$. All queries in the simulation dataset terminate within 10^6 checks, while for the handcrafted dataset only 0.25 and 0.55 % of the vertex-face and edge-edge queries required more than 10^6 checks, reaching an actual maximal tolerance δ of 2.14×10^{-5} and 6.41×10^{-5} for vertex-face and edge-edge respectively. We note that, despite the percentages begin small, by removing m_I the handcrafted queries take 0.015774 and 0.042477 *seconds* on average for vertex-face and edge-edge respectively. This is due to the large number of degenerate queries, as can be seen from the long tail in the histogram of the run-times (Figure 3.7). We did not observe any noticeable change of running time for the simulation dataset.

Our algorithm has two user-controlled parameters (δ and m_I) to control the accuracy and running time. The tolerance δ provides a direct control on the achieved accuracy and provides an indirect effect on the running time (Figure 3.8). The other parameter, m_I, directly controls the maximal running time of each query: for small m_I our algorithm will terminate earlier, resulting

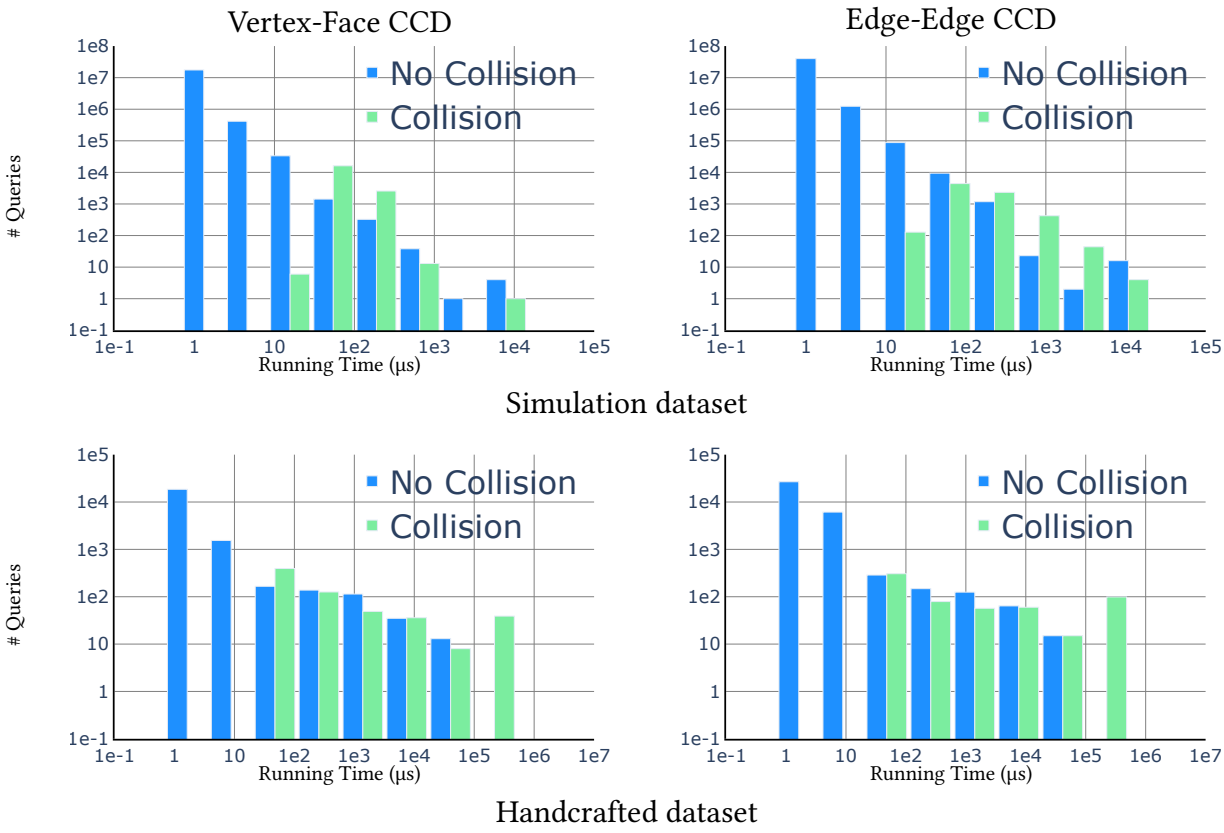


Figure 3.7: Log histograms of the running time of positive queries and negative queries on both datasets.

in a lower accuracy and thus more chances of false positives (Figure 3.9 top). We remark that, in practice, very few queries require so many subdivisions: by reducing m_I to the very low value of 100, our algorithm early-terminates only on ~0.07 % of the 60 million queries in the simulation dataset.

3.6 Minimum Separation CCD

An additional feature of some CCD algorithms is *minimum separation*, that is, the option to report collision at a controlled distance from an object, which is used to ensure that objects are never too close. This is useful to avoid possible inter-penetrations introduced by numerical rounding after the collision response, or for modeling fabrication tolerances for additive or subtractive manufacturing. A minimum separation CCD (MSCCD) query is similar to a standard

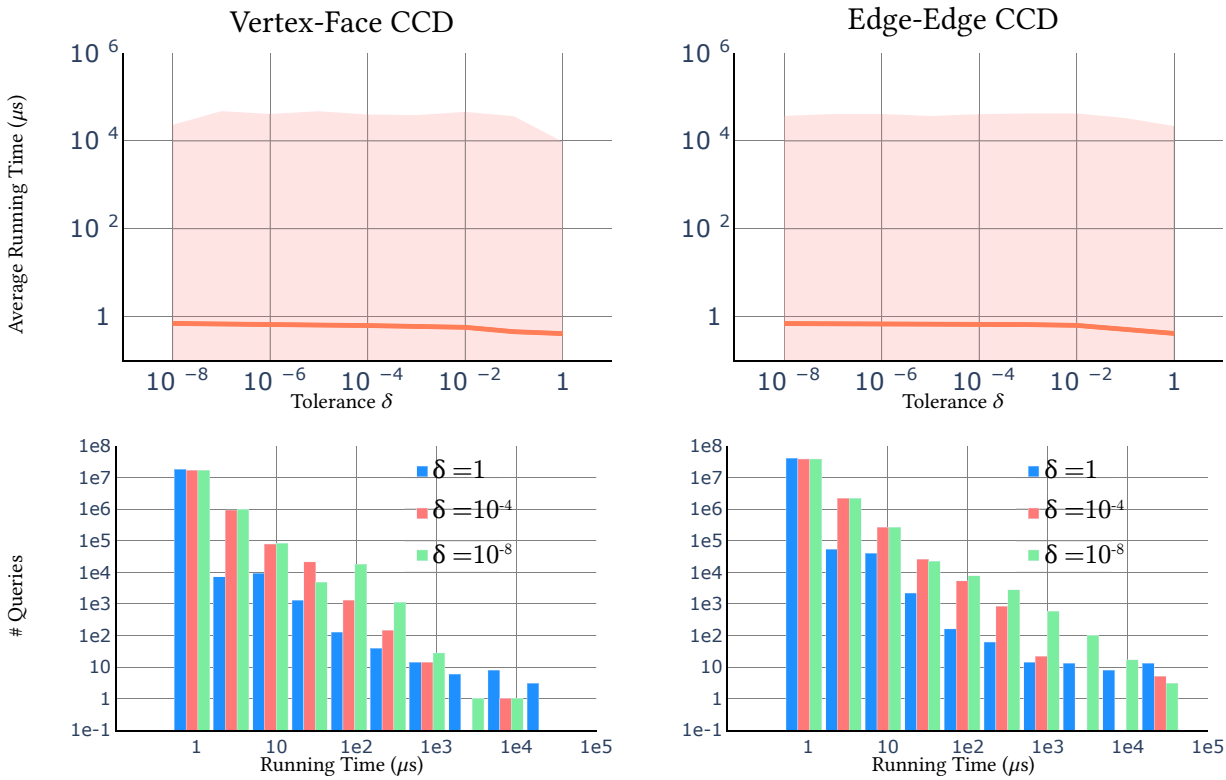


Figure 3.8: Top, average runtime of our algorithm for different tolerances δ for the simulation dataset. The shaded area shows the range of the distribution (min and max). Bottom, distribution of running times of our algorithm for three different tolerances $\delta = 10^{-8},\ 10^{-4}$, and 1 over the simulation dataset.

query: instead of checking if a point and a triangle (or two edges) are exactly overlapping, we want to ensure that they are always separated by a user-defined distance d during the entire linear trajectory. Similarly to the standard CCD (Section 3.3) MSCCD can be express using a multivariate or a univariate formulation, usually measuring distances using the Euclidean distance. We focus on the multivariate formulation since it does not require to filter spurious roots, we refer to Section 3.4.2 for a more detailed justification of this choice.

Multivariate Formulation. We observed that using the Euclidean distance leads to a challenging problem, which can be geometrically visualized as follows: the primitives will not be closer than d if $F(\Omega)$ does not intersect a sphere of radius d centered on the origin. This is a hard problem, since it requires checking conservatively the intersection between a sphere (which is a

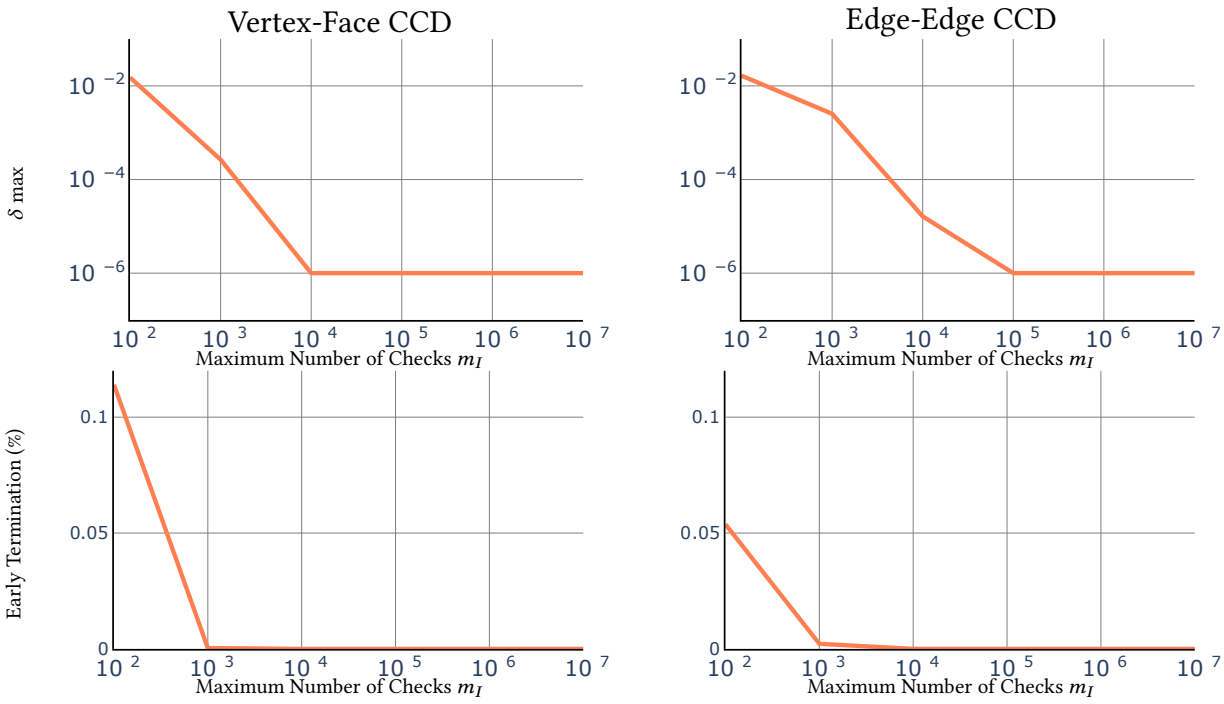


Figure 3.9: The percentage of early-termination and maximum value of the tolerance δ for different m_I for the simulation dataset.

rational polynomial when explicitly parametrized) and $F(\Omega)$.

Studying the applications currently using minimum separation, we realized that they are not affected by a different choice of the distance function. Therefore, we propose to change the distance definition from Euclidean to Chebyshev distance (i.e., from the L^2 to the L^∞ distance). With this minor change the problem dramatically simplifies: instead of solving for $F = 0$ (Section 3.5), we need to solve for $|F| \leq d$. The corresponding geometric problem becomes checking if $F(\Omega)$ intersects a cube of side $2d$ centered on the origin.

Univariate Formulation. The univariate formulation is more complex since it requires to redefine the notion of co-planarity for minimum separation. We remark that the function f in Equation (3.3) measures the length of the projection of $q(t)$ along the normal, thus to find point at distance d the equation becomes $f(t) \leq \langle n(t), q(t) \rangle = d\|n(t)\|$. To keep the equation polynomial, remove the inequality, and avoid square roots, the univariate MSCCD root finder

Table 3.2: MSCCD benchmark. Summary of the average runtime in µs (t), number of FPs, and number of FNs for MSRF and our method.

Handcrafted Dataset

	Vertex-Face MSCCD						Edge-Edge MSCCD					
	MSRF			Ours			MSRF			Ours		
d	t	FP	FN	t	FP	FN	t	FP	FN	t	FP	FN
10^{-2}	12.89	854	114	18.86K	2.6K	0	3.84	774	189	9.64K	4.8K	0
10^{-8}	15.05	216	2	1.60K	159	0	2.89	230	18	3.42K	309	0
10^{-16}	13.90	151	35	1.51K	108	0	2.90	231	21	2.92K	214	0
10^{-30}	13.59	87	141	1.39K	108	0	2.89	118	157	2.79K	214	0
10^{-100}	14.45	16	384	1.43K	108	0	3.05	14	335	2.82K	214	0

Simulation Dataset

	Vertex-Face MSCCD						Edge-Edge MSCCD					
	MSRF			Ours			MSRF			Ours		
d	t	FP	FN	t	FP	FN	t	FP	FN	t	FP	FN
10^{-2}	55.47	156.8K	18.3K	12.04	8.1M	0	14.42	354.1K	7.0K	19.12	8.3M	0
10^{-8}	55.26	75	0	0.72	8	0	11.12	228	1	0.73	40	0
10^{-16}	54.83	4	3.8K	0.71	2	0	10.70	10	4	0.72	17	0
10^{-30}	53.73	0	10.2K	0.66	2	0	10.68	0	1.7K	0.67	17	0
10^{-100}	53.53	0	18.6K	0.66	2	0	10.59	0	5.0K	0.68	17	0

becomes

$$\langle n(t), q(t) \rangle^2 - d^2 \|n(t)\|^2.$$

We note that this polynomial becomes sextic, and not cubic as in the zero-distance version. To account for replacing the inequality with an equality, we also need to check for distance between q and the edges and vertices of the triangle [Harmon et al. 2011]. In addition to finding the roots of several high-order polynomials, this formulation, similarly to the standard CCD, suffers from infinite roots when the two primitives are moving on a plane at distance d from each other.

3.6.1 Method

The input to our MSCCD algorithm are the same as the standard CCD (eight coordinates, δ, and m_I) and the minimum separation distance $d \geq 0$. Our algorithm returns the earliest TOI indicating if two primitives become closer than d as measured by the L^∞ norm.

We wish to check whether the box $B_F(\Omega)$ intersects a cube of side $2d$ centered on the ori-

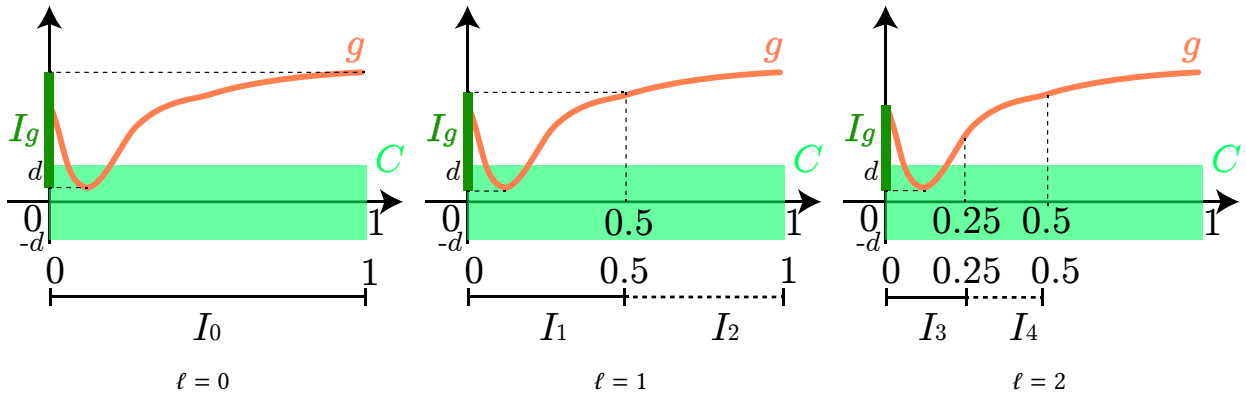


Figure 3.10: 1D illustration of the first three levels of our MSCCD inclusion-based root-finder. Instead of checking if I_g intersects with the origin, we check if it intersects the interval $[-d, d]$ marked in light green.

gin (Figure 3.10). Equivalently, we can construct another box $B'_F(\Omega)$ by displacing the six faces of $B_F(\Omega)$ outward at a distance d, and then check whether this enlarged box contains the origin. This check can be done as for the standard CCD (Section 3.5), but the floating point filters must be recalculated to account for the additional sum (indeed, we add/subtract d to/from all the coordinates). Hence, the filters for F' are:

$$\begin{aligned} \epsilon^x_{\text{ee}} &= 7.105427357601002 \times 10^{-15} \gamma_x^3 \\ \epsilon^x_{\text{vf}} &= 7.549516567451064 \times 10^{-15} \gamma_x^3 \end{aligned} \tag{3.8}$$

As before, the filters are calculated as described in [Attene 2020] and they additionally assume that $d < \gamma_x$.

To account for minimum separations, the only change in our algorithm is at Line 7 where we need to enlarge B by d and in Lines 9 and 15 since C_ε needs to be replaced with $C_\epsilon = [-\epsilon^x, \epsilon^x] \times [-\epsilon^y, \epsilon^y] \times [-\epsilon^z, \epsilon^z]$.

3.6.2 Results

To the best of our knowledge, the MSRF [Harmon et al. 2011] implemented in [Lu et al. 2019], is the only public code supporting minimum separation queries. While not explicitly constructed for MSCCD, FPRF uses a distance tolerance to limit false negatives, similarly to an explicit minimum separation. We compare the results and performance in Appendix E.5.

MSRF. uses the univariate formulation, which requires to find the roots of a high-order polynomial, and it is thus unstable when implemented using floating-point arithmetic.

Table 3.2 reports timings, false positive, and false negatives for different separation distances d. As d shrinks (around 10^{-16}) the results of our method with MSCDD coincide with the ones with $d = 0$ since the separation is small. For these small tolerances, MSRF runs into numerical problems and the number of false negatives increases. Figure 3.11 shows the average query time versus the separation distance d for the simulation dataset, since our method only requires to check the intersection between boxes, the running time largely depends on the number of detected collision, and the average is only mildly affected by the choice of d.

3.7 Integration in Existing Simulators

In a typical simulation the objects are represented using triangular meshes and the vertices are moving along a linear trajectory in a timestep. At each timestep, collisions might happen when a vertex hits a triangle, or when an edge hits another edge. A CCD algorithm is then used to prevent interpenetration; this can be done in different ways. In an active set construction method (Section 3.7.1) the CCD is used to compute contact forces to avoid penetration assuming linearized contact behaviour. For a line-search based method (Section 3.7.2), CCD and TOI are used to prevent the Newton trajectory from causing penetration by limiting the step length. Note that, the latter approach requires a conservative CCD, while the former can tolerate false negatives.

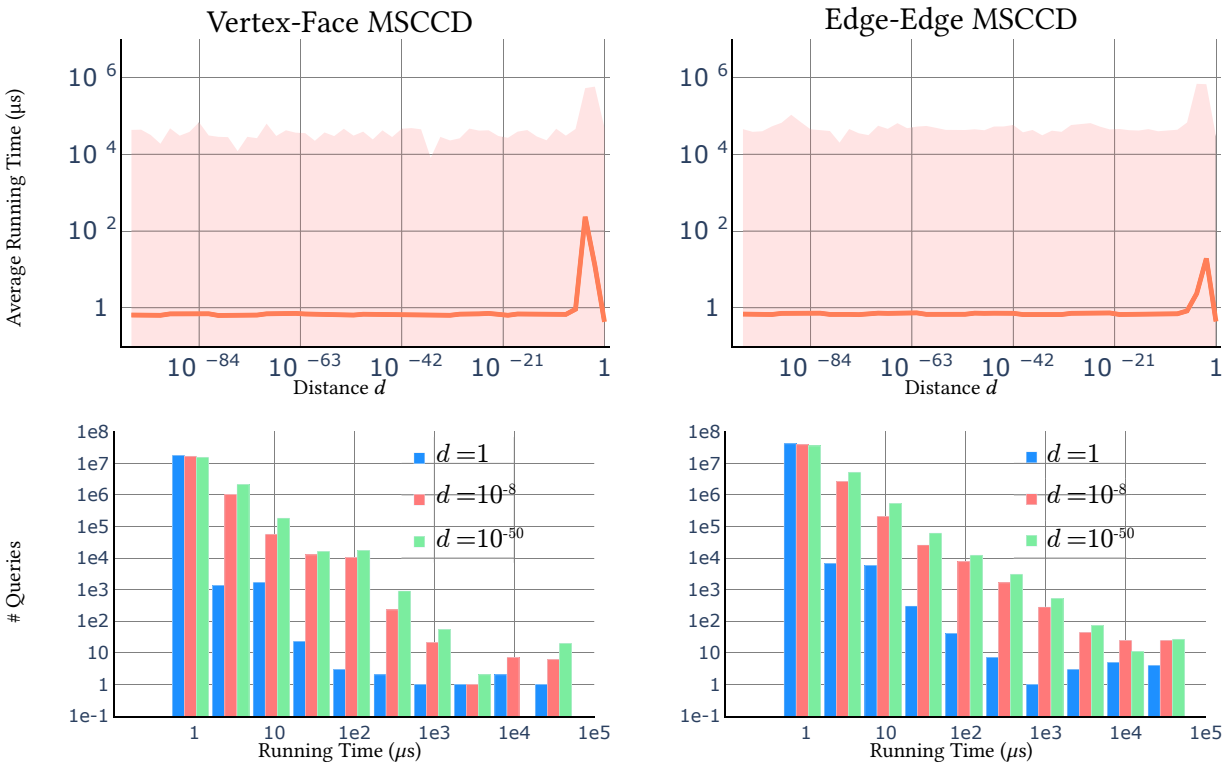


Figure 3.11: Top: average runtime of our algorithm for varying minimum separation d in the simulation dataset. The shaded area depicts the range of the values. Bottom: distribution of running time for three different minimum separation distances $d = 10^{-50}$, 10^{-8}, and 1 over the simulation dataset.

The integration of a CCD algorithm with collision response algorithms is a challenging problem on its own, which is beyond the scope of this paper. As a preliminary study, to show that our method can be integrated in existing response algorithm, we examine two use cases in elastodynamic simulations:

1. constructing an active set of collision constraints [Harmon et al. 2008; Verschoor and Jalba 2019; Wriggers 1995], Section 3.7.1;

2. during a line search to prevent intersections, Section 3.7.2.

We leave as future work a more comprehensive study including how to use our CCD to further improve the physical fidelity of existing simulators or how to deal with challenging cases such as sliding contact response.

To keep consistency across queries, we compute the numerical tolerances (Equations (3.5) and (3.8)) for the whole scene. That is, $x_{\max}$, $y_{\max}$, and $z_{\max}$ are computed as the maximum over all the vertices in the simulation. In Algorithms 3.4 and 3.5 we utilize a broad phase method (e.g., spatial hash) to reduce the number of candidates C that need to be evaluated without narrow phase CCD algorithm.

3.7.1 Active Set Construction

Algorithm 3.4 Active Set Construction Using Exact CCD

```
1:  function ConstructActiveSet(x_0, x_1, δ, m_I)
2:      C ← BroadPhase(x_0, x_1)
3:      C_A ← ∅
4:      for c ∈ C do                          ▷ Iterate over the collision candidates
5:          t ← CCD(x_0 ∩ c, x_1 ∩ c, δ, m_I)
6:          if 0 ≤ t ≤ 1 then
7:              C_A ← C_A ∪ {(c, t)}
8:      return C_A
9:
10: function CCD(c_0, c_1, δ, m_I)
11:     if c_0 and c_1 are edges then
12:         F ← build F_ee from c_0 and c_1          ▷ Equation (3.2)
13:     else
14:         F ← build F_vf from c_0 and c_1          ▷ Equation (3.1)
15:     return Solve(F, δ, m_I)
```

In the traditional constraint based collision handling (such as that of Verschoor and Jalba [2019]), collision response is handled by performing an implicit timestep as a constrained optimization. The goal is to minimize a elastic potential while avoiding interpenetration through gap constraints. To avoid handling all possible collisions during a simulation, a subset of *active* collisions constraints C_A is usually constructed. This set not only avoids infeasibilities, but also improves performance by having fewer constraints. There are many activation strategies, but for the sake of brevity we focus here on the strategies used by Verschoor and Jalba [2019].

Algorithm 3.4 shows how CCD is used to compute the active set C_A. Given the starting and ending vertex positions, x_0 and x_1, we compute the TOI for each collision candidate $c \in C$. We use the notation $x_i \cap c$ to indicate selecting the constrained vertices from x_i. If the candidate c is an actual collision, that is $0 \leq t \leq 1$, then we add this constraint and the TOI, t, to the active set, C_A.

From the active constraint set the constraints of Verschoor and Jalba [2019] are computed as

$$\langle n, p_c^1 - \boldsymbol{p}_c^2 \rangle \geq 0,$$

where n is the contact normal (i.e., for a point-triangle the triangle normal at the TOI and for edge-edge the edge-edge cross product at the TOI), p_c^1 is the point (or the contact point on the first edge), and p_c^2 is the point of contact on the triangle (or on the second edge) at the end of the timestep. Note that, this constraint requires to compute the point of contact, which depends on the the time-of-impact which can be obtained directly from our method.

Because of the difficulty for a simulation solver to maintain and not violate constraints, it is common to offset the constraints such that

$$\langle n, p_c^1 - p_c^2 \rangle \geq \eta > 0.$$

In such a way, even if the η constraint is violated, the real constraint is still satisfied. This common trick, implies that the constraints need to be activated early (i.e., when the distance between two objects is smaller than η) which is exactly what our MSCCD can compute when using $d = \eta$. In Figure 3.12, we use a value of $\eta = 0.001\,\text{m}$. When using large values of η, the constraint of Verschoor and Jalba [2019] can lead to infeasibilities because all triangles are extended to planes and edges to lines.

Figure 3.12 shows example of simulations run with different numerical tolerance δ. Changing δ has little effect on the simulation in terms of run-time, but for large values of δ, it can affect

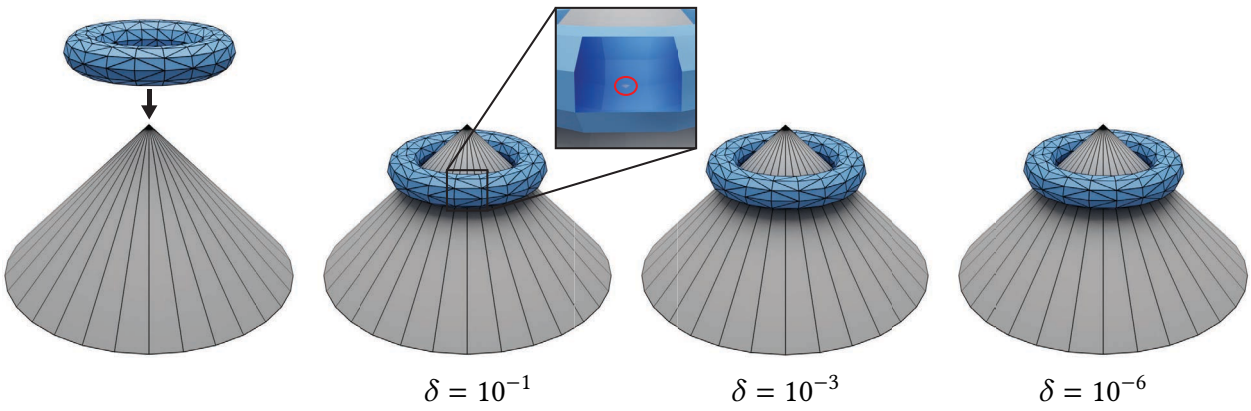


Figure 3.12: Active-set construction. An elastic simulation using the constraints and active set method of Verschoor and Jalba [2019]. From an initial configuration (left) we simulate an elastic torus falling on a fixed cone using three values of δ (from left to right: $10^{-1}, 10^{-3}, 10^{-6}$). The total runtime of the simulation is affected little by the change in δ (24.7, 25.2, and 26.2 seconds from left to right compared to 32.3 seconds when using FPRF). For $\delta = 10^{-1}$, inaccuracies in the time-of-impact lead to inaccurate contact points in the constraints and, ultimately, intersections (inset).

accuracy. We observe that for a $\delta \geq 10^{-2}$ the simulation is more likely to contain intersections. This is most likely due to the inaccuracies in the contact points used in the constraints.

3.7.2 Line Search

A line search is used in a optimization to ensure that every update decreases the energy E. That is, given an update, Δx, to the optimization variable x, we want to find a step size α such that $E(x + \alpha\Delta x) < E(x)$. This ensure that we make progress towards a minimum.

When used in a line search algorithm, CCD can be used to prevent intersections and tunneling. This requires modifying the maximum step length to the TOI. As observed in Chapter 2, the standard CCD formulation without a minimum separation cannot be used directly in a line search algorithm. Let $t^\star$ the earliest TOI (i.e., $F(t^\star, \tilde{u}, \tilde{v}) = 0$ for some $\tilde{u}, \tilde{v}$ and there is no collision between 0 and $t^\star$) and assume that the energy at $E(x_0 + t^\star\Delta x) < E(x_0)$ (Algorithm 3.5, Line 22). In this case the step $\alpha = t^\star$ is a valid descent step which will be used to update the position x in outer iteration (e.g., Newton optimization loop). In the next iteration, the line search will be called with the updated position and the earliest TOI will be zero since we selected $t^\star$ in the previous

Algorithm 3.5 Line Search with Exact CCD

1: **function** LineSearch($E, x_0, \Delta x, p, \delta, m_I$)
2: $x_1 \leftarrow x_0 + \Delta x$
3: $C \leftarrow$ BroadPhase(x_0, x_1) ▷ Collision candidates
4: $\alpha \leftarrow 1$
5: $d_i, \rho_i \leftarrow$ Distance(C)
6: Compute ϵ_i from Equation (3.8)
7: $d \leftarrow \max(pd_i, \delta)$
8: **while** $p < (d - \delta - \epsilon_i - \rho_i)/d$ **do**
9: $p \leftarrow p/2$
10: $d \leftarrow pd_i$
11: $\delta_i \leftarrow \delta$
12: **for** $c \in C$ **do** ▷ α is bounded by earliest time-of-impact
13: $t, \bar{\delta}_i \leftarrow \text{MSCCD}(x_0 \cap c, x_1 \cap c, d, \alpha, \delta, m_I)$
14: $\alpha \leftarrow \min(t, \alpha)$
15: $\delta_i \leftarrow \max(\bar{\delta}_i, \delta_i)$
16: **if** $p < (d - \delta_i - \epsilon_i - \rho_i)/d$ **then**
17: $\delta \leftarrow \delta_i$ ▷ Repeat with p validated from δ_i
18: Go to Line 8.
19:
20: **while** $\alpha > \alpha_{\min}$ **do** ▷ Backtracking line-search
21: $x_1 \leftarrow x_0 + \alpha\Delta x$
22: **if** $E(x_1) < E(x_0)$ **then** ▷ Objective energy decrease
23: **break**
24: $\alpha \leftarrow \alpha/2$
25: **return** α
26:
27: **function** MSCCD($c_0, c_1, d, t, \delta, m_I$)
28: **if** c_0 and c_1 are edges **then**
29: $F \leftarrow$ build F_{ee} from c_0 and c_1 ▷ Equation (3.2)
30: **else**
31: $F \leftarrow$ build F_{vf} from c_0 and c_1 ▷ Equation (3.1)
32: **return** SolveMSCCD(F, t, δ, m_I, d)

iteration. This prevents the optimization from making progress because any direction Δx will lead to a TOI $t = 0$. To avoid this problem we need the line search to find an appropriate step-size α along the update direction that leaves "sufficient space" for the next iteration, so that the barrier in Chapter 2 will be active and steer the optimization away from the contact position. Formally,

we aim at finding a *valid CCD sequence* $\{t_i\}$ such that

$$t_i < t_{i+1}, \quad \lim_{i \to \infty} t_i = t^\star, \quad \text{and} \quad t_i / t_{i+1} \approx 1.$$

The first requirement ensures that successive CCD checks will report an increasing time, the second one ensures that we will converge to the true minimum, and the last one aims at having a "slowly" convergent sequence (necessary for numerical stability). In Chapter 2, we exploit a feature of FPRF to simulate a minimum separation CCD: in this work we propose to directly use our MSCCD algorithm (Section 3.6).

Constructing a Sequence.. Let $0 < p < 1$ be a user-defined tolerance (p close to 1 will produce a sequence $\{t_i\}$ converging faster) and d_i be the distance between two primitives. We propose to set $d = pd_i$, and ensure that no primitive are closer than d. Without loss of generality, we assume that $F(x + \Delta x) = 0$, that is, taking the full step will lead to contact. By taking successive steps in the same direction, d_i will shrink to zero ensuring t_i to converge to $t^\star$. Similarly we will obtain a growing sequence t_i since d decreases as we proceed with the iterations. Finally, it is easy to see that $p = t_i / t_{i+1}$ which can be close to one.

To account for the aforementioned problem, we propose to use our MSCCD algorithm to return a valid CCD sequence when employed in a line search scenario. For a step i, we define δ^i as the tolerance, ϵ_i the numerical error Equation (3.8), and ρ_i as the maximum numerical error in computing the distances d_i from the candidates set C (Line 5). ρ_i should be computed using forward error analysis on the application-specific distance computation: since the applications are not the focus of our paper, we used a fixed $\rho_i = 10^{-9}$, and we leave the complete forward analysis as a future work. (We note that our approximation might thus introduce zero length steps, this however did not happen in our experiments.) If $d_i - (\delta_i + \epsilon_i + \rho_i) > d$, our MSCCD is guaranteed to find a TOI larger than zero. Thus if we set $d = pd_i$ (Line 7), we are guaranteed to

find a positive TOI if

$$d_i > \frac{\delta_i + \epsilon_i + \rho_i}{1 - p}.$$

To ensure that this inequality holds, we propose to validate p before using the MSCCD with δ (Line 8), find the TOI and the actual δ_i (Line 12), and check if the used p is valid (Line 16). In case p is too large, we divide it by two until it is small enough. Note that, it might be that

$$d_i < \delta_i + \epsilon_i + \rho_i,$$

in this case we can still enforce the inequality by increasing the number of iterations, decreasing δ, or using multi-precision in the MSCCD to reduce ϵ_i. However, this was never necessary in any of our queries, and we thus leave a proper study of these options as future work.

As visible from Table 3.2, our MSCCD slows down as d grows. Since the actual minimum distance is not relevant in the line search algorithm, our experiments suggest to cap it at δ (Line 7). To avoid unnecessary computations and speedup the MSCCD computations, our algorithm, as suggested by Redon et al. [2002a], can be easily modified to accept a shorter time interval (Line 13): it only requires to change the initialization of I (Algorithm 3.2 Line 3). These two modifications lead to a 8× speedup in our experiments. We refer to this algorithm with MSCCD (i.e., Algorithm 3.2 with MSCDD, Section 3.6.1, and modified initialization of I) as `SolveMSCCD`.

Figure 3.13 shows a simulation using our MSCCD in line search to keep the bodies from intersecting for different δ. As illustrated in the previous section, the effect of δ is negligible as long as $\delta \leq 10^{-3}$. Timings vary depending on the maximum number of iterations. Because the distance d varies throughout the simulation, some steps take longer than others (as seen in Figure 3.11). We note that, if we use the standard CCD formulation $F = 0$, the line search gets stuck in all our experiments, and we were not able to find a solution. Note that for a line search based method it is crucial to have a conservative CCD/MSCCD algorithm: the videos in the additional material shows that a false negative leads to an artifact in the simulation.

Figure 3.13: Collision-aware line-search. An example of an elastic simulation using our line search (Section 3.7.2) and the method in Chapter 2 to keep the bodies from intersecting. An octocat is falling under gravity onto a triangulated plane. From left to right: the initial configuration, the final frame with $\delta = 10^{-3}$, $\delta = 10^{-4.5}$, $\delta = 10^{-6}$ all with a maximum of 10^6 iterations. There are no noticeable differences in the results, and the entire simulations takes 63.3, 67.9, and 67.0 seconds from left to right (a speed up compared to using FPRF which takes 102 seconds).

3.8 Discussion

We constructed a benchmark of CCD queries and used it to study the properties of existing CCD algorithms. The study highlighted that the multivariate formulation is more amenable to robust implementations, as it avoids a challenging filtering of spurious roots. This formulation, paired with an interval root finder and modern predicate construction techniques leads to a novel simple, robust, and efficient algorithm, supporting minimum separation queries with runtime comparable to state-of-the-art, non-conservative, methods.

While we believe that it is practically acceptable, our algorithm still suffers from false positive and it will be interesting to see if the multivariate root finding could be done exactly with reasonable performances, for example employing expansion arithmetic in the predicates. Our definition of minimum separation distance is slightly different from the classical definition, and it would be interesting to study how to extend our method to directly support Euclidean distances. Another interesting venue for future work is the extension of our inclusion function to non-linear trajectories and their efficient evaluation using static filters or exact arithmetic.

Our benchmark focuses only on CPU implementations: reimplementing our algorithm on a graphics processing unit (GPU) with our current guarantees is a major challenge. It will require

to control the floating-point rounding on the GPU (and compliant with the Institute of Electrical and Electronics Engineers (IEEE) floating-point standard), to ensure that the compiler does not reorder the operations or skip the computation of temporaries. Additionally, it would require to recompute the ground truth and the numerical constants for single precision arithmetic, as most GPUs do not yet support double computation. This is an exciting direction for future work to further improve the performance of our approach.

We release an open-source reference implementation of our technique with an MIT license to foster the adoption of our technique by existing commercial and academic simulators. We also release the dataset and the code for all the algorithms in our benchmark to allow researchers working on CCD to easily compare the performance and correctness of future CCD algorithms.

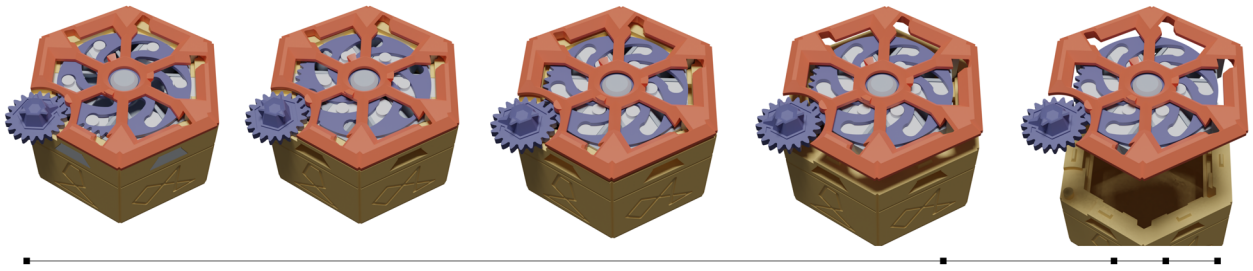
4 Intersection-free Rigid Body Dynamics

4.1 Introduction

Simulations of rigid objects with contact resolution and friction are ubiquitous in computer graphics and robotics. Rigid body models do not deform. Equipped with just rotational and translational degrees of freedom (DOF) they are a critical simplification enabling simulations with orders of magnitude less DOF when material deformation effects are either not significant or can be safely ignored.

An ideal rigid body simulator should take a scene description, initial conditions, and a set of (possibly time-dependent) boundary conditions, and integrate the system through time. This is unfortunately not the case with existing algorithms, which require extensive parameter tuning to produce sensible results (Section 4.6). In this work, we revisit the problem with a very different focus: automation and robustness. We propose an algorithm that does not require per-scene parameter tuning and can timestep large scenes with complex geometry, contacts, and friction interactions. Our algorithm is the first rigid body simulator that guarantees a lack of interpenetrations for all trajectories (and consequently on each timestep) of a simulation.

Our algorithm extends the IPC formulation (Chapter 2) for large deformation dynamics to rigid body dynamics. We rely on the same core ideas: model contacts via a set of barrier func-



Model ©Angus Deveson.

Figure 4.1: Expanding Lock Box. An intricate locking mechanism designed for 3D printing can be directly simulated with our algorithm. As the "key" turns, the central spiral is rotated which in turn pulls in each of the five locking pins. When all pins have been retracted the bottom is able to freely fall. Our algorithm's intersection-free guarantee enables the automatic testing of designs without the need to tune simulation parameters.

tions, and use an incremental potential formulation to timestep the system while ensuring no intersection at all intermediate stages of the computation. These ideas are extended to rigid body dynamics with reduced coordinates, where each body is parametrized by a rigid transformation. Our formulation supports large time steps, co-dimensional objects, and complex scenes with hundreds of inter-linked rigid bodies in resting or sliding contact. We compare our solution against the original IPC volumetric formulation (proxying the rigid bodies using a material with high Young's modulus) showing that our approach is, as expected, more efficient on large scenes due to the smaller number of degrees of freedom while being able to exactly model rigid motion.

As part of our algorithm, we need to conservatively detect collisions on a special type of curved trajectories obtained by linearly interpolating rigid motions in rotation vector representation.

We propose the first conservative broad and narrow phase solution for triangle-point and edge-edge collision detection queries for rigid body motion. The narrow phase query is based on a simple and effective observation: the problem can be reduced to a sequence of linear CCD queries with a minimum separation. For the broad phase, we propose to use interval arithmetic to compute conservative bounding boxes that can be used in a standard bounding volume hierarchy (BVH) data structure.

The resulting algorithm handles complex scenes that cannot be simulated with existing rigid body simulators, or that otherwise require laborious fine-tuning and hand-tweaking of simulation parameters to achieve, opening the doors to new applications in graphics, robotics, and fabrication. To quantitatively and qualitatively compare our algorithm with competing solutions, we introduce a benchmark for rigid body simulation, and compare our results against four popular simulators (Bullet [Coumans and Bai 2019], MuJoCo [Todorov et al. 2012], Chrono [Tasora et al. 2016], and Houdini's rigid body dynamics (RBD) [SideFX 2023]).

To foster future research and make our results reproducible, we attach a reference implementation of our algorithm, the benchmark, and scripts to reproduce all results in the paper in the additional material. This material will be released publicly as an open-source project.

Our main contributions are:

- An IPC formulation for rigid body dynamic;
- An efficient, provably conservative CCD query for curved trajectories;
- A benchmark for rigid body simulation.

4.2 Related Work

4.2.1 Rigid Body Simulation

Dating back to Euler the rigid body model is a fundamental primitive for physical modeling and simulation [Marsden and Ratiu 2013]. While it offers an exceedingly compact representation for body dynamics it comes with unique challenges as well. The first being that tracing a piecewise rigid trajectory is much more challenging than for a piecewise linear one. We cover the implications this has for integrating collision detection with time stepping in detail in Section 4.2.2. The second being that because rigid bodies are infinitely stiff, applied forces and contact

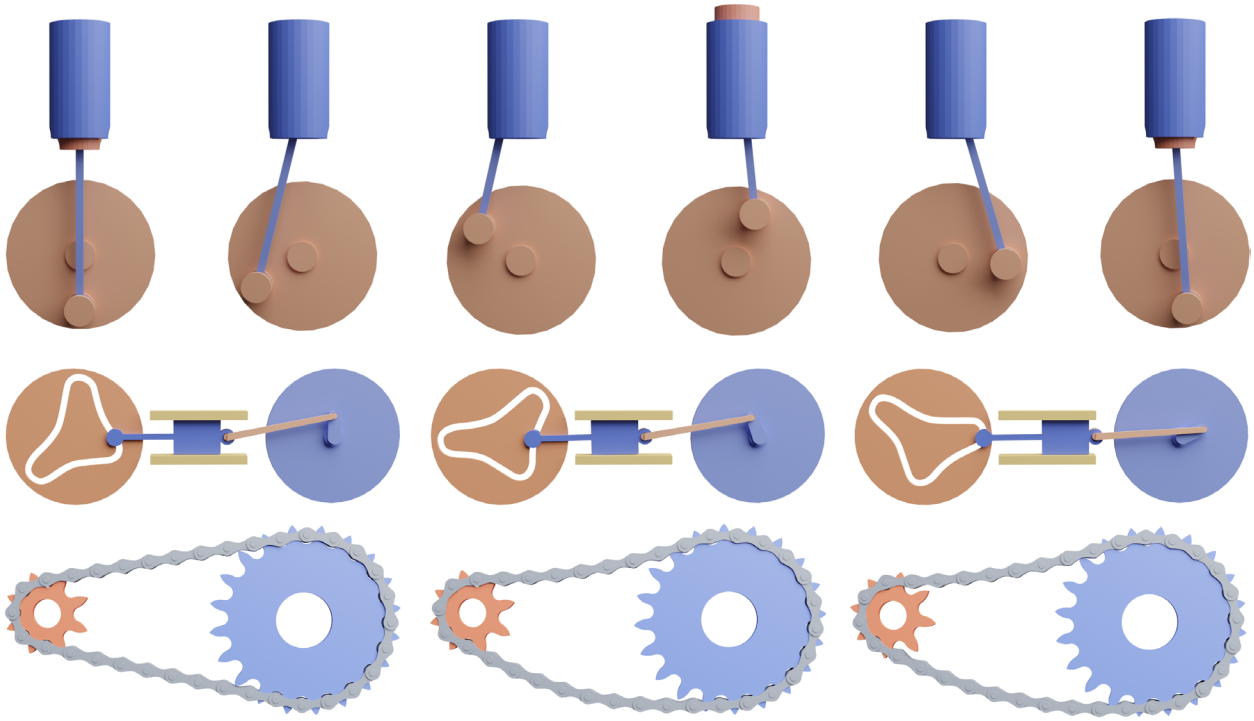


Figure 4.2: Mechanisms. We demonstrate the robustness of our method on various mechanisms with tight conforming contact. Top: a piston is attached to a rotating disk and a static cylinder is used to constrain the motion of the piston. Middle: A wheel with complex geometry rotates smoothly, but results in intermittent motion on the connected wheel. Bottom: a bike chain is attached to a kinematic sprocket. Each link is modeled using a realistic joint consisting of a roller, pin, and two plates.

responses are communicated instantaneously across the material domain. This sensitivity has long challenged the stability, accuracy, and effectiveness of time-stepping methods and friction models applied to simulate multibody systems [Stewart 2000].

Rigid-body contact simulation has been extensively investigated in mechanics, robotics, and graphics [Baraff 1989; Bender et al. 2012; Brogliato 1999; Hahn 1988; Mirtich and Canny 1995; Stewart 2000; Witkin and Baraff 2001]. In graphics, beginning with pioneering work of Baraff [1991] rigid body contact has especially focused on LCP models [Anitescu and Hart 2004b; Anitescu and Potra 1997; Baraff 1994; Kaufman et al. 2008; Lötstedt 1982; Stewart and Trinkle 2000; Trinkle et al. 1995]. Here the semi-implicit models employed enforce contact constraints at the velocity level. This linearized constraint enforcement then results in constraint drift and tunneling.

In turn, these artifacts can be partially mitigated by constraint stabilization methods [Anitescu and Hart 2004a; Cline and Pai 2003; Erleben 2007; Moreau 1988] at the cost of physical accuracy.

LCP and related contact models can also equivalently be formulated *variationally* [Moreau 1966; Redon et al. 2002b] and are amenable to both primal and dual constructions [Macklin et al. 2020]. However, as these rely on velocity level arguments and linearized contact constraints they can not be employed for IPC-based optimization. Here, to extend IPC to rigid coordinates, we construct an incremental potential for rigid bodies based directly on positions and rotations rather than velocities.

Focusing on efficiency and speed a wide range of faster, iterative methods for rigid bodies have also been developed building off of LCP [Erleben 2007; Guendelman et al. 2003], proximal [Erleben 2017], gradient descent [Mazhar et al. 2015], and decomposition [Coevoet et al. 2020; Hsu and Keyser 2010; Tonge et al. 2012] methods to name just a few. With speed, however, comes additional accuracy trade-offs [Kaufman et al. 2008]. In turn, this inherent loss of accuracy and the resultant impact on stability and robustness generally requires compensation in the form of hand-tuning and often large amounts of non-physical constraint stabilization.

A potential benefit of our work, which we leave as future work, is the easy coupling of the original IPC formulation for deformable bodies with our new IPC formulation for rigid bodies. Similar joint formulations have been introduced, for example, Müller et al. [2020] simulate rigid bodies through extended position-based dynamics allowing them to easily couple soft and rigid bodies.

There is also a rich history of simulating rigid bodies with guarantees. Time integration methods, starting with Moser and Veselov's [1991] celebrated work, focus on preserving geometric invariants of free rigid bodies [Hairer et al. 2006]. Recent complementary work [Smith et al. 2012; Vouga et al. 2017] focuses on designing methods for preserving geometric invariants (energy and momentum) as well desirable collision properties for contacting rigid bodies. For maintaining intersection-free rigid-body trajectories Mirtich [2000, 1996] and Snyder et al. [1993] construct

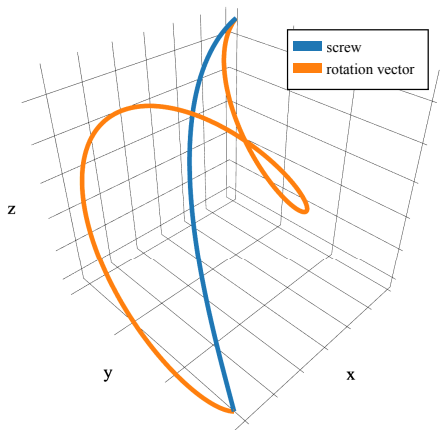


Figure 4.3: Trajectories of interpolating rotation vectors can be wildly different form the traditional screw motion used by others.

conservative, explicit time-stepping methods. Mirtich [1996] explicitly forward steps rigid bodies with conservative advancement to the time of contact and later extends intersection-free resolution with efficient roll-backs [Mirtich 2000]. Snyder et al. [1993] applies interval analysis to detect collision between bodies. Further discussion and comparisons to our CCD are provided in Section 4.2.2 and Appendix D.2. The use of an explicit time-stepping scheme can extremely limit step size (and so progress) as each collision must be detected and resolved before the simulation can proceed. In comparison, our method is fully implicit enabling large time-steps and global analysis of all collisions in a time step simultaneously.

4.2.2 Collision Detection

We restrict our overview to CCD algorithms for curved trajectories, as we are interested in rigid motions, and to CCD algorithms for linear trajectories with a minimum separation, as our algorithm needs to tackle this subproblem. We refer to Wang et al. [2021] for an overview of CCD methods for linear trajectories without a minimum separation.

Curved. There has been extensive research on curved CCD algorithms, both in graphics and in robotics. The trajectories considered are interpolation of rotation matrices, screw motions,

and spline curves. We are not aware of any method designed to handle the trajectories obtained interpolating rotation vectors that we consider in this paper.

There are two major approaches: interval-based root-finding on a system on non-linear equations and conservative advancement.

Interval-Based Root-Finding. One of the first approaches was introduced in [Snyder 1992; Snyder et al. 1993], where they propose to use an interval-based root finder to conservatively detect if there are collisions and at which time. The approach is robust but slow, as it heavily relies on interval arithmetic. To reduce the dimensions in the domain, and correspondingly improve performances, Redon et al. [2002a] proposes to use a similar strategy to only a part of the problem and rewriting the CCD problem as a univariate system. However, this approach leads to an infinite number of roots in degenerate cases, which dramatically slow down certain queries [Wang et al. 2021]. A similar formulation, but for trajectories obtained by interpolating quaternions is introduced in [Canny 1986]. We provide an explicit comparison against these approaches for both the multivariate and univariate formulations in Appendix D.2.

Conservative Advancement. The most popular family of methods is conservative advancement, which iteratively builds conservative convex proxies for a substep of the trajectory [Mirtich 2000, 1996]. These methods have been proposed for spline trajectories [Pan et al. 2012], trajectories with constant rotational and linear velocities [Tang et al. 2009b], screw motion [Tang et al. 2011a]. Different primitives are used such as bounding boxes or spheres [Schwarzer et al. 2005]. While most methods can be applied only to convex primitives, there are extensions for nonconvex polyhedra [Zhang et al. 2006]. In Zhang et al. [2007c], conservative advancement is extended to articulated bodies, with a novel technique based on Taylor expansion to compute tight approximations even for long body chains. A useful tool for computing the conservative proxies is the computation of distances between polyhedra. Specialized methods for rigid body motions are introduced in [Zhang et al. 2007a,b] and used within a conservative advancement framework to

design a CCD algorithm.

None of these techniques can directly handle the trajectories that we consider in our work, obtained by interpolating rotation vectors.

Other Methods. In addition to the above classifications, van Waveren [2005] introduces a unique method for handling rotational contacts between polyhedral features. By using Plücker coordinates and accounting for errors in floating-point rounding, van Waveren [2005] is able to robustly detect and respond to collision in real-time applications. Unfortunately, this method is limited to screw motions and is not immediately applicable to our current framework (interpolation of rotation vectors).

Numerical Accuracy. Snyder [1992] and Snyder et al. [1993] consider the problem of floating-point rounding, and can thus ensure a correct result when a floating-point implementation is used. Other methods are non-conservative when implemented using floating-point arithmetic. Since any missed collision would be fatal in our setting as it will break our interpenetration-free invariance, the only method that we can use is [Snyder 1992; Snyder et al. 1993] both on the original multivariate formulation, or on the one-dimensional formulation proposed in [Redon et al. 2002a] (and adapted to rotation vector interpolation trajectories). We provide a discussion of these two methods in Section 4.4.3 and provide a comparison with our technique in Appendix D.2.

Minimum Separation Linear CCD. Linear CCD with minimum separation [Harmon et al. 2011; Lu et al. 2019; Provot 1997; Stam 2009; Wang et al. 2021] detects collisions when two primitives are at a small user-specified distance. In our work, we reduce the curved CCD problem to a sequence of linear CCD with minimum separation. While any of the methods above could be used, we opt for [Wang et al. 2021], as it is the only one that is guaranteed to be correct when implemented using floating-point arithmetic, and it also has a public implementation available on GitHub. Our curved CCD algorithm can also be extended to support conservative minimum sep-

aration (Section 4.5), a feature that, to the best of our knowledge, no other curved CCD method considered before and that is useful in fabrication applications to ensure the satisfaction of clearance constraints.

4.3 IPC Overview

We briefly overview the IPC solver introduced in Chapter 2 to make this chapter self-contained.

Chapter 2 proposes a novel way to handle large deformation dynamics with frictional contact, reducing a single time step to the minimization of a *unconstrained* non-linear energy:

$$x^{t+1} = \underset{x}{\operatorname{argmin}} \; E_d(x, x^t, v^t) + B(x, \hat{d}) + D(x, \hat{d}), \tag{4.1}$$

where x^t is the set of nodal position, v^t the velocities, $E_d(x, x^t, v^t)$ is an IP for numerical time stepping [Kane et al. 2000], D is the friction potential, and B is the barrier potential. The later vanishes when primitives are further than a user-defined geometric accuracy $\hat{d}$ and diverges when two objects are in contact. Here we first review the barrier potential, as we will need to extend it in this work. In the next section we cover the necessary work to extend the incremental potential for rigid bodies and so enable the IPC formulation. For further details on the friction model and solving we refer to Chapter 2.

Solver and Line Search CCD. IPC requires an initial state that is free of self-intersections and uses a custom Projected Newton solver to time step the system by minimizing Equation (4.1) to a user-controlled accuracy. The solver ensures that the trajectories of all surface primitive pairs are intersection-free during the optimization. The guarantee comes from explicitly validating the linear trajectory in every line search using a conservative linear CCD query: if the CCD query returns a collision, the step length is reduced until a step is possible. The solver requires the

energy to be C^2 (as the Newton method requires the computation of the second derivatives) and thus a careful definition for all terms of the energy is necessary.

Barrier Functions and Distances. Let C be a set containing all non-incident point-triangle and all non-adjacent edge-edge pairs in surface meshes. The barrier potential is then defined as:

$$B(x, \hat{d}) = \kappa \sum_{k \in C} b\big(d_k(x), \hat{d}\big), \tag{4.2}$$

where κ is the barrier stiffness, d_k is the mollified unsigned distance between the k pair of primitives (we refer to Chapter 2 for the detail on the computation of the mollified distances d_k between the primitive pairs), and b is a logarithmic barrier function defined as

$$b(d, \hat{d}) = \begin{cases} -(d - \hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d}. \end{cases} \tag{4.3}$$

We note that while C contains a number of pairs that is quadratic with respect to the number of primitives, most of the pairs will result in a zero contribution to Equation (4.3) as the support of the barrier is local.

4.4 Method

Input. The input for our algorithm is a desired time step size h, a computational distance accuracy target, $\hat{d}$, and a set of n rigid bodies. Each rigid body i has a set of k_i vertices in axis-aligned, body-frame local coordinates $\hat{X}_i$, a set of triangular faces $\hat{F}_i$, a mass m_i, and an inertial frame I_i. For each symbol, we use the subscript i to identify per-body quantities, and the same symbol without the subscript denotes a stacked vector (or matrix, as appropriate) of that quantity concatenated across the set of all simulated objects (e.g., $\hat{X}_i$ give the coordinates of the i-th body,

while $\hat{X}$ is the stacked coordinates of all bodies). The position of each rigid body is then given by a parametrization with a rotation vector[1] $\boldsymbol{\theta}_i \in \mathbb{R}^3$ and a translation $q_i \in \mathbb{R}^3$ that together map each body from its local frame to world coordinates with

$$\phi_i(\boldsymbol{\theta}_i, q_i) = \mathcal{R}(\boldsymbol{\theta}_i)\hat{X}_i + q_i, \tag{4.4}$$

Here, the function $\phi_i : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^{3\times k_i}$ maps the k_i vertices (in local coordinates) of the i-th body into world coordinates with Rodrigues's rotation formula $\mathcal{R}$ (see Equation (4.14)) mapping from a rotation vector to a rotation matrix [Grassia 1998; Rodrigues 1840].

We initialize each simulation with a starting configuration of rotations $\boldsymbol{\theta}^0$ and translations q^0 for all bodies. We require a non-interpenetrating starting configuration and call any intersection free configuration *valid*.

Output. Simulation output is a final valid configuration $(\boldsymbol{\theta}^{t_{\text{end}}}, q^{t_{\text{end}}})$ obtained by time integrating the rigid body system, and the corresponding trajectory from $(\boldsymbol{\theta}^0, q^0)$ to $(\boldsymbol{\theta}^{t_{\text{end}}}, q^{t_{\text{end}}})$ guaranteed free of intersections. The generated trajectory is piecewise linear in generalized coordinates, $(\boldsymbol{\theta}, q)$, and is a curved trajectory in world coordinates.

Overview. Our approach follows the same high-level ideas as Chapter 2. Our first step requires us to formulate rigid body system time integrators as IP – these are not previously available. With rigid body IP in hand, we then can follow Chapter 2 by augmenting it with both a barrier and friction potential (remapped via ϕ) to resolve contact and friction forces, respectively. Below we first construct our incremental potential formulation (Section 4.4.1) and then describe how we adapt line search, constraint set generation, and a Newton-type solver to the rigid body time step problem. As a key part of this solution, during line search, we must process a special type of curved trajectories for continuous collision detection. For this, we develop a conservative CCD

[1]This parameterization, also often called an "Euler vector", gives a rotation around the vector's direction prescribed by an angle equal to the vector's magnitude.

query in Section 4.4.3. We provide an extensive comparison of our rigid body formulation and the original formulation of Chapter 2 in Section 4.6.1.

4.4.1 Rigid Body Incremental Potential

Following Chapter 2, we construct a discrete energy whose stationary points give an unconstrained time step method's *configurational* update. The Newton-Euler rigid body equations of motion are naturally defined at the acceleration level, however, they don't (due to parameterization) naturally integrate up to an obvious variational formulation whose extremizers give an updated rotation for a rigid body time step.

We then construct an IP formulation directly on rotation matrices Q_i that map points on rigid bodies i from their local frames to a frame axis aligned with the world. At any time t we then have $Q_i^t = \mathcal{R}(\boldsymbol{\theta}_i^t)$. Our first step is to recall that we can define angular kinetic energy directly on rotation matrix velocities [Hairer et al. 2006] as $\frac{1}{2}\operatorname{tr}(\dot{Q}_i J_i \dot{Q}_i^\top)$ where

$$J_i = \frac{1}{2}\operatorname{diag}(-I_i^x + I_i^y + I_i^z,\ I_i^x - I_i^y + I_i^z,\ I_i^x + I_i^y - I_i^z) = \operatorname{diag}(-I_i^x + I_i^y + I_i^z,\ I_i^x - I_i^y + I_i^z,\ I_i^x + I_i^y - I_i^z)$$

is the inertial matrix, and I_i^x, I_i^y, and I_i^z are components of the inertial frame I_i.

With the inertial matrix defined we now target *flat* equations of motion that will allow us to compose IPs for arbitrary numerical time integrators on Q_i. To do so we simply apply constrained Lagrangian dynamics with orthogonality $Q_i^\top Q_i - \boldsymbol{I} = 0$ as a constraint. We then can directly apply standard form, constrained time integrators with flat coordinates [Ascher and Petzold 1998]. With our construction, we derive here the IP formulation for a rigid body system integrated with

implicit Euler. For our formulation, the constrained implicit Euler time stepper is then

$$Q_i^{t+1} = Q_i^t + h\dot{Q}_i^t - h^2 \nabla V(Q_i^{t+1}) J_i^{-1} + Q_i^{t+1} \Lambda J_i^{-1} + h^2 [\tau_i] J_i^{-1}, \tag{4.5}$$

$$Q_i^{t+1\top} Q_i^{t+1} - \boldsymbol{I} = 0, \tag{4.6}$$

$$\dot{Q}_i^t = \frac{Q_i^t - Q_i^{t-1}}{h}, \tag{4.7}$$

where Λ is the symmetric Lagrange-multiplier matrix for our constraint, τ_i are any external, applied torques to body i at time t and V are any potential energies defined on Q_i. We use the notation $[.]$ to indicate the construction of the skew-symmetric (cross-product) matrix[2].

In turn, to create an implicit Euler rigid body IP we can next convert this to a corresponding variational form

$$\begin{aligned} \tilde{Q}_i^{\,t} &= Q_i^t + h\dot{Q}_i^t + h^2 [\tau_i] J_i^{-1} \\ Q_i^{t+1} &= \underset{Q}{\operatorname{argmin}} \tfrac{1}{2} \operatorname{tr}\left(Q J_i Q^\top\right) + \operatorname{tr}\left(Q J_i (\tilde{Q}_i^t)^\top\right) + h^2 V(Q), \\ &\qquad \text{s.t. } Q^\top Q - \boldsymbol{I} = 0. \end{aligned} \tag{4.8}$$

Then, for our entire rigid body system (presuming w.l.o.g. for now no potentials) the implicit Euler IP for rotational coordinates is

$$E_Q(Q) = \sum_{i=1}^{n} \left(\tfrac{1}{2} \operatorname{tr}(Q_i J_i Q_i^\top) - \operatorname{tr}(Q_i J_i (\tilde{Q}_i^{\,t})^\top) \right), \tag{4.9}$$

[2]
$$[v] = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}$$

and correspondingly for translational coordinates (directly from standard implicit Euler) we have

$$\tilde{q_i}^t = q_i^t + h\dot{q}_i^t + h^2(g + m^{-1}f_i)$$
$$E_q(q) = \sum_{i=1}^{n} \left(\tfrac{1}{2} m_i q_i^\top q_i - m_i q_i^\top \tilde{q}_i^t\right), \tag{4.10}$$

where g is the acceleration due to gravity, f_i are any external, applied forces to body i's center of mass at time t, and velocities are updated by

$$\dot{Q}^t = \frac{Q^t - Q^{t-1}}{h} \quad \text{and} \quad \dot{q}^t = \frac{q^t - q^{t-1}}{h}.$$

Finally, the complete implicit Euler rigid body IP is

$$E(Q, q) = E_Q(Q) + E_q(q),$$

Now that it is defined entirely in terms of Q and q it can be, as per our strategy, directly applied to swap for E_d in Equation (4.1), when we wish to apply rigid body coordinates. This gives us the following *constrained* optimization problem to solve

$$(Q^{t+1}, q^{t+1}) = \underset{Q,q}{\operatorname{argmin}}\, E(Q, q) + B(\phi(Q, q), \hat{d}) + D(\phi(Q, q)) \tag{4.11}$$
$$\text{s.t. } Q_i^\top Q_i = \boldsymbol{I},\ i = \{1, \ldots, n\}, \tag{4.12}$$

where the constraint is necessary to ensure that minimizer Q^{t+1} gives rotation matrices.

Rotation Vector Parametrization. Our goal remains to use *unconstrained* optimization in order to apply as Newton-type solver with line-search filtering and so robustly minimize the IP with guarantees. To do so parameterizing rotations with the rotation vector, $\boldsymbol{\theta}_i$, allows us to then directly apply Rodrigues' rotation formula to drop equality constraints from Equation (4.12).

This finally leads us to an unconstrained optimization problem, and so gives us our rigid body incremental potential for frictional contact

$$(\boldsymbol{\theta}^{t+1}, q^{t+1}) = \underset{\boldsymbol{\theta},q}{\operatorname{argmin}}\, E(\mathcal{R}(\boldsymbol{\theta}), q) + B(\phi(\mathcal{R}(\boldsymbol{\theta}), q)) + D(\phi(\mathcal{R}(\boldsymbol{\theta}), q)).$$

In turn, as we discuss next it can now be solved with a filtered projected Newton solver.

Our rotation vector parametrization is then critical to obtaining our unconstrained minimization form of the IP, as it avoids additional constraints and enables us to solve the optimization with an unconstrained projected Newton solver. While alternatives exist to minimize energies like our IP in the space of SO(3) [Owren and Welfert 2000], it is not immediately obvious how to integrate our barrier in these methods as they do not offer filtered line-search.

Adding differently scaled rotation vectors can require an increased number of updates to change the axis of rotation. However, do to warm-starting each solve from the last time step, this problem never arises in practice even in scenes with large time step sizes. We discuss a synthetic example of this more and provide a solution (if ever needed) in Appendix D.5.

4.4.2 Projected Newton Solver

Now that we have constructed an unconstrained barrier IP for rigid bodies we apply the Newton-type solver proposed in Chapter 2, with a few modifications that are necessary to address numerical challenges specific to the rigid body IP formulation.

Rodrigues' Rotation Formula and its Derivatives. Rodrigues' rotation formula provides a way of computing a rotation matrix from a rotation vector. Rodrigues' rotation formula is commonly written as

$$\mathcal{R}(\boldsymbol{\theta}) = \boldsymbol{I} + \sin(\|\boldsymbol{\theta}\|)\left[\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}\right] + (1 - \cos(\|\boldsymbol{\theta}\|))\left[\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}\right]^2, \tag{4.13}$$

where $\mathcal{R}(0) = \boldsymbol{I}$. For numerical stability (around $\boldsymbol{\theta} = \mathbf{0}$), we rewrite $\mathcal{R}$ as

$$\mathcal{R}(\boldsymbol{\theta}) = \boldsymbol{I} + \operatorname{sinc}(\|\boldsymbol{\theta}\|)[\boldsymbol{\theta}] + 2\operatorname{sinc}^2\left(\frac{\|\boldsymbol{\theta}\|}{2}\right)[\boldsymbol{\theta}]^2, \tag{4.14}$$

where

$$\operatorname{sinc}(x) = \begin{cases} 1 & x = 0 \\ \frac{\sin(x)}{x} & \text{otherwise} \end{cases}.$$

Note, we compute values close to zero computed using a Taylor series expansion (see Appendix D.1.1) [Grassia 1998].

While sinc is C^∞, special care is needed to compute its gradient and Hessian to avoid divisions by 0 (or small numbers). A full derivation of the derivatives of $\operatorname{sinc}(\|\boldsymbol{\theta}\|)$ is provided in Appendix D.1.2.

Additionally, when computing $\operatorname{sinc}(x)$ with interval arithmetic a naïve implementation using interval division can result in intervals far outside the range of $\operatorname{sinc}(x)$ (due to divisions of small numbers). We instead utilize the monotonic domain near zero by computing the real values (or a small interval to account for rounding errors) of the interval's endpoints. We discuss this strategy further in Appendix D.1.3.

Stabilization. Because of our transformation from axis-angle to rotation matrix, the Hessian $\nabla^2(E_Q(Q))$ may not be PSD. Unlike in the elastodynamic case, a projection to PSD is not balanced by the addition of a mass matrix and so can result in a singular matrix. Instead, we first apply the unprojected Hessian (inexpensive when compared to the finite element formulation in the original IPC) and if the linear solve fails or the computed direction is not a descent direction we apply standard offsetting by adding an identity scaled by ξ and solving. We continue the process, increasing ξ by a factor of two until either the $\xi > \xi^{\max} = 1e12$ or the solve is successful. In practice, this offset is rarely needed, and we never reach $\xi^{\max}$ in any of our experiments.

Evaluation of the Barrier Term B. The set C contains all possible collision pairs. However, due to the local support of the barrier functions, it is unnecessary to consider pairs whose distance is larger than $\hat{d}$, as they do not contribute to the barrier potential B (Equation (4.2)). In Chapter 2, the pairs of primitives closer than $\hat{d}$ are quickly detected using a spatial hashing data structure. For the rigid case, we can exploit the rigidity of the objects to avoid the construction of a hash grid for every evaluation of the barrier potential.

We explicitly consider the relative position of a pair of rigid bodies a and b. In the reference system of b, the relative position of the vertices of a are:

$$s_{ba} = \mathcal{R}(\boldsymbol{\theta}_b)^\top (\mathcal{R}(\boldsymbol{\theta}_a)\hat{X}_a + q_a - q_b). \tag{4.15}$$

We can thus build a BVH for every rigid body independently made by one bounding box for every primitive, only once when a model is loaded. We can then build a bounding box for each primitive in a, enlarge it by $\hat{d}$, map it to the reference system of b using Equation (4.15), and then query the BVH of b to find candidate pairs for the set C. To ensure that the check is conservative, we evaluate Equation (4.15) using interval arithmetic [Tucker 2011] (note that an axis-aligned bounding box is simply a triplet of one-dimensional intervals). Additionally, we also use a scene BVH containing one bounding box for every body to discard any pair of rigid bodies that do not contain potential pairs.

4.4.3 Curved CCD

To ensure that there are no intersections at any time during the simulation, we explicitly check for collisions during every line search. Following the common approach used in linear CCD, we proceed in two phases: a broad phase to quickly identify pairs of primitives that are likely to be in contact, and the narrow phase, to certify every candidate pair. We first introduce the special type of curved trajectories that we consider in this work and then propose a broad

phase algorithm that takes advantage of the rigidity of the bodies.

Curved Trajectories. The trajectory of the vertices of a primitive (i.e., a vertex, edge, or triangle) a_i in a body $\hat{X}_i$ are mapped from a configuration $(\boldsymbol{\theta}_i^0, q_i^0)$ to a configuration $(\boldsymbol{\theta}_i^1, q_i^1)$, by

$$\phi_{a_i}(t) = \mathcal{R}(\boldsymbol{\theta}_i(t))a_i + q_i(t), \quad t \in [0,1]. \tag{4.16}$$

where

$$\boldsymbol{\theta}_i(t) = (1-t)\boldsymbol{\theta}_i^0 + t\boldsymbol{\theta}_i^1 \quad \text{and} \quad q_i(t) = (1-t)q_i^0 + tq_i^1.$$

Note that $\phi_{a_i}(t)$ is non-linear in t due to the presence of Rodrigues' formula $\mathcal{R}$.

Broad-Phase. To reduce the computational cost, we express the trajectory in the reference system of one body extending Equation (4.15) to the time dependent case,

$$s_{ba}(t) = \mathcal{R}(\boldsymbol{\theta}_b(t))^\top(\mathcal{R}(\boldsymbol{\theta}_a(t))\hat{X}_a + q_a(t) - q_b(t)). \tag{4.17}$$

We propose to use interval arithmetic [Tucker 2011] to automatically compute a bound. That is, we evaluate $s_{ba}(t)$ over the interval $[0,1]$ to obtain a bounding box for every point in $\hat{X}_a$ representing a conservative estimation of the trajectory with respect to b. The bounding boxes can then be used in a standard spatial acceleration data structure where we reuse the same BVH we built for evaluating the barrier potential.

Narrow Phase Curved CCD. After identifying potential pairs of primitives colliding, the goal of the narrow phase is to find the earliest time t (if any) for which a pair of primitives (either triangle-point or edge-edge) intersects.

Consider the trajectory of a point $\boldsymbol{p}(t)$ and the trajectories of the three vertices of a triangle $\boldsymbol{p}_1(t), \boldsymbol{p}_2(t), \boldsymbol{p}_3(t)$. The most direct formulation of continuous collision detection is to explicitly

look for the earliest root of the following non-linear system of equations

$$F_{\text{vf}}(t, \alpha, \beta) = \boldsymbol{p}(t) - \big((1 - \alpha - \beta)\boldsymbol{p}_1(t) + \alpha\boldsymbol{p}_2(t) + \beta\boldsymbol{p}_3(t)\big), \tag{4.18}$$

for $t, \alpha, \beta \in [0, 1]$, and $\alpha + \beta \leq 1$. If no root exists the two primitives do not intersect. Similarly, consider the trajectory of two edges whose vertices are $\boldsymbol{p}_1, \boldsymbol{p}_2$ and $\boldsymbol{p}_3, \boldsymbol{p}_4$

$$F_{\text{ee}}(t, \alpha, \beta) = \big((1 - \alpha)\boldsymbol{p}_1(t) + \alpha\boldsymbol{p}_2(t)\big) - \big((1 - \beta)\boldsymbol{p}_3(t) + \beta\boldsymbol{p}_4(t)\big) \tag{4.19}$$

for $t, \alpha, \beta \in [0, 1]$.

Baseline Solutions. To the best of our knowledge, there are no existing algorithms developed specifically for our problem, that is the particular formulation of $\phi_{a_i}(t)$. However, there are two approaches that can be easily adapted. The first is the generic interval root finder proposed by Snyder [1992], which can directly be used to find roots of the non-linear system of Equations (4.18) or (4.19). The second is an adaptation for our problem of the screw CCD proposed by Redon et al. [2002a], which uses a univariate formulation to improve performances. Unfortunately, after experimenting with both approaches, we conclude that they cannot be used for our purposes. The former has a very long runtime due to the expensive interval computation and a large number of dimensions of the domain to subdivide, while the latter cannot handle degenerate cases linked to the univariate formulation (see [Wang et al. 2021] for a more detailed explanation of the intrinsic limitation of univariate formulations for linear CCD). We provide a comparison between our algorithm and the two baselines in Appendix D.2.

Linearization Error. We propose a novel algorithm based on the following idea: if we can compute an upper bound b of the maximal error between a curved trajectory and its piecewise linear approximation, then we can conservatively check for collisions using a linear CCD with

a minimum separation of b. Let us consider the curved trajectory $\phi_{a_i}(t)$ (Equation (4.16)) of a single vertex $a_i \in \hat{X}_i$. The time-dependent distance between the curved trajectory and the linear approximation is:

$$e_{a_i}(t) = \|\phi_{a_i}(t) - ((1-t)\boldsymbol{p}^0 + t\boldsymbol{p}^1)\|, \quad t \in [0,1]. \tag{4.20}$$

with $\boldsymbol{p}^0 = \phi_{a_i}(0)$ and $\boldsymbol{p}^1 = \phi_{a_i}(1)$. By evaluating e_{a_i} over the interval $[0,1]$ using interval arithmetic, we obtain our desired bound b. This construction can be extended to find a distance bound for all points between two convex primitives by evaluating e_{a_i} for every vertex in both primitives and taking the maximum. Given the pair of primitives and the bound b we conservatively check for intersections using the linear minimum separation CCD proposed by Wang et al. [2021], using the L^∞ metric for minimum separation. This idea is used in Algorithm 4.1 to adaptively refine the linear approximation depending on the error bound.

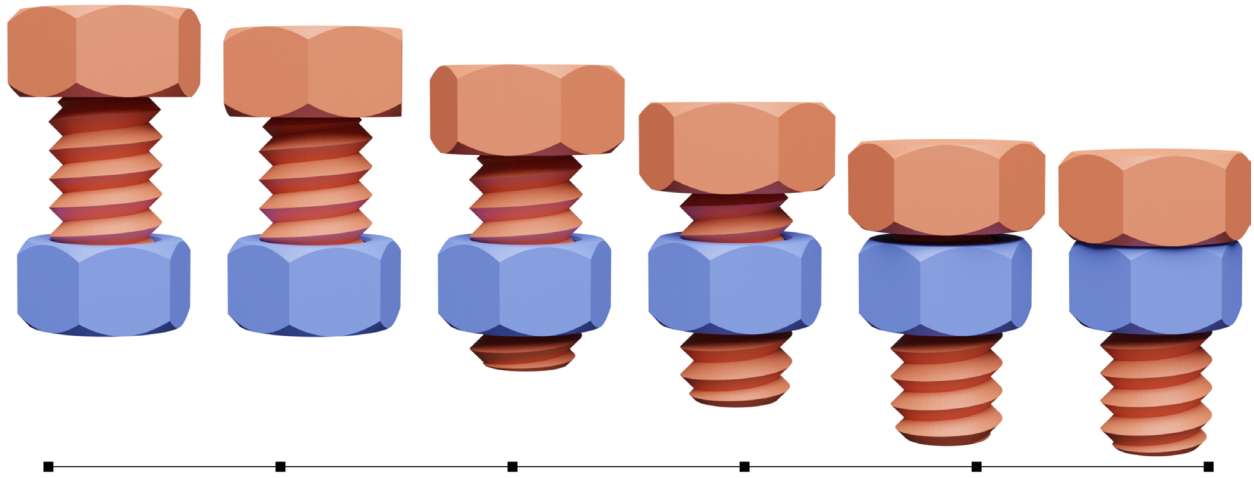
Algorithm Description. The algorithm keeps track of the earliest time guaranteed to be collision-free in a variable t_0 (initially equal to 0), which is incremented whenever the linear CCD is able to validate a section of the trajectory (Line 21). The algorithm iteratively subdivides the linear approximation, keeping track of the endpoint of every segment in a stack ts. After a segment is retrieved from the stack (Line 6), we compute the initial distance between the two objects (Line 7) and an upper bound on the error of the linear approximation of the trajectory (Line 8). If the bound is larger than the initial distance (Line 9) the linear CCD will find a collision at the beginning of the time since the linear approximation is poor. We thus refine the linear approximation. The parameter $\delta \in (0,1)$ (Line 9) allows us to trade off the cost between the CCD and the refinement. A value close to 1 will lead to minimal refinement, but potentially more challenging queries for the linear CCD, while a smaller value will preemptively refine the linear approximation, making the CCD queries easier. We experimentally found that a value of 0.5 is a good tradeoff (see the parameter study in Appendix D.3). To bound the cost of the linear CCD and prevent overrefinement, we set an upper bound $N^{\max}$ on the maximal number of sub-

Algorithm 4.1 Determine if and when the earliest impact occurs along a curved trajectory.

1: **procedure** CurvedCCD($a_i, b_j, \delta, N^{\max}$)
2: $t_0 \leftarrow 0$
3: $ts \leftarrow \{1\}$
4: $N \leftarrow 1$
5: **while** $ts \neq \emptyset$ **do**
6: $t_1 \leftarrow \text{top}(ts)$
7: $d_{t_0} \leftarrow d(t_0, a_i, b_j)$ ▷ Equations (2.18) and (2.19)
8: $b \leftarrow e_{a_i}([t_0, t_1]) + e_{b_j}([t_0, t_1])$
9: **if** $b \geq \delta d_{t_0}$ and ($N < N^{\max}$ or $t_0 = 0$) **then**
10: $ts \leftarrow ts \cup \{(t_1 + t_0)/2)\}$
11: $N \leftarrow N + 1$
12: **continue**
13: impact, toi $\leftarrow$ LinearCCD($\phi_{a_i}(t_0), \phi_{a_i}(t_1), \phi_{b_j}(t_0), \phi_{b_j}(t_1), b$)
14: **if** $t_0 = 0$ and toi $= 0$ **then**
15: $ts \leftarrow ts \cup \{t_1/2\}$
16: $N \leftarrow N + 1$
17: **continue**
18: **if** impact **then**
19: **return** true, $t_0 + \text{toi}(t_1 - t_0)$
20: pop(ts)
21: $t_0 \leftarrow t_1$
22: **return** false, ∞

divisions (we use 1000 in our experiments). The bound is however disabled when $t_0 = 0$, as we need to have a strictly positive TOI to make progress in the Newton optimization and we know that a non-zero t always exists due to our barrier formulation. If the interval passes the distance check, we apply linear CCD (Line 13), and we further refine in case the linear CCD returns a ToI of 0 and if $t_0 = 0$ as this must be due to the poor approximation of b since a non-zero t always exist. If the linear CCD finds a collision we report it and return, otherwise we continue with the next segment in the stack. If we reach the end of the trajectory without finding a collision, the algorithm terminates and reports that the trajectory is collision-free.

For linear CCD with minimum separation, we use [Wang et al. 2021] with default parameters.



Model ©YSoft be3D under CC BY-SA 3.0.

Figure 4.4: Bolt. A bolt spins inside a static nut under gravity. Without friction, the bolt is quickly able to follow the threading and begins to rotate.

Shared Earliest Time of Impact. As in Chapter 2, we compute an upper bound on the step size using the earliest time-of-impact for a given step. To speed up this process, we follow the advice of Redon et al. [2002a] who suggests reusing the earliest time-of-impact from the previous CCD queries for the same step. This reduces the number of queries and is achieved by replacing Line 3 with $ts \leftarrow t^{\text{earliest}}$, where t^{earliest} is the earliest time-of-impact for the current step (initially 1).

Minimum Separation. To extend our algorithm to guarantee a minimum separation we make three minor modifications to our formulation. First, we shift the input to our distance barrier (Equation (4.2)) by subtracting the minimum separation distance from the primitive pair's distance. Second, we inflate all bounding boxes used in the broad-phase to account for the added minimum separation. Last, we take advantage of the linear minimum separation CCD to add an additional offset to the minimum separation (before Line 13 perform $b \leftarrow b + d_{\text{min}}$).

4.4.4 Boundary Conditions

A kinematic rigid body moves under its own velocity but does not respond to collision forces. We implement kinematic bodies using an augmented Lagrangian (AL) based on the method in Appendix A.5 to enforce Dirichlet boundary conditions. For each kinematic rigid body k, we construct the AL from the two terms,

$$E_{\mathrm{A,q}}(q) = \frac{\kappa_{\mathrm{A},q}}{2} m_k \left\| q_k - \hat{q}_k^{t+1} \right\|^2 - \sqrt{m_k} \lambda_{\mathrm{A},k}^T (q_k - \hat{q}_k^{t+1})$$

$$E_{\mathrm{A,Q}}(Q) = \frac{\kappa_{\mathrm{A},Q}}{2} \operatorname{tr}(Q_k - \hat{Q}_k^{t+1}) J_k (Q_k - \hat{Q}_k^{t+1})^\top) - \operatorname{tr}(\Lambda_{\mathrm{A},k}^\top (Q_k - \hat{Q}_k^{t+1}) J_k^{\frac{1}{2}})$$

where $(\hat{q}_k^{t+1}, \hat{Q}_k^{t+1})$ is the prescribed configuration at time $t+1$.

Following the algorithm in Appendix A.5, we initialize the Lagrange multipliers to $\lambda_{\mathrm{A},k} = 0$ and $\Lambda_{\mathrm{A},k} = 0$ and penalty stiffnesses to $\kappa_{\mathrm{A},q} = 10^3$ and $\kappa_{\mathrm{A},Q} = 10^3$. These potentials are then added to Equation (4.12).

The convergence criteria of each time step optimization is then modified to account for the satisfaction of the kinematic bodies' motion. Concretely, we compute

$$\eta_q = 1 - \sqrt{\frac{\sum_k \|\hat{q}_k^{t+1} - q_k\|^2}{\sum_k \|\hat{q}_k^{t+1} - q_k^t\|^2}}$$

and

$$\eta_Q = 1 - \sqrt{\frac{\sum_k \|\hat{Q}_k^{t+1} - Q_k\|_F^2}{\sum_k \|\hat{Q}_k^{t+1} - Q_k^t\|_F^2}}$$

and converge iff the optimization's stationarity criteria is satisfied with $\eta_q > 0.999$, and $\eta_Q > 0.999$.

If only stationarity is satisfied, we update the AL parameters. For brevity we only describe

the update scheme for $\kappa_{\mathrm{A},q}$, λ_{A} as the others follow closely. If $\eta_{\mathrm{A},q} < 0.99$ and $\kappa_{\mathrm{A}} < 10^8$, then

$$\kappa_{\mathrm{A},q} \leftarrow 2\kappa_{\mathrm{A},q}.$$

Otherwise, for each kinematic body k,

$$\lambda_{\mathrm{A},k} \leftarrow \lambda_{\mathrm{A},k} - \kappa_{\mathrm{A}}\sqrt{m_k}(q_k^i - \hat{q}_k^{t+1}).$$

Additionally, whenever the AL convergence criteria are satisfied, we fix all prescribed DOF and remove the AL from Equation (4.12) for the remainder of the optimization. This helps by removing unnecessary stiffness in our objective function.

4.5 Results

Our algorithm is implemented in C++ and uses Eigen [Guennebaud et al. 2010a] for the linear algebra routines, libigl [Jacobson et al. 2018] for basic geometry processing routines, and filib for interval arithmetic [Lerch et al. 2006]. We run our experiments on a workstation with two AMD EPYC™ 7452 Processors. The reference implementation used to generate the results is attached to the submission and will be released as an open-source project. We provide a video for every simulation shown in the paper as part of our additional material.

We first present our results and postpone a comparison against existing rigid body simulators to Section 4.6 and to a volumetric IPC formulation in Section 4.6.1.

Rigid Body Mechanisms with Complex Geometry. The first example is a bolt that spins under gravity inside a nut. This is a challenging scene for many rigid body simulators (although others have shown success [Wang et al. 2012; Xu et al. 2014]) due to the tight sliding contacts on an extended curved area (Figure 4.4).

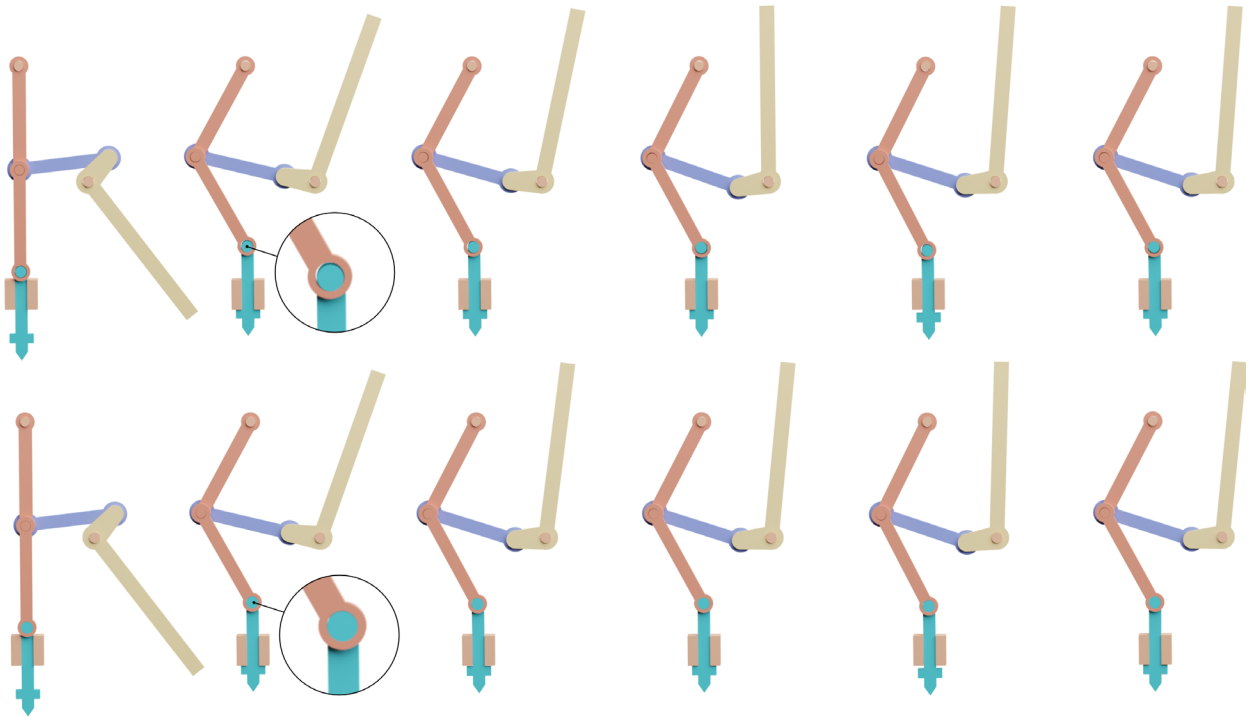


Figure 4.5: Punching Press. We designed two variations of a punching press mechanism: one with loose joints (top row) and one with tight (bottom row). By applying a force to raise the punch, our use of full rigid DOF instead of articulated bodies allows us to model and test varying tolerance in joints.

We show a collection of more complex mechanisms in Figure 4.2, including a piston, a rotating wheel that generates intermittent motion, and a bike chain. In all cases, we do not use any constraint on the reduced coordinates.

Note that the contacts are reliably handled by our approach, enabling us to experiment with variations in the mechanisms, for example by adding additional tolerance in the holes of a punching press (Figure 4.5). Note that explicit collision modeling is necessary to capture this effect.

Simulation for Fabrication. Our method can be used to design and simulate complex mechanisms before real-world fabrication. To mock-up this use case, we purchased the 3D model of a 3D printed locking box from Maker's Muse[3], and directly use the STL files in our simulator (Figure 4.1). The mechanisms can be studied in our simulation, where it is easy to modify the

[3]Maker's Muse: https://www.makersmuse.com/expanding-lock-box

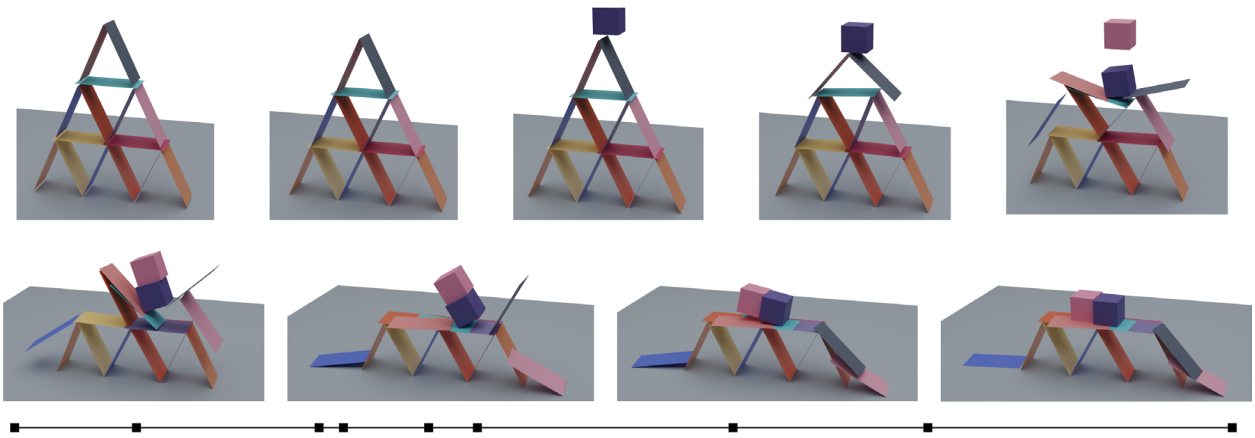


Figure 4.6: Codimensional card house. We design a codimensional variant of the standard frictional benchmark of Kaufman et al. [2008], where each card is composed of only two triangles. The cards are briefly allowed to stably come to rest ($\mu = 0.9$), before being impacted by two cubes. The top two levels collapse, but the final floor is able to catch the cubes demonstrating our ability to quickly handle transitions between static and dynamic friction.

design and test it in a virtual environment.

Codimensional Rigid Bodies. Our algorithm supports simulating codimensional bodies. We show a card house composed of 2D codimensional, rigid cards in Figure 4.6. 1D co-dimensional objects are also supported and can be used, for example, to efficiently simulate a large chain net (Figure 4.7 Top). As a stress test, we drop a heavy ball on top of the chain net. We can even simulate 0D codimensional point-clouds. As a demonstration, we roll a point cloud ball (with friction) towards another ball composed of planar slices (Figure 4.7 Bottom).

Large Angular Velocity. We can simulate objects moving at high angular velocities to capture interesting real-world effects involving rigid body objects, such as a spolling coin (Figure 4.8), with a timestep of 10^{-4} s.

Large Numbers of Bodies. Our algorithm can stably simulate large collections of rigid bodies, as demonstrated by a stack of boxes displaced by a wrecking ball (Figure 4.9).

We can also stably simulate long chains of interlinked bodies. We show an example in Fig-

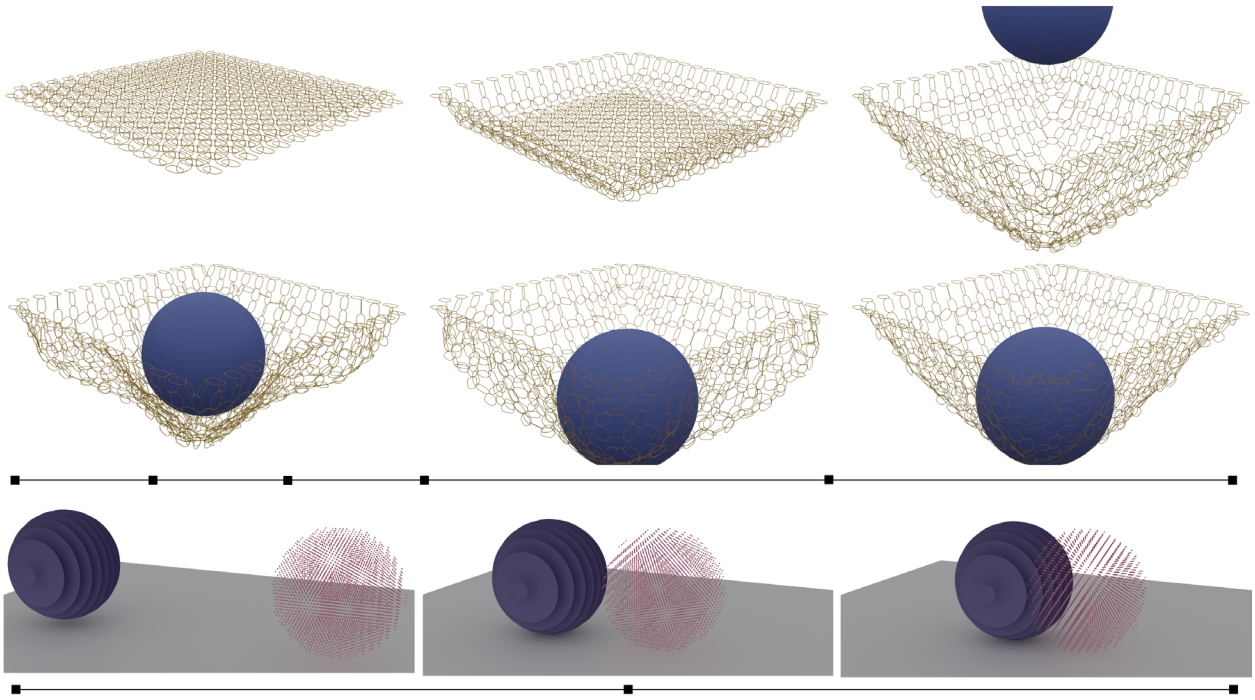


Figure 4.7: Codimensional bodies. The IPC formulation allows us to easily simulate codimensional objects. Top: A ball is dropped onto a chain net composed of 1D codimensional edges. Bottom: A sphere of disconnected codimensional planes and a point cloud ball roll into each other. Upon contact, the geometry locks, and both spheres rock back and forth before coming to rest.

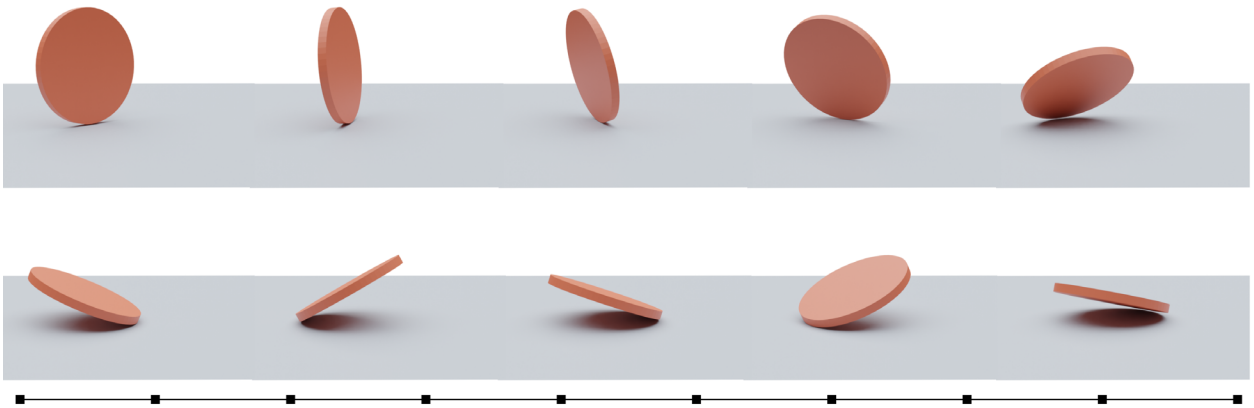


Figure 4.8: Spolling coin. A coin spolls (spins while rolling) on a surface with friction ($\mu = 0.2$). As the coin falls it continues to rotate while only a single point touches the ground. To accurately capture these high-speed dynamics, we use a small timestep of $h = 10^{-4}$ s.

ure 4.10, where a heavy anchor is lifted by rolling up a chain composed of 21 individual links.

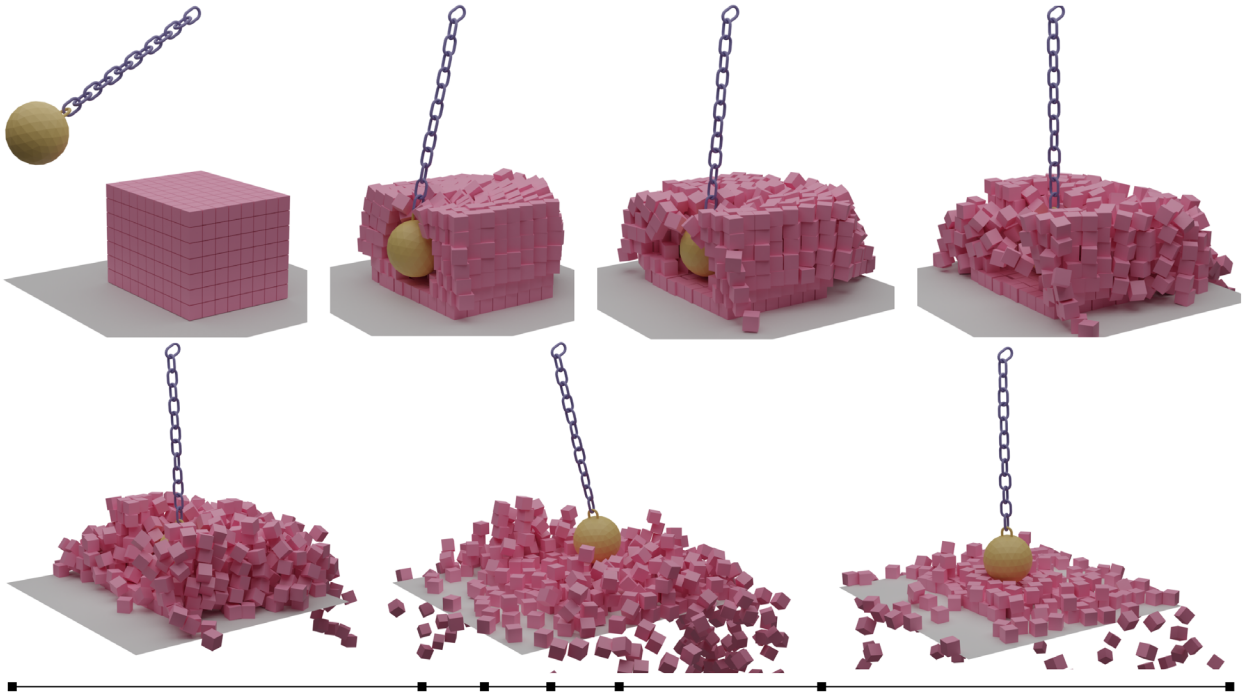


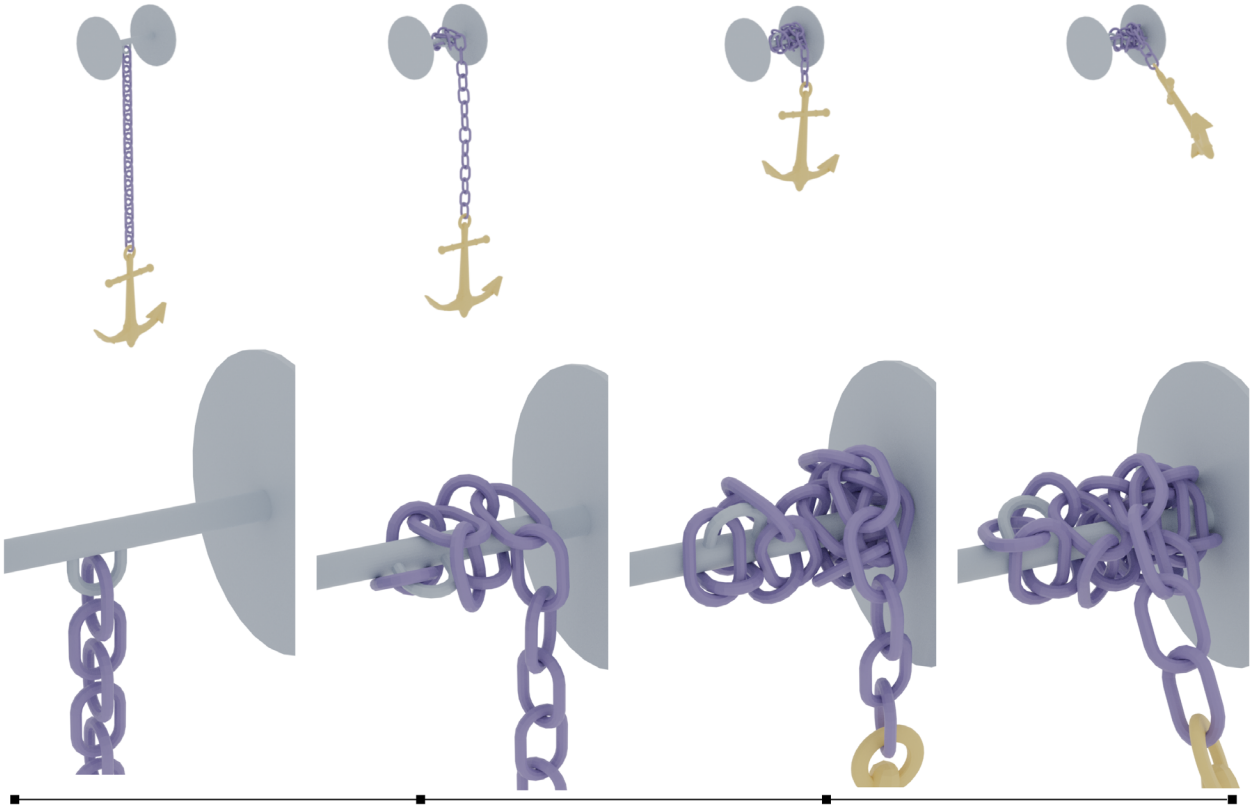
Figure 4.9: Wrecking ball. A stack of 560 boxes is hit by a wrecking ball made from a chain of interlinked bodies.

Friction. We repeat the arch scene experiment used to benchmark the friction model in Chapter 2, replacing the deformable yet stiff blocks with rigid objects (Figure 4.11). The results are indistinguishable (see also Figure 4.19).

The Lewis is an interesting mechanism used to lift heavy bodies, relying on static friction (Figure 4.12). As a final friction experiment, we place a box on a spinning disk with four different coefficients of friction. As the rotational velocity of the disk increases, the box loses contact and flies away (Figure 4.13) for $\mu \neq 1$.

Packing for 3D Printing with a Minimum Separation. The stability of our algorithm over large time steps and the possibility to add controlled minimum separation makes it ideal for packing multiple objects within the bed of a 3D printer. A common way to solve this problem is inflating the objects by the printer clearance and then use bin packing [Fogleman 2017].

Our algorithm can be used as a simple alternative to packing for 3D printing (Figure 4.14): we



Model ©Animation Anchor Line (anchor) under TurboSquid 3D Model License.

Figure 4.10: Anchor. A heavy anchor attached to a chain briefly falls under gravity before being lifted by rolling the chain around an axle. Natural bunching and kinking behaviors are visible.

can compute a packing of a collection of objects by dropping them in a box and extending our algorithm to ensure that the printer clearance is respected.

This is just a prototype, and more research will be necessary to evaluate the effectiveness of this approach in practical applications and compare it with bin packing, especially since the runtime of Wang et al. [2021] (and consequently of our curved CCD) increases considerably for large minimum separation distances.

Scalability. Our reference implementation exploits parallelization in the following algorithmic stages: energy gradients and Hessians are constructed in parallel, all body pairs in the barrier and CCD broad phase are evaluated in parallel, and the narrow phase CCD is performed in parallel to compute the earliest time-of-impact. Overall, this allows our algorithm to take advantage

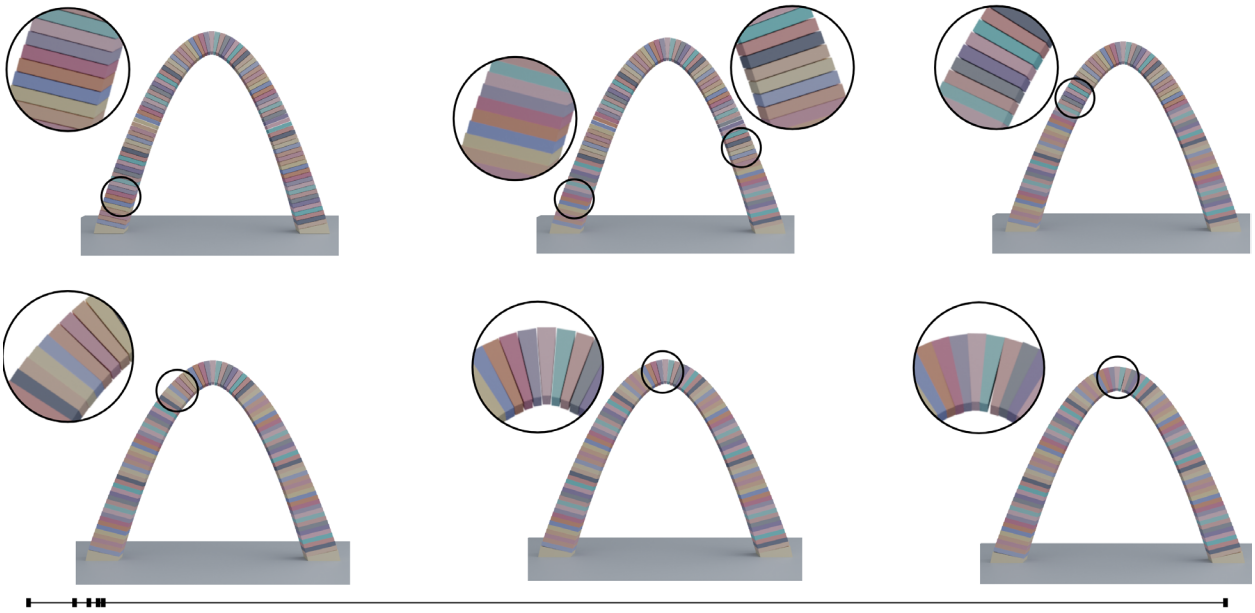


Figure 4.11: Arch. An arch composed of 101 rigid blocks is in equilibrium under gravity due to friction forces.

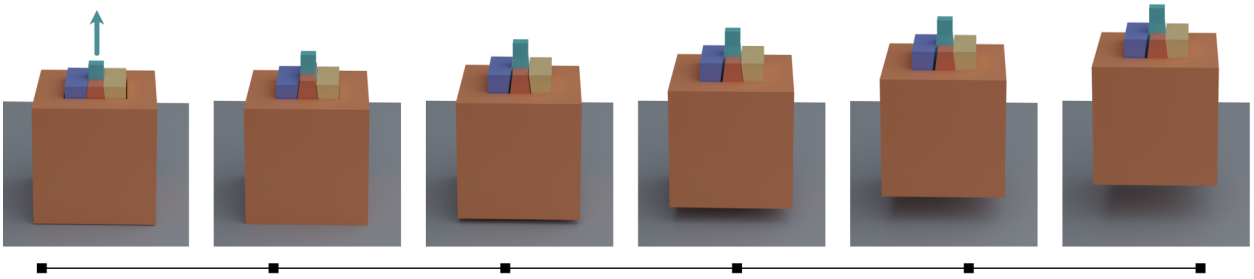


Figure 4.12: Lewis lifting mechanism. Utilizing friction and geometry the Lewis is able to lift large weights. A pyramid-shaped piece is placed between wedge-shaped pieces. When the center piece is pulled up the surrounding pieces are pressed into the outer block. The center is moved kinematically at 0.5 m/s with $\mu = 0.3$ and is able to lift a block 10× its mass.

of modern multi-core processors. We test the weak (i.e., we increase the complexity of the scene as we increase the number of threads) and strong (i.e., we keep the scene the same as we increase the number of threads) scaling of our method by simulating a chain of densely meshed links (Figure 4.15).

[4]Shapeway's PA11 material: https://www.shapeways.com/materials/pa11

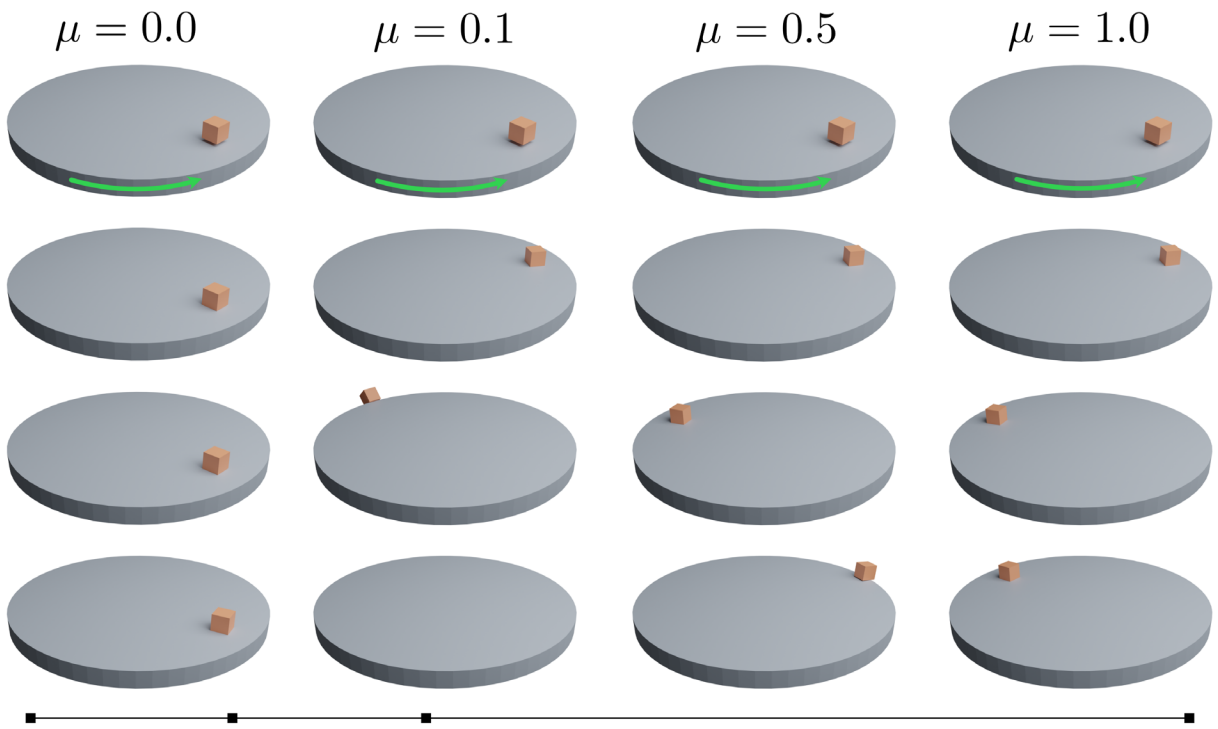


Figure 4.13: Turntable. A block is dropped on an accelerating turntable with four different coefficients of friction ($\mu = 0, 0.1, 0.5, 1.0$). With $\mu = 0$, the block rests on top of the table, slowly drifting. With $\mu = 0.1$, the block quickly catches and is flung away by the table. With $\mu = 0.5$, the block is able to hang on longer but eventually slides to the edge and falls off. With $\mu = 1$, the block sticks to the table and remains in the same relative position throughout the simulation.

4.6 Benchmark

We perform an extensive benchmark comparison on some of the most popular rigid body simulators (Bullet, MuJoCo, Chrono, and Houdini's RBD), focusing on evaluating the methods' ability on: (1) maintaining stability, (2) avoiding interpenetration, and (3) producing accurate dynamics. Our benchmark includes unit tests composed of simple primitive geometries like tetrahedron and cubes (Figure 4.16), degenerate test cases proposed by Erleben [2018] (Figure 4.17), and some of our more complex, large scale examples. In general, existing methods are orders of magnitude faster than our method, but fail severely even on simple scenes, depending on the parameters. Additionally, we show that, even with extensive parameter tuning, these methods cannot simu-

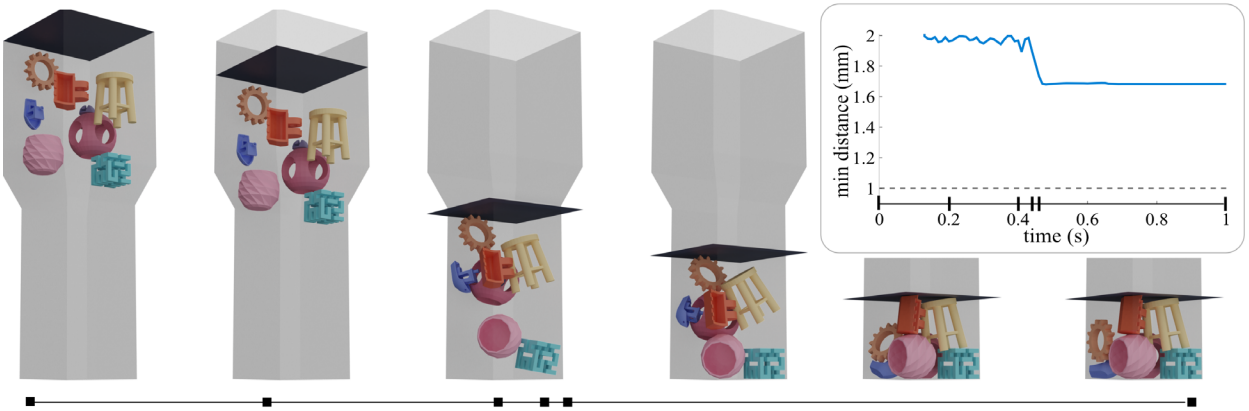


Figure 4.14: 3D packing. Based on the tolerance of Shapeway's PA11 material[4], we pack eight models into the bed of a 3D printer of size $290 \times 290 \times 600$ mm, with a clearance of 1mm enforced by our minimum separation. (Inset) We plot the minimum distance throughout the simulation showing that we always maintain the desired minimum distance between objects.

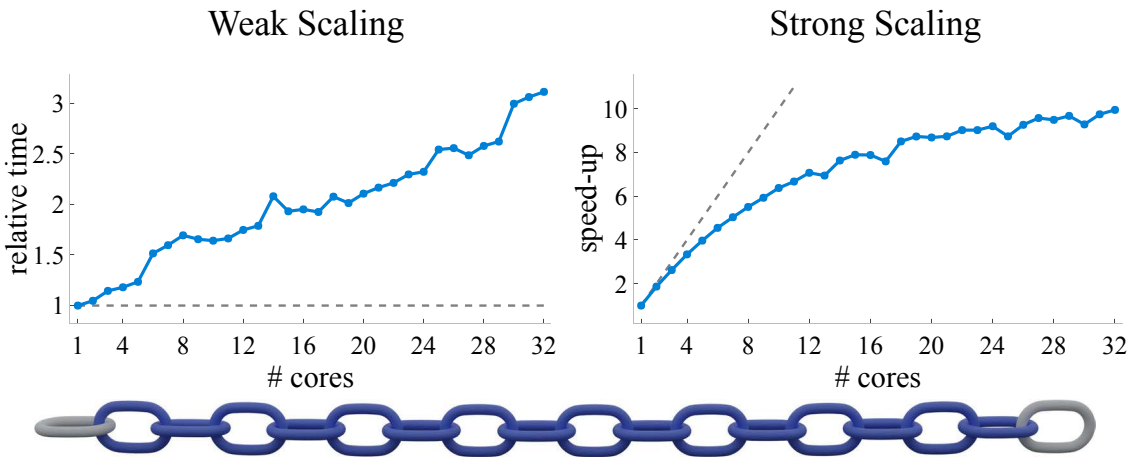


Figure 4.15: Scalability. We test both the weak (left) and strong (right) scalability on a chain of densely meshed links (bottom). For weak scaling, we set the number of free links equal to the number of threads and plot the runtime divided by the single link time. For strong scaling, we use a chain of 64 links and plot the speed-up over the single-core time. In each case, we plot the ideal value in grey. While our method greatly benefits from parallelization we see diminishing returns after 16 cores and observe little improvement when testing up to 64 cores.

late certain scenes. All scripts with simulation parameters tested will be publicly released as part of our open-source project.

Bullet. The primary method for modeling contacts between moving concave geometries in Bullet is via convex collision resolution employing convex decomposition proxies for input mesh geometries. Bullet provides automated construction of approximate convex decompositions for meshes via V-HACD [Mammou 2020]. Hand-crafted custom decompositions are often employed instead which can provide better approximation of the geometry and so improved collision proxies. In the following experiments we use input meshes for convex geometries (all of the unit test and Erleben's tests) or else, for concave geometries, an expert-constructed manual decomposition.

Bullet performs well on the unit tests and tests of Erleben [2018], but generates interpenetrations at larger time steps (0.01 s). Bullet performs best when the timestep is not too large (the default is 1/240 s and "several parameters are tuned with this value in mind" [Coumans and Bai 2019]). We find that $h = 10^{-3}$ s works for most scenes, but some scenes (e.g., five-cube stack and spikes) require time steps as small as (10^{-4} s) to completely avoid interpenetrations. In the volumetric chain-net, one of our more complex benchmark scenes, large time steps generate intersections and constraint drift that eventually lead to tunneling. A smaller time step (0.001 s) helps avoids tunneling artifacts, but small intersections still occur.

We also test Bullet's *experimental* collision handling between arbitrary input triangle meshes directly (without convex decomposition proxies) and find it fails on almost all unit tests and the tests of Erleben [2018] using default parameters. We observe large amounts of energy injected into the system as an effect of position stabilization: once an intersection appears, the simulator quickly pulls the objects which produces large velocities.

Additionally, we note that Bullet successfully manages to prevent interpenetration in examples at larger dimensions. However, for smaller scenes we see severe interpenetrations even at small time steps. For example, we tested Bullet on a 0.1×-scale chain net scene, and observed severe interpenetration and instabilities even with $h = 10^{-4}$ s. This could certainly be related to Bullet's design, as stated in Bullet's documentation, being tuned to work on scenes with larger

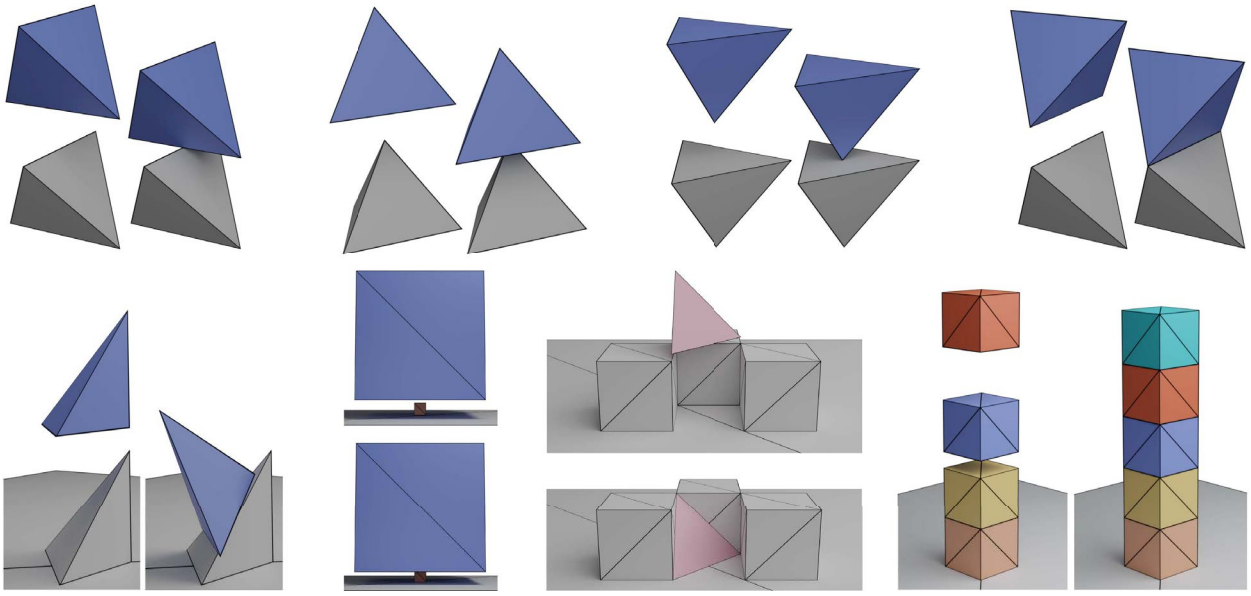


Figure 4.16: Unit tests. A set of unit test scenes used to benchmark the accuracy and robustness of each method. We show the initial configuration and the resulting simulation using our method.

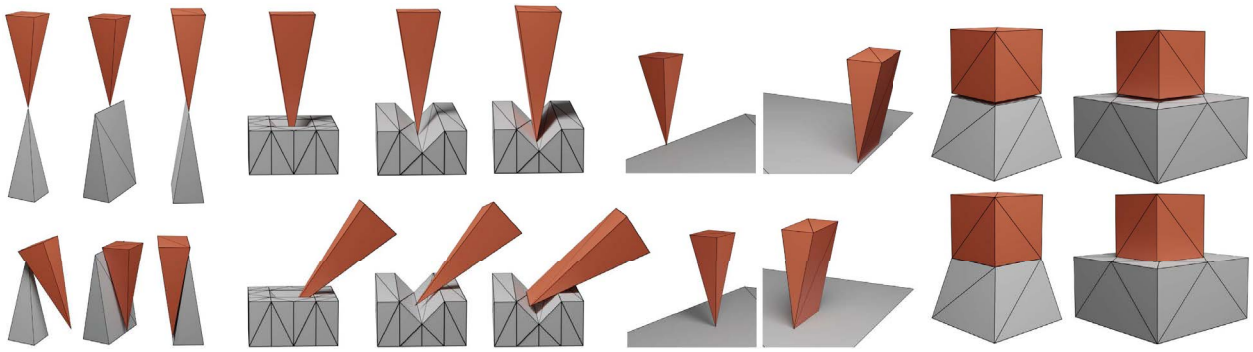


Figure 4.17: Erleben's degenerate test cases. Our method can easily handle the challenging degenerate cases proposed by Erleben [2018].

dimensions. Interestingly, we also note that Bullet simulates the 0.1×-scale chain net example roughly 8× slower than at the original scale, reflecting Bullet's parameters controlling collision detection and activation distance with respect to scale.

MuJoCo. MuJoCo works well on almost all unit tests and Erleben's test cases, without severe explosion or interpenetrations. Note that, for this method, we do not report small intersections that exist in almost all MuJoCo results as a failure since this is the expected behavior for the con-

tact resolution used in MuJoCo. For the tet-corner example, even with frame-rate time step size $h = 0.01$ s, MuJoCo successfully simulates the tetrahedron falling down into the tight space. However, we found that MuJoCo fails on all our large-scale examples independently from time step size. Nearly half of the examples crash the program, either because huge velocity or bounding boxes are detected (suggesting explosion), or the contact buffer is full and the slow progressive memory reallocation does not help. Similar to Bullet, we find that MuJoCo runs the 0.1×-scale much slower (more than 3×) comparing to the larger dimension counterpart. Compared to Bullet, MuJoCo is generally several times faster for the same time step sizes. We tried to avoid interpenetrations on the 4×4 chain example by (1) swapping integrators from Implicit Euler to RK4, (2) changing solver from Newton to PCG, and (3) increasing solver iteration from 100 to 1000. None of these changes avoided interpenetrations.

Chrono. Chrono provides two methods for rigid body contacts: smooth contacts (SMC) and non-smooth contacts (NSC). SMC uses a penalty-based formulation, so it is known to have intersections with large time steps or velocities. NSC uses a complementarity-based approach and is, therefore, more robust. We focus our benchmark on the NSC model. Chrono also provides several solver and time-stepper methods. We benchmark the Barzilai-Borwein solver and projected implicit Euler time-stepper as we found they are the most robust for a wide range of scenes and the documentation recommends them for "fast dynamics with hard (NSC) contacts and low inter-penetration" [Tasora et al. 2016].

Similar to Bullet, Chrono performs well on the unit tests and Erleben's test cases, but we find noticeable interpenetrations at large time steps ($h = 0.01$ s). In particular, sharp features and parallel edge-edge contacts (e.g., five-cube stack or parallel-edge tetrahedrons) are more prone to interpenetrations. Overall we find that Chrono is robust at smaller time steps, and only the five-cube stack requires a time step smaller than 0.001 s to avoid interpenetrations.

However, Chrono struggles in some of our more complex scenes. For example, the bolt scene

initially works as the bolt turns in the nut, but after a short time they intersect and the bolt stops moving. Testing with different time steps ($h = 10^{-2}, 10^{-3}$, and 10^{-4} s), we get the same results. In an effort to get the bolt to work we tested various parameters and discovered adjusting the scale of the scene resolves the problems. When we scale the scene by 10× (and so change the overall physical system), we find Chrono performs remarkably well and is able to simulate the bolt at a time step of 0.01 s without interpenetration. Avoiding this kind of unintuitive parameter tuning that is necessary to prevent intersections and produce plausible results is a motivation of our work.

Houdini RBD. Since Houdini RBD (not the binding to Bullet) is harder to script than the two former methods, we modeled only three scenes: five cubes, bolt, and wrecking ball. For the five cubes scene, the simulation quickly stabilizes without artifacts, but it fails on resting contacts after a few seconds, and the stack starts to collapse (even using a small time step of $h = 10^{-4}$ s). Improving over Bullet and MuJoCo, Houdini successfully simulates the bolt scene, in real physical dimensions (i.e., small since all units are in meters) without explosion. However, the bolt intersects with the nut even when the time step is set to $h = 10^{-4}$ s. Finally, for the wrecking ball scene, Houdini does not support a plane geometry composed of 2 triangles holding the large cube matrix, therefore we make the problem easier by using a built-in ground plane. Still, just like in the five cubes scene, the cube matrix collapses after becoming static (before being hit by the wrecking ball). For this scene, we further tested with a higher resolution for the signed distance field used in RBD for collision detection: However, the cube matrix still collapses.

Friction Tests. We compare the different friction models by placing a block on a slope at 26.565°, which has a critical value for the coefficient of friction at 0.5 (Figure 4.18). In our results, the block does not move for $\mu = 0.5$ and starts to slide at $\mu = 0.49$. Bullet is able to closely match the expected behavior: The block does not move for a value of $\mu \geq 0.505$. MuJoCo requires a value of $\mu = 0.9$ to prevent the block from sliding. Chrono perfectly matches the expected results

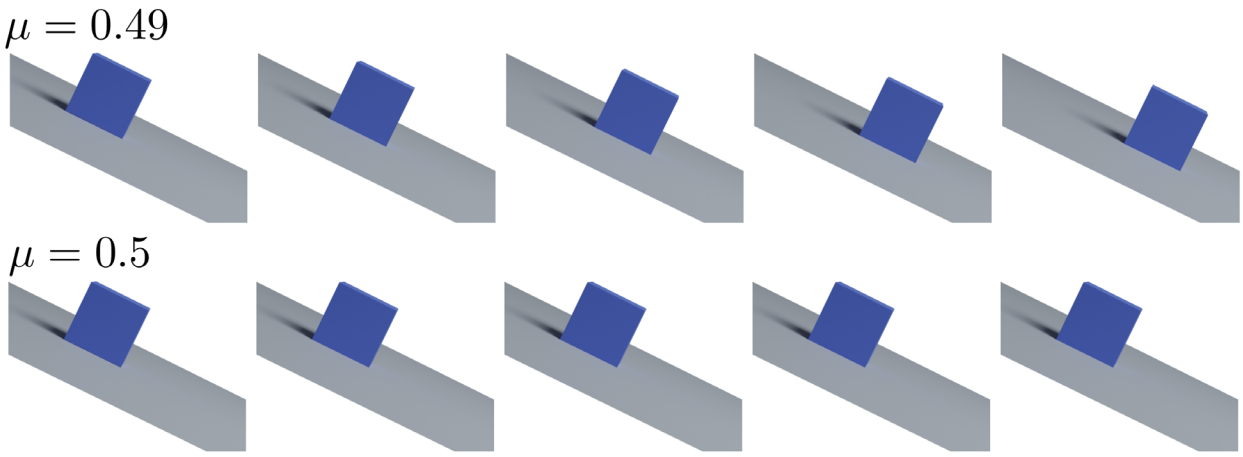


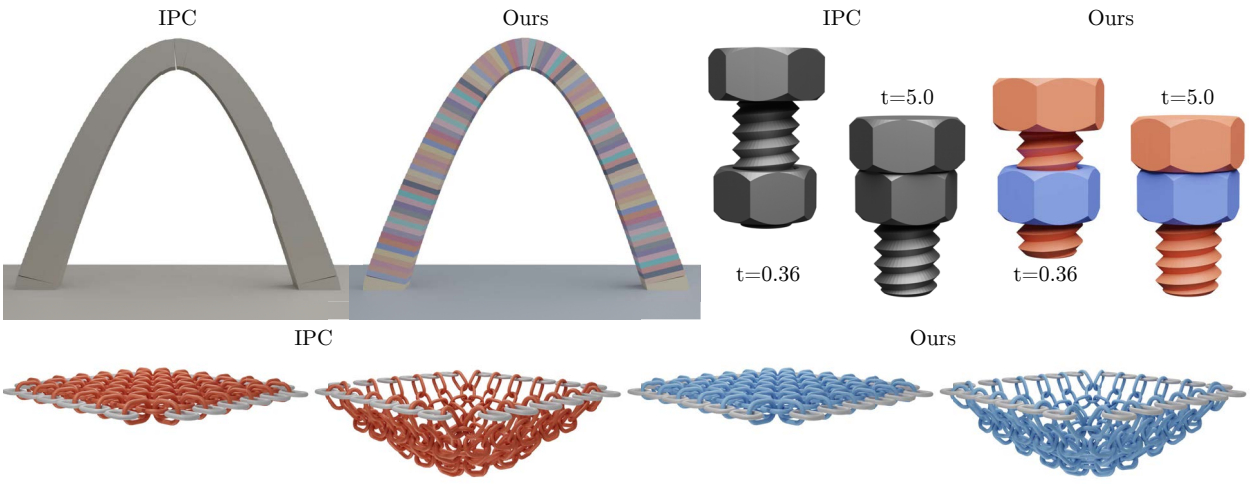
Figure 4.18: High school physics friction test. We perform a simple test of high school physics by placing a block on an inclined plane with a slope of 26.565°. For a value of $\mu \geq \tan(26.565°) \approx 0.5$, the friction force will counter the acceleration due to gravity. We accurately replicate this by showing for $\mu = 0.49$ the block slides and for $\mu = 0.5$ the block does not slide.

with a critical value of $\mu = 0.5$. Houdini's RBD requires a value of $\mu = 0.7$ to prevent the block from sliding.

For our arch test (Figure 4.11), Bullet's convex collision handling is able to reach a stable equilibrium, but for large time steps (0.01 s) the blocks intersect. Bullet's concave triangle mesh collision handling, experiences large "ghost" forces that cause it to collapse even for varying time step sizes (10^{-2}, 10^{-3}, and 10^{-4} s). With MuJoCo, Chrono, and Houdini the arch is unable to support itself as large intersections occur between the bottom blocks (tested with $h = 10^{-2}$, 10^{-3}, and 10^{-4} s).

4.6.1 IPC

While not designed for rigid body simulations, the IPC algorithm in Chapter 2 can handle very stiff materials, and it is thus possible to use it to approximate dynamic systems of rigid bodies. While the bodies are not exactly rigid when simulated with IPC, the major advantage is that restitution effects are directly simulated (while we do not account for them in our current rigid body formulation). The disadvantage is that the interior of the objects needs to be filled



Model ©YSoft be3D (bolt) under CC BY-SA 3.0.

Figure 4.19: IPC comparison. Comparison of the original volumetric, deformable IPC formulation (using material parameters for steel: Young's modulus $E = 200\,\text{GPa}$ and Poisson ratio Poisson's ratio $\nu = 0.3$) and our rigid body formulation.

with tetrahedra, increasing the solve time, especially for complex geometries. We show three representative scenes in Figure 4.19: in the arch, there is no need to insert any internal vertices and IPC is faster than the rigid version (2× slower), due to the cheaper linear CCD. On the bolt and chain-net scenes, the geometry is more complex, and the reduced set of coordinates of the rigid body formulation makes our algorithm faster (2.8 and 7.0×). In all scenes, the overall dynamic is very similar between the two formulations. We provide a more detailed comparison over a selection of nine scenes in Appendix D.4.

4.7 Discussion

We revisited the rigid body simulation problem focusing on robustness and automation. By introducing a new IP formulation for rigid body dynamics and a new conservative curved CCD formulation, we designed a system that can reliably simulate complex scenes, with large time steps, and without parameter tuning.

Table 4.1: Simulation statistics. for all scenes presented in Section 4.5. We report the number of bodies, number of primitives, simulation parameters, and the average timings and Newton iterations per timestep. All timings are generated on a machine with a 2x32-core 2.35 GHz AMD EPYC™ 7452 32-Core Processor with 1 TB of memory. Each simulation is limited to a maximum of 16 cores (* indicates up to 64 cores). The suffix ℓ indicates the value is relative to the world diagonal. We also report friction parameters and the Newton convergence tolerance. Please refer to Chapter 2 for full definition of these parameters.

Example	bodies	vertices	edges	faces	h (s)	d (m)	μ	ϵ_v (m/s)	max friction iterations	newton tol. (ϵ_d (m/s))	contacts avg. (per timestep)	contacts max. (per timestep)	memory (MB)	iterations (per timestep)	timing (s) (per timestep)
Expanding lock box	11	66K	66K	22K	0.01	10^{-5}	N/a			$10^{-2}\ell$	5K	9K	811	24.9	9.0
Bolt	2	3K	8K	5K	0.01	10^{-4}	N/a			$10^{-2}\ell$	332	759	86	8.3	3.5
Piston	4	6K	6K	2K	0.01	10^{-3}	N/a			$10^{-2}\ell$	370	1K	1323	9.9	1.5
Intermitten motion	5	2K	6 K	4K	0.01	10^{-4}	N/a			$10^{-2}\ell$	21	498	875	5.7	0.2
Bike chain*	138	48K	143K	96K	0.01	10^{-5}	N/a			$10^{-2}\ell$	6K	11K	12403	42.0	21.6
Punch	6	2K	7K	5K	0.01	10^{-4}	N/a			$10^{-2}\ell$	57	136	591	9.2	1.3
Punch (loose)	6	2K	7K	5K	0.01	10^{-4}	N/a			$10^{-2}\ell$	47	78	903	12.5	1.3
Codim. house of cards	18	80	116	56	0.01	10^{-3}	0.9	10^{-5}	1	$10^{-3}\ell$	78	204	1045	161.9	3.2
Codm. chain net	673	23K	25K	1K	0.01	10^{-3}	N/a			$10^{-2}\ell$	2K	2K	3234	130.5	20.5
Disconnected components	3	5K	2K	850	0.01	10^{-3}	0.1	10^{-3}	1	$10^{-2}\ell$	7	10	41	1.9	0.1
Spolling coin	2	134	389	258	10^{-4}	10^{-4}	0.2	10^{-5}	∞	$10^{-4}\ell$	12	84	229	3.8	0.01
Wrecking ball	575	8K	20K	13K	0.01	10^{-3}	N/a			$10^{-2}\ell$	4K	14K	1959	17.1	4.8
Anchor	23	9K	27K	18K	0.01	10^{-3}	N/a			$10^{-2}\ell$	267	1K	1178	21.4	3.0
Arch	102	812	2K	1K	0.01	10^{-3}	0.5	10^{-3}	1	$10^{-3}\ell$	649	695	1590	2.1	0.21
Lewis	7	64	149	98	0.01	10^{-3}	0.3	10^{-3}	1	$10^{-2}\ell$	50	55	26	2.7	0.02
Turntable ($\mu = 0.0$)	2	138	402	268	0.025	10^{-3}	0	10^{-5}	∞	$10^{-4}\ell$	9	15	178	2.3	0.004
Turntable ($\mu = 0.1$)	2	138	402	268	0.025	10^{-3}	0.1	10^{-5}	∞	$10^{-4}\ell$	2	13	284	2.9	0.003
Turntable ($\mu = 0.5$)	2	138	402	268	0.025	10^{-3}	0.5	10^{-5}	∞	$10^{-4}\ell$	7	14	255	5.9	0.01
Turntable ($\mu = 1.0$)	2	138	402	268	0.025	10^{-3}	1	10^{-5}	∞	$10^{-4}\ell$	10	10	246	9.1	0.01
3D packing*	10	7K	20K	13K	0.01	10^{-3}	N/a			$10^{-2}\ell$	184	456	8860	91.3	33.5

Limitations. Our method has three major limitations. (1) The robustness of the algorithm comes at a computational cost, our algorithm is (two to three orders of magnitude) slower than other rigid body simulators. (2) The current formulation does not preserve energy. (3) Our current formulation does not provide direct control for restitution.

While (1) is an intrinsic limitation, which could be ameliorated with more code optimizations or the use of GPU accelerators, (2) and (3) are very interesting venues for future work.

Future Work. Our work opens the door to robust rigid body simulation, over a wider range of geometries and contact scenarios. While our algorithm is slower than competing methods, our method requires no parameter tuning to generate feasible results, and therefore can be potentially used to generate simulation data in one shot for reinforcement learning in robotics. In that setting, it would be interesting to add support for articulated bodies, add support for accurate actuators,

and merge the deformable and rigid body formulation to allow robots to interact with deformable objects. For applications in graphics, it would be interesting to add additional collision primitives, such as spheres, capsules, and boxes, to lower the runtime in cases where geometrical accuracy is less important.

Concluding Remarks. To conclude, we believe our formulation will foster the development of a new family of robust rigid body simulations while supporting exciting simulation applications in graphics, robotics, and digital fabrication.

5 High-Order Incremental Potential Contact for Elastodynamic Simulation on Curved Meshes

5.1 Introduction

Elastodynamic simulation of deformable and rigid objects is used in countless algorithms and applications in graphics, robotics, mechanical engineering, scientific computing, and biomechanics. While the elastodynamic formulations used in these fields are similar, the accuracy requirements differ: while graphics and robotics applications usually favor high efficiency to fit within strict time budgets, other fields require higher accuracy. In both regimes, FE approaches based on a conforming mesh to explicitly partition the object volume are a popular choice due to their maturity, flexibility in handling non-linear material models and contact/friction forces, and convergence guarantees under refinement.

In a FE simulation, a set of elements is used to represent the computational domain and a set of basis functions are used within each element to represent the physical quantities of interest (e.g., the displacement in an elastodynamic simulation). Many options exist for both elements and bases. Due to the simplicity of their creation, linear tetrahedral elements are a common choice for the element shape. Similarly, linear Lagrangian functions (often called the hat functions) are

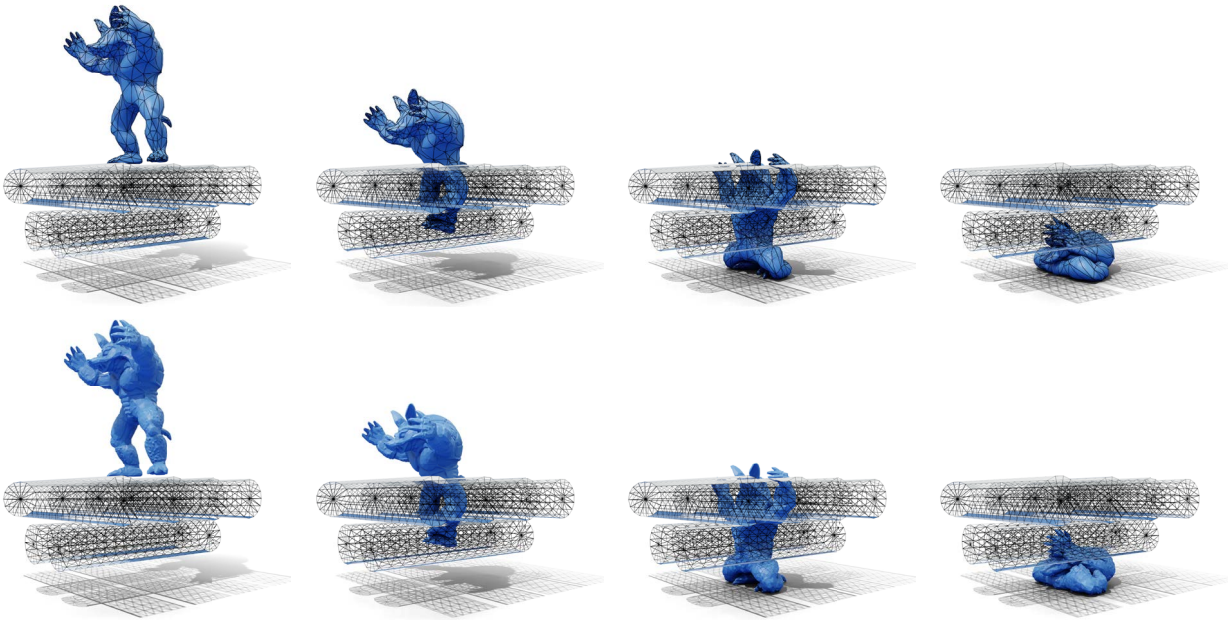


Figure 5.1: High-order armadillo-rollers. A simulation of an armadillo squished by rollers. We use a high-order volumetric mesh (top row) and deform it with quadratic displacement. To solve collision and compute contact forces, we use a dense linear surface mesh (bottom row) and transfer the deformation and contact forces between the two meshes.

often used to represent the displacement field. The linearity in both shape and basis leads to a major and crucial benefit for dynamic simulations: after the displacement is applied to the rest shape, the resulting mesh remains a piece-wise linear mesh. This is an essential property in order to robustly and efficiently detect and resolve collisions [Wang et al. 2021]. Collisions between arbitrary curved meshes or between linear meshes over curved trajectories are computationally expensive, especially if done in a conservative way [Ferguson et al. 2021].

However, these two choices are restrictive: meshes with curved edges represent shapes, at a given accuracy, with a lower number of elements than linear meshes, especially if tight geometric tolerances are required. Curved meshes are often favored over linear meshes in mechanical engineering [Hughes et al. 2005]. The use of linear bases, especially on simplicial meshes, is problematic as it introduces arbitrary stiffness (a phenomenon known as locking [Schneider et al. 2018]). Additionally, high-order bases are more efficient, in the sense that they provide the same

accuracy (compared to a reference solution) as linear bases for a lower running time [Babuška and Guo 1992; Schneider et al. 2022]. Elasto-static problems in computational fabrication (e.g., [Panetta et al. 2015]), mechanics, and biomechanics [Maas et al. 2012] often use high-order bases, but their use for dynamic problems with contact is very limited or the high-order displacements are ignored for contact purposes.

Contribution. We propose a novel elastodynamic formulation supporting both high-order geometry and high-order bases (Figure 5.1). Our key observation is that a linear transformation of the displacements degrees of freedom leads to linear trajectories of a carefully designed collision proxy. We use this observation to extend the recently proposed IPC formulation, enabling us to use both high-order geometry and high-order bases. Additionally, we can now use arbitrary collision proxies in lieu of the boundary of the FE mesh, a feature that is useful, for example, for the simulation of nearly rigid materials. To evaluate the effectiveness of our approach, we explore its use in graphics applications, where we use the additional flexibility to efficiently simulate complex scenes with a low error tolerance, and we show that our approach can be used to capture complex buckling behaviors with a fraction of the computational cost of traditional approaches. Note that in this work we focus on tetrahedral meshes, but there are no theoretical limitations to applying our method to hexahedral or other polyhedral elements.

Reproducibility. To foster further adoption of our method we release an open-source implementation based on PolyFEM [Schneider et al. 2019b] which can be found at polyfem.github.io.

5.2 Related Work

High-Order Contacts. Contact between curved geometries has been investigated in multiple communities, as the benefits of p-refinement (i.e., refinement of the basis order) for elasticity have been shown to transfer to problems with contact in cases where an analytic solution is

known, such as Hertzian contact [Aldakheel et al. 2020; Franke et al. 2010, 2008; Konyukhov and Schweizerhof 2009].

One of the simplest forms of handling contact, penalty methods [Moore and Wilhelms 1988; Terzopoulos et al. 1987] apply penalty force when objects contact and intersect. However, despite their simplicity and computational advantages, it is well known that the behavior of penalty methods strongly depends on the choice of penalty stiffness (and a global and constant in-time choice ensuring stability may not be possible). Li et al. [2020] propose IPC to address these issues, and we choose to use their formulation and benefit from their strong robustness guarantees.

Mortar methods [Belgacem et al. 1998; Hüeber and Wohlmuth 2006; Puso and Laursen 2004] are also a popular choice for contact handling, especially in engineering [Krause and Zulian 2016] and biomechanics [Maas et al. 2012]. Extensions to high-order non-uniform rational B-spline (NURBS) surfaces have also been proposed [Seitz et al. 2016]. Mortar methods require to (a priori) mark the contacting surfaces. A clear limitation of this method is that they cannot handle collisions in regions with more than two contacting surfaces or self-collisions. Li et al. [2020] provide a didactic comparison of the IPC method and one such mortar method ([Krause and Zulian 2016]). They show such methods enforce contact constraints weakly and therefore allow intersections (especially at large timesteps and/or velocities). Nitsche's method is a method for soft Dirichlet boundary conditions (eliminating the need to tune the penalty stiffness) [Nitsche 1971]. Stenberg [1998] and recent work [Chouly et al. 2022; Gustafsson et al. 2020] extend Nitsche's method to handle contacts through a penalty or mortaring method. While this eliminates the need to tune penalty stiffnesses, these methods still suffer from the same limitations as mortaring methods.

Another way to overcome the challenges with high-order contact is the use of a *third medium* mesh to fill the empty space between objects [Wriggers et al. 2013]. This mesh is handled as a deformable material with carefully specified material properties and internal forces which act in lieu of the contact forces. In this setting, high-order formulations using p-refinement have been

shown to be very effective [Bog et al. 2015]. Similar methods have been used in graphics (referred to as an "air mesh"), as a replacement for traditional collision detection and response methods [Jiang et al. 2017; Müller et al. 2015]. The challenge for these approaches is the maintenance of a high-quality tetrahedral mesh in the thin contact regions, a problem that is solved in 2D, but still open for tetrahedral meshes.

The detection and response to collisions between spline surfaces are major open problems in isogeometric analysis, where over a hundred papers have been published on this topic (we refer to Temizer et al. [2011] and Cardoso and Adetoro [2017] for an overview). However, automatic mesh generation for isogeometric analysis (IGA) is still an open issue [Schneider et al. 2021], limiting the applicability of these methods to simple geometries manually modeled, and often to surface-only problems.

In comparison, we introduce the first technique using the IPC formulation to solve elastodynamic problems with contact and friction forces on curved meshes using high-order elements. We also show that an automatic high-order meshing and simulation pipeline is possible when our algorithm is paired with [Jiang et al. 2021].

High-Order Collision Detection. IPC utilizes CCD to ensure that every step taken is intersection-free. The numerical exactness of CCD can make or break the guarantees provided by the IPC algorithm [Wang et al. 2021]. While several authors have proposed methods for collision detection between curved surfaces and nonlinear trajectories [Ferguson et al. 2021; Kry and Pai 2003; Nelson and Cohen 1998; Nelson et al. 2005; Snyder et al. 1993; Von Herzen et al. 1990], there still does not exist a method that is computationally efficient while being conservative (i.e. never misses collisions). Therefore, we are unable to utilize existing methods and instead, propose a method of coupling linear surface representations with curved volumetric geometry.

High-Order Bases. Linear FE bases are overwhelmingly used in graphics applications, as they have the smallest number of DOF per element and are simpler to implement. High-order bases

have been shown to be beneficial to animate deformable bodies [Bargteil and Cohen 2014], to accelerate approximate elastic deformations [Mezger et al. 2009], and to compute displacements for embedded deformations [Longva et al. 2020]. Higher-order bases have also been used in meshless methods for improved accuracy and faster convergence [Faure et al. 2011; Martin et al. 2010]. High-order bases are routinely used in engineering analysis [Jameson et al. 2002] where p-refinement is often favored over h-refinement (i.e., refinement of the number of elements) as it reduces the geometric discretization error faster and using fewer degrees of freedom [Babuska and Guo 1988; Babuška and Guo 1992; Bassi and Rebay 1997; Luo et al. 2001; Oden 1994].

We propose a method that allows using high-order bases within the IPC framework, thus enabling us to resolve the IPC contact model at a higher efficiency for elastodynamic problems with complex geometry, i.e. we can obtain similar accuracy as with linear bases with a lower computation budget. Additionally, our method allows us to explicitly control the accuracy of the collision approximation by changing the collision mesh sampling (Section 5.4).

High-order bases can be used as a reduced representation and the high-order displacements can be transferred to higher resolution meshes for visualization purposes [Suwelack et al. 2013]. We use this approach to extend our method to support arbitrary collision proxies, which enables us to utilize our method to accelerate elastodynamic simulations by sacrificing accuracy in the elastic forces.

Physically-Based Simulation. There is a large literature on the simulation of deformable and rigid bodies in graphics [Bargteil and Shinar 2018; Kim and Eberle 2022], mechanics, and robotics [Choi et al. 2021]. In particular, a large emphasis is on the modeling of contact and friction forces [Brogliato 1999; Kikuchi and Oden 1988; Stewart 2001; Wriggers 1995].

Longva et al. [2020] propose a method for embedding geometries in coarser FE meshes. By doing so they can reduce the complexity while utilizing higher-order elements to generate accurate elastic deformations. To apply Dirichlet boundary conditions they design the spaces such

that they share a common boundary. This scheme, however, cannot capture self-contacts without resorting to using the full mesh. As such they do not consider the handling of contacts. They do, however, suggest a variant of the Mortar method could be future work, but this has known limitations as outlined above. We do not provide a comparison against this method as it does not support contact, and adding contact to it is a major research project on its own, as discussed by the authors.

In our work, we build upon the recently introduced IPC [Li et al. 2020] approach, as it offers higher robustness and automation compared to traditional formulations allowing interpenetrations between objects. We review only papers using the IPC formulation in this section, and we refer to [Li et al. 2020] for a detailed overview of the state of the art.

Li et al. [2020] proposes to use a linear FE method to model the elastic potential, and an interior point formulation to ensure that the entire trajectory is free of collisions. While the approach leads to accurate results when dense meshes are used, the computational cost is high, thus stemming a series of works proposing to use reduced models to accelerate the computation. Li et al. [2021] propose Codimensional Incremental Potential Contact (C-IPC), a new formulation for codimensional objects is introduced that optionally avoids using volumetric elements to model thin sheets and rod-like objects. An acceleration of multiple orders of magnitude is possible for specific scenes where the majority of objects are codimensional. Ferguson et al. [2021] propose a formulation of IPC for rigid body dynamics, dramatically reducing the number of DOF but adding a major cost and complexity to the collision detection stage, as the trajectories spanned by rigid objects are curved.

Longva et al. [2020] demonstrate their ability to approximately model a rigid body using a single stiff element. This idea is further expanded upon by Lan et al. [2022a] who propose to relax the rigidity assumption: they use an affine transformation to approximate the rigid ones, thus reducing the problem of collision detection to a much more tractable linear CCD. Massive speedups are possible for rigid scenes, up to three orders of magnitude compared to the original

formulation. While these methods provide major acceleration for specific types of scenes, they are not directly usable for scenes with deformable objects.

Lan et al. [2021] proposes to use medial elastics [Lan et al. 2020], a family of reduced models tailored for real-time graphics applications. In their work, the shape is approximated by a medial skeleton which is used to model both the elastic behavior and as a proxy for collision detection. The approach can simulate deformable objects, however, it cannot reproduce a given polyhedral mesh and it is also specialized for medial elasticity simulations.

In our work, we enable the use of high-order meshes and high-order elements in a standard FE framework. Our approach decouples the mesh used to model the elastic potential from the mesh used for the contact and friction potentials, thus providing finer-grained control between efficiency and accuracy.

Convergence and use of C^0 Lagrangian Elements. Studies compare C^0 (p-FEM) and IGA bases' convergence under p-refinement [Sevilla et al. 2011], in the presence of contact [Seitz et al. 2016; Temizer et al. 2011] and in other settings such as electromechanics [Poya et al. 2018]. IGA bases have been shown, in specific problems with simple geometries, to have slightly higher accuracy compared to Lagrangian C^0 elements. In this work, we favor Lagrangian C^0 elements as IGA meshes are hard to generate for complex geometries and, additionally, some of their benefits are lost when non-regular grid meshes are required to represent complex geometry [Schneider et al. 2019a, 2022]. Our paper does not study the convergence of the method, we leave a convergence (h and p) study as future work jointly with a convergence study for the IPC contact model. Our goal is restricted to show that elastodynamic simulations with high-order geometry and bases are possible on complex geometry and provide a practical speedup over the linear geometry representation and linear bases that are commonly used in graphics applications.

5.3 IPC Overview

Our approach builds upon the IPC solver introduced in Chapter 2. In this section, we review the original formulation and introduce the notation.

Li et al. [2020] computes the updated displacements u^{t+1} of the objects at the next time step by solving an *unconstrained* non-linear energy minimization:

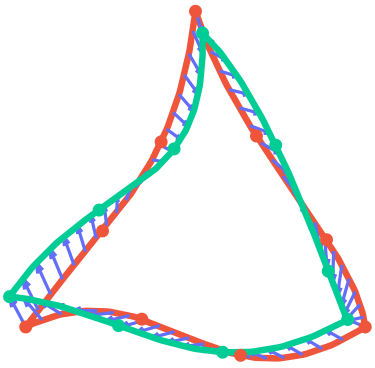
$$u^{t+1} = \underset{u}{\operatorname{argmin}}\ E(u, u^t, v^t) + B(x + u, \hat{d}) + D(x + u, \epsilon_v), \tag{5.1}$$

where x is the vertex coordinates of the rest position, u^t is the displacement at the current step, v^t the velocities, $E(u, u^t, v^t)$ is a time-stepping IP [Kane et al. 2000], B is the barrier potential, and D is the lagged dissipative potential for friction [Li et al. 2020]. The user-defined geometric accuracy $\hat{d}$ controls the maximal distance at which the barrier potential will have an effect. Similarly, the smooth friction parameter ϵ_v controls the smooth transition between static and dynamic friction. We refer to Li et al. [2020] for a complete description of the potentials, as for our purposes we will not need to modify them.

Solver and Line Search CCD. The advantage of the IPC formulation is that it is possible to prevent intersections from happening by using a custom Newton solver with a line-search that explicitly checks for collisions using a continuous collision detection algorithm [Provot 1997; Wang et al. 2021], while keeping the overall simulation cost comparable to the more established LCP based contact solvers [Li et al. 2020].

5.4 Method

We introduce an extension of IPC for a curved mesh $\mathcal{M} = (V_\mathcal{M}, T_\mathcal{M})$ where $V_\mathcal{M}$ and $T_\mathcal{M}$ are the nodes and volumetric elements of $\mathcal{M}$, respectivly. The formulation reduces to standard IPC when



$$\left(\sum_{i=1}^{n} u_i^{t+1}\varphi_i\right) - \left(\sum_{i=1}^{n} u_i^{t}\varphi_i\right)$$
$$= \sum_{i=1}^{n}(u_i^{t+1} - u_i^{t})\varphi_i = \sum_{i=1}^{n} \Delta u_i \varphi_i$$

Figure 5.2: Linearity of displacement update. Even with nonlinear bases φ_i, the update to displacement still constitutes a linear combination of nodal displacements. Therefore from a starting position (in red), the update to displacements of any point on the surface (in blue) is linear, and as such we need not use expensive nonlinear CCD.

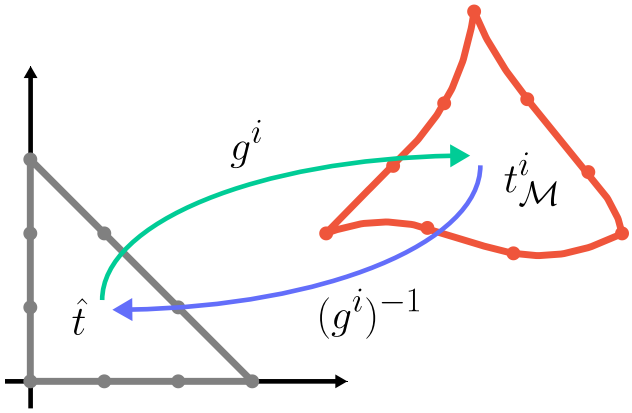


Figure 5.3: Geometric mapping. The geometric map g^i maps from the reference element $\hat{t}$ to the global positions of an element $t^i_{\mathcal{M}} = g^i(\hat{t})$. Additionally, g^i is bijective and it is easy to invert for linear meshes (barycentric coordinates), but requires a small nonlinear optimization for higher-order elements.

linear meshes and linear bases are used, but other combinations are also possible: for example, it is possible to use high-order bases on standard piece-wise linear meshes, as we demonstrate in Section 5.5.

We first introduce explicit definitions for functions defined on the volume and the contact surface corresponding to its boundary. Let $f_{\mathcal{M}} \colon \mathcal{M} \to \mathbb{R}^3$ be a volumetric function (in our case the volumetric displacement u) defined as

$$f_{\mathcal{M}} = \sum_{i=1}^{n} f^i_{\mathcal{M}} \varphi^i_{\mathcal{M}}, \tag{5.2}$$

where $\varphi^i_{\mathcal{M}}$ are the n FE bases defined on $\mathcal{M}$ and $f^i_{\mathcal{M}}$ their coefficient.

Similarly on the surface $\mathcal{S} = (V_{\mathcal{S}}, T_{\mathcal{S}})$ used for collision, with vertices $V_{\mathcal{S}}$ and triangular faces $T_{\mathcal{S}}$, we define $f_{\mathcal{S}} \colon \mathcal{S} \to \mathbb{R}^3$ (in our case the displacement u restricted to the surface) as

$$f_{\mathcal{S}} = \sum_{j=1}^{m} f^j_{\mathcal{S}} \varphi^j_{\mathcal{S}}, \tag{5.3}$$

where $\varphi^j_{\mathcal{S}}$ are the m FE bases defined on $\mathcal{S}$ and $f^j_{\mathcal{S}}$ their coefficient. We can now rewrite Equation (5.1) to make explicit that the potential E depends on $\mathcal{M}$, while B and D only depend on $\mathcal{S}$:

$$\begin{aligned} u^{t+1} = \operatorname*{argmin}_u \, & E_{\mathcal{M}}(u, u^t, v^t) + B_{\mathcal{S}}(V_{\mathcal{S}} + \Phi(u), \hat{d}) \\ & + D_{\mathcal{S}}(V_{\mathcal{S}} + \Phi(u), \epsilon_v), \end{aligned} \tag{5.4}$$

where $\Phi \colon \Theta_{\mathcal{M}} \to \Theta_{\mathcal{S}}$ is an operator where $\Theta_{\mathcal{M}} = \operatorname{span}\{\varphi^i_{\mathcal{M}}\}$ and $\Theta_{\mathcal{S}} = \operatorname{span}\{\varphi^j_{\mathcal{S}}\}$ that transfers volumetric functions on $\mathcal{M}$ to $\mathcal{S}$. In the context of [Li et al. 2020] (i.e., Equation (5.1)), Φ is a restriction of the volumetric function to its surface. While in general, Φ could be an arbitrary operator, IPC takes advantage of its linearity: if Φ is linear, then the trajectories of surface vertices in one optimization step of Equation (5.4) will be linear (Figure 5.2), and it is thus possible to use standard continuous collision detection methods [Provot 1997; Wang et al. 2021]. If Φ is nonlinear, for example in the rigid-body formulation introduced by Ferguson et al. [2021], the collision detection becomes considerably more expensive [Lan et al. 2022a].

We observe that arbitrary *linear* operators can be used for Φ, and note that increasing the order of the bases used to represent $f_{\mathcal{M}}$ and $f_{\mathcal{S}}$ does not affect the linearity of the operator. An additional advantage of this reformulation is that the space $\Theta_{\mathcal{S}}$ does not have to be a subspace of $\Theta_{\mathcal{M}}$. For example, the collision mesh can be at a much higher resolution than the volumetric mesh used to resolve the elastic forces (Section 5.5).

We first discuss how to build a linear operator Φ for high-order meshes, high-order elements, and arbitrary collision proxies, and we postpone the discussion on how to adapt the IPC algorithm to work with arbitrary Φ to Section 5.4.2.

5.4.1 Construction of Φ

We present two methods for constructing Φ: upsampling the surface of $\mathcal{M}$ to obtain a dense piecewise linear approximation of its boundary, which we use as $\mathcal{S}$ (Section 5.4.1.1), or using an arbitrary surface triangle mesh as $\mathcal{S}$ and determining closest point correspondences used to evaluate bases (Section 5.4.1.2). Our results in Section 5.5 show a mix of both approaches: Figures 5.4, 5.5, and 5.7 to 5.9 use an upsampling while Figures 5.1, 5.5, 5.6, and 5.9 to 5.12 use an arbitrary triangle mesh proxy.

Since Φ is a linear operator, a discrete function $f_{\mathcal{M}} \in \Theta_{\mathcal{M}}$ with coefficients $f^i_{\mathcal{M}}$ can be transferred to $f_{\mathcal{S}} \in \Theta_{\mathcal{S}}$ using its m coefficients $f^j_{\mathcal{S}}$ as

$$\mathbf{f}_{\mathcal{S}} = W\mathbf{f}_{\mathcal{M}},$$

where $\mathbf{f}_{\mathcal{M}}$ and $\mathbf{f}_{\mathcal{S}}$ are the stacked coefficients $f^i_{\mathcal{M}}$ and $f^j_{\mathcal{S}}$, respectively. The tetrahedron $t^i_{\mathcal{M}} \in T_{\mathcal{M}}$ of a high-order mesh $\mathcal{M}$ is defined as the image of the *geometric mapping* g^i applied to reference right-angle tetrahedron $\hat{t}$; that is

$$t^i_{\mathcal{M}} = g^i(\hat{t}).$$

On $\mathcal{S}$, the geometric map is a vectorial function and has the same form as Equation (5.3).

5.4.1.1 Upsampled linear boundary

To construct $\mathcal{S}$ we need to use the *geometric map* to find the initial vertex positions, while to define the operator to transfer functions from the volumetric mesh to $\mathcal{S}$ we will use the *basis functions* of $\mathcal{M}$.

Vertex Positions. Every vertex of the piece-wise linear approximation $v_S^j \in V_S$ has coordinates $\hat{v}^j$ in the reference tetrahedron of t_M^i, so its global coordinates can be computed as

$$v_S^j = g^i(\hat{v}^j),$$

and stacked into the vector V_S used in Equation (5.4).

Transfer. To construct the linear operator Φ encoded with the matrix W transferring from a higher-order polynomial basis on the boundary of $\mathcal{M}$ to the piecewise linear approximation $\mathcal{S}$, we observe that, since $\mathcal{S}$ is an upsampling of $\mathcal{M}$, we can use $\hat{v}^j$ to directly evaluate the bases of $\mathcal{M}$ (for all non-zero bases) and use them as a weight to transfer the function from $\mathcal{M}$ to $\mathcal{S}$ and define

$$W_{ji} = \varphi_M^i(\hat{v}^j),$$

which is a linear operator, independent of the degree of the basis functions.

5.4.1.2 Arbitrary Triangle Mesh Proxy

The same construction applies to arbitrary mesh proxies (e.g., Figure 5.1), but we need to compute $\hat{v}^j$ for every vertex. When $\mathcal{M}$ is linear we can simply compute $\hat{v}^j$ as the barycentric coordinates of the closest tetrahedron in $\mathcal{M}$, but when $\mathcal{M}$ is nonlinear we use an optimization to invert g^i [Suwelack et al. 2013]. However, unlike Suwelack et al. [2013], we found that using a normal field to define correspondences is fragile when the surfaces have a very different geometric shape, so we opt for a simpler formulation based on distances.

Algorithm 5.1 outlines our method for computing Φ for an arbitrary triangle proxy. Namely, given a volumetric mesh $\mathcal{M}$ and an arbitrary triangle mesh $\mathcal{S}$ we do not have the pre-image under the geometric mapping of the vertices $v_S^j \in V_S$, so we compute one by determining the closest element in $\mathcal{M}$ to v_S^j and use an optimization to compute the inverse geometric mapping to obtain

Algorithm 5.1 Construct $\Phi = W\mathbf{f}_{\mathcal{M}}$ for arbitrary triangle mesh

procedure ConstructW($\mathcal{M}, \mathcal{S}$)
  $W \leftarrow \mathbf{0}$ ▷ $W \in \mathbb{R}^{m \times n}$
  $\widetilde{\mathcal{M}} \leftarrow$ linearize($\mathcal{M}, 4$) ▷ 4 linear tetrahedra per curved tet
  for $v_{\mathcal{S}}^j \in V_{\mathcal{S}}$ **do**
    $\boxdot \leftarrow$ inflate(AABB($v_{\mathcal{S}}^j$), 10^{-3})
    while [**do***n* = 3 in our examples]$| \boxdot \cap \widetilde{\mathcal{M}}| < n$
      $\boxdot \leftarrow$ inflate($\boxdot$, 10%)
    for $t_{\mathcal{M}}^i \in (\boxdot \cap \mathcal{M})$ **do**
      $\tilde{b}_i \leftarrow$ BC($v_{\mathcal{S}}^j$, linearize($t_{\mathcal{M}}^i$)) ▷ barycentric coords.
      $\hat{v}_i^j \leftarrow \operatorname{argmin}_{\hat{v}} \ \|g^i(\hat{v}) - v_{\mathcal{S}}^j\|_2^2$ ▷ limited-memory BFGS (L-BFGS) with $\hat{v}_0 = \tilde{b}_i$
    $i^* \leftarrow \operatorname{argmin}_i \ \|\hat{v}_i^j\|_1$ ▷ Closest to the interior
    $\hat{v}^j = \hat{v}_{i^*}^j$ ▷ pre-image of $v_{\mathcal{S}}^j$
    $W_{ji} = \varphi_{\mathcal{M}}^i(\hat{v}^j)$
  return W

the coordinates $\hat{v}^j$. This procedure only needs to be performed once because W depends only on the rest geometry.

5.4.2 Gradient and Hessian of Surface Terms

Adapting IPC to work with arbitrary linear Φ mapping requires only changing the *assembly* phase, which requires gradients and Hessian of the surface potentials. Similar to IPC, we use Newton's method to minimize the newly formulated potential in Equation (5.4), and we thus need its gradient and Hessian.

For a surface potential $B_{\mathcal{S}}(V_{\mathcal{S}} + \Phi(u), \hat{d})$ and transfer

$$\Phi(u) = \Phi\left(\sum_{i=1}^{n} u_i \varphi_{\mathcal{M}}^i\right) = \sum_{j=1}^{m} (W\mathbf{u})_j \varphi_{\mathcal{S}}^j,$$

where $\mathbf{u}$ is the *vector* containing all the coefficients u_i; we use the definition of W to express the

gradient of the barrier (or the friction) potential as

$$\begin{aligned}\nabla_u B_S(V_S + \Phi(u), \hat{d}) &= \nabla_u (V_S + \Phi(u)))^\top \nabla_{S_u} B_S(\mathcal{S}_u, \hat{d}) \\ &= \nabla_u (V_S + (W\mathbf{u}))^\top \nabla_{S_u} B_S(\mathcal{S}_u, \hat{d}) = W^\top \nabla_{S_u} B_S(\mathcal{S}_u, \hat{d}),\end{aligned}$$

where $\mathcal{S}_u = V_S + \Phi(u)$. The Hessian is computed similarly

$$\nabla_u^2 B_S(V_S + \Phi(u), \hat{d}) = W^\top [\nabla_{S_u}^2 B_S(\mathcal{S}_u, \hat{d})] W.$$

The formulas for $\nabla_{S_u} B_S(\mathcal{S}_u, \hat{d})$, $\nabla_{S_u} D_S(\mathcal{S}_u, \epsilon_v)$, and their Hessians are the same as in [Li et al. 2020], thus requiring minimal modifications to an existing implementation. As in [Li et al. 2020], we mollify the edge-edge distance computation to avoid numerical issues with nearly parallel edges.

5.5 Results

All experiments are run on individual nodes of an HPC cluster each using two Intel Xeon Platinum 8268 24C 205W 2.9 GHz Processors and 16 threads. All results are generated using the PolyFEM library [Schneider et al. 2019b] coupled with the IPC Toolkit [Ferguson et al. 2020], and use the direct linear solver Pardiso [Alappat et al. 2020; Bollhöfer et al. 2019, 2020]. We use the notation P_n to define the FE bases order (e.g., P_2 indicates quadratic Lagrange bases) and all our curved meshes are quartic. All simulation parameters and a summary of the results can be found in Table 5.1 and 5.2, respectively.

5.5.1 Test Cases

Bending beam. We first showcase the advantages of high-order bases and meshes. Figure 5.4 shows that linear bases on a coarse mesh introduce artificial stiffness and the result is far from

Figure 5.4: Bending beam. Squared-section coarse beam pressed by two planes. Linear elements exhibit artificial stiffness as they cannot bend. The reference P_1 solution and P_3 are rendered in isolation on the right. The results are indistinguishable, but P_3 is an order of magnitude faster.

the reference (a dense P_1 mesh). As we increase the order, the beam bends more. Using P_3 on such a coarse mesh leads to results indistinguishable from the reference at a fraction of the cost. We also compare the results of a higher resolution P_1 mesh with a limited time budget. That is, the number of elements is chosen to produce a similar running time as the P_3 results (1,124 tetrahedra compared to 52 in the coarse version). Even in this case, the differences are obvious and far from the expected results.

Bouncing ball. Figure 5.5 shows the movement of the barycenter of a coarse bouncing sphere on a plane. When using linear bases on the coarse mesh, the ball tips over and starts rolling as the geometry is poorly approximated (yellow line). Replacing the coarse collision mesh using our method (blue line) improves the results for a small cost (125 frames/s versus 83.3 frames/s); however, since the sphere boundary is poorly approximated and the bases are linear, the results are still far from the accurate trajectory (green line). Finally, replacing $\mathcal{M}$ with a curved mesh and using P_2 bases leads almost to the correct dynamics (red line) while maintaining a real-time simulation (38.4 frames/s). As a reference, the dense P_1 linear mesh (green line) runs at 3.9 frames/s.

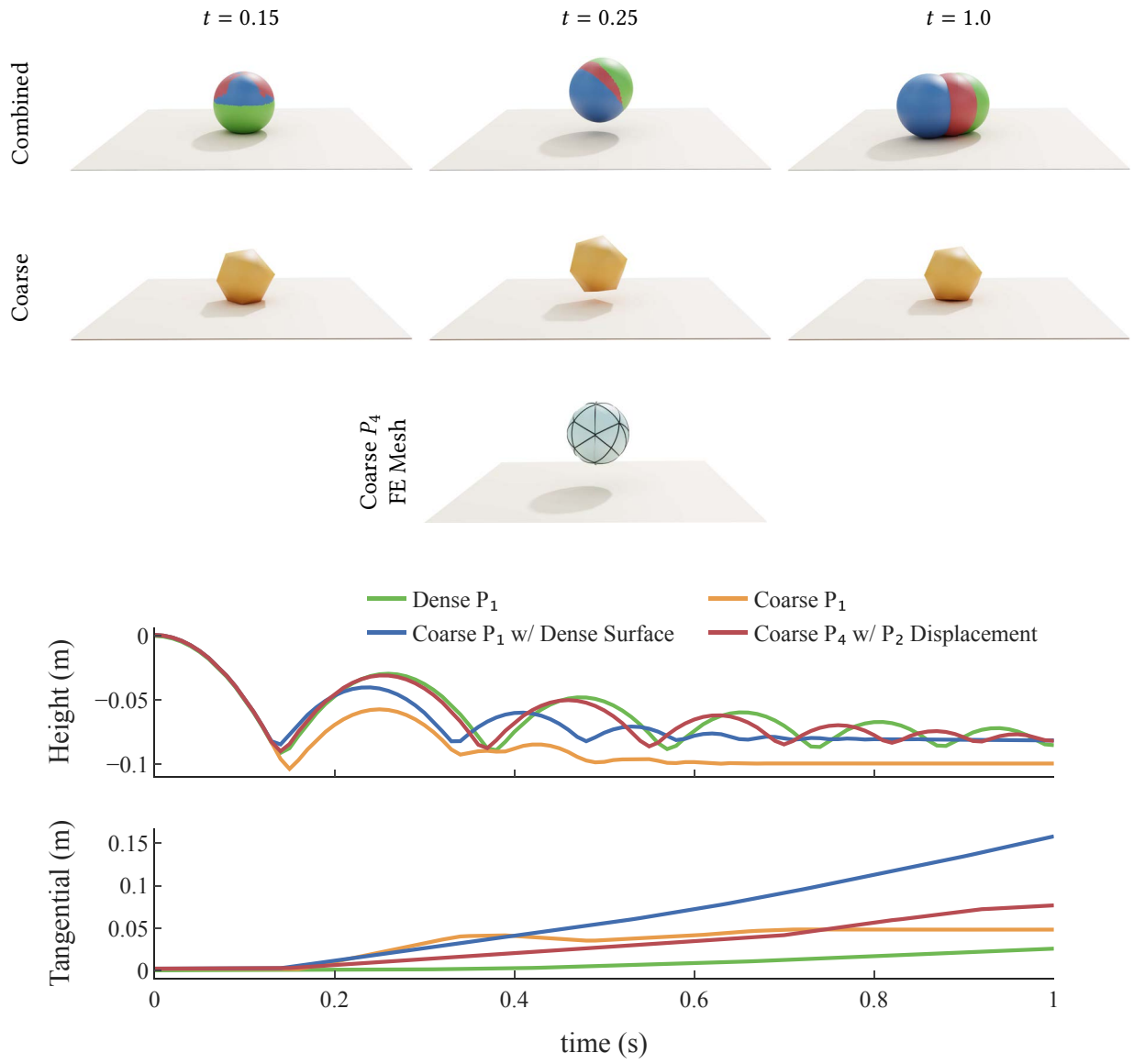


Figure 5.5: Bouncing ball. Simulation of a bouncing sphere on a plane. The yellow image and line are the baseline, a coarse linear mesh with linear displacement. The results can be improved using our method and replacing $\mathcal{S}$ with a dense sphere in blue. When using a high-order mesh with P_2 displacement, red, the results are similar to the dense linear simulation in green.

Rolling ball. Figure 5.6 shows our method is able to maintain purely tangential friction forces on the FE mesh while rolling a ball down a slope. The baseline spherical FE mesh (8.8K P_1 tetrahedra) and our method using a cube FE mesh (26 P_1 tetrahedra), both using the same collision geometry, produce very similar dynamics, but our method is 7.5× faster. However, while the

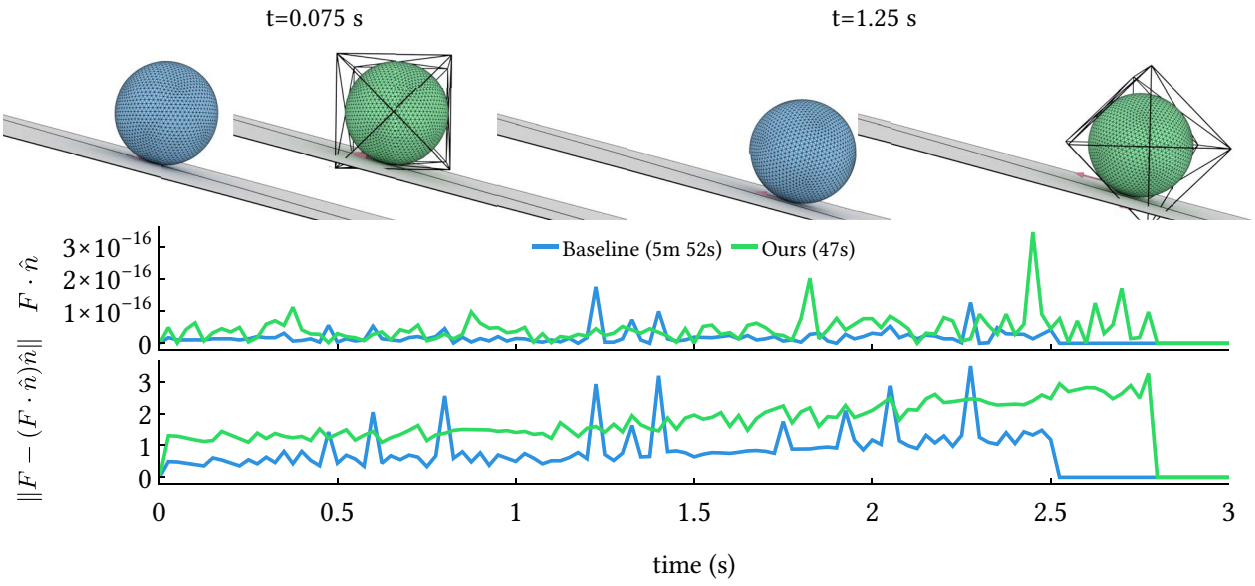


Figure 5.6: Rolling-ball. We demonstrate a ball rolling down a slope, while maintaining non-slip rolling contact, produces purely tangential friction forces on the FEM mesh. Our method uses a symmetric cube mesh (black wireframe) as the FE mesh and a high-resolution sphere (green) as the collision mesh. The friction forces on the FE mesh are shown as pink arrows. We plot the out-of-plane friction force ($F \cdot \hat{n}$) and norm of the in-plane friction force ($\|F - (F \cdot \hat{n})\hat{n}\|$). Compared to a high-resolution baseline, the out-of-plane error shows negligible differences but the in-plane force is around 2× greater. This is due to the increased numerical stiffness of our course mesh leading to less localized deformation, smaller distances, and, ultimately, a larger normal force.

ball's material is stiff ($E = 10^9$ Pa), it is not rigid, so the baseline model deforms slightly at the point of contact. Our model exhibits extra numerical stiffness from the large linear elements and so deforms less. This results in a 5% difference on average in the minimum distance which translates to a normal force (and ultimately friction force) that is 2× greater. This inaccuracy is a limitation of using such a course FE mesh within our framework.

5.5.2 Examples

Mat twist. We reproduce the mat twist example in [Li et al. 2020] using a thin linear mesh $\mathcal{M}$ with 2K tetrahedra and simulate the self-collisions arising from rotating the two sides using a collision mesh $\mathcal{S}$ with 65K vertices (Figure 5.7). Simulating this result using standard IPC on the coarse (left) is fast but leads to visible artifacts; by using P_2 bases for displacements the results are

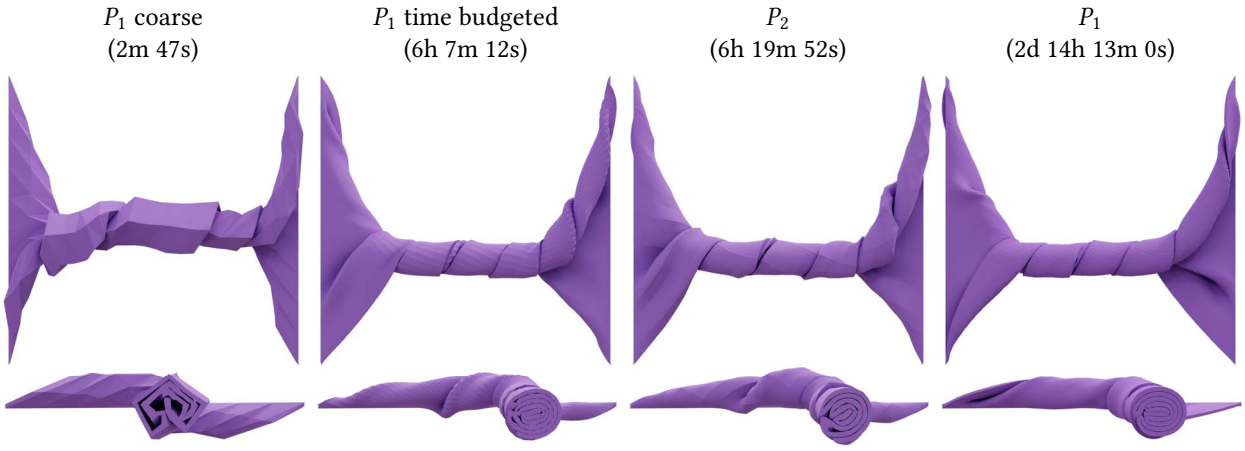


Figure 5.7: Mat-twist. Simulation of twisting for different bases' order and mesh resolutions. The cross-section (bottom row) shows that the coarse linear mesh (left) has huge artifacts. The coarse P_2 bases (middle-right) produce smooth results similar to a dense mesh (right) for a tenth of the time. A "time-budgeted" version shows similar results but exhibits checker patterns around the folds.

smooth and the simulation is faster (91 s/frame). For reference, a finer linear solution with more elements, to get a result similar to ours but only using linear elements, requires 230K elements and a runtime 10× higher.

We find a P_1 mesh with 51K tetrahedra (25× the number used in the P_2 variant) that produces a similar running time. The P_2 collision mesh uses 3.5× more triangles leading to 3.1× slower collision detection while the linear solver for the P_1 mesh is only 2.2× slower. This results in similar dynamics and final state (see Figure 5.7) with some notable differences around the folds of the mat.

Microstructure. In Figure 5.8, we simulate the compression of an extremely coarse (6K P_4 tetrahedra) curved microstructure mesh from [Jiang et al. 2021]. We upsample its surface to generate a collision mesh with 143K triangles. We demonstrate our method's ability to simulate *anisoparametric* scenarios (i.e., the shape and basis functions differ) by using P_1 and P_2 bases. In this case, both simulations take a similar amount of time (6h 34m 9s versus 6h 4m 48s).

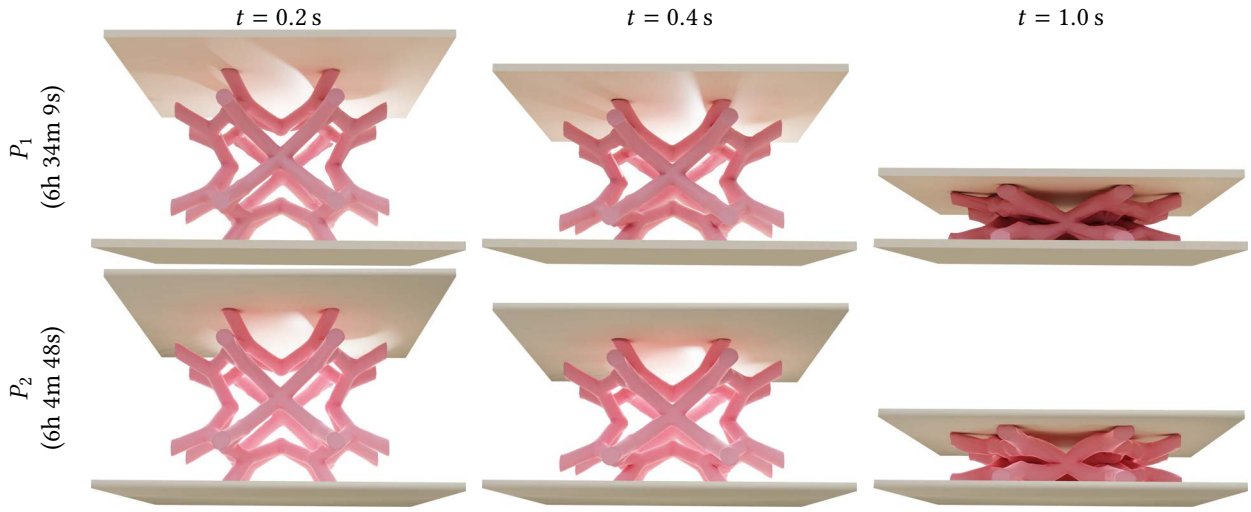


Figure 5.8: Microstructure. Compression of a curved microstructure using linear and quadratic bases. While the choice of bases only leads to marginal running time savings, it demonstrates our method's ability to simulate anisoparametric scenarios where the P_4 shape functions differ from the P_1/P_2 bases.

Armadillo on a Roller. In Figure 5.9, we replicate the armadillo roller from [Verschoor and Jalba 2019] and use fTetWild [Hu et al. 2020] to generate $\mathcal{M}$ with 1.8K tetrahedra (original mesh has 386K). With our method, we combine $\mathcal{M}$ with the original surface with 21K faces with linear element and obtain a speedup of 60× (row*). We used [Jiang et al. 2021] to generate a coarse curved mesh (with only 4.7K tetrahedra) and use an optimization to invert the geometric mapping and simulate the result using P_2, this leads to a simulation 30× faster (row†). Finally, we upsampled the surface of the curved mesh to generate a new collision mesh $\mathcal{S}$ with 20K faces, this simulation is only 8× faster (row‡).

Trash-compactor. We reproduce the trash compactor from [Li et al. 2020] using a coarse mesh $\mathcal{M}$ with 21K tetrahedra and compress it with five planes. Since the input mesh is already coarse and the models have thin features in the tentacles, we use fTetWild to generate a coarser mesh with 3.5K tetrahedra. Using this mesh with P_1 displacements while using the same surface mesh for collisions provides a 2.5× speedup. Since both coarse and input meshes have similar resolution, using P_2 leads to a more accurate but much slower (around 10×) result as the number of DOF for P_2 is similar to the denser mesh but with 5× the number of surface triangles.

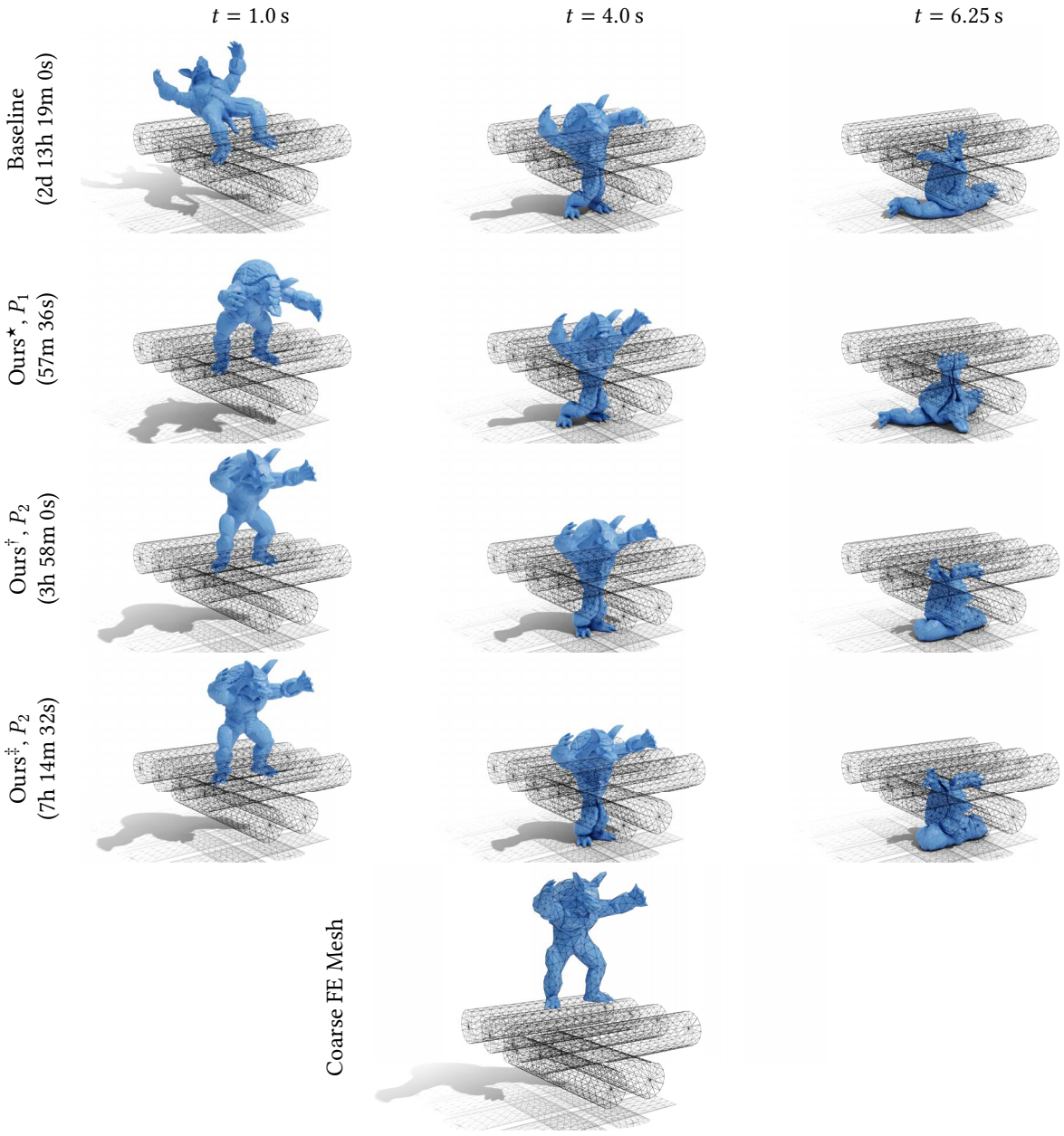


Figure 5.9: Armadillo-rollers. Armadillo roller simulation for the different variants of our method. Ours* uses a coarse linear mesh with linear displacement and the original geometry for the collision. Ours† uses a curved mesh with P_2 displacement and an upsampled geometry for the collision. Ours‡ uses a curved mesh with P_2 displacement and the original geometry for the collision.

5.5.3 Extreme Coarsening

Nut and Bolt. As mentioned in Section 5.4, our method can be used with linear meshes and linear bases. This is best suited to stiff objects where the deformation is minimal. Figure 5.11

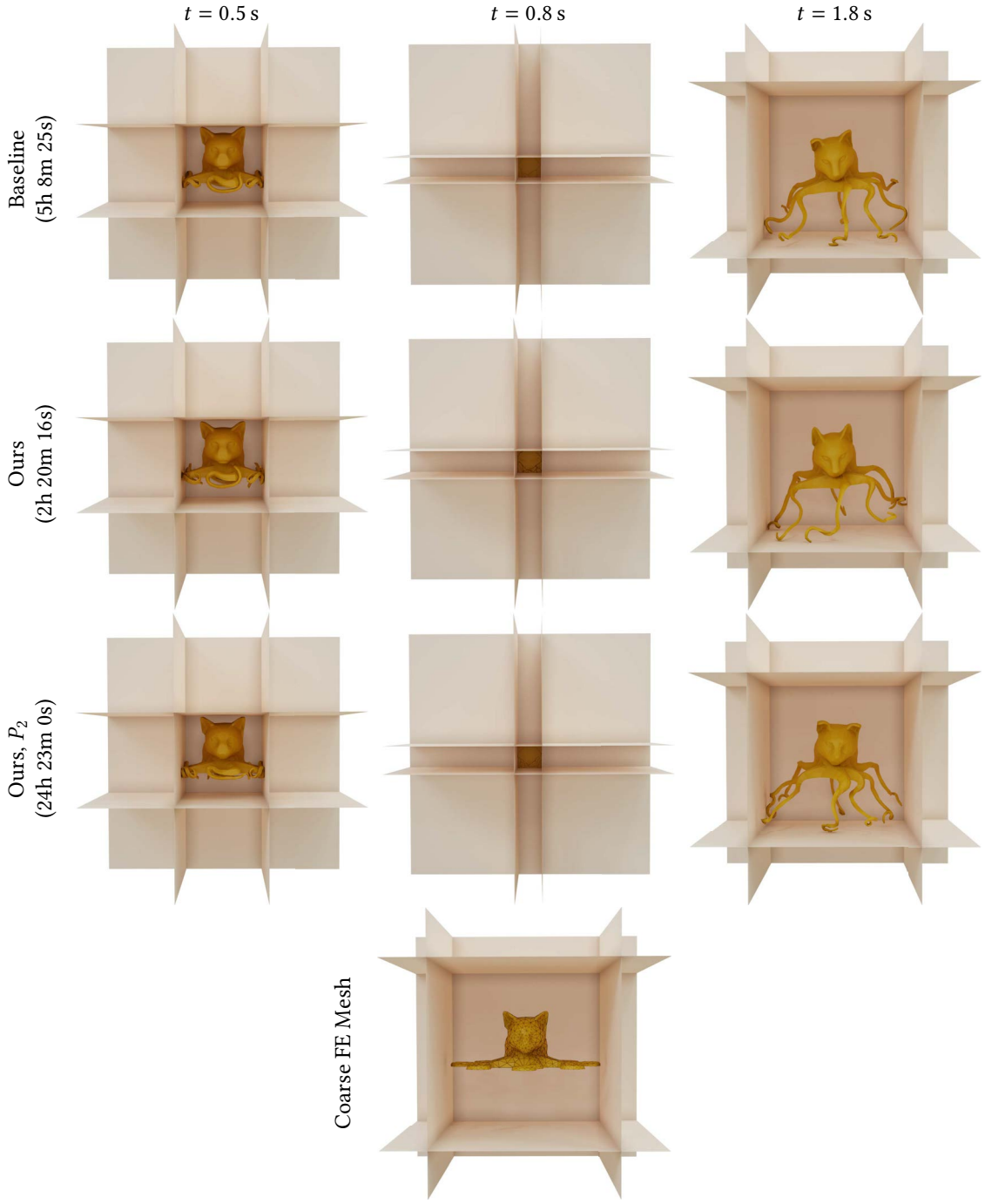
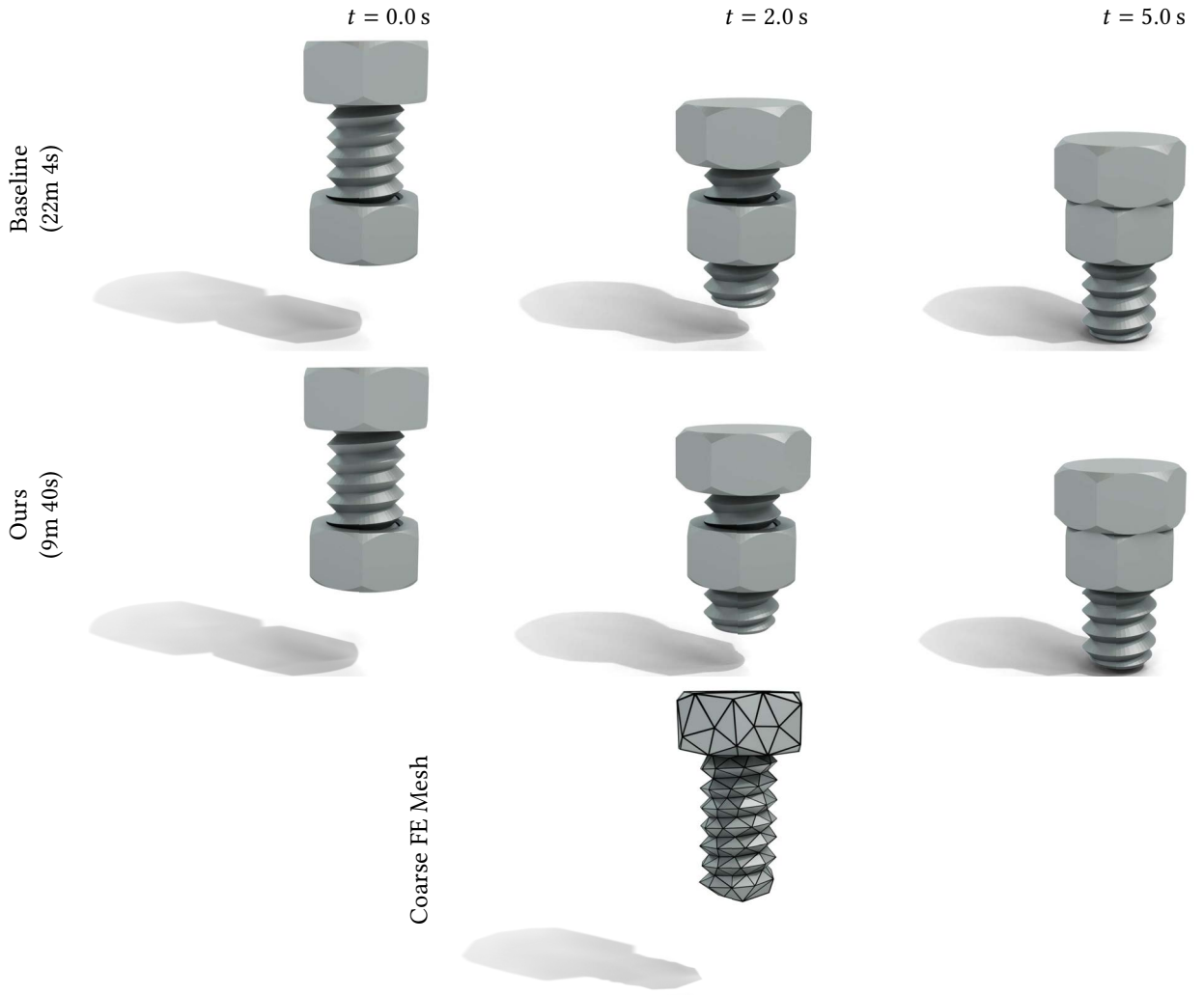


Figure 5.10: Trash compactor. The Octocat model is compressed by five planes. Using the original input mesh (top) is two times slower than using our method with linear elements (middle). Since we cannot coarsen the input too much without losing the tentacles, using P_2 leads to longer running times and similar results (bottom).



Model ©YSoft be3D under CC BY-SA 3.0.

Figure 5.11: Nut-and-bolt. Simulation of a bolt rotating into a bolt under gravity. Directly meshing the input mesh (top) generate similar results as using our method with a coarse simulation mesh (right).

shows an example of a nut sliding inside a bolt, since both materials are stiff ($E = 200\,\text{GPa}$), we coarsen $\mathcal{M}$ using fTetWild [Hu et al. 2020] from 6K tetrahedra and 1.7K vertices to 492 and 186, respectively. This change allows our method to be twice as fast without visible differences.

Balancing Armadillo. When generating a coarse mesh $\mathcal{M}$ the center of mass and mass of the object might change dramatically. Figure 5.12 shows that the coarse mesh cannot balance anymore as the center of mass is outside the contact area. To prevent this artifact, similarly

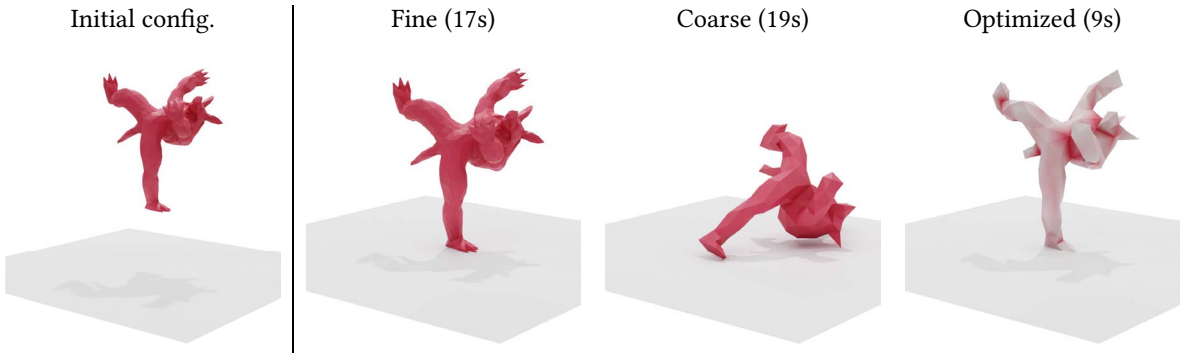


Figure 5.12: Balancing armadillo. We simulate the dancing armadillo from [Prévost et al. 2013] falling on a plane (left). The coarse model (middle) tips over because the center of mass falls outside the foot. We optimize the density (shown in red) to match the input center of mass and the armadillo is balanced (right). Differences in running time can be attributed to the different dynamics (i.e., the coarse model experiences more contacts when it falls over).

to [Prévost et al. 2013], we modify the density (in red in the third figure) of the material to move the center of mass.

5.6 Discussion

We introduce a robust and efficient simulator for deformable objects with contact supporting high-order meshes and high-order bases to simulate geometrically complex scenes. We show that there are major computational advantages in increasing the order of the geometric map and bases and that they can be used in the IPC formulation with modest code changes.

Limitations. At a high level, we are proposing to use p-refinement for elasticity, coupled with h-refinement approach for contacts, to sidestep the high computational cost of curved continuous collision detection. The downside of our approach is that our contact surface is still an approximation of the curved geometry, and while we can reduce the error by further refinement, we cannot reduce it to zero. While for graphics applications this is an acceptable compromise, as the scene we use for collision is guaranteed to be collision-free and we inherit the robustness properties of the original IPC formulation, there could be engineering applications where it is

important to model a high-order surface *exactly*. In this case, our approach could not be used as we might miss the collisions of the curved FE mesh.

A second limitation of our approach is that the definition of a robust, guaranteed positivity check for high-order elements is still an open research problem [Johnen et al. 2013]. In our implementation, we check positivity only at the quadrature points, which is a reasonable approximation but might still lead to unphysical results as the element might have a negative determinant in other interior points.

While our method for mapping between an arbitrary triangle mesh proxy and the curved tetrahedral mesh works well enough for the examples shown in this paper, it is not a robust implementation, as the closest point query can lead to wrong correspondences. In the future, it will be interesting to explore the use of bijective maps between the two geometries to avoid this issue (for example by using the work of Jiang et al. [2020]).

Our choice of Φ is not unique as there are a large number of basis functions to choose from. We explored other options such as mean value coordinates and linearized L2-projection, but we found their global mappings produce dense weight matrices. This results in slower running times with only minor quality improvements. A future direction might be the exploration of more localized operators such as bounded bi-harmonic weights [Jacobson et al. 2011].

Future Work. Beyond these limitations, we see three major avenues for future work: (1) existing curved mesh generators are still not as reliable in producing high-quality meshes as their linear counterparts: more work is needed in this direction, and our approach can be used as a testbed for evaluating the benefits curved mesh provides in the context of elastodynamic simulations, (2) our approach could be modified to work with hexahedral elements, spline bases, and isogeometric analysis simulation frameworks, and (3) we speculate that integrating our approach with high-order time integrators could provide additional benefits for further reducing numerical damping and we believe this is a promising direction for a future study.

Table 5.1: Simulation parameters used in the results. For each example, we report the time step size (h), density (ρ with * indicating multi-density), Young's modulus (E), Poisson ratio (ν), barrier activation distance ($\hat{d}$), coefficient of friction (μ), friction smoothing parameter (ϵ_v), maximum friction iterations, and Newton tolerance. For all examples, we use implicit Euler time integration and the Neo-Hookean material model.

Scene	h (s)	ρ (kg/m^3), E (Pa), ν	$\hat{d}$ (m)	μ, ϵ_v (m/s), friction iters.	Newton tol. (m)
Armadillo-rollers (Figures 5.1 and 5.9)	0.025	1e3, 5e5, 0.2	1e-3	0.5, 1e-3, 1	1e-3
Bending beam (Figure 5.4)	0.1	1e3, 1e7, 0.4	1e-3	0.5, 1e-3, 10	1e-5
Bouncing ball (Figure 5.5)	0.001	700, 5.91e5, 0.45	1e-3	0.2, 1e-3, 1	1e-12
Mat-twist (Figure 5.7)	0.04	1e3, 2e4, 0.4	1e-3	-	1e-5
Microstructure (Figure 5.8)	0.01	1030, 6e5, 0.48	1e-5	0.3, 1e-3, 1	1e-4
Trash-compactor (Figure 5.10)	0.01	1e3, 1e4, 0.4	1e-3	-	1e-5
Nut-and-bolt (Figure 5.11)	0.01	8050, 2e11, 0.28	1e-4	-	1e-5
Balancing armadillo (Figure 5.12)	0.1	$1e3^\star$, 1e11, 0.2	1e-5	0.1, 1e-3, 20	1e-5
Rolling ball (Figure 5.6)	0.025	1e3, 1e9, 0.4	1e-3	1.0, 1e-3, ∞	1e-5

Our approach is a first step toward the introduction of high-order meshes and high-order FEM in elastodynamic simulation with the IPC contact model, and we believe that our reference implementation will reduce the entry barrier for the use of these approaches in industry and academia.

Table 5.2: Summary of results shown in Section 5.5. For each example, we report the number of tetrahedra (#T) used for elasticity, the number of surface triangles (#F) used for collision processing, and the total running time of the simulation. Names correspond to the same given in each figure.

Scene		#T	#F	Running Time
Armadillo-rollers (Figure 5.9)	Baseline	386K	24K	2d 13h 19m 00s
	Ours*, P_1	1.8K	24K	57m 36s
	Ours†, P_2	4.7K	23K	3h 58m 00s
	Ours‡, P_2	4.7K	24K	7h 14m 32s
Bending beam (Figure 5.4)	P_1 coarse	48	72	15s
	P_1 reference	25K	4.4K	7m 43s
	P_2	1.1K	2.8K	32s
	P_3	48	5.5K	58s
	P_1 time budgeted	48	5.5K	57s
Bouncing ball (Figure 5.5)	Dense P_1	8.8K	5.1K	4m 16s
	Coarse P_1	30	32	8s
	Dense Surface	30	2.4K	12s
	P_4	30	2.4K	26s
Mat-twist (Figure 5.7)	P_1 coarse	2.2K	1.6K	2m 47s
	P_1 time budgeted	54K	37K	6h 7m 12s
	P_2	2.2K	129K	6h 19m 52s
	P_1	230K	141K	2d 14h 13m 00s
Microstructure (Figure 5.8)	P_1	6.4K	143K	6h 34m 09s
	P_2	6.4K	143K	6h 04m 48s
Trash-compactor (Figure 5.10)	Baseline	21K	8.6K	5h 08m 25s
	Ours	3.5K	8.5K	2h 20m 16s
	Ours, P_2	3.5K	41K	24h 23m 00s
Nut and bolt (Figure 5.11)	Baseline	6.1K	5.2K	22m 04s
	Ours	492	5.2K	9m 40s
Balancing armadillo (Figure 5.12)	Fine	5.9K	3.7K	17s
	Coarse	585	486	19s
	Optimized	585	486	9s
Rolling ball (Figure 5.6)	Baseline	8.8K	5.1K	5m 52s
	Ours	26	5.1K	47s

6 In-Timestep Remeshing for Contacting Elastodynamics

6.1 Introduction

We propose, In-Timestep Remeshing (ITR), a new algorithm for simulating frictionally contacting elastodynamics where remeshing criteria, remeshing operations, and variable mappings are all tightly coupled implicitly, within the timestep solve. Our algorithm automatically adapts meshing in-timestep to account for time-local conditions of both the internal forces and frictional contacts of a trajectory. At the same time, by careful construction, non-intersection and non-inversion are respected as invariants over each operation within the remeshing, and so across each timestep. This provides consistent improvement across extreme variations in materials, severe boundary conditions, fine surface contact details, large friction, and even under the extreme compressions and tensions regularly imposed by contacting and impacting domains.

Large-deformation elastodynamic simulations often require exceedingly dense spatial discretizations to capture critical and often transient features like shockwaves and indentations. At the same time, meshes dense enough to capture these behaviors can be prohibitively expensive in both runtime and memory for practical applications with real-world examples – especially in 3D. These challenges motivate the application of AM methods that seek to locally introduce and remove simulation DOF on the fly, in order to concentrate them where they are most needed.

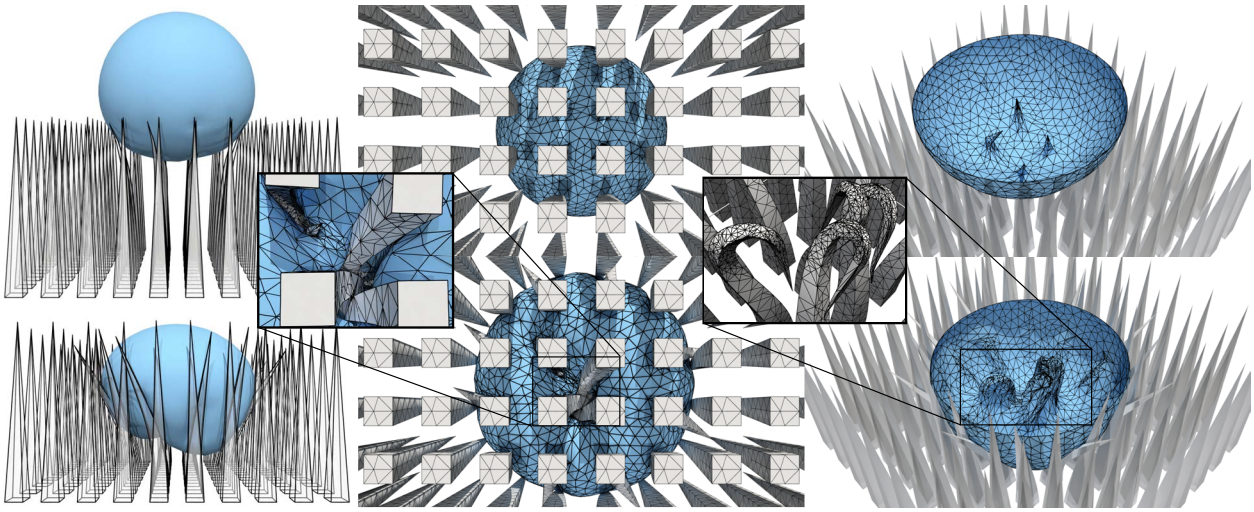


Figure 6.1: Ball on spikes. In-Timestep Remeshing (ITR) enables physics-aware adaptive refinement and coarsening to robustly capture detailed contact-driven deformations in simulated trajectories. Here we drop a soft (neo-Hookean material, $E = 10^5\,\text{Pa}$) ball at large timesteps ($h = 0.01\,\text{s}$) onto very stiff ($E = 10^8\,\text{Pa}$) sharp spikes. Starting with coarse, unstructured finite-element meshes for all geometries (see Figure 6.2(a)) we show here two later steps in the trajectory as the ball initially collides with and then comes to rest on the spikes (top and bottom left respectively). Views from below (middle and middle inset) for each of these steps highlight how our physics-aware remeshing automatically and locally adapts the tetrahedral mesh in time to capture the changing detailed deformations within the material and at contact regions. In a cutaway view (right), we remove the tetrahedral interior elements from the ball, leaving just its bottom surface mesh faces to highlight how ITR tightly conforms, per timestep, without intersection, to the sharp and challenging contacts without over-refining (please compare to the sizing field method in Figure 6.2(b)). Correspondingly we cut the ball geometry from the view altogether (right inset) and zoom in on the tightly wound spike geometries that form the severe indentation on the ball, evidencing the accurate solution of the challenging timestep problem resolving forces between highly disparate material stiffnesses.

Generally, AM for simulating dynamics is currently applied in-between simulation timesteps. This often fits well within optimized physics pipelines in graphics but keeps mesh changes, and the resultant necessary remapping of physical quantities, decoupled from the actual timestep simulation solves. In turn, this decoupling introduces a number of fundamental challenges that we address in this work.

Meshing Criteria. First, the measures evaluated on-mesh that decide where and how to change discretization, can not directly evaluate how remeshing options will impact the solution of the physical problem when decoupled in this way. Applied post-solve, these criteria instead

provide approximations, based on the current generated state, at the current, fixed discretization. Indirect proxies are then generally applied, using geometric criteria and/or snapshots of physical quantities to guide the refinement and coarsening, which, in turn, then need to be re-tuned as the specific physical system simulated (e.g., materials, speeds, boundary conditions) change.

Invariants. Second, necessary invariants for accurate, large-deformation contact simulation are often broken in remeshing pipelines, with element inversions and intersections regularly generated. A range of fail safes and stabilizations have been applied to fix these issues *post hoc* [Narain et al. 2013; Spillmann and Teschner 2008]. However, these fixes all have trade-offs: they generally introduce errors, can inject energy (potentially creating instabilities) [Narain et al. 2013], and require per-example tuning even as they work to remove intersections and/or fix elements.

Mapping. Third, physical quantities, e.g., displacements, velocities, and accelerations, must be mapped to new discretizations. Inherently, all such mappings, aside from happy nesting cases, introduce errors. However, the process of alternating timestep solves, meshing, and mapping, additionally introduces inconsistencies between the physical state and the mesh discretization, while mapping operations themselves can also generate intersections and inversions. As we cover in the next sections, this leads to unacceptable artifacts, additional instabilities, and even simulation failures. Prior work in simulating dynamics with adaptive-meshing, in dealing with these issues, often seeks to minimize refinement operations to reduce error [Wicke et al. 2010]. However, this often opposes the original goal of adapting where needed.

Contact. Contact-driven dynamics particularly pose both significant challenges to, and high demand for AM, where large and highly singular contact forces generate significant and localized deformations in simulation meshes. In such cases, the above-covered issues are especially critical to consider as the separation of meshing steps and solves breaks temporal coherence, introduces infeasible states and unnecessarily perturbs system energies (with attendant numerical artifacts

and jittering), and so often undoes much of the immediate benefit of improved accuracy and quality targeted by AM operations in the first place. Likewise, existing contact-aware AM methods, applying solely geometric criteria significantly over-refine boundaries (see Figure 6.5), in many cases again directly opposing the original intent of AM.

In-Timestep AM. We address the above challenges with, to our knowledge, a first fully coupled AM method for contacting elastodynamics with meshing criteria, operations, and mappings, tightly coupled within each timestep solve. To do so we apply the recently proposed, IPC model [Li et al. 2020] which provides a convergent [Li et al. 2023] and smooth model for frictionally contacting solids Applying the IPC model, we construct In-Timestep Remeshing where meshing criteria have access to the current, ongoing, nonlinear timestep solve's merit function. With this framework, we can apply efficient "micro" simulations per mesh operation, and so make physics-informed and invariant-safe decisions on how to update the discretization.

Contributions. ITR thus refines and coarsens by measuring the change in improvement within each ongoing timestep solve and so avoids recourse to geometric meshing criteria that are physics-oblivious and require per-example tuning.

To build ITR our technical contributions include:

- a "safe" constrained L^2-projection method for variable mapping that minimizes mapping error while preserving invariants by ensuring a globally injective mapping;
- a consistent, smooth remeshing criteria function for frictionally contacting elastodynamics built upon the IPC model; and
- a refinement and coarsening algorithm with provably safe operations, operation filtering heuristics for limiting per-step cost while ensuring solution improvement, and local nonlinear analysis leading to a final, convergent timestep solution on each step's new mesh.

Runtime Efficiency. When compared to uniform mesh refinement, ITR judiciously adds and removes DOF, reducing linear solve times in the inner loop of the nonlinear timestepping algorithm with a DOF improvement ranging from 2.6 to 185× less DOF per example with corresponding 2.7 to 1,444× linear solve speedups. However, additional computation is applied to select where and when the spatial discretization is modified. The interplay between resultant time-savings in linear solves and this overhead varies significantly depending on the scene (and how suitable a naive refinement is). Scenes requiring localized refinement (e.g., Figure 6.1) are up to 3.3× faster with our implementation of ITR, while others (e.g., Figure 6.6) can be up to 9.6× slower. We provide a detailed analysis of this trade-off and discuss its long-term implications for this technology in Section 6.4.3.

We demonstrate the effectiveness of our approach across a wide range of challenging 3D (and 2D) examples, where we highlight the benefits of physics-aware AM. While simple methods are desirable, remeshing necessarily comes with the cost of complex implementation: we release a modular, open-source implementation of our methods at polyfem.github.io to enable replicability and future application.

6.2 Related Work

Meshes are ubiquitous in graphics and there are a wide range of algorithms and applications that create and/or modify them. We focus here specifically on related work on unstructured meshes and the application of mesh modifications for elastodynamic simulation, both with and without frictional contact. For a broad overview of adaptive methods in graphics covering a wide range of physical problems, models, and structured discretizations, please refer to Manteaux et al. [2017]. Hu et al. [2018] similarly review dedicated meshing algorithms, and Mitchell and McClain [2014] cover methods combining mesh modifications with *basis refinement* (p and hp-refinement), which we do not consider in this work.

Adaptive remeshing also plays a critical role in modeling fracture and cutting [Chentanez et al. 2009; Hahn and Wojtan 2015; Koschier et al. 2015; Manteaux et al. 2015; O'Brien et al. 2002; O'Brien and Hodgins 1999; Pfaff et al. 2014] as well as in surface tracking methods which employ complex and robust remeshing operations to explicitly track the movement of colliding and merging boundaries [Brochu and Bridson 2009; Da et al. 2014; Jiang et al. 2017; Klingner et al. 2006; Menon et al. 2015; Misztal et al. 2014; Misztal and Bærentzen 2012; Müller et al. 2015; Stein et al. 2004; Wojtan et al. 2009]. Here we focus solely on elastodynamic simulation without fracture and look to extensions in these areas as exciting potential future directions.

Changing a physical model's spatial discretization during elastodynamic simulation requires four high-level algorithmic components:

1. *Criteria:* where to change the discretization and, when doing so, where to increase or decrease the number of DOF;

2. *Operations:* which operations are applied to change the discretization;

3. *Mapping:* once a discretization is changed, how physical quantities are mapped from the prior discretization to the new one; and

4. *Solution Schedule:* how and when these mapped quantities are applied to update the physical model's solution.

In the following, we next categorize and consider related works with respect to their treatment of these four core components.

6.2.1 Criteria

Geometry. Starting from the seminal work of Hutchinson et al. [1996] for mass-spring systems, a popular way of guiding simulation mesh adaptation is to rely on the geometry of the discretization, either in rest configuration [Bargteil et al. 2007], deformed configuration [Dunyach et al.

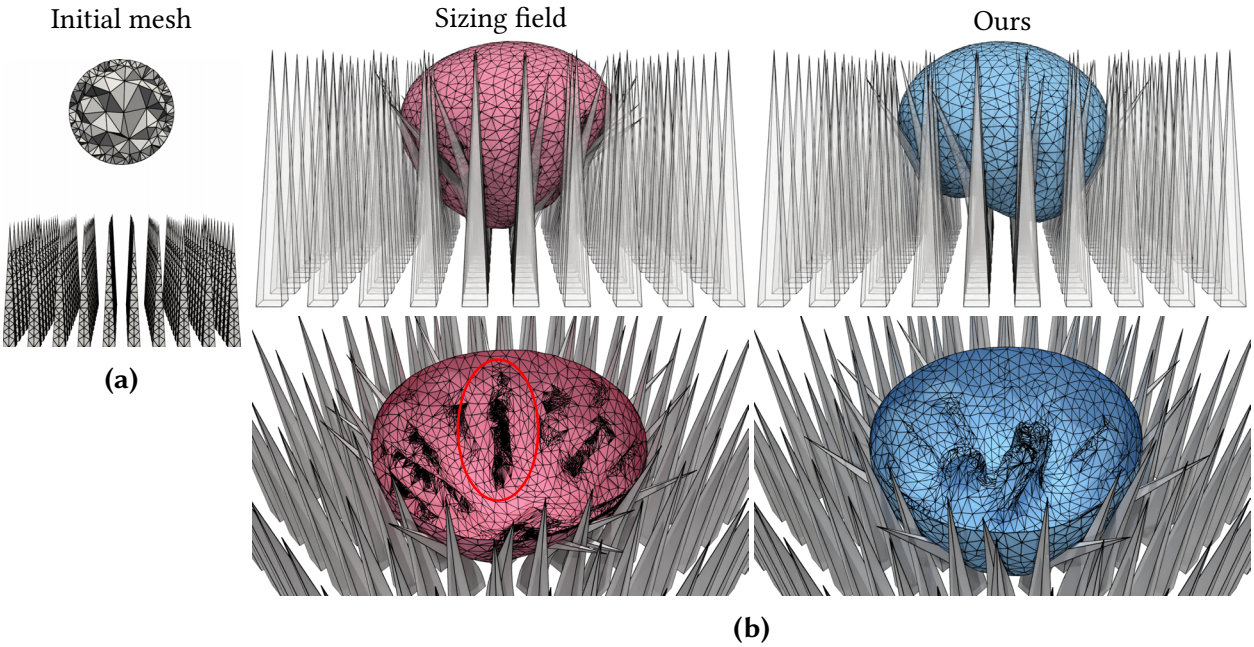


Figure 6.2: Sizing field comparison. (a) The initial conditions and mesh used for the "ball on spikes" simulations in Figures 6.1 and 6.2(b). (b) A comparison of our algorithm (right) and results of applying a contact-aware sizing field-based adaptive meshing criteria [Li et al. 2018a; Narain et al. 2012; Wicke et al. 2010] (left) for in-timestep simulation. We show a cutaway view (bottom row) where we have clipped the geometry to see the inside of the sphere's surface. While the sizing field result refines around the contacts, it severely over-refines right away (circled in red) and so fails to capture intricate interactions. In comparison, our method adaptively updates while tracking both contact and internal forces and so locally refines to capture the spikes pushing into the ball (see Figure 6.1 for a closer view of our results).

2013], or both, enabling the use of a snapshot of strains or stresses [Bargteil et al. 2007; Debunne et al. 2001; Spillmann and Teschner 2008; Wicke et al. 2010]. Similar criteria have been proposed for shells [Li and Volkov 2005; Narain et al. 2013, 2012; Simnett et al. 2009; Villard and Borouchaki 2005], where additional considerations for the complex in-plane and bending behaviors of thin materials play an important role. Additionally, and interestingly, user-dependent geometric criteria such as camera view [Koh et al. 2015] can be considered for refinement. These measures are then primarily proxies for the variations in physical energy in the system, and for the quality of the underlying discretization to represent it. They are, however, approximations based solely on the current rest and deformed configurations at the current, fixed discretization.

Contact. Contacts pose both significant challenges to, and high demand for, adaptive remeshing. Contact forces generate large, yet localized, deformations in many simulation meshes and regularly introduce highly singular strains on boundaries for which it is often desirable to improve resolution. Geometric *proximity* criteria are often applied to help select regions for remeshing on simulation mesh boundaries where two or more surfaces are geometrically close [Bender and Deul 2013; Erhart et al. 2006; Simnett et al. 2009]. Proximity alone is often insufficient and so is sometimes augmented by temporal continuity conditions of the detected collisions [Spillmann and Teschner 2008], and even higher-order approximations that consider a contact region's curvature via contact tangents across mesh faces [Li et al. 2018a; Narain et al. 2013, 2012; Pfaff et al. 2014]. While often effective, these measures do not account for the actual contacting geometry, force balance between contacts and the elastic materials (e.g., considering whether materials involved are equally stiff and so less likely to deform and require adaptation), contact force magnitudes, nor the frictional forces involved. With purely geometric analysis these underlying material and configurational aspects are unaccounted for and so opportunities for necessary refinement and useful coarsening on the contact regions are missed – leading to significant over-refinement or under-refinement in many cases; see Section 6.4.1 and Figures 6.2(b) and 6.5 for examples and evaluation.

Elastic Energy. Rather than apply geometric proxies, a number of recent works focus on applying criteria that measure a model's elastic energy as a criterion in assessing the effectiveness of remeshing [Demkowicz 2006; Mitchell and McClain 2014]. Most closely related to our approach Mosler and Ortiz [2007] propose elastic- and incremental-plastic energy decrease as criteria for small remeshing problems in elastostatics and plasticity, but are limited to solely refinement operations, and do not address contact, friction, nor dynamics.

6.2.2 Operations

Global. Global methods [Jiang et al. 2017; Klingner et al. 2006; Skouras et al. 2014; Stein et al. 2004], create a new mesh for every timestep. Often this is applied via an external meshing tool and so gives the advantage of reducing the implementation effort that would otherwise be required for tighter integration. However, in building a new mesh from scratch, opportunities for problem-aware and ideally smaller updates are lost while such large global changes in the simulation mesh, necessarily incur larger errors in mapping; see Section 6.2.3 below.

Local. Local methods applied in simulation [Li et al. 2018a; Narain et al. 2013, 2012; Spillmann and Teschner 2008] utilize a sequence of local remeshing operations (splits, collapses, swaps/flips) to modify the mesh according to the criteria applied (Section 6.2.1). While applied locally these operations can still cause trouble by creating intersections that must be prevented [Brochu and Bridson 2009] and inversions [Wicke et al. 2010], while also potentially injecting error by introducing instabilities if not resolved carefully (Section 6.2.4). We apply local mesh operations in concert with invariant checks and post-operation energy evaluations to guarantee effective (error-decreasing) and safe (invariant-preserving) mesh adaptations.

Basis and r-Refinement. An alternative to explicit changes in the mesh is to adaptively refine the basis, either via h-refinement within-element [Grinspun et al. 2002] or via p-refinement [Mitchell and McClain 2014]. While adaptive, these methods are not designed to deal with large deformations as they cannot change the shape of the elements (the mesh is fixed). Complementary adaptivity is also provided by r-adaptive or "moving-mesh" methods [Budd et al. 2009] which update the nodal locations in the deforming model's rest mesh but not the topology. While effective in capturing localized dynamics behavior [Zielonka et al. 2008], on its own r-adaptivity is not suited for dynamic contact problems, which generally require concentrated refinement in highly local and often rapidly changing regions.

6.2.3 Mapping

Closest Point. An efficient approach to transfer vertex-based quantities between two meshes in close spatial proximity is to transfer the attributes from a vertex/quadrature point of one mesh to its closest neighbor on the other [Molinari and Ortiz 2002]. This approach introduces large errors when the meshes have elements of different sizes, and usually requires post-stabilization techniques (Section 6.2.4) to avoid simulation artifacts, especially in the presence of stiff materials and contact.

Interpolation. A more accurate method with a bit larger computational overhead is to interpolate via the finite element basis – when linear elements are applied, this is equivalent to the barycentric coordinate interpolation commonly applied in graphics [Spillmann and Teschner 2008; Wicke et al. 2010]. Despite higher accuracy, significant errors still accumulate and post-stabilization techniques remain necessary [Narain et al. 2013; Spillmann and Teschner 2008].

L^2 Projection. Given the above issues, a natural strategy is to compute a mapping that minimizes error [Léger et al. 2014; Vavourakis et al. 2013]. The L^2 projection finds the representation of the function in the finite element space of the target mesh that is a least-squares fit of the function in the finite element space of the source mesh [Léger et al. 2014], and so minimizes the residual of the mapping. Considerably more expensive and challenging to implement than the above alternatives, this projection is commonly applied in scientific computing and mechanical engineering.

We advocate, to our knowledge, for the first time in the graphics community, the L^2 projection for adaptive mesh refinement, as it is robust to both varying mesh densities and low-quality elements. However, despite these important properties, the L^2 projection, on its own, remains insufficient for large-deformation dynamics as it can not ensure necessary invariants in elastodynamics are preserved. In particular, the projection can create intersections and element inver-

sions. In Section 6.3.5, we provide a brief overview of the L^2 projection and then propose our extension to obtain an invariant-preserving, error-minimizing mapping.

6.2.4 Solution Schedule

For elastodynamic simulation, a fundamental question is how to integrate remeshing and mapping variables into each timestep's solution of the physical model.

Interleaving. The standard strategy is to decouple timestepping from remeshing, generally by interleaving timestep solve, remeshing, and mapping. This leaves the remeshing criteria to the mercy of *post hoc* quantities, while after remeshing, the newly mapped variables are finalized as the updated state for the timestep [Narain et al. 2012; Wicke et al. 2010]. Except for local remeshing operations that create nested spaces, the above-covered mappings (Section 6.2.3) all necessarily introduce errors in the projected quantities – meaning the newly mapped solution is inaccurate and inconsistent with the underlying mesh it is defined on and so introduces artifacts including instabilities and jittering [Narain et al. 2013]. Moreover, the updated solution can introduce intersections and inversions, generated by the prior mapping. Earlier works, recognizing these issues, do apply geometric corrections for intersections [Narain et al. 2012] but, at the same time, generally strive to minimize the overall number of remeshing operations to reduce total error [Narain et al. 2012; Wicke et al. 2010].

Post-Stabilization. Post-stabilization methods, recognizing the instability and jittering introduced by direct mapping of timestepped variables to the new mesh, introduce an additional step, after mapping, to improve stability (although not accuracy). Narain et al. [2013], apply a nonlinear least-squares solve to perturb mapped positions to find a more stable configuration, while Spillmann and Teschner [2008] apply a similar strategy with additional collision response phases to also correct for intersections. These methods, with proper parameter tuning, can be effec-

tive at removing visual artifacts, such as jittering, but they also introduce significant errors in the physical model, as they can apply arbitrary perturbations to a solution that already contains errors.

6.2.5 IPC and In-Timestep Remeshing

The above analysis leads us to the conclusion that for simulating elastodynamics, the remeshing criteria, remeshing operations, and variable mappings, should all be tightly coupled, and so integrated together within *each* timestep solve. The next question is how. Mosler and Ortiz's [2007] work on simulating elastostatics using the elasticity potential is our starting point for contacting elastodynamics. We begin by applying the recently proposed, IPC model [Li et al. 2023, 2020] which provides a convergent and *smooth* model for frictionally contacting solids. In turn, because IPC contact forces are smooth and the IPC timestep update is variational, this allows us to formulate, per timestep, a spatially smooth merit function as our meshing criteria. This merit function includes elasticity, contact, and friction, and its decrease guarantees solution improvement. We then carefully design our solver to refine, coarsen and safely L^2-project, within each timestep solve, to maintain consistent updates, while ensuring that the final output of each timestep is an accurate, intersection-free, and inversion-free solution progressing the simulation dynamics forward in time.

Hierarchical Methods. Hierarchical methods provide solver strategies, complementary to AM methods, that can be applied to improve timestep solves. These methods (e.g., [Hormann et al. 1998; Zhang et al. 2022]) apply a hierarchy of pre-determined resolutions to better compute a solution for a final (pre-specified) and generally uniform, high-resolution target mesh.

In contrast, AM methods (including ITR) locally adapt solution meshes to apply detailed resolution where it can be better used. In future work, it should be an interesting extension to consider the application of hierarchical methods within ITR to obtain faster nonlinear solves. For this, the

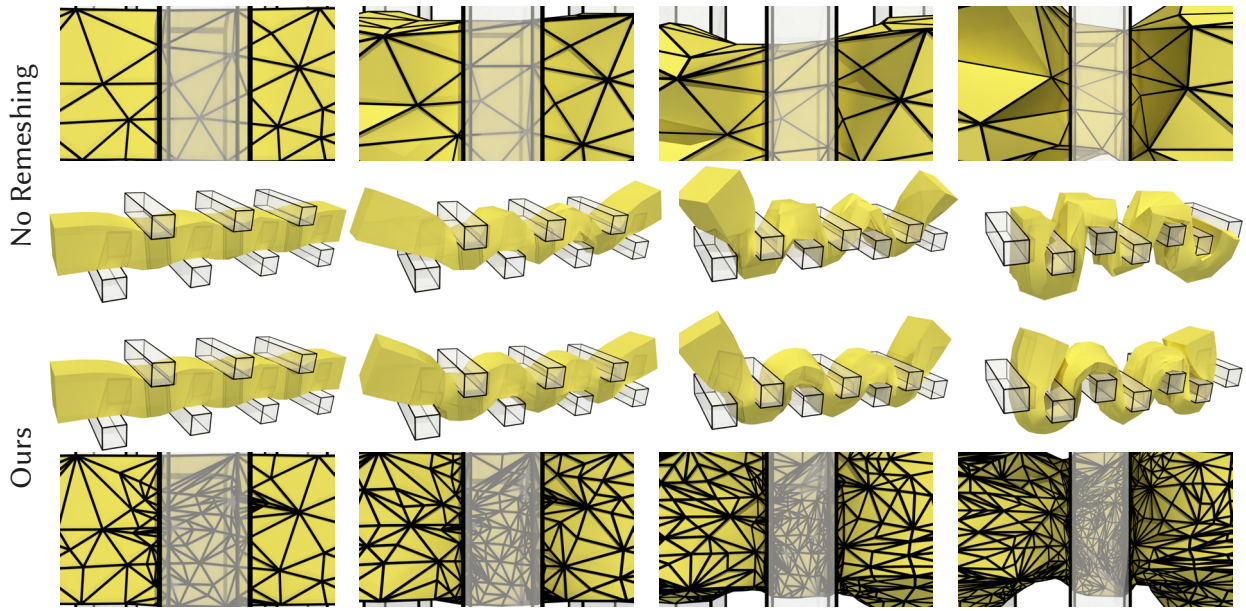


Figure 6.3: Masticator. A challenging 3D compression example, simulated without refinement (top) and with (bottom) our algorithm, starting from the same initial mesh. The insets highlight how our method is able to capture the sharp contact features and buckling under compression by increasing mesh resolution. Without remeshing, these details are lost, resulting in a different deformation.

most closely related approach to ITR in the hierarchical literature, is the recent work of Zhang et al. [2022] who build a hierarchical solver for IPC. They propose a Euclidean projection to find non-intersecting geometries nearest to possibly-intersecting, Loop-subdivision-upsampled targets, by applying barrier-enforced, CCD-filtering to the direct path from a "safe", midpoint-upsampled triangle mesh, to a target. Here we construct a complementary, error-minimizing, L2-projection, for tetrahedral meshes, constructed by constrained quadratic energy minimization, supplemented with CCD-filtered collision barriers, suitable for refinement and coarsening operations.

6.3 In-Timestep Remeshing

6.3.1 Spatially Continuous Setting

We consider the solution of simplicial simulation meshes (triangles in 2D, tetrahedra in 3D) undergoing large-deformation elastodynamics with frictional contact. Before discretizing to a spatial mesh, we first begin by discretizing *in time*: we construct the solution of each timestep's problem in semi-discrete optimization form,

$$x^{t+1} = \operatorname*{argmin}_x E_t(x) \tag{6.1}$$

with a spatially continuous Incremental Potential,

$$\begin{aligned} E_t(x) = &\int_\Omega \frac{\rho}{2} \left\| x(X) - \tilde{x}^t(X) \right\|^2 \, \mathrm{d}V \\ &+ \alpha h^2 \int_\Omega \Psi\big(x(X)\big) - x(X)^\top f(X) \, \mathrm{d}V \\ &+ \alpha h^2 \int_{\partial\Omega} B\big(x(X)\big) + D\big(x(X)\big) \, \mathrm{d}A. \end{aligned} \tag{6.2}$$

Here Ψ is a hyperelastic deformation-energy density (e.g. neo-Hookean), f encodes the sum of body forces and (when ranging over boundary regions) any applied reactions, and B and D are the spatially-continuous analogs of the IPC energies [Li et al. 2023] for, respectively, contact barrier and friction pseudo-potential. In turn, the choice of predictor position, $\tilde{x}^t$ (an explicit function of prior deformation, velocity, and possibly acceleration fields: $x^t, x^{t-1}, \ldots, v^t, v^{t-1}, \ldots, a^t, a^{t-1}, \ldots$), scaling term $\alpha \in \mathbb{R}^+$, and an explicit update equation for velocity (and acceleration as needed) from optimal solution x^{t+1}, jointly define the specific choice of numerical time-integration method. Here, in the main text, for simplicity, we will keep in mind implicit Euler

with

$$\tilde{x}^t = x^t + hv^t,\ v^{t+1} = \tfrac{1}{h}(x^{t+1} - x^t),\ \alpha = 1, \tag{6.3}$$

while similarly, a wide range of additional time integration methods are directly covered[1].

6.3.2 Solution Quality per Timestep

In this continuum form, the optimization timestep solve in Equation (6.2) highlights an important deciding feature: when we are allowed to range over the space of all valid deformations x, a deformation giving the (locally) smaller value of E_t is the better solution to the timestep. Looking ahead to our next step of spatial discretization, this provides a simple, physics-focused metric for ranking finite-element meshes in a solution space, that is custom-suited to each *timestep*. Of course, a corollary is that this energy decrease is always "easily" obtained via uniform refinement to finer and finer meshes but this also comes with the associated cost of more, and generally too much, computation. Instead, here we focus on applying this metric to locally adapt our mesh in regions of high value. To do so we focus our adaptivity on this actual, temporally local, measured change in the timestep's solution quality itself, rather than on intermediate proxies via mesh qualities or physical properties.

[1]For example, other time-integration methods we applied are Implicit Newmark, with

$$\alpha = 1/4, \tilde{x}^t = x^t + hv + h^2/4a^t, v^{t+1} = 2/h\left(x^{t+1} - x^t\right) - v^t, \text{and}$$
$$a^{t+1} = 2/h\left(v^{t+1} - v^t\right) - a^t,$$

and second-order backward differentiation formula (BDF2), with

$$\alpha = 4/9, \tilde{x}^t = \tfrac{1}{3}(4x^t - x^{t-1}) + \tfrac{2h}{9}(4v^t - v^{t-1}), a^{t+1} = \tfrac{4h^2}{9}(x^{t+1} - \tilde{x}^t), \text{and}$$
$$v^{t+1} = \tfrac{1}{3}(4v^t - v^{t-1}) + \tfrac{2h}{3}a^{t+1}.$$

Small changes by additional terms in the arguments of the energy functions extend the range of our application even further to a yet wider range of numerical time integration methods without loss of generality [Li et al. 2023].

6.3.3 Spatial Discretization

We apply piecewise-linear discretization of Equation (6.2) on meshes $\mathcal{T}$ with discrete fields defined, per triangulation/tetrahedralization, at the n vertices of the mesh in 3D (respectively 2D) space and stored in vectors $x, v, a \in \mathbb{R}^{3n}$ (respectively $\mathbb{R}^{2n}$).

Each of the spatially discrete energy terms in our incremental potential are now expressed as weighted sums of energy functions over mesh element stencils, s (tetrahedral, triangle, edge, point or pairings thereof depending on energy and dimension) in $\mathcal{T}$,

$$\sum_{s \in \mathcal{T}} w_s W_s\left(x\right),$$

where $w_s > 0$ is the volume, area or length-weighted scaling of the rest shape element s, and W_s is the respective energy density function of each potential restricted to this element's stencil.

For a fixed mesh $\mathcal{T}$, the timestep solution is then a local minimizer of a fully discrete Incremental Potential, per timestep

$$\begin{aligned} E_t(x, \mathcal{T}) =& E(x, \mathcal{T}, \tilde{x}^t) \\ =& \frac{1}{2}(x - \tilde{x}^t)^\top M_{\mathcal{T}}(x - \tilde{x}^t) \\ &+ \alpha h^2 \left(\Psi_{\mathcal{T}}(x) + B_{\mathcal{T}}(x) + D_{\mathcal{T}}(x) - x^\top f^t\right), \end{aligned} \tag{6.4}$$

where $M_{\mathcal{T}}$ is the mesh's consistent mass matrix, and $\Psi_{\mathcal{T}}$, $B_{\mathcal{T}}$, and $D_{\mathcal{T}}$ are the total resultant energy potentials generated, respectively, by the aforementioned discretizations of the corresponding deformation, contact barrier, and friction energies on $\mathcal{T}$ [Li et al. 2023].

6.3.4 Timestepping Framework and Invariants

We advance our simulation domain through time using an incrementally updating triangulation of the domain, $\mathcal{T}(t)$, with deformations $x(t)$, velocities $v(t)$, and rest positions, $\bar{x}(t)$ defined at $\mathcal{T}(t)$'s vertices. Input for the solve of each timestep optimization is then: $\bar{x}^t, x^t, v^t$, and current applied forces (body and external), f^t, defined on mesh $\mathcal{T}^t$ with the *deformed* mesh $(x^t, \mathcal{T}^t)$ giving a penetration- and inversion-free configuration.

In turn output for each of our timestep solves is then a new mesh $\mathcal{T}^{t+1}$ and updated fields, $\bar{x}^{t+1}, x^{t+1}, v^{t+1}$ that maintain the invariants of non-intersection and non-inversion at end state, while accurately satisfying the numerical time-integration model by minimizing the incremental potential with $\left\|\nabla_x E_t(x, \mathcal{T}^{t+1})\right\| \leq \epsilon_d$.

At the same time, as we cover in detail below, to better resolve dynamics, each of our timestep solves also incrementally updates the simulation mesh $\mathcal{T}$, as a "configurational" degree of freedom with mesh-refinement to lower the total value of the incremental potential solution in Equation (6.2) measured by

$$m_t(\mathcal{T}) = \min_x E_t(x, \mathcal{T}), \tag{6.5}$$

and similarly coarsening where this does not significantly increase this same value.

Maintaining Invariants. We apply Newton iterations to minimize $E_t(x, \mathcal{T})$ when holding the mesh fixed. Preserving invariants for these steps we follow the IPC method's filtered line-search step [Li et al. 2020] which applies CCD and inversion-checking to descent steps. This ensures that all applied displacements for position updates to x ensure both safety and energy decrease towards convergence. In our setting with remeshing, this is not enough – all operations during each timestep computation, including remeshing, must maintain non-intersection and non-inversion at every update.

6.3.5 Safe Projections Between Spaces

Each remeshing operation, i, applied during a timestep solve changes the mesh, $\mathcal{T}^i \rightarrow \mathcal{T}^{i+1}$. This means that all quantities, (x^t, v^t, f^t, $\bar{x}$, ...) defined in the prior mesh must, of course, be mapped, or *projected*, to the new one.

When simulating dynamics these quantities are generally transferred in-between timestep solves (see Section 6.2.3); that is given $\bar{x}, x^t, v^t,$ and $\mathcal{T}^1$ we would first solve for a new timestep solution, x^{t+1}, v^{t+1}, then update the mesh to a new one $\mathcal{T}^2$. Then, only after remeshing, x^{t+1}, v^{t+1}, are projected to the new mesh. Unfortunately, this staggered process means that

$$x^{t+1} \neq \underset{x}{\operatorname{argmin}}\, E_t(x, \mathcal{T}^2)$$

and so is not a solution to the timestep problem on the current mesh. This inconsistency between solution space and deformation then commonly generates instabilities and jittering artifacts, especially when dealing with stiffer materials and collisions (see e.g., Narain et al. [2013] and their supplemental video).

Another fundamental challenge for remapping variables is the actual definition of the projection operator itself. As alluded to above, changing the mesh also changes the underlying function space of the model. This is why inconsistencies from staggering the timestep solves and projections can generate such significant errors when timestepping.

A cheap, popular, and perhaps simplest projection strategy is closest-point sampling where we assign new nodal values from the closest node in the prior mesh. While tempting, this "projection" introduces large artifacts and instabilities when elements' sizes differ [Vavourakis et al. 2013], e.g., under refinement, where many new nodes' values are often assigned from the same source node in the prior mesh. A popular alternative is to apply interpolation from the finite-element basis – barycentric interpolation in our linear-element setting. This generally gives better results than closest-point sampling, but still introduces large projection errors, again leading

to artifacts [Léger et al. 2014; Vavourakis et al. 2013; Wicke et al. 2010].

We instead begin with the L^2 projection that *minimizes* the two-norm error residual when we map from starting to target finite-element space [Léger et al. 2014]. Consider again updating from $\mathcal{T}^1$ with function space V^1 and basis $\{\varphi_i^1 \mid 1 \leq i \leq n\}$ to $\mathcal{T}^2$ with corresponding function space V^2 and basis $\{\varphi_i^2 \mid 1 \leq i \leq m\}$. We now define a least-squares projection operator

$$\mathcal{P} : V^1 \to V^2, \tag{6.6}$$

so that for functions $f^1 \in V^1$ their projection $f^2 = \mathcal{P}(f^1) \in V^2$ minimizes the L^2 residual $\frac{1}{2}\left\|f^1 - f^2\right\|^2$. Optimality conditions minimizing this residual [Léger et al. 2014] give the projection of a quantity u, defined on the vertices of $\mathcal{T}^1$ (e.g., the coefficient vector of f^1), to the vertices of $\mathcal{T}^2$ as

$$\widetilde{M}_{\mathcal{T}^2}^{-1} A_{\mathcal{T}^2}^{\mathcal{T}^1} u, \tag{6.7}$$

where $\widetilde{M}_{\mathcal{T}^2} \in \mathbb{R}^{m \times m}$ is the density-normalized mass matrix on $\mathcal{T}^2$ and $A_{\mathcal{T}^2}^{\mathcal{T}^1} \in \mathbb{R}^{m \times n}$ is a transfer matrix between bases so that $a_{ij} = \int_\Omega \varphi_i^2 \varphi_j^1 \, dV$. The two bases are then defined on two different meshes and so we use arrangement via PolyClipper [Powell 2021] to compute the quadrature points necessary for the integral [Krause and Zulian 2016].

Although not, to our knowledge, previously applied in graphics, this L^2-projection has long been appreciated in mechanics applications for its better preservation of quantities [Léger et al. 2014] due to minimized error. However, while well-projecting *unconstrained* quantities the L^2 projection (and all others) are oblivious to our invariants. Projections can and will create both element inversions and intersections, meaning we can not apply the L^2 projection operator as-is.

To make the L^2 projection safe we return to the variational picture and now rebuild a constrained least-squares residual minimization, *subject to non-intersection and non-inversion* con-

straints, that safely projects quantities from an old mesh $\mathcal{T}^{\text{old}}$ to a new one, $\mathcal{T}$,

$$P_{\mathcal{T}}(u) = \underset{v}{\operatorname{argmin}} \tfrac{1}{2} v^\top \widetilde{M}_{\mathcal{T}} v - v^\top A_{\mathcal{T}}^{\mathcal{T}^{\text{old}}} u + B_{\mathcal{T}}(x) + I_{\mathcal{T}}(x). \tag{6.8}$$

The first two terms form the least squares condition, $B_{\mathcal{T}}$ is our discretized IPC contact barrier defined on the new mesh, and $I_{\mathcal{T}}$ discretizes a new barrier we propose to enforce non-inversion during projection without biasing the solution with elastic material behavior,

$$I_{\mathcal{T}}(x) = \sum_{t \in \mathcal{T}} w_t c_t(x, \hat{v}), \tag{6.9}$$

where, the function c_t returns a log barrier on the volumes $v_t(x)$ of tetrahedra t, that is smoothly activated when volume falls below $\hat{v}$,

$$c_t(x, \hat{v}) = \begin{cases} -\kappa_v \left(\frac{v_t(x)}{\hat{v}} - 1\right)^2 \ln\left(\frac{v_t(x)}{\hat{v}}\right), & 0 < v_t(x) < \hat{v} \\ 0 & v_t(x) \geq \hat{v}. \end{cases} \tag{6.10}$$

We use $\hat{v} = 10^{-12}\,\text{m}^3$ and $\kappa_v = 0.1E$ (same as contact barrier stiffness) throughout where E is the material's Young's modulus. To apply each constrained projection we first safely initialize our displacement variables on the new mesh via linear interpolation and then directly reuse our same line-search-filtered Newton method to solve Equation (6.8) and so minimize the L^2-residual while ensuring safe new variables on the updated mesh.

For all of our ITR phases, detailed in the next two sections, all prior timestep quantities $(\cdot^t)$ must be projected to ensure consistency. However, in our setting, we are able to take advantage of a simple optimization: during refinement (only), barycentric interpolation is equivalent to our L^2 projection and so can be safely and cheaply applied rather than Equation (6.8) for all our edge-split operations.

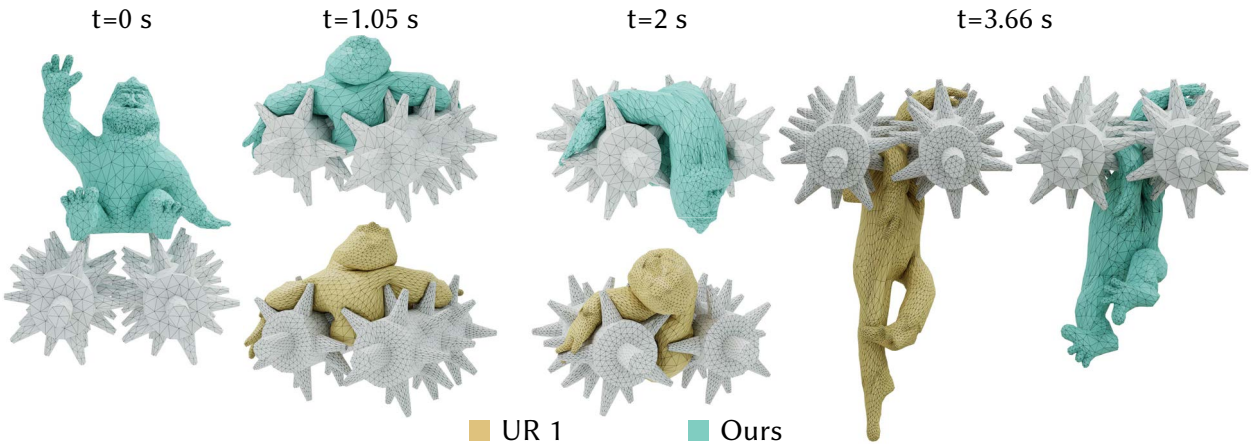


Figure 6.4: Gorilla rollers. a very soft gorilla model ($E = 2 \times 10^4$ Pa) is dropped on a pair of stiff rotating elastic rollers ($E = 2 \times 10^8$ Pa) with softer spikes ($E = 2 \times 10^7$ Pa). As the gorilla impacts the spikes, the mesh is refined to account both for the large elastic forces in thin features and for the rapidly changing contact forces. Our method adapts to the different material stiffness, by refining the softer gorilla in the necessary regions of contact, much less for the stiffer spikes, and leaves the even stiffer roller unadapted. The dynamics for the single-level uniformly-refined (UR) solution (UR 1) is comparable up to $t = 2$ s where the spike is (unlike the adapted mesh solution) is unable to push into the gorilla's left shoulder.

6.3.6 Remeshing with Local Operations

Changing the *geometry* (i.e., vertex rest-positions in our setting) of a mesh is attractive for mesh adaptation as it leads to continuous changes in the underlying finite element space, and so is amenable to gradient-based optimization of functionals depending on them. However, such r-adaptive-type updates are insufficient to capture the large, often transient, and highly localized deformations we capture in accurate elastodynamic modeling. Instead, we often require increasing (and decreasing) the number of DOF and so the number of vertices in our meshes. However, in changing the *connectivity* of a mesh we obtain discontinuous changes in our finite-element space. In turn, this will drive nonsmooth changes in our meshing criteria functional in Equation (6.2) and so here we will apply a discrete optimization strategy.

We consider two types of operations that are applicable to both triangle and tetrahedral meshes: 1) an edge split, which splits every triangle/tetrahedra touching an edge into two while inserting a vertex, and 2) its inverse, an edge collapse. These operations are discrete in nature,

but depend on both discrete and continuous parameters: a split operation is applied to a discrete edge, but the position of the newly inserted vertex is controlled by two or three continuous coordinates.

Due to the discrete nature of the problem, it is not practical to seek an optimal sequence of operations minimizing Equation (6.5) (presuming a fixed sequence length or a targeted given tolerance). We instead apply a greedy block-coordinate descent strategy: we test a set of potential operations and pick those that provide maximal local improvement in the energy for the inserted DOF. We first discuss how we evaluate the effect of a single operation, and then how to greedily select a sequence of operations reducing Equation (6.5).

Effect of an individual operation. To evaluate the effect of an operation on Equation (6.5) a naive approach would be to perform the mesh modification, project quantities (Section 6.2.3), ensure that the invariants are still valid, and minimize Equation (6.5) globally. However, this is computationally prohibitive: inspired by approaches used for *a posteriori* error estimators [Mitchell 1991; Schmidt and Siebert 2000], we perform local solves in the neighborhood of the mesh modification. This enables a sound approximation of each operation's impact, under the assumption that this effect decays as we move from the operation's stencil. By changing the neighborhood's size, we trade accuracy of our estimator with computational cost.

Scheduling. While we can not tractably obtain globally optimal sequences for meshes, we could potentially find locally optimal solutions by always continually selecting splits that satisfy a sufficient amount of energy decrease (i.e., a minimal necessary reduction) of our energy until no more remain. However, as we scale to larger (and 3D) meshes, this approach is no longer practical either. Detailed below we thus introduce a culling method that preemptively discards candidate split operations that are not likely to lead to large energy reductions (and correspondingly discards candidate edge collapses that are likely to lead to energy increases). We do this by filtering based on the elastic and contact energies per mesh element, with greater local energy concen-

tration indicating a higher likelihood (although not guaranteed) of energy reduction benefit by splitting. Note that our filtering heuristic is applied solely to *cull* likely ineffective operations – our criteria for acceptance remains unchanged by it. We detail our filtering method in the next section.

Implementation. Implementing this discrete optimization algorithm is challenging, especially for tetrahedral meshes, as we need a mechanism to *preview* each connectivity change, extract its patch, and minimize Equation (6.5). If the operation is invalid, or else does not satisfy our criteria, changes to the connectivity and to its associated fields defined on our mesh need to be rolled back. This is significantly challenging to implement via low-level libraries, e.g., CGAL, libigl, or OpenMesh. We opt to implement our remeshing with declarative specification in Jiang et al. [2022], which allows us to explicitly work on the mesh before and after the operation, and directly supports invariant checks and rollbacks.

6.3.7 In-Timestep Remeshing Algorithm

We provide a high-level overview of our method in Algorithm 6.1.

Initial Timestep Solution. Give a current solution state x^t, v^t from the last timestep solve[2] at time t, our ITR first computes a new predictor timestep solution x' by minimizing Equation (6.4) on the current mesh $\mathcal{T}_t$ (Line 3).

Refinement. Using the new solution x', we sort every edge e_i according to its elastic energy $\Psi_{\mathcal{T}_t}(e_i)$ (area-weighting all adjacent cells) to form list E_Ψ, and create list E_B by sorting e_i according to its contact energy $B_{\mathcal{T}_t}(e_i)$ (averaged over two adjacent faces in 3D) (Line 6). We then select

[2]For clarity in pseudocode and discussion we do not track the update of a^t, a^{t+1} nor x^{t-1} here. Treatment for acceleration terms, when time-integration methods are applied that use them, follow identically to v^t, v^{t+1} throughout, similarly treatment of x^{t-1} follows identically to x^t.

Algorithm 6.1 Overview of our in-timestep remeshing algorithm (Part 1 of 2).

1: **procedure** InTimestepRemeshing($x_t, v_t, \mathcal{T}_t$)
2: // **Initial Timestep**
3: $x' \leftarrow \text{argmin}_x E_t(x, \mathcal{T}_t)$
4:
5: // **Refinement**
6: $E_\Psi \leftarrow \text{Sort}(\{\Psi_{\mathcal{T}_t}(e_i)\})$, $E_B \leftarrow \text{Sort}(\{B_{\mathcal{T}_t}(e_i)\})$
7: $S \leftarrow E_\Psi > \epsilon_S \cup E_B > \epsilon_S$,
8: $\mathcal{T}'_t, x'_{t+1}, x'_t, v'_t \leftarrow \text{Split}(S, \mathcal{T}_t, x', x_t, v_t)$
9: $x_t, v_t \leftarrow x'_t, v'_t$
10:
11: // **Coarsening**
12: $C \leftarrow E_\Psi < \epsilon_C \cap E_B < \epsilon_C$,
13: $\mathcal{T}_{t+1}, x'_{t+1}, x'_t, v'_t \leftarrow \text{Collapse}(C, \mathcal{T}'_t, x'_{t+1}, x'_t, v'_t)$
14: $x_t, v_t \leftarrow \text{SafeProject}(\mathcal{T}_t, \mathcal{T}_{t+1}, x_t, v_t)$
15:
16: // **Global Solve**
17: $x_{t+1} \leftarrow \text{argmin}_x E_t(x, \mathcal{T}_{t+1})$
18: **return** $\mathcal{T}_{t+1}, x_{t+1}, x_t, v_t$

the top ϵ_S% of both lists to form the filtered set S of edges as candidates for splitting operations (Line 7).

We then proceed to the Split procedure (Line 8). The Split procedure takes the set of candidate operations S, and for each operation performs the split (Line 23), obtaining a new mesh $\mathcal{T}'_t$, updates the variable on the mesh by linear interpolation along the split edge[3] (Line 24), which for each split is equivalent to a zero-error L^2 projection, and then performs a small local solve (Line 25). See the next section below for details on the *Local Solve*. The split operation is accepted if we obtain sufficient decrease, $\delta E = E_{t+1}(x_i, \mathcal{T}_i) - E_{t+1}(x_p, \mathcal{T}_p) > \delta_s$, and the newly created edges are applied to update the queue (Line 29). Otherwise, if the operation is rejected for providing insufficient improvement, we undo the split.

Local Solve. Local solves applied in both the *Split* procedure above and the *Collapse* procedure below follow the same procedure. A timestep re-solve is performed in a local patch by minimizing

[3]In practice, we use a simple averaging of endpoint values.

Algorithm 6.2 Overview of our in-timestep remeshing algorithm (Part 2 of 2).

procedure SPLIT($S, \mathcal{T}, x_{t+1}, x_t, v_t$)
  $Q \leftarrow$BUILDPRIORITY(S)
  while $Q \neq \emptyset$ **do**
    $e \leftarrow$ POP(Q)
    $\mathcal{T}' \leftarrow$ SPLITEDGE($e, \mathcal{T}$)
    $x'_{t+1}, x'_t, v'_t \leftarrow$ INTERPOLATE($x_{t+1}, x_t, v_t, \mathcal{T}, \mathcal{T}'$)
    $x'_{t+1} \leftarrow$ LOCALSOLVE($x'_{t+1}, \mathcal{T}'$)
    if $E_t(x_{t+1}, \mathcal{T}) - E_t(x'_{t+1}, \mathcal{T}') > \delta_s$ **then**
      $\mathcal{T} \leftarrow \mathcal{T}'$
      $x_{t+1} \leftarrow x'_{t+1}, x_t \leftarrow x'_t, v_t \leftarrow v'_t$
      $Q \leftarrow$ UPDATEQUEUE($Q, \mathcal{T}$)
  return $\mathcal{T}, x_{t+1}, x_t, v_t$

procedure COLLAPSE($C, \mathcal{T}, x_{t+1}, x_t, v_t$)
  $Q \leftarrow$BUILDPRIORITY(C)
  while $Q \neq \emptyset$ **do**
    $e \leftarrow$ POP(Q)
    $\mathcal{T}' \leftarrow$ COLLAPSEEDGE($e, \mathcal{T}$)
    $x'_{t+1}, x'_t, v'_t \leftarrow$ INTERPOLATE($x_{t+1}, x_t, v_t, \mathcal{T}, \mathcal{T}'$)
    if INVARIANTCHECK($x'_t, x'_{t+1}, \mathcal{T}'$) **then**
      $x'_{t+1} \leftarrow$ LOCALSOLVE($x'_{t+1}, \mathcal{T}'$)
      if $E_t(x_{t+1}, \mathcal{T}) - E_t(x'_{t+1}, \mathcal{T}') > \delta_c$ **then**
        $\mathcal{T} \leftarrow \mathcal{T}'$
        $x_{t+1} \leftarrow x'_{t+1}, x_t \leftarrow x'_t, v_t \leftarrow v'_t$
        $Q \leftarrow$ UPDATEQUEUE($Q, \mathcal{T}_t$)
  return $\mathcal{T}, x_{t+1}, x_t, v_t$

Equation (6.4) on the current mesh, but now fixing *all nodes* in the system except for DOF in a local patch with a size that is the maximum between the 2-ring of the edge and 1% of the domain's volume, and using x' as a safe and "near-to-solution" warm start. A first i iterations are run ($i = 4$ for contacting patches and 1 otherwise) and then checked to see if it reaches respectively, sufficient decrease for a split (see above) or small (by $|\delta_c|$) acceptable increase for a collapse (see below). If the remeshing criteria *is not reached* the operation is abandoned (as covered) as the Newton decrement shows no progress. Otherwise, if the criteria are met and we will be accepting the operation we continue the local-patch solve to convergence (same termination tolerance as

the global solve) ensuring that downstream operations (and the final solve) all start from well-resolved regions.

Coarsening. Next, we then select the bottom $\epsilon_C\%$ from E_Ψ and E_B to form the set C of candidate edges for potential *collapse* operations (Line 12) and attempt to collapse them (Line 13).

As in *Split*, the *Collapse* procedure takes a set of candidate operations C, and for each operation performs the collapse (Line 36), obtaining a new mesh $\mathcal{T}'_t$. However, differently from *Split*, the collapse operation *does not* create a nested space, and so interpolation will introduce mapping-errors. In turn, these errors can occasionally break our invariants. To avoid this problem, we locally perform an interpolation, and then explicitly check if the invariants are violated (Line 38). If they are violated, we reject the operation, otherwise we proceed similarly to check that the *Split* operation *does not increase* system energy by more than a small, prescribed tolerance via a $\delta_c \leq 0$ and otherwise follow as in the *Collapse* procedure.

As in the interpolation applied in our *Split* operations, we interpolate the endpoints of the collapsed edge to determine each new vertex's attributes. We collapse boundary edges if and only if neighboring faces are coplanar in order to preserve the mesh's rest shape. For the same reason, when an edge has a single vertex on the boundary, we collapse it to the boundary endpoint. For all other edges, we average the endpoints.

After all collapse operations are complete, unlike after splits, we now require a L^2-projection of prior displacements and velocities on $\mathcal{T}_{t+1}$ by means of the safe L^2 projection (Line 14) described in Section 6.3.5, using our interpolated quantities as safe initialization.

Global Solve. Finally, warm-starting with the latest solution estimate x'_{t+1}, we perform a final re-solve of Equation (6.4) on the full domain using the finalized new mesh $\mathcal{T}_{t+1}$ and then explicitly update velocity to v_{t+1}. As we have been incrementally updating (effectively relaxing) the solution throughout this process this final solve is efficient (the number of iterations for convergence is low) as the majority of the effort has been performed in both the initial and intermediary solves

during remeshing.

6.4 Evaluation

Our algorithm is implemented in C++, using Eigen [Guennebaud et al. 2010b] for basic linear-algebra, PolyFEM [Schneider et al. 2019b] for FE system construction, IPC Toolkit [Ferguson et al. 2020] for evaluating IPC potentials and collision detection, Wildmeshing-toolkit [Jiang et al. 2022] for mesh data structures and editing, Pardiso [Alappat et al. 2020; Bollhöfer et al. 2019, 2020] for the large linear systems in our global Newton solves, and Eigen's dense Cholesky decomposition ($LL^\top$) for the small linear systems in our local Newton solves. All experiments are run on a cluster node with an Intel Cascade Lake Platinum 8268 processor limited to 16 threads. Our reference implementation, used to generate all results, will be released as an open-source project. Please see our supplemental video for result animations and Table 6.3 for parameters used.

6.4.1 Comparisons

To our knowledge, our ITR algorithm is the only AM method that can ensure the preservation of IPC invariants and so can be combined with the IPC contact model. This is because mappings and contact failsafes applied in previous works can and will fail with intersections and downstream failures in challenging contacting scenarios like those we test here (Section 6.2). In order to compare with prior methods on these challenging scenarios we focus on comparing *meshing criteria* in a comparable side-by-side setting allowing all methods to utilize within-timestep simulation and IPC solves.

To robustly process contacts and implicitly solve dynamics with IPC we replace our physics-aware meshing criteria within ITR with Wicke et al.'s [2010] sizing field, based on the deformation gradient [Wicke et al. 2010, Equation (7)], for internal deformation criteria, and on the most recent, state-of-the-art contact sizing criteria proposed by Li et al. [2018a]. We compare our

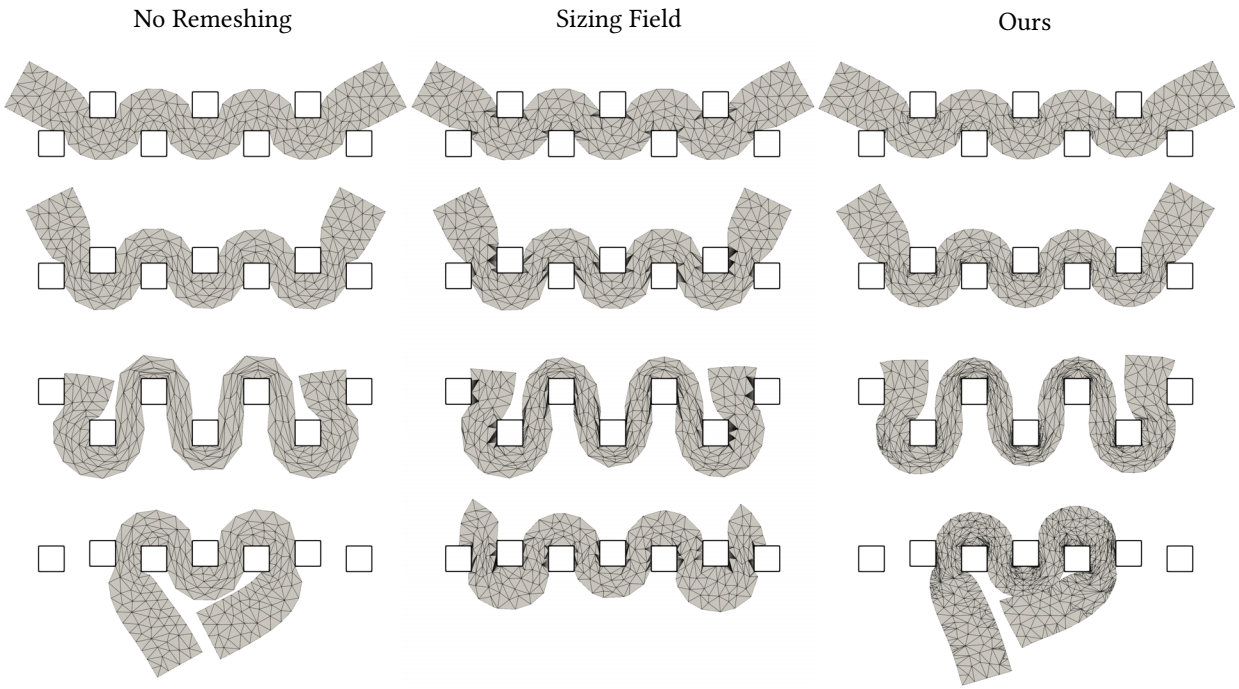


Figure 6.5: 2D Masticator. Simulation of the deformation of a 2D bar deformed by a set of squares. the same mesh is used for the simulation without remeshing (left column), remeshed using a sizing field based on [Li et al. 2018a; Narain et al. 2012; Wicke et al. 2010] within our in-timestep framework (middle column), and with ITR (Ours, right column). ITR produces a more detailed simulation compared to the run without remeshing, adding DOF to accurately capture the sharp contact with the cubes and the large deformation of the bar. The method based on the sizing field overrefines the contact regions (and does not refine elsewhere) and leads to a different "snagged" final configuration.

energy-based acceptance criteria with this sizing field[4]. To do so we replace our criteria in our implementation with the above sizing field and accept edge-split and edge-collapses following the scheduling of Narain et al. [2012] (same as Li et al. [2018a]).

We instrument two side-by-side comparison examples: one in 2D (*Masticator*) and the other in 3D (*Ball-on-Spikes*). Initially, we observe that the contact-based sizing field leads to runaway endless refinement on contacting surfaces – rapidly leading to intractable simulations. On closer inspection, we see that division by contact distance in the denominator of Li et al.'s [2018a] sizing tensors is the source: here accurate IPC contact-processing allows for exceedingly close

[4]Note that we do not apply the additional deformation sizing field criteria from [Li et al. 2018a; Narain et al. 2012] as those are customized for shell models – for this, we take our deformation sizing component instead from the volumetric work of Wicke et al. [2010].

compliance between surfaces. To enable the contact sizing field strategy to progress we add a limit to the method restricting edge lengths to 0.01.

For our 2D example, we see in Figure 6.5 that both our algorithm and the sizing field method improve on the simulation of the original unrefined mesh (left column). However, the sizing field greedily refines in contact regions with large numbers of unnecessary faces that can significantly slow simulation and lead to overly compliant surfaces locally, that "snag" on boundaries. Please see our supplemental video for detailed trajectories of all three simulations.

Similar results bear out for our 3D test in Figure 6.2(b). Here we again see that our remeshing criterion automatically adapts to both the contact geometries, the relative material stiffnesses of the domains, and the force balance between the coiled spikes and the dropped ball – leading to the resolution and local mesh adaptation necessary to capture all these details. In contrast, we again see that the sizing field rapidly over-refines for the first initial contacts, leading to highly meshed, but minor, side indentations for the first initial collisions with the spikes, but entirely misses the later spike protrusions and compressed coiling as it does not account for the physical solution and relative forces (compare with Figure 6.1). Please also see our supplemental video for more details and a comparison with the simulation of the unrefined starting mesh.

6.4.2 Results

Sharp Contact. In Figure 6.3, we reproduce the *Masticator* example of Wicke et al. [2010] with a large timestep ($h = 0.05\,\text{s}$) to stress-test a deformable bar compressed by a set of rigid boxes. Our algorithm quickly captures the sharp contact interfaces upon collision, followed by more refinement to allow compliance along the block, curvature on top, and initially symmetric bulging of the bar out-of-plane, followed by the start of buckling. In contrast, without refinement, the simulated bar's initial mesh does not have sufficient DOF to capture contact and compliance – these behaviors are missed and instead we obtain a jagged and twisted deformation.

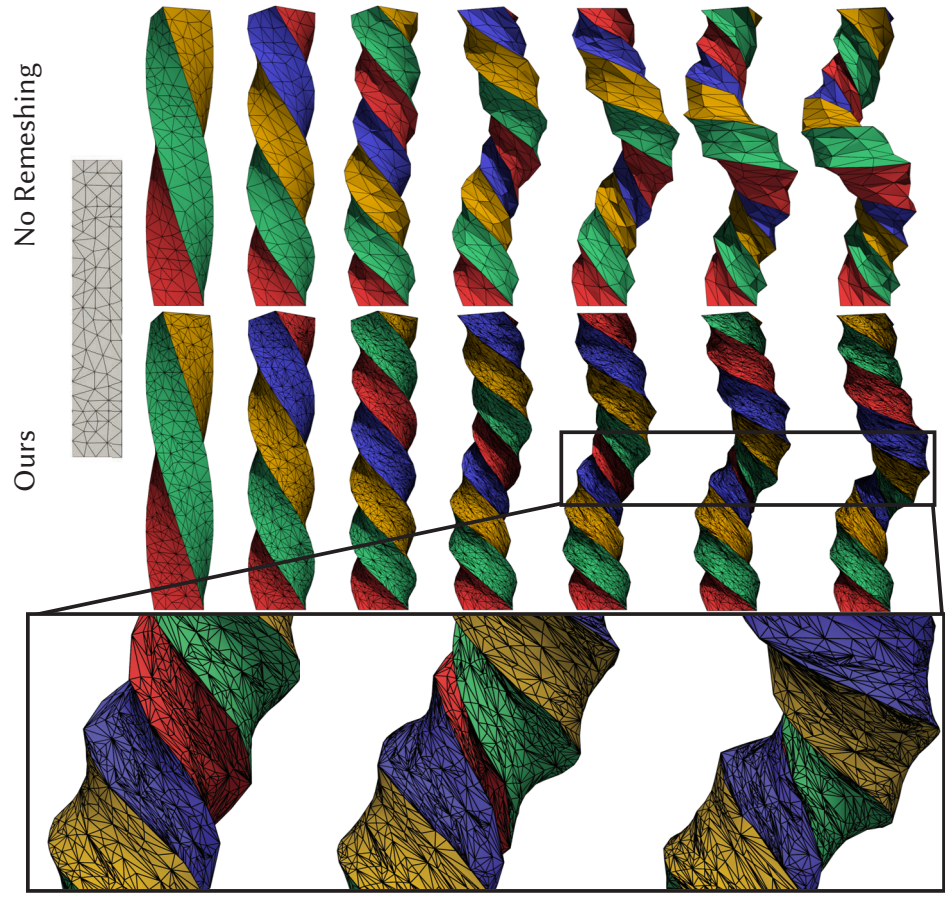


Figure 6.6: Bar-twist. Starting from the same coarse geometry (shown in grey) the resulting deformed mesh is very different without (top) and with (middle) ITR. Our algorithm adaptively adds (and removes) DOF in the mesh to better resolve elastodynamics. As our adaptive simulation progresses we move from regular twisting to buckling. Bottom inset: during buckling our physics-aware remeshing allows the face to collapse (see the center red face) and fold in on itself with tightly resolved contact.

Large Deformation with Self-Contact. In Figure 6.6, a stiff bar ($E = 10^7$ Pa) is anchored on both sides and twisted by rotating its top. In the close-ups we see how prior to contact our algorithm progressively refines the tetrahedral domain as more winding introduces greater curvature and more stress. As winding continues, the simulation adapts to provide even twisting along bar faces and edges until buckling. Upon buckling, we observe in the zoom-in of Figure 6.6 how our simulation of the bar adapts the mesh to capture the collapse, fold-in, and exceedingly tight frictional contact of its faces (e.g., the middle red face). In contrast, simulating directly (unrefined) with the initial bar misses these details and leads to large deformation errors even before

the onset of buckling.

Complex geometry and material-awareness. Varying surface complexity and material stiffness across domains are likewise simultaneously resolved by our algorithm. Here a stiff roller is scripted to rotate, with slightly softer spikes and much softer dropped gorilla geometry in Figure 6.4. On contact, we see the gorilla geometry increasingly refines around the impact site with the bar (which refines less due to a stiffer reaction) and then coarsens as it rebounds.

High-Speed Impact. While above we stress-test ITR with a number of large timestep examples, for many phenomena we may wish to capture detailed deformation occurring at much finer time scales. In Figure 6.8, we model the high-speed impact test from Li et al. [2020] with ITR. A soft ($E = 10^6$ Pa) ball is fired at a wall obstacle with high velocity ($v_0 = 67$ m/s) with a timestep of $h = 2 \times 10^{-5}$ s. Here ITR begins with a much coarser (17× fewer tetrahedra) initial mesh. Then, on impact, ITR *automatically* begins refining the mesh to capture both the rapidly changing contact interface on the surface and the internal propagation of shock waves across the material. It then *coarsens* the mesh as the shockwave passes through, with light, secondary refinement and coarsening applications capturing the subsequent oscillations in free-flight. Please see our supplemental video for the resulting dynamics of the simulation and the changing mesh supporting it.

Dynamic Wave Propagation. In many cases, emergent behavior in a system's dynamics would best define a suitable mesh choice – but this is hard to predict without a higher-resolution simulation result to guide us in the first place. In Figure 6.7, we fix the left side of a coarsely triangulated beam and then drive its other end with periodic vertical oscillations. Appropriately modeled, this driven system should lead to a steady state of attenuating waves damping as they traverse the bar from right to left. Over multiple timesteps, we see that ITR progressively adapts the simulation mesh with increasing corresponding resolution left to right and local adaptations in appropriate

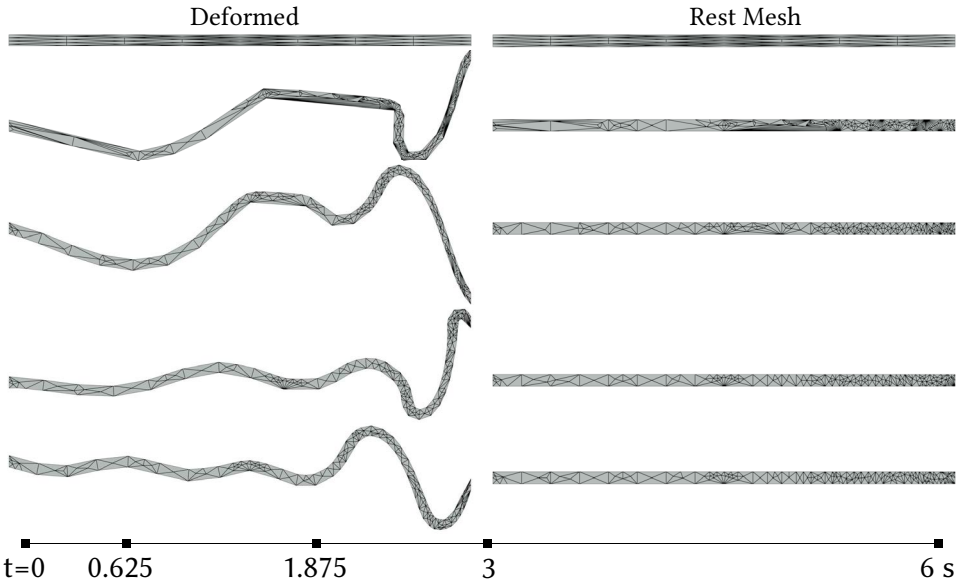


Figure 6.7: Elastic wave. Simulation of a driven periodic wave motion. Starting from a coarse rectangular mesh (top), ITR progressively adapts the simulation mesh with increasing corresponding resolution from left to right and local adaptations in appropriate regions to capture the steady wave dynamics (bottom).

regions to smoothly capture the steady wave dynamics.

Energy Effectiveness of Remeshing. We instrument the above ball-impact example to study the effectiveness of our ITR (Figure 6.9). Across the entire simulation, we compute the energy decrease per timestep of the total incremental potential energy obtained by remeshing from the beginning of the timestep solve (prior to our algorithm initializing the remeshing operation: Algorithm 6.1 Line 3) to the final solution output (Algorithm 6.1 Line 17) on the timestep's adapted mesh: $\Delta E = E(x_{t+1}, \mathcal{T}_{t+1}) - E(x', \mathcal{T}_t)$. In Figure 6.9 we see that as we refine the mesh (noticeably just around the first contact, marker (a)) our method dramatically improves the energy, while during coarsening (e.g., after complete separation in (d)) the energy does not increase, despite the removal of DOF. Looking more closely at the trends we also see proportionately more energy decrease when more refinement operations are performed demonstrating the effectiveness of our criteria's selection and timing of operations.

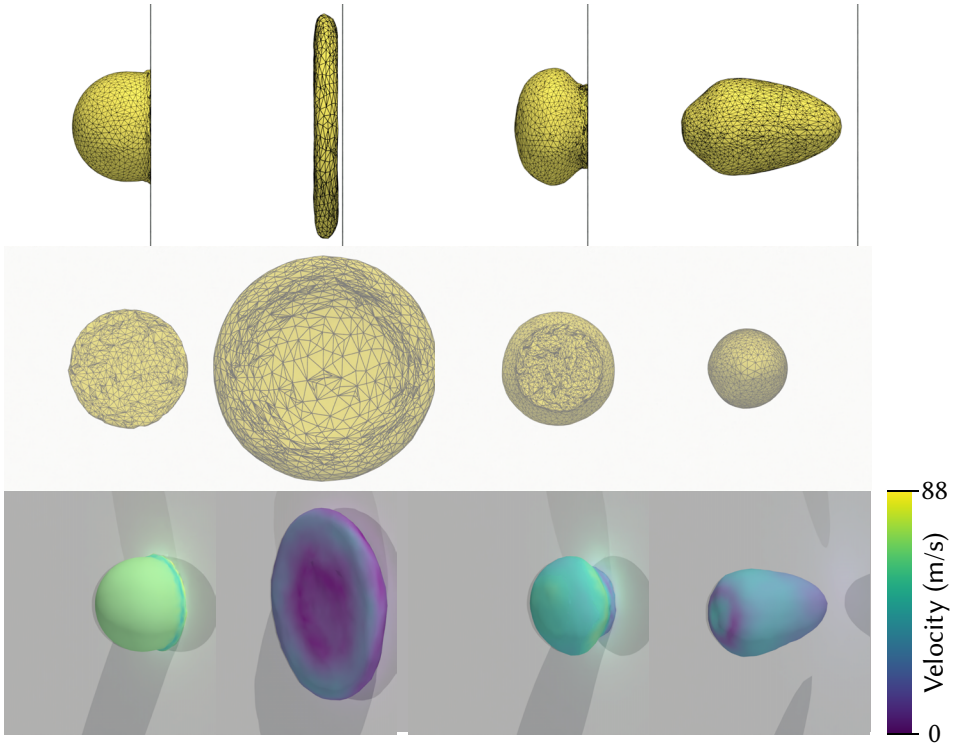


Figure 6.8: Impacting ball. We replicate the high-speed impact test from [Li et al. 2020] where a soft ($E = 10^6$ Pa) ball is fired at a static wall with a high velocity ($v_0 = 67$ m/s). Beginning with a much coarser initial mesh, ITR's adaptive remeshing determines that refinement only begins during initial collision (left column). As the ball bounces away from the wall, ITR then begins removing DOF, which were earlier added to capture the contact dynamics and are now unnecessary. The bottom row shows the velocity magnitude throughout this process.

Stability. As covered in Section 6.2, a fundamental challenge in AM methods for dynamics, especially during coarsening, is stability. Each ITR timestep solve in all the above examples is solved to convergence on the timestep's final output mesh with the prior state safely L2-projected to it. We observe that qualitatively (see our supplemental video), all the trajectories generated by ITR remain stable, and so free of jittering and instability artifacts, e.g. as demonstrated in prior methods by Narain et al. [2013].

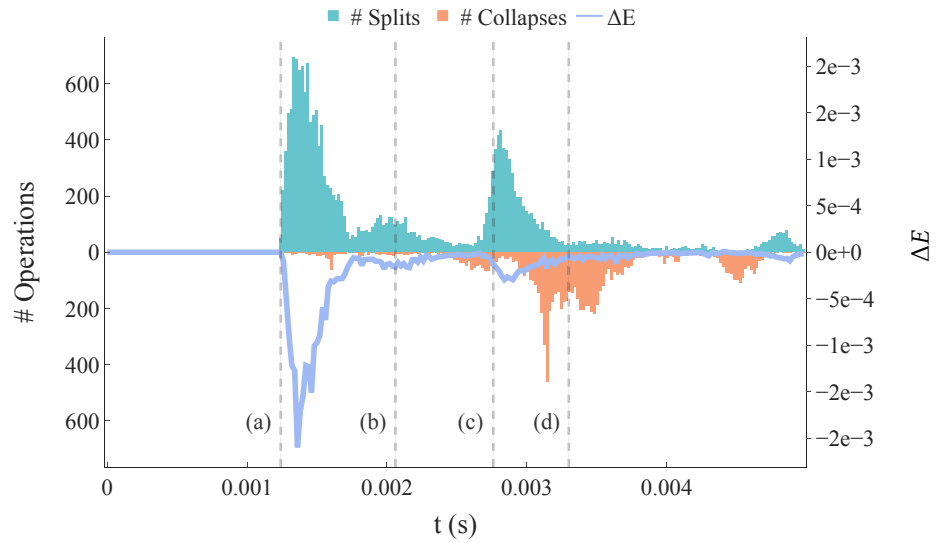


Figure 6.9: For each timestep of the impacting ball simulation (Figure 6.8), we plot the number of splits (green bars), the number of collapses (orange bars), and the change in energy (blue line) from the initial solve of the timestep (prior to remeshing operations) via minimization of E to the final solution of the timestep on the final updated mesh. Key times in the simulation are indicated by virtual lines: (a) first contact, (b) maximal compression, (c) rebound of the material as peels away, and (d) complete separation. The plot shows significant improvements (decrease) in the energy as we apply splitting operations, and at the same time, the coarsening operations do not negatively affect the energy, while increasing efficiency.

6.4.3 Performance and Resolution

For non-adapted (fixed mesh) timestep solves of IPC there are three primary sources of computational cost per Newton iterate: (1) evaluation of the energy potential gradients and Hessians, and their assembly to a global linear system, (2) the linear solve of each such system to compute a descent direction, and (3) line search along this direction. Both (1) and (3) involve the evaluation of potential-energy stencils and collision detection. As such, they generally dominate costs only for exceedingly small systems since, with modern acceleration strategies and easy parallelization, they generally scale close to linear in the number of elements. On the other hand, the large, ill-scaled, sparse linear system solves per Newton iteration in (2) dominate the costs for all practical examples as they require direct solvers [Li et al. 2020] with poor parallel scaling of a memory

bound problem [Lipton et al. 1979].

By adapting the mesh ITR significantly lowers DOF count for high-quality simulation output (see Table 6.1) and so reduces the size of the largest (super-linear cost) solver bottleneck: linear system sizes. At the same time, ITR introduces a new potentially large (albeit linear and currently unoptimized) overhead cost per timestep to evaluate the suitability of each mesh-adaptation proposal.

To evaluate the current runtime performance of ITR with respect to these computational costs, we compare ITR with successive uniform refinements [Ong 1994] (splitting along the first diagonal) in two inter-related analyses. In the first, we measure the *wall-clock* time used by each implementation, including current (unoptimized) costs for ITR's remeshing overhead. Here we report both running time and memory consumption, noting that there is only one value for memory as linear solves are the bottleneck for memory usage. In the second, we consider an *ideal* analysis; we keep in mind that linear solver technology is an advanced, exceptionally well-studied domain with little expectation of significant improvement, while costs for ITR mesh adaptivity are currently linear and not yet addressed in our ITR implementation with significant optimization nor even low-hanging opportunities for parallelization. For the latter, we focus on the DOF difference (for comparable quality output) and the corresponding difference in global system linear solve times for the IPC timestep solver.

Statistics for these comparisons are reported in Tables 6.1 and 6.2. We begin by visually identifying, per benchmark example, the artifact-free baseline UR simulation with qualitatively comparable results to our ITR simulation. As a concrete example, consider the ball on-spike scene, where we observe that simulations with both one and two levels of refinement exhibit significant snagging and severe element distortion; please see Figure 6.10 for examples. On the other hand, for the gorilla roller scene, two-levels of uniform refinement are sufficient to remove most artifacts and obtain qualitatively similar deformation and contact compliance to our ITR result, please see Figure 6.11.

Table 6.1: In-Timestep Remeshing performance comparison. The average running time per timestep, peak memory, and the number of DOF for an unrefined mesh (UR 0), three levels of uniform refinement (UR 1–3), and our method. We bold the values corresponding to the lowest resolution showing comparable and artifact-free results to ITR.

Scene	Average running time per step (s)					Peak memory (GiB)					Number of DOF				
	UR 0	UR 1	UR 2	UR 3	Ours	UR 0	UR 1	UR 2	UR 3	Ours	UR 0	UR 1	UR 2	UR 3	Ours avg (max)
Ball on spikes (Figure 6.1)	20.1	418.1	8,371.6	**40,648.1**	**12,316.6**	0.9	3.7	24.3	**151.5**	**2.0**	27k	144k	900k	**6M**	**43k (55k)**
Masticator (Figure 6.3)	70.8	480.7	**9,781.7**	16,775.8	**8,230.8**	0.9	0.5	**2.7**	16.2	**4.1**	1k	9k	**63k**	476k	**16k (57k)**
Gorilla rollers (Figure 6.4)	15.2	162.0	**3,317.3**	23,372.5	**1,077.8**	1.0	3.8	**24.7**	182.6	**7.7**	18k	115k	**800k**	5M	**22k (27k)**
Bar-twist (Figure 6.6)	0.2	1.5	**232.5**	2,733.6	**2,234.9**	0.1	0.4	**3.1**	21.1	**3.7**	1k	9k	**63k**	476k	**24k (78k)**
Impacting ball (Figure 6.8)	0.3	5.9	**115.8**	8,564.0	**960.8**	0.2	1.1	**8.3**	71.8	**2.9**	5k	34k	**248k**	1M	**50k (61k)**

Table 6.2: Average linear solver running time.

Scene	UR 0	UR 1	UR 2	UR 3	Ours
Ball on spikes (Figure 6.1)	0.08	0.81	9.88	201.77	0.25
Masticator (Figure 6.3)	0.01	0.06	0.78	9.39	0.22
Gorilla rollers (Figure 6.4)	0.12	1.23	16.82	245.44	0.17
Bar-twist (Figure 6.6)	0.00	0.05	0.57	8.77	0.21
Impacting ball (Figure 6.8)	0.02	0.28	3.60	126.68	0.37

In summary, we see ITR's DOF improvement ranging from 2.6 to 185× less DOF per example with corresponding 2.7 to 1,444× linear solve speedups. At the same time, the impact of our initial, unoptimized implementation of our remeshing procedures on wall-clock time varies significantly across examples, ranging from 3.3× speedup for the complex ball on spikes scene to 9.6× slowdown for the much simpler bar-twist scene.

Opportunities for wall-clock performance improvement. We identify four high-impact and immediate directions for future extensions that we believe will likely provide significant improvement in remeshing costs (and so runtimes) for ITR: (1) the most immediate and low-hanging opportunity is the development of parallel and distributed versions of ITR; (2) similarly low-hanging is the application of custom collision-detection that is spatially localized to leverage the small local support that our individual mesh operations evaluate in our local-solve updates (currently this is still applied globally), (3) exploiting both temporal- and spatial-coherence during

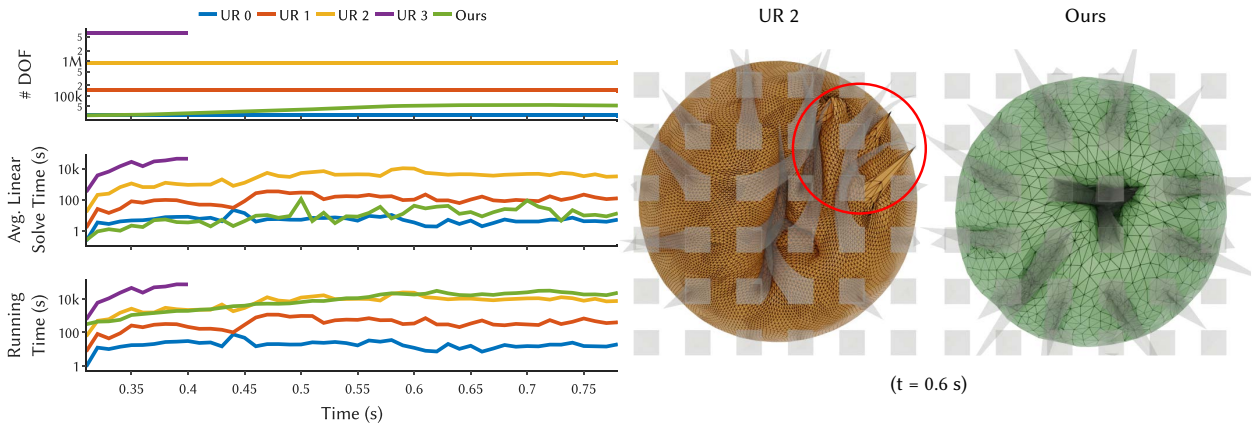


Figure 6.10: Ball on spikes uniform comparison. Here we plot a detailed view of the performance of the ball on spikes simulation (Figure 6.1). We compare an unrefined mesh (UR 0), three levels of uniform refinement (UR 1–3), and our method. Circled in the rendering on the right, it is clear that UR 2 was insufficient in capturing the local deformations and stretching caused by the spike tips.

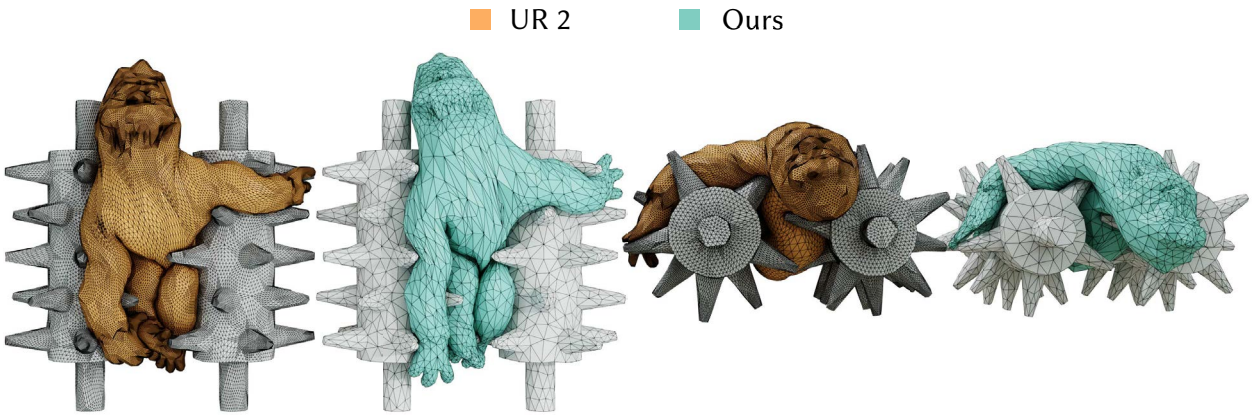


Figure 6.11: Gorilla rollers uniform comparison demonstrates results of the gorilla roller simulation with two levels of uniform refinement (UR 2) and ITR ("Ours") at the halfway point of the simulation.

collision-detection and culling, and (4) exploration of higher-order bases and geometry to further reduce DOF count.

6.5 Discussion

We have proposed ITR, a first fully-coupled adaptive-remeshing algorithm for implicit time-stepping elastodynamics with frictional contact via a spatially continuous incremental potential

Table 6.3: IPC simulation and ITR parameters. For each example, we report the timestep size (h), density (ρ), Young's modulus (E), Poisson ratio (ν), barrier activation distance ($\hat{d}$), barrier stiffness (κ), coefficient of friction (μ), friction accuracy parameter (ϵ_v), and max friction iteration setting. We also report the split and collapse acceptance tolerances ($\delta_{S,C}$) and the culling thresholds ($\epsilon_{S,C}$). For all examples, we use a Newton convergence criteria of $\|\Delta x\|/h \leq 10^{-3}$ m/s, implicit Euler time integration, and a neo-Hookean material.

Scene	h (s)	ρ (kg/m^3), E (Pa), ν	$\hat{d}$ (m), κ (Pa)	μ, ϵ_v (m/s), friction iters.	δ_s (J), δ_c (J)	ϵ_S, ϵ_C
Ball on spikes (Figure 6.1)	0.01	2000/1100, 1e5/1e8, 0.4	1e-3, 6e4	0.1, 1e-3, 1	1e-5, -1e-8	0.95, 0.01
2D Masticator (Figure 6.5)	0.05	1e3, 1e4, 0.4	1e-3, 1e3	0.1, 1e-3, 1e3	1e-3, -1e-8	0.95, 0.01
3D Masticator (Figure 6.3)	0.05	1e3, 1e4, 0.4	1e-3, 1e3	0.1, 1e-3, 1e3	1e-5, -1e-8	0.95, 0.01
Gorilla rollers (Figure 6.4)	0.01	1e3, 5e4/2e7/2e8, 0.3	1e-3, 3e5	0.5, 4e-3, 1	1e-5, -1e-8	0.95, 0.01
Bar-twist (Figure 6.6)	0.01	1e3, 1e7, 0.4	1e-3, 1e6	-	1e-3, -1e-8	0.95, 0.01
Elastic wave (Figure 6.7)	0.025	1340, 1e4, 0.495	1e-3, 1e3	-	5e-5, -1e-8	0.85, 0.01
Impacting ball (Figure 6.8)	2e-5	1150, 1e6, 0.45	6.9e-5, 1e5	-	1e-14, -1e-16	0.95, 0.01
Cantilever (Figure 6.12)	0.1	1e6, 1.1e9, 0.3	1e-3, 1.1e8	-	*,-1e-13	0.6, 0.4

merit function. To do so ITR ensures non-penetration and non-inversion throughout all operations in both remeshing and solving. In turn, it applies robust physics-aware remeshing to generate stable and accurate trajectories with low DOF counts. Simulated geometries conform well to necessary contacting interfaces and deformations with parsimonious refinement where new DOF are needed to improve the solution, and effective coarsening where they are not.

6.5.1 Limitations and Future Work

Along with the opportunities for improved remeshing operation performance discussed above in Section 6.4.3 we see a number of additional avenues for fruitful improvements and extensions.

Currently, we empirically demonstrate improved solution behavior on a wide range of challenging examples. However, an important next step, which we do not address here is a formal convergence study of our refinement. We provide a preview of such a study in Figure 6.12, where

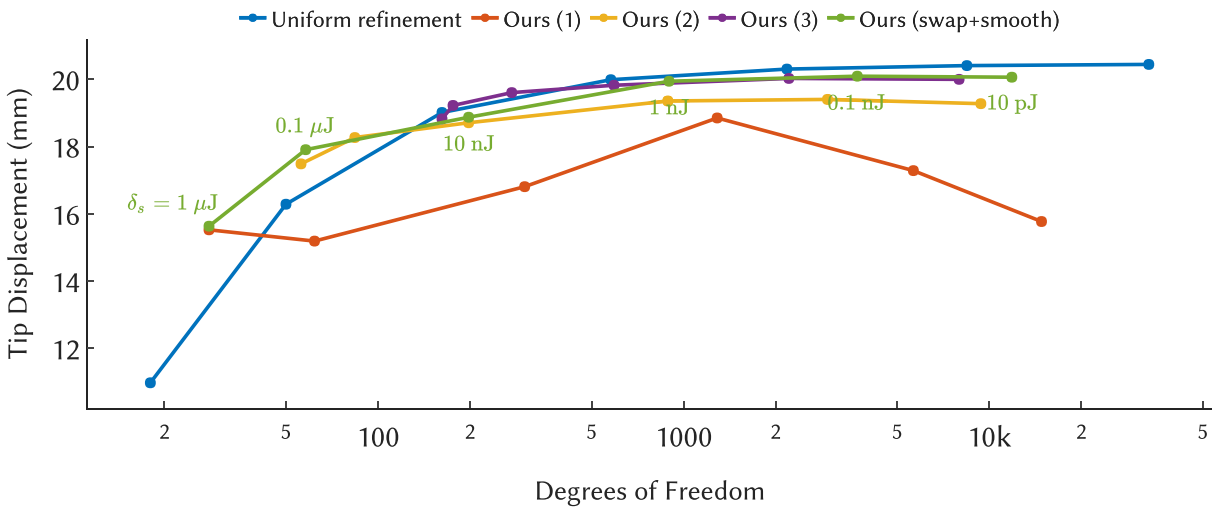


Figure 6.12: Cantilever convergence. Using a cantilever example (see e.g., Pelteret [2016]), we examine the convergence behavior of ITR with varying refinement acceptance tolerances δ_s from 1 μJ to 10 pJ. We observe that the accuracy of our method is largely dependent on the initial discretization: Ours (1–3) start from 1 to 3 levels of initial refinement, respectively. As a proof-of-concept, we also test a preliminary extension of ITR that additionally utilizes edge-swapping and vertex smoothing, starting from the same mesh as Ours (1). Here we see that these operations are important to "breakaway" from the initial discretization.

we consider a cantilever convergence test (see e.g., Pelteret [2016]). As can be seen, ITR's convergence is currently highly dependent on the initial discretization. At least in part, this dependence appears closely related to controlling for mesh *quality*.

We observe that while split and collapse operations are effective for adaptive updates, they are insufficient to preserve mesh quality and so limit the convergence and range of refinement that ITR can currently apply. The inclusion of edge/face flips/swaps will be a simple and direct improvement that naturally fits within the ITR framework, as will be explicit optimization for mesh quality [Wicke et al. 2010]. As a proof-of-concept investigation, in the above cantilever experiment, we have updated ITR operations to additionally include a preliminary version of edge flips and vertex smoothing. As we see in Figure 6.12 this improves ITR's convergence.

Additional extensions of ITR's adaptivity to also include r-refinement should also be valuable. Likewise, while we focus here solely on volumetric elastodynamics, ITR should usefully extend to codimensional models for shell and rod simulations and even alternative contact models.

We hope that this work and its reference implementation will encourage further research on the application of adaptive unstructured remeshing. As simulation methods advance and problem complexities grow, it becomes all the more important to judiciously apply computation where it can be most effective.

7 Conclusion

To summarize the contributions of this thesis, we have developed a new method for the simulation of contacts that is provably intersection- and inversion-free. This method, IPC, is able to simulate elastodynamics and rigid body dynamics with complex scenes while exhibiting high accuracy and robustness. We additionally, introduce a systematic way of testing CCD methods and have proposed an efficient floating-point-based method that is provably correct. Finally, we have shown that IPC can be used to simulate scenarios with high accuracy through the use of high-order elements and adaptive meshing. All of this is without the need to tune parameters.

This opens the door to many future research opportunities both within computer graphics and across disciplines. It is astonishing to see how quickly these methods have been adopted and adapted [Fang et al. 2021; Gholamalizadeh et al. 2022; Kim et al. 2022; Lan et al. 2022a,b, 2021; Li et al. 2021; Moshfeghifar et al. 2022], yet there is still a large amount of work that can be done in the near to distant future.

7.1 Ongoing and Future Work

Real-time simulation. A large area of computer graphics research is dedicated to devising methods for real-time simulation. The methods proposed here, however, are far from real-time for complex scenes. To address this limitation, Lan et al. [2022a] revisit the problem of rigid body dynamics (Chapter 4) and show that by relaxing the rigidity constraints they can get up to three

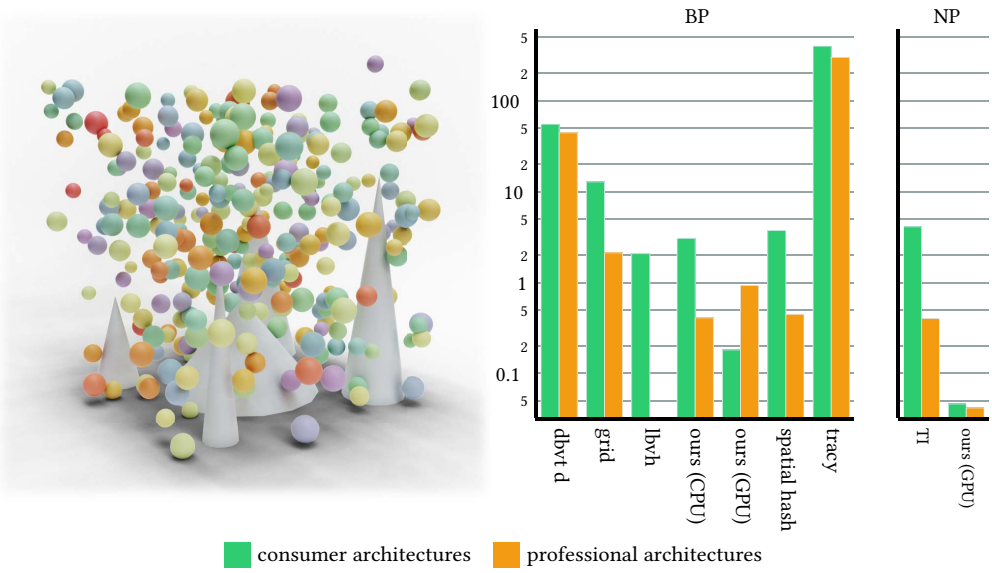


Figure 7.1: GPU CCD. By utilizing the massively parallel nature of GPU's we are able to get up to a two-orders-of-magnitude improvement in CCD performance [Belgrod et al. 2022]. Here we show one such result where we take a scene from the UNC Dynamic Scene Benchmarks [Curtis et al. 2012] of several rigid spheres floating through space. On the right, we plot the performance for both the broad-phase (BP) and narrow-phase (NP). Compared to existing BP methods, we are able to get around an orders-of-magnitude speed-up, and compared to Tight Inclusion (TI) (Chapter 3) we are able to get two orders-of-magnitude faster performance. This is all while maintaining the same provable guarantees provided by our original method (Chapter 3).

orders-of-magnitude speed-up. This so-called Affine Body Dynamics (ABD), is able to completely avoid the need for expense nonlinear CCD. Simultaneously, Lan et al. [2022b] show that by using an approximate model of elasticity known as Projective Dynamics (PD) [Bouaziz et al. 2014] in conjunction with IPC, they are able to get qualitatively reasonable animations in real-time while maintaining the same penetration-free guarantees of IPC. Further investigating these approaches for their quantitative accuracy and exploring other avenues of real-time performance (e.g., PBD [Müller et al. 2007]) is exciting future work.

CCD FOR PARALLEL ARCHITECTURES. A large opportunity for potential performance improvements of our method is the utilization of massively parallel architectures such as the GPU. As a first effort in creating a full GPU implementation, we investigate how to best utilize GPU archi-

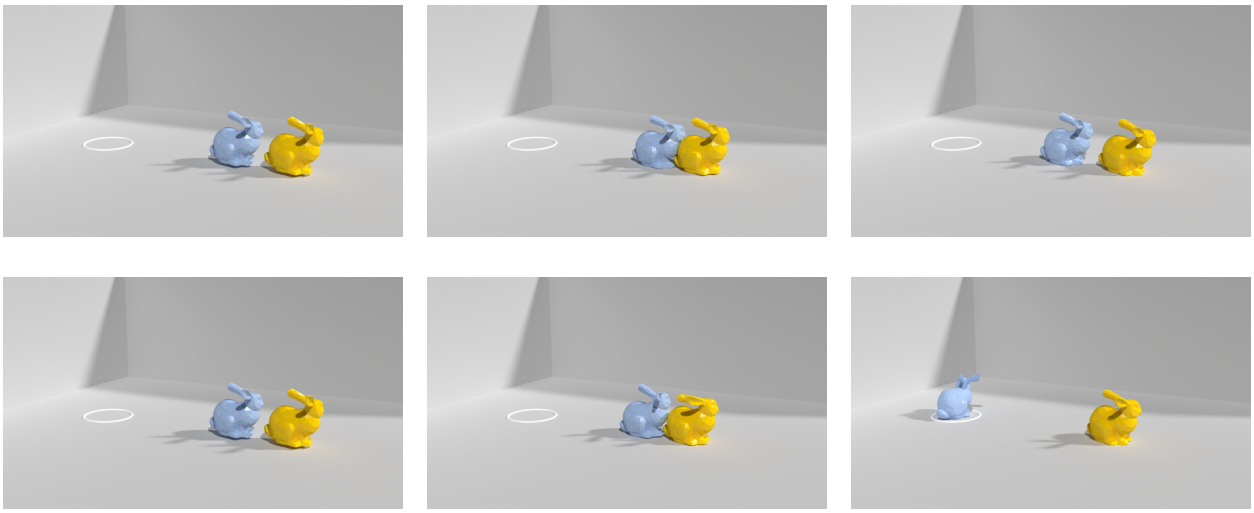


Figure 7.2: Differentiable simulation: bunny pool. The direction and magnitude of the initial velocity of the yellow bunny are optimized to push, after contact, the blue bunny into the white circle marker. The top row shows the initial configuration, and the bottom row shows our optimized result. This scene involves an elastodynamic simulation with a non-linear material model with contact and friction forces.

tecture for collision detection [Belgrod et al. 2022].

Differentiable simulation. Because we model contact and friction via a smooth potential, the entire model is fully differentiable. This allows us to utilize IPC within inverse design applications such as shape optimization [Huang et al. 2022] or topology optimization[Zong et al. 2022]. We can also use IPC for control problems such as in Figure 7.2 or material and coefficient of friction estimation [Huang et al. 2022]. Unlike prior methods which utilize penalty-based contact handling [Du et al. 2021; Geilinger et al. 2020], we finally are able to utilize IPC in complex scenarios such as Figure 1.1 with guarantees of feasible results over designs.

Robotics. Simulation tools are widely used in robotics for reinforcement learning [Zhao et al. 2020]. However, they often rely on simulation frameworks designed for efficiency and not robustness or accuracy (e.g., Bullet [Coumans and Bai 2019], MuJoCo [Todorov et al. 2012], or NVIDIA PhysX [Nvidia Corporation 2021]). Recently several papers have shown the applicability of our work to robotics [Chen et al. 2022; Kim et al. 2022]. In particular, Kim et al. [2022] utilize our

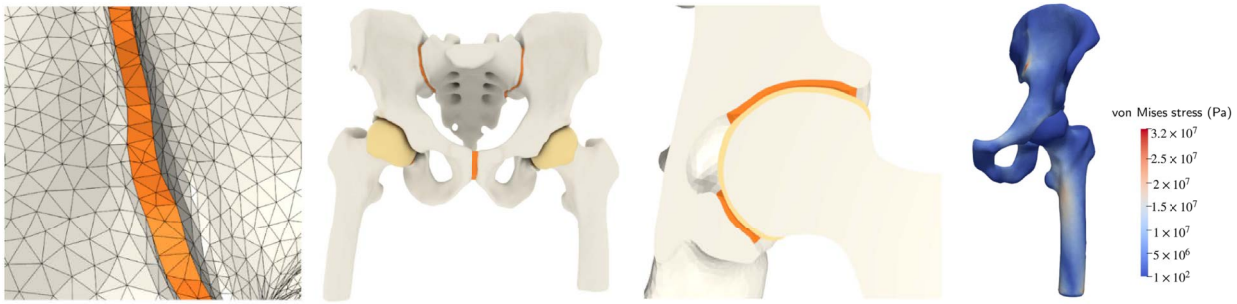


Figure 7.3: Biomechanics: hip geometry and simulation. Simulations in biomechanics pose a challenge for prior methods because they commonly feature large ratios of stiffness (e.g., on the right is a soft tissue compressed between stiff bones), thin geometries (see the sliced view on the center-right), and a need for high accuracy. On the right, we show IPC is able to seamlessly simulate even these challenging scenarios [Moshfeghifar et al. 2022].

open-source implementation to show that with IPC they are able to more accurately predict if a robotic arm will successfully grasp and pick up an object. Chen et al. [2022] propose an entire multi-joint robotics simulator based on IPC and the work of [Lan et al. 2022a].

An interesting future direction that has yet to be fully explored is the use of IPC for *soft robotics*. Soft robotics focuses on the manipulation of the world through the use of compliance. There is a lot of potential in directly applying our accurate FE methods to compliant mechanisms, pneumatics, and more.

Biomechanics. Simulations in biomechanics feature a lot of traditionally challenging parts: larger material stiffness ratios (e.g., muscle and bone), thin materials (e.g., ligaments between bones), high demand for realism (e.g., pre-simulation of fabricated prostheses), etc. Figure 7.3 provides one such case when calculating the stresses exerted under gravity on a hip and femur. The state-of-the-art for simulation in biomechanics is FEBio [Maas et al. 2012], but it commonly suffers from intersections and inversions. This is not only unphysical but can lead to an outright inability to simulate certain scenarios. Working with these methods requires tuning simulation parameters (e.g., timestep size) and material parameters (e.g., stiffness). However, it can be challenging to find a good set of parameters, and comparing results with differing parameters is

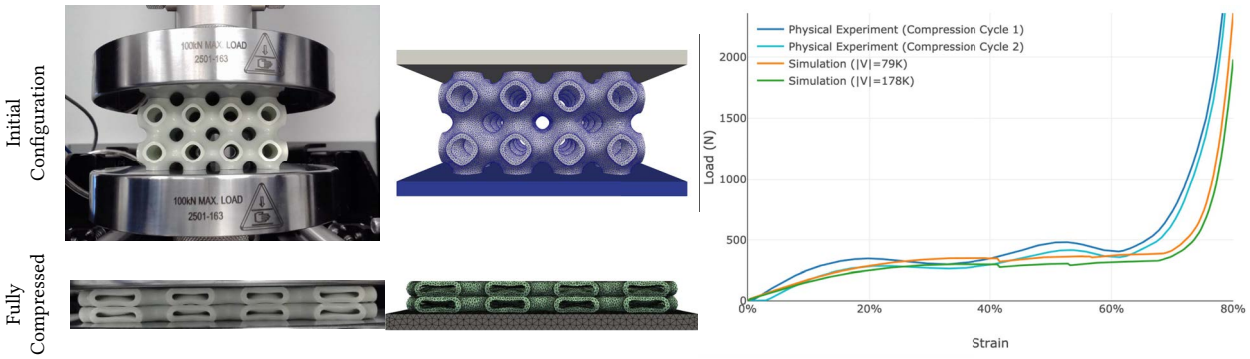


Figure 7.4: Physical validation. We compress a 3D-printed lattice (Schwarz) structure and measure the force exerted onto a sensor plate at the bottom. Replicating this setup virtually, we are able to analyze the physical accuracy of IPC. On the right, we show the strain vs. load curves match well up to ~ 60% strain where the simulation starts to lag behind the measured values. This shows great potential in the ability of our work to predict the physical behavior of new designs before fabrication.

challenging as they lead to differing dynamics.

In contrast, our work on IPC guarantees intersection and inversion-free dynamics independent of parameters. As part of an ongoing effort to fully utilize IPC in biomechanics, we have tested our method on various scenarios and find it can match and even exceed the accuracy of FEBio especially on complex scenes where FEBio often crashes [Gholamalizadeh et al. 2022; Moshfeghifar et al. 2022]. The robustness provided by our work opens a new door to the large-scale utilization of simulation tools in biomechanics.

Physical validation and accuracy. A wider question when utilizing simulation tools is how accurate are they to the physical world. To answer this, we collaborate with industry partners at Carbon. In particular, Carbon works with several companies to release replacements for traditional foam parts using 3D-printed lattice structures (see Figure 7.4). To aid in the design of these parts, simulation can be used to predict the mechanical behaviors of lattices and perform a search of material properties based on simulated predictions.

Simulation of lattices is particularly challenging due to their thin parts, nonlinear buckling, a large number of contacts when in compression, and the high degree of compression these parts undergo. Existing commercial methods struggle to simulate these scenarios, so we explore the

applicability of IPC to high-accuracy settings. As a first step in this effort, we compare the forces exhibited by various 3D-printed lattices under compression (Figure 7.4). Early, results show our method can match the expected behavior of the real-world lattices and robustly handle scenarios for which commercial solutions fail (e.g., large buckling and large compression). These results show great promise in the ability of our research to improve traditional design and manufacturing pipelines.

A | Incremental Potential Contact: Technical Details

A.1 Smoothing

Let $f(x)$ be a function we wish to smooth. It is C^1 continuous everywhere except at $x = a$ where it is only C^0 continuous. Applying a function $g(x)$ that is C^1 continuous everywhere with $g(a) = 0$, we have a smoothed function $f(x)g(x)$ that is C^1 continuous everywhere. For $x \neq a$ we have $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$ is C^0 continuous everywhere. At $x = a$, the left and right derivatives of $f(x)g(x)$ are then

$$\begin{aligned} \lim_{x \to a^-} (f(x)g(x))' &= \lim_{x \to a^-} f'(x)g(a) + f(a)g'(a), \\ \lim_{x \to a^+} (f(x)g(x))' &= \lim_{x \to a^+} f'(x)g(a) + f(a)g'(a). \end{aligned} \tag{A.1}$$

As $f(x)$ is C^0 continuous at $x = a$, $\lim_{x \to a^-} f'(x)$ and $\lim_{x \to a^+} f'(x)$ are both bounded. Then with $g(a) = 0$ we then have left and right derivatives of $f(x)g(x)$ both equal $f(a)g'(a)$ at $x = a$,

$$\lim_{x \to a^-} (f(x)g(x))' = \lim_{x \to a^+} (f(x)g(x))' = f(a)g'(a) \tag{A.2}$$

Thus $(f(x)g(x))'$ is likewise C^0 continuous at $x = a$ and $f(x)g(x)$ is correspondingly C^1 continuous.

Table A.1: Ablation study on barrier functions with different continuity. NC means not converging after 10000 PN iterations when solving a certain time step.

Examples	CO # iters, **t** (s) / time	C1 # iters, t (s) / time	C2 (IPC) # iters, t (s) / time
Mat on knives	5.34, 1.66	5.59, 1.43	5.47, 1.40
2 mats 40 × 40 fall	NC	26.83, 10.64	26.17, 10.48
Octocat on knives	NC	9.29, 4.21	7.73, 3.76
Sphere (1K) roller	NC	47.64, 9.41	45.22, 9.02
Mat 40 × 40 twist (10 s)	7.74, 1.93	8.04, 2.07	7.56, 1.89
Sphere (1K) pin-cushion	12.82, 1.27	9.22, 0.75	8.93, 0.69
Rods (630) twist (10 s)	6.65, 1.05	3.05, 0.38	3.07, 0.40

A.2 Barrier Continuity and Testing

The continuity of our C^2 barrier function is confirmed when $d < \hat{d}$, as $\frac{\partial b}{\partial d} = (\hat{d}-d)(2\ln\frac{d}{\hat{d}}-\frac{\hat{d}}{d}+1)$ and $\frac{\partial^2 b}{\partial d^2} = -2\ln\frac{d}{\hat{d}} + (\hat{d}-d)\frac{\hat{d}+3d}{d^2}$ both vanish as $d \to \hat{d}$. Thus the left and right derivatives of b at $d = \hat{d}$ are both equal at zero up to 2nd order.

Our motivation for applying a C^2 clamped barrier rather than a less nonlinear, but still smooth, C^1 barrier is to provide 2nd-order derivatives suitable for our Newton-type solver. Thus it is genralluy better to have a continuous Hessian for improved convergence [Nocedal and Wright 2006]. Nevertheless, here we also provide an ablation study applying all both C^0 ($b = -\ln(d/\hat{d})$) and C^1 ($b = (d-\hat{d})\ln(d/\hat{d})$), along with our final choice of our C^2 ($b = -(d-\hat{d})^2\ln(d/\hat{d})$) barrier in IPC on a set of examples in Table A.1.

From the results we see that for the C^0 barrier, optimization can be non-convergent. Here this can be a result if the local minima is right inside the clamped region of the barrier, where the gradient does not change smoothly – here intermediate values may not be found to balance terms so decrease the total gradient. While, in comparison to C^1, our C^2 barrier is generally 5 % to 10 % faster due to the continuity of the Hessian – this is reflected in less iteration counts.

A.3 CFL-inspired Culling of CCD

As IPC requires performing CCD for every Newton iteration, CCD clearly becomes a bottleneck. Therefore we propose a novel CFL-inspired culling strategy to accelerate CCD.

Recall that our culled constraint set contains all surface primitive pairs with distances smaller than $\hat{d}$. Thus all remaining primitives outside the culled constraint set have farther distances lower-bounded by $\hat{d}$. We place all vertices participating in these primitives in a set $\mathcal{F}$. At each iteration i with a search direction p^i, we find the index ℓ of the simulation node with the largest component in p (i.e., its effective search-direction "velocity") among all remaining node pairs, $\ell = \text{argmax}_{\kappa \in \mathcal{F}} \|p_k^i\|$. We then compute a conservative bound on the largest-feasible step size for surface primitives not in $\hat{C}$ as

$$\alpha_{\mathcal{F}} = \frac{\hat{d}}{2\|p_\ell^i\|}. \tag{A.3}$$

We then apply conservative CCD (see below) to the remaining primitive pairs in $\hat{C}(x^i)$, obtaining a large feasible step size, $\alpha_{\hat{C}}$, for that set. Then $\alpha_0 = \min(\alpha_{\mathcal{F}}, \alpha_{\hat{C}})$ is our maximum *feasible* step size for iteration i. We then apply α_0 as starting step size to bootstrap backtracking for Newton iteration i and so ensure decrease with feasibility.

Computing large, feasible step sizes in this way effectively reduces CCD cost by two-orders of magnitude. However, for scenes that contain high speed motions, $\alpha_{\mathcal{F}}$ can be overly conservative (smaller than needed) which then would increase iteration counts and so cause energy related computations to increase overall cost unnecessarily.

Thus we adapt by balancing between applying full CCD for all candidate pairs provided by spatial hash and applying our CFL-like strategy. Here we will designate the exact feasible bound computed from applying CCD to all primitive pairs in the spatial hash as $\alpha_{\mathcal{S}}$.

In each step we first compute $\alpha_{\mathcal{F}}$ and $\alpha_{\hat{C}}$ – they are both efficient and inexpensive to find. Next we observe that the culled bound $\alpha_{\hat{C}}$ is often very close to the exact bound $\alpha_{\mathcal{S}}$, while computing

Table A.2: CCD strategies ablation.

Examples	Full CCD # iters, t (s)/time step	CFL combined (IPC) # iters, t (s)/time step	Full CCD CCD related timing (s)	CFL combined (IPC) CCD related timing (s)
Mat on knives	5.34, 1.66	5.47, 1.40	97.50	47.25
2 mats 40 × 40 fall	23.22, 10.21	26.17, 10.48	273.38	162.10
Octocat on knives	6.99, 3.92	7.73, 3.76	206.61	129.93
Sphere (1K) roller	49.38, 9.89	45.22, 9.02	482.13	394.01
Mat 40 × 40 twist (10 s)	7.37, 2.13	7.56, 1.89	147.60	62.91
Sphere (1K) pin-cushion	8.35, 0.77	8.93, 0.69	37.25	19.61
Rods (630) twist (10 s)	3.04, 0.47	3.07, 0.40	23.89	9.13

this exact bound is generally one-third of the total timing cost in a single iteration. We thus proceed by computing $\alpha_{\mathcal{S}}$ if $\alpha_{\mathcal{F}} < \frac{1}{2}\alpha_{\hat{\mathcal{C}}}$. Otherwise we apply our CFL-type bound and apply $\alpha_0 = \min(\alpha_{\mathcal{F}}, \alpha_{\hat{\mathcal{C}}})$.

We thus avoid overly restrictive step sizes when $\alpha_{\mathcal{F}}$ and $\alpha_{\hat{\mathcal{C}}}$ are already quite close – meaning their minimum should also be quite close to $\alpha_{\mathcal{S}}$. In practice we observe that our CFL-like assisted CCD culling strategy provides a 50 % speed-up for all CCD related costs with nearly the same iteration counts. Ovxerall this results in an average 10 % speedup for IPC; see Table A.2.

A.4 Conservative CCD

CCD is generally applied to compute a TOI corresponding to a step size that would bring distances between primitives to 0. In our barrier setting this "largest feasible step size" needs to be made conservative by backing away from an exact zero distance. A simple strategy would be a conservative rescaling with a factor $c \in (0, 1)$; e.g., by starting the line search at 0.5 or 0.9 of the total step along the descent direction. However, for the CCD computations rounding error can be severe for the tiny contacting distances we allow and so even small naive scaling factors (e.g., 0.1) can allow unacceptable intersections in such cases while in others is a much too conservative bound unnecessarily slowing convergence.

Rather than directly finding and then conservatively rescaling a CCD-computed step length that takes us to intersection we directly compute via CCD a step size along p^i that will bring

primitives to a distance of $(1-c)d_c > 0$. Here d_c is the current distance between the primitives. Standard CCD libraries [1] directly provide this option, e.g., exposed as an η parameter. In turn this modifies coefficients of the polynomial equations solved during CCD, and effectively reduces numerical rounding errors that cause issues for direct scaling. In our implementation we apply $c = 0.8$ between mesh primitives, $c = 0.9$ for mesh-to-plane, and similarly $c = 0.8$ for computing the large feasible step size to avoid element inversion when barrier elasticity energies, e.g., neo-Hookean, are applied.

Finally, to ensure we remain intersection- and inversion- free at each step. We apply a post-step check whenever nodal positions are displaced (e.g. line search and initial movements of boundary conditions and collision objects. These include the edge-triangle intersection check filtered by our spatial hash and a volume check for every tetrahedral element. In exceedingly rare cases when an edge-triangle intersection or negative volume are detected, we half the final step size bound.

A.5 Equality Constraints for Moving Collision Objects and Time-Varying Boundary Conditions

In many scenarios, e.g., scripting animations, kinematic objects, and engineering tasks, scripted kinematic collision objects (CO) and/or moving positional (Dirichlet-like) boundary condition (BC) are required. Contact algorithms generally handle these functionalities by either directly prescribing and updating nodal positions of CO/BC nodes at start of each time step, or interpolating them in substeps across a step. Remainder of simulation DOF are then solved w.r.t. the prescribed nodes being fixed at "current" positions. However, such strategies are extremely limiting, with simulations generally restricted to small time step sizes, speeds, and/or deformations as the BC and/or CO become faster and more challenging. For example, directly prescribing

[1] we use https://github.com/evouga/collisiondetection

CO or BC nodes can often generate tunneling artifacts, and for moving BC, simulations can often fail simply by inverting elements when barrier energies like neo-Hookean energy are applied. To address these issues we formulate scripted dynamic BC and CO as equality constraints in IPC. This simultaneously ensures that intersections (and inversions) are avoided even while scripted motions are applied at large time step.

We start by computing the prescribed nodal positions at the beginning of each time step, and then apply CCD and element inversion detection to find a large feasible step size towards the prescribed position from the current one (see above).

If it is safe to apply them fully without causing intersection or inversion, we simply update prescribed nodal positions, solve the remainder of simulation DOF and are done. However, if our feasible step size is smaller than taking a full step (we use a criterion of 0.999), we first move the prescribed nodes as far as we can conservatively (see Appendix A.4) and then add new *equality constraints* for each of these prescribed nodes provided by either CO or BC scripting.

As in our treatment of intersection-free *inequality* constraints, we build an unconstrained form for each by applying an augmented Lagrangian. For every such prescribed node, we add a Lagrangian and a penalty potential. We initialize each Lagrange multiplier to 0 and each penalty stiffness to 10^6. Concretely, we add an energy

$$-\sqrt{m_k}\lambda_{\mathrm{A},k}^{\top}(x_k - \hat{x}_k) + \frac{\kappa_{\mathrm{A}}}{2} m_k \|x_k - \hat{x}_k\|^2 \tag{A.4}$$

to our barrier-form incremental potential for each prescribed node k with corresponding destination $\hat{x}_k$ in the current time step, if *any* of the prescribed nodes could not reach its destination during our start-of-time-step test.

We measure satisfaction of BC/CO node constraints at each Newton iteration i by calculating

Algorithm A.1 Augmented Lagrangian Update Rule

for each PN iteration i **do**
    ...
    if $\frac{1}{h}\|p^i\|_\infty < \max(10^{-2}l, \epsilon_d)$ **and** $\eta_{\text{AL}} < 0.999$ **then**
        if $\eta_{\text{AL}} < 0.99$ **and** $\kappa_{\text{AL}} < 10^8$ **then**
            $\kappa_{\text{AL}} \leftarrow 2\kappa_{\text{AL}}$
        else
            for each constrained node k **do**
                $\lambda_{\text{AL},k} \leftarrow \lambda_{\text{AL},k} - \kappa_{\text{AL}}\sqrt{m_k}(x_k^i - \hat{x}_k^{t+1})$

their total in-time-step progress as

$$\eta_{\text{A}} = 1 - \sqrt{\frac{\sum_k \|\hat{x}_k^{t+1} - x_k^i\|^2}{\sum_k \|\hat{x}_k^{t+1} - x_k^t\|^2}}, \tag{A.5}$$

where $\hat{x}^{t+1}$ is the prescribed BC/CO positions for time step $t+1$. Then, whenever a current iterate is both close to satisfying stationarity, via the stopping criteria, and progress, with $\eta_{\text{A}} < 0.999$, we either increase the BC/CO penalty stiffness or else update the Lagrange multipliers, via the first-order update rule, see Algorithm A.1. Finally, whenever $\eta_{\text{A}} \geq 0.999$, we fix all prescribed nodes at current position and solve for remaining DOF in order to avoid unnecessary slow down of convergence due to added stiffness.

For codimensional surface and segment collision objects their nodal mass is computed by estimating their nodal volume as the half sphere with diameter being the average length of the incident edges. For point collision objects we set their mass to be the average nodal mass of the simulated objects. Alternatively, simply setting all codimensional nodal masses to be the average of the simulated objects should also be fine. These estimated masses are only used for moving codimensional collision objects and they do not affect any physical accuracy.

A.6 Adaptive Barrier Stiffness

Stiffness, and so difficulty in solution of the barrier comes from two sources: $\hat{d}$ and likewise κ. As we can improve accuracy by directly decreasing $\hat{d}$, this frees κ to adaptively condition our solves for improved convergence.

A feature of our barrier framework is that the estimation of Lagrange multipliers are efficiently self-adjusted by the constraint values, in our case, the d_k's. However, if κ is not set appropriately, the contact primitives would either need to get extremely close to get enough repulsion (barrier gradient) when κ is too small, or need to get a distance right below $\hat{d}$ for a small repulsion if κ is set too large, both resulting in slow convergence because of ill-conditioning and nonsmoothness. Thus adaptively setting and/or adjusting κ is essential.

Intuitively, the smaller d_k is, the better the complementarity condition is satisfied. This is certainly true from optimization theory but is troublesome in numerical computation. In numerical optimization, since double precision floating point numbers are used, when d_k gets tiny, the rounding errors will significantly slow down convergence and even result in incorrect intermediate computations. Therefore, we must also prevent d_k from being too tiny.

If we view $\hat{d}$ as a control on the upper bound acting distance of our contact forces, κ can be seen as an indirect control on the lower bound of the acting distance as larger κ can provide the same amount of "repulsion" at a larger distance, avoiding the need to push distances to become too tiny. Tiny distances not only make CCD less robust and make the optimization less efficient in our numerical simulation, but also are not physically reasonable in science. Generally speaking, the space between the nucleus of two bonded atoms is around 10^{-10} meters. There is no way for two macroscopic touching objects to get to that close. On the other hand, recall that the energy gradient of our optimization is

$$g(x, \kappa, \hat{d}) = \nabla E(x) + \kappa \sum_k \frac{\partial b}{\partial d_k} \nabla d_k(x, \hat{d}), \tag{A.6}$$

where at subproblem convergence, the IP gradient $\nabla E(x)$ balances with the barrier gradient $\kappa \sum_k \frac{\partial b}{\partial d_k} \nabla d_k(x, \hat{d})$. In addition, the barrier stiffness κ also influences the condition of the energy Hessian. Thus we adapt barrier stiffness strategy based on balancing the two gradients, iteratively increasing κ when needed, and applying lower and upper bounds obtained from conditioning analysis on the Hessian. Here d_k can then avoid being too small or too close to $\hat{d}$ to provide improved convergence regardless of $\hat{d}$, material, h, and our other input settings change.

The idea of balancing gradients can be traced back in optimization literature [Nocedal and Wright 2006] for estimating appropriate initial stiffnesses of barriers. In our case, we solve $\text{argmin}_\kappa \|g(x^j, \kappa, \hat{d})\|^2$ which gives us $-g_c \cdot \nabla E(x)/\|g_c\|^2$ (where $g_c = \sum_k \frac{\partial b}{\partial d_k} \nabla d_k(x, \hat{d})$) at start of each time step to obtain an estimate of κ that seeks to balance the two gradients at x^j. However, we observe that the effectiveness of this balancing term is highly dependent on the x^j applied. It can be quote far from the configuration at solution and so potentially can leads to poorly scaled or even negative values for κ. Thus, we extend our analysis from the balancing gradients to additionally include conditioning of the Hessian. This, in turn, obtains an effective estimate to provide a lower bound of κ in support of the gradient balancing strategy.

Our analysis seeks to keep the scaling of the diagonal entries of the Hessian, at small distances, close to the mass. Specifically, a scaling characterization of the Hessian diagonal in $\nabla^2 b$ at $d = 10^{-8} l$ is easily estimated by taking its first term $\nabla d^\top \frac{\partial^2 b}{\partial d^2} \nabla d$ in point-point formula as

$$c_{\nabla^2 b} = \left(\left\| \nabla d^\top \right\|^2 \frac{\partial^2 b}{\partial d^2} \right) \Bigg|_{d=10^{-8} l} = 4 \times 10^{-16} l^2 \frac{\partial^2 b}{\partial d^2}(10^{-8} l).$$

We then set the lower bound of κ to provide at least 10^{11} times the average lumped nodal mass $\bar{m}$ on the diagonal entries when $d = 10^{-8} l$ and so enabling production of sufficient repulsion at larger distances, that is

$$\kappa_{\min} = 10^{11} \bar{m} / c_{\nabla^2 b}$$

Note that our κ lower bound will be different for different $\hat{d}$, effectively capturing the curvature

change of the barrier. With this lower bound, we now can safely use the gradient scheme with bounded κ value.

Distance can still become small if resultant stress or applied compression (e.g, BCs) in the scene are extreme. We thus add an additional, final κ adjustment that doubles κ, when needed, in between Newton iterations. After every Newton iteration, if we detect that there are contact pairs having a characteristic distance smaller than minimum $\hat{d}_\epsilon = 10^{-9}l$ both before and after this iteration, and the distance is decreasing in this Newton iteration, we double the κ value. Although we do not observe divergence of the adapted κ values we apply a fixed upper bound of $\kappa_{\max} = 100\kappa_{\min}$.

To summarize, our adaptive barrier stiffness strategy is:

1. At start of each time step, compute κ_g giving smallest gradient, and set $\kappa \leftarrow \min(\kappa_{\max}, \max(\kappa_{\min}, \kappa_g))$.

2. After each Newton iteration, if any contact pair has distance smaller than $\hat{d}_\epsilon$ both before and after this iteration, and the distance is decreasing, set $\kappa \leftarrow \min(\kappa_{\max}, 2\kappa)$.

A.7 Distance Computation Implementation

A.7.1 Point-point and point-edge constraint duplications

As we discuss in Chapter 2 many point-triangle and edge-edge distances can and will reduce to point-point or point-edge distance in computation. Thus there can be multiples of exactly the same point-point and/or point-edge stencils in our constraint set. While it is tempting to simply either ignore or remove these duplicates, neither strategy is effective. Ignoring duplication in code can lead to significant redundant computation of the same force and Hessian. On the other hand removing them introduces inconsistency into our objective energy, leading to poor convergence or even divergence over iterations. Instead, we track duplicate stencils, computing their

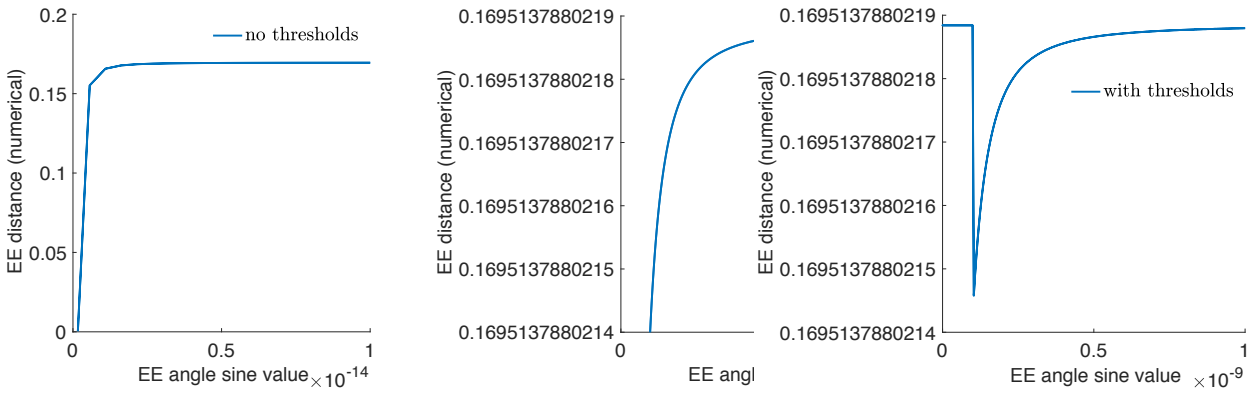


Figure A.1: Parallel-edge degeneracy handling. Left: Due to numerical rounding errors for distances of edge-edge pairs decreases to 0 when the edges get more and more parallel (see Figure 2.7 for an example); Middle: a zoom-in view show clearly that the distance really starts decreasing when their angle's sine value is at around 10^{-9}; Right: we here set a threshold to force the use of point-point or point-edge formulas to compute the distance of edge-edge pairs when their angle's sine value is below 10^{-10}, which introduces a negligible (for optimization) nonsmoothness.

energy, gradient, and Hessian evaluations only once for each distinct stencil and then multiply their entries appropriately so that all terms are correctly applied but still avoiding redundant and expensive computation.

A.7.2 Nearly parallel edge-edge distance

When computing distances between two nearly parallel edges using the edge-edge plane distance formula (Equation (2.23)), numerical rounding errors will generate huge gradient and Hessian values, and even results in wrong distances (Figure A.1) because of the ill-defined normal, making our optimization intractable with double precision floating point numbers. Therefore, we check the angle between edges, forcing each case to reduce to the most appropriate point-point or point-edge constraints if the sine value is smaller than 10^{-10}. This clearly makes the distance function C^0 continuous again at the threshold. However, the nonsmoothness is nearly negligible (Figure A.1), while our multiplying energy smoother $e_{k,l}(x)$ is also extremely small when edges are nearly parallel. In practice we find this robustly avoids numerical issues and converges well for all benchmark tests; see Section 2.7.

A.8 Tangent and Sliding Modes

After reducing the general point-triangle, and edge-edge distances to one of the closed form formulas, see Section 2.6, we can directly compute the sliding basis operators for each of the four types of contact pairs required for computing friction.

We start by defining the basis, $P_k(x) \in R^{3\times 2}$, formed by the two orthogonal 3D unit-length column vectors spanning the tangent space of the contact pair k, and a selection matrix $\Gamma_k \in R^{3\times 3n}$ which computes relative velocity $v_k = \Gamma_k v$ of each contact pair k. Then we can define the sliding basis $T_k(x) = \Gamma_k^\top P_k(x)$ that maps tangent space relative velocity or displacement to the stacked global vector. Here we then list the construction for P and Γ for each contact distance type:

Point (x_0) – Triangle ($x_1x_2x_3$).

$$P_k(x) = \left[\frac{x_2-x_1}{\|x_2-x_1\|}, n \times \frac{x_2-x_1}{\|x_2-x_1\|}\right] \tag{A.7}$$

where $n = \frac{(x_2-x_1)\times(x_3-x_1)}{\|(x_2-x_1)\times(x_3-x_1)\|}$. Each row of Γ_k is

$$[..., 1, ..., (-1+\beta_1+\beta_2), ..., -\beta_1, ..., -\beta_2, ...] \tag{A.8}$$

where β's are those defined in $\mathcal{D}^{P-T}$ (17, main paper).

Edge (x_0x_1) – Edge (x_2x_3).

$$P_k(x) = \left[\frac{x_1-x_0}{\|x_1-x_0\|}, n \times \frac{x_1-x_0}{\|x_1-x_0\|}\right] \tag{A.9}$$

where $n = \frac{(x_1-x_0)\times(x_3-x_2)}{\|(x_1-x_0)\times(x_3-x_2)\|}$. Each row of Γ_k is

$$[..., 1-\gamma_1, ..., \gamma_1, ..., \gamma_2-1, ..., -\gamma_2, ...] \tag{A.10}$$

where γ's are those defined in $\mathcal{D}^{E-E}$ (18, main paper).

Point (x_0) – Edge (x_1x_2).

$$P_k(x) = \left[\frac{x_2-x_1}{\|x_2-x_1\|}, \frac{(x_2-x_1)\times(x_0-x_1)}{\|(x_2-x_1)\times(x_0-x_1)\|}\right] \tag{A.11}$$

Each row of Γ_k is

$$[..., 1, ..., \eta - 1, ..., -\eta, ...] \tag{A.12}$$

where $(1-\eta)x_1 + \eta x_2$ is the closest point to x_0 on edge x_1x_2.

Point x_0 – Point x_1.

$$P_k(x) = \left[t, \frac{x_1-x_0}{\|x_1-x_0\|} \times t\right] \tag{A.13}$$

where $t = \frac{e\times(x_1-x_0)}{\|e\times(x_1-x_0)\|}$ and e is $(1,0,0)$ if $(x_1 - x_0)$ is not colinear with $(1,0,0)$, or e is $(0,1,0)$. Each row of Γ_k is $[..., 1, ..., -1, ...]$.

A.9 Friction Implementation

Since we lag the sliding basis in friction computations to $T^n = T(x^n)$ and normal forces to λ^n in (see Section 2.5) from either the last time step or the last friction update iteration n, all other terms are integrable. The lagged friction is then

$$F_k(x, \lambda^n, T^n, \mu) = -\mu\lambda^n T_k^n f_1(\|u_k\|)\frac{u_k}{\|u_k\|} \tag{A.14}$$

and gives us a simple and compact friction potential

$$D_k(x) = \mu\lambda_k^n f_0(\|u_k\|). \tag{A.15}$$

Here f_0 is given by $f_0' = f_1$ and $f_0(\epsilon_v h) = \epsilon_v h$ so that $F_k(x) = -\nabla D_k(x)$. In turn this likewise provides a simple-to-compute Hessian contribution

$$\nabla^2 D_k(x) = \mu \lambda_k^n T_k^n \left(\frac{f_1'(\|u_k\|)\|u_k\| - f_1(\|u_k\|)}{\|u_k\|^3} u_k u_k^\top + \frac{f_1(\|u_k\|)}{\|u_k\|} I_2 \right) T_k^{n\top}. \tag{A.16}$$

where $I_2 = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Projecting this Hessian to PSD then simply requires projecting the 2×2 matrix

$$\frac{f_1'(\|u_k\|)\|u_k\| - f_1(\|u_k\|)}{\|u_k\|^3} u_k u_k^\top + \frac{f_1(\|u_k\|)}{\|u_k\|} I_2 \tag{A.17}$$

to SPD as T_k^n is symmetrically multiplied on both of its two sides.

The above model is general so that f_0 and f_1' are both easy to define for a range of f_1 choices:

1. C^0 F_f: $f_0(x) = \frac{x^2}{2\epsilon_v h} + \frac{\epsilon_v h}{2}$, $f_1(x) = \frac{x}{\epsilon_v h}$, and $f_1'(x) = \frac{1}{\epsilon_v h}$;
2. C^1 F_f: $f_0(x) = -\frac{x^3}{3\epsilon_v^2 h^2} + \frac{x^2}{\epsilon_v h} + \frac{\epsilon_v h}{3}$, $f_1(x) = -\frac{x^2}{\epsilon_v^2 h^2} + \frac{2x}{\epsilon_v h}$, and $f_1'(x) = -\frac{2x}{\epsilon_v^2 h^2} + \frac{2}{\epsilon_v h}$;
3. C^2 F_f: $f_0(x) = \frac{x^4}{4\epsilon_v^3 h^3} - \frac{x^3}{\epsilon_v^2 h^2} + \frac{3x^2}{2\epsilon_v h} + \frac{\epsilon_v h}{4}$, $f_1(x) = \frac{x^3}{\epsilon_v^3 h^3} - \frac{3x^2}{\epsilon_v^2 h^2} + \frac{3x}{\epsilon_v h}$, and $f_1'(x) = \frac{3x^2}{\epsilon_v^3 h^3} - \frac{6x}{\epsilon_v^2 h^2} + \frac{3}{\epsilon_v h}$.

Importantly this also emphasizes that there are never any divisions by $\|u_k\|$ in all of our energy, gradient, and Hessian computations for friction as they are always cancelled out. This ensures that so that the implemented computation can be robust and accurate.

Our friction model applies our C^1 – (2) in the above.. This design choice again provides a continuous Hessian for better convergence in our Newton-type method. Here we also provide a comparison of behavior of the C^1 model w.r.t. to the different orders of smoothed friction model on our arch and ball roller example (Table A.3).

We observe that C^1 friction model provides a "sweet-spot": improvement over C^0 due to the C^1 model's continuous hessian, while it also improves over the C^2 model with less additional nonlinearity.

Table A.3: Ablation study on smoothed static friction.

Examples	C0 # iters, t(s) / time step	C1 (IPC) # iters, t(s) / time step	C2 # iters, t(s) / time step
Sphere (1K) roller	1.37, 0.01	1.24, 0.01	1.26, 0.02
1 m-high arch (static)	53.60, 12.21	45.22, 9.79	52.83, 11.42

A.10 Squared Terms

In our implementation, we apply squared distances in our evaluations to avoid numerical errors and inefficiencies that can be introduced by taking squared roots – especially in gradient and Hessian computations. Concretely, our barrier terms are applied as $b(d^2, \hat{d}^2)$ throughout our implementation. This manipulation leaves our problem formulation with unsigned distances unchanged as have $d > 0 \Leftrightarrow d^2 > 0$. However, we must be careful with units in order to preserve appropriate scaling of dimensions – especially in terms of frictio. Fortunately, we can sort this out directly via the chain rule.

To ensure that units of normal force remain correct we now observe that directly plugging in d^2 and $\hat{d}^2$ into Equation (2.9) no longer computes the contact force magnitudes λ_k defined Chapter 2. Here we now have $-\frac{\kappa}{h^2}\frac{\partial b}{\partial (d_k^2)}(d_k^2, \hat{d}_k^2)$. Applying the squared formulation we rewrite the stationarity of the barrier as

$$M(\frac{x - \hat{x}}{h^2}) = -\nabla\Psi(x) - \frac{\kappa}{h^2}\sum_{k\in C}\frac{\partial b}{\partial (d_k^2)}\frac{\partial (d_k^2)}{\partial (d_k)}\nabla d_k(x) \tag{A.18}$$

which is $M(\frac{x-\hat{x}}{h^2}) = -\nabla\Psi(x) - \frac{\kappa}{h^2}\sum_{k\in C}\frac{\partial b}{\partial (d_k^2)}2d_k\nabla d_k(x)$. In turn this allows us to extract the correct contact force magnitudes (when using squared distances) as $\lambda_k = -\frac{\kappa}{h^2}\frac{\partial b}{\partial (d_k^2)}2d_k$.

B | Incremental Potential Contact: Comparison Details

B.1 COMSOL

Scene Setup. We set up a simple scene to illustrate issues we encounter for contact modeling in COMSOL. See Figure B.1 top and middle, for our initial positions where we have, from left to right, a fixed rectangle ($0.2\,\text{m} \times 0.7\,\text{m}$), two free squares ($0.5\,\text{m} \times 0.5\,\text{m}$), and a moving rectangle ($0.2\,\text{m} \times 0.7\,\text{m}$) with symmetric boundary conditions in the y-direction. We impose displacement boundary conditions on the right-most rectangle of $-0.6\,\text{m}$, use densities of $1000\,\text{kg/m}^3$ and a linear material model with $E = 1000\,\text{Pa}$ and $\nu = 0.4$. We experimented with different discretizations and applications of the boundary conditions. With a quad model we apply the boundary conditions incrementally ($-0.6s$) with s going from zero to one, and steps of size 0.01, while with a triangle mesh we applied steps of 0.005. The first four figures in Figure B.1 show respectively the initial configurations and the solutions for these two experiments, the last figure shows increased failure if we simply apply the boundary conditions directly without incremental loading.

Discussion. In COMSOL, we initially attempted to simulate dynamic contacts. However, the contact formulation is documented as being strictly valid only for stationary problems [COMSOL Inc. 2019, Time-Dependent Contact Analysis, Page 199]. We then attempted to solve a simpler

Figure B.1: COMSOL comparison. Experiment setup (first and third figure) and results (second and fourth figure) of our COMSOL experiment. Disabling the incremental application of the boundary conditions leads to massive penetrations (last figure).

stationary problem. We take two cubes compressed by two rectangle, again, as above, one fixed, and one displaced. After an extensive correspondence with COMSOL technical support (see Additional Material), we were told to apply a few necessary "tricks": (i) since the problem is symmetric, it is better to cut it in half and impose symmetry boundary conditions; (ii) the mesh on the destination object should be twice as fine as the mesh on the source object along all contact interface boundaries; (iii) the scene should be designed so that there are no initial gaps between

the objects that are in contact (no changing contact sets); (iv) the system should be solved applying an auxiliary sweep with small parametric steps, and so apply the displacement incrementally. Applying all of these additional customized tricks we were able to solve the simulation with both COMSOL's penalty and augmented Lagrangian options. Our experiments, agreeing with COMSOL technical support, show that the number of parametric steps that must be applied depend on the mesh size, with different values (larger or smaller) leading to reasonable solutions or not that must be discovered per example. For both simulations penetration is expected, more, we observe for the penalty method, as intersection is how contact pressure develops in the solver. Additionally, we learned that it is never a good idea to have corners being a part of a contact boundaries, as the corners will be numerically singular and cause penetration between the contact boundaries. Thus, according to COMSOL guidelines[1] corners should be filleted.

B.2 Ansys

Scene Setup. We again set up two simple problems to test contact modeling in Ansys. For the first example we generate a "C"-shaped mesh (formed from five $10\,\text{mm} \times 10\,\text{mm} \times 10\,\text{mm}$ cubes) supported on the bottom and a dropped cube ($10\,\text{mm} \times 10\,\text{mm} \times 10\,\text{mm}$) subject to gravity $981\,\text{mm/s}^2$. For the second example we placed a large cube ($14\,\text{mm} \times 14\,\text{mm} \times 14\,\text{mm}$) with fixed support on the bottom and placed a smaller cube on top ($10\,\text{mm} \times 10\,\text{mm} \times 10\,\text{mm}$ cubes), beveled at 1 mm; both were subject to gravity of $100\,\text{mm/s}^2$. Both setups apply densities of $10^{-6}\,\text{kg/mm}^3$ and neo-Hookean material with $E = 0.01\,\text{MPa}$ and $\nu = 0.4$. Figure B.2 shows the two set ups and the resulting simulation failures. Note for the second experiment the spacing between the two objects is 1mm, from the plot on the bottom we see that the top, falling cube has a maximum displacement of more than 1.4mm for an implicit time step, and 1.02mm for an explicit time step.

[1] https://www.comsol.com/support/knowledgebase/1102

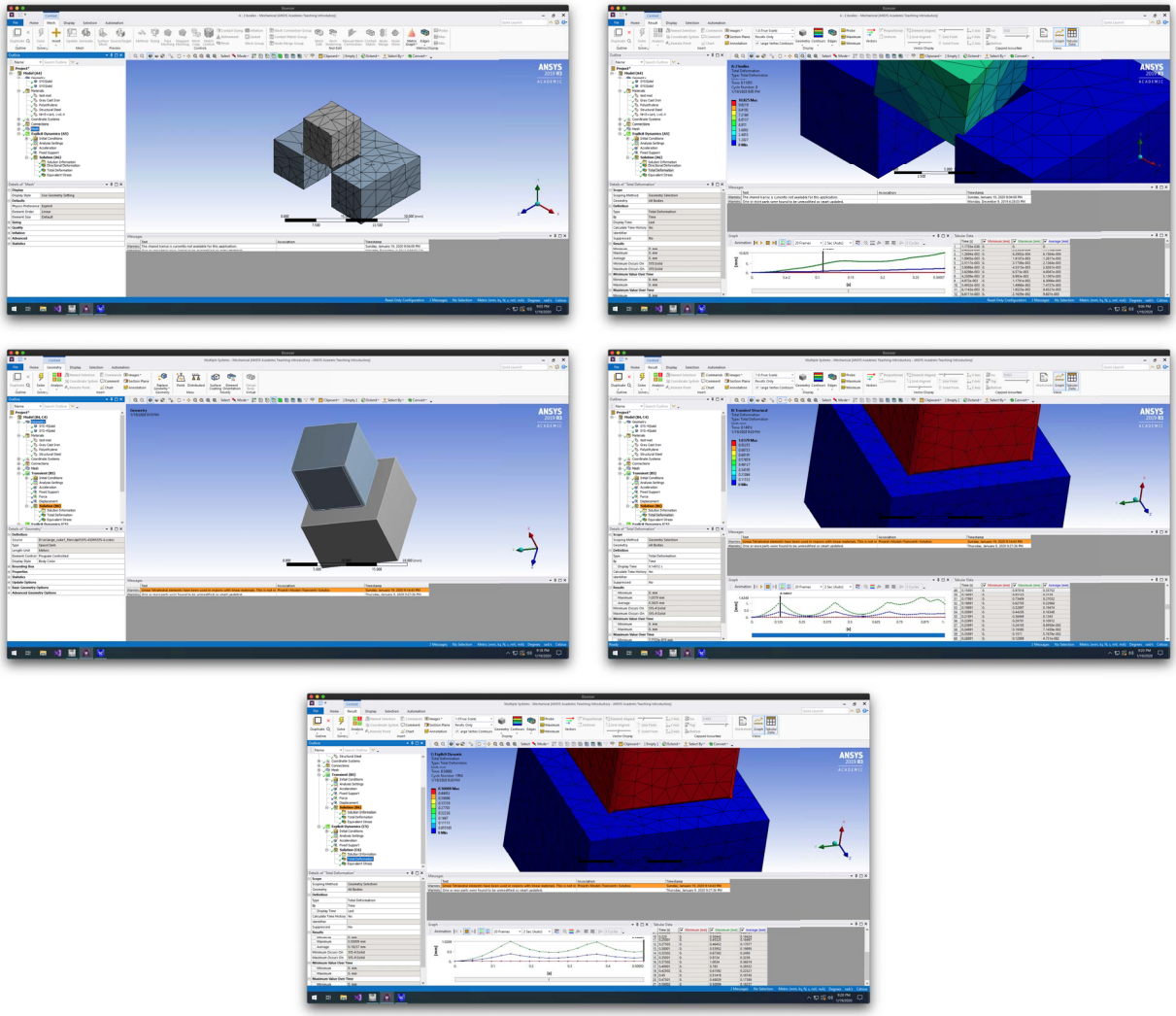


Figure B.2: Ansys comparison. First and second Ansys experiment (first and third image). All experiments all have inter-penetration. The second-to-last image applies implicit time-stepping, and the last one explicit time-stepping.

Discussion. Ansys provides explicit (through its explicit dynamics modules) and implicit (through its transient structural module) numerical time-integration options. The two models are fundamentally different in the way they handle collisions, but they both allow for small inter-penetrations. The explicit module is mostly automatic both in how it detects possible contact pairs and then resolves them. The results of the simulations are reasonable, although they do contain some small penetrations. The implicit module, is more complex, requires identifying contact pairs

and much hand- tweaking of many exposed parameters in order to gain a successful simulation. This is true even for a simple scenario where two cubes are colliding. Here the default parameters do not produce a reasonable solution. For such simple scenarios Ansys technical support advised us to apply explicit dynamics and did not provide us with a reasonable set of parameters to fix the implicit simulation.

B.3 Utopia [Krause and Zulian 2016]

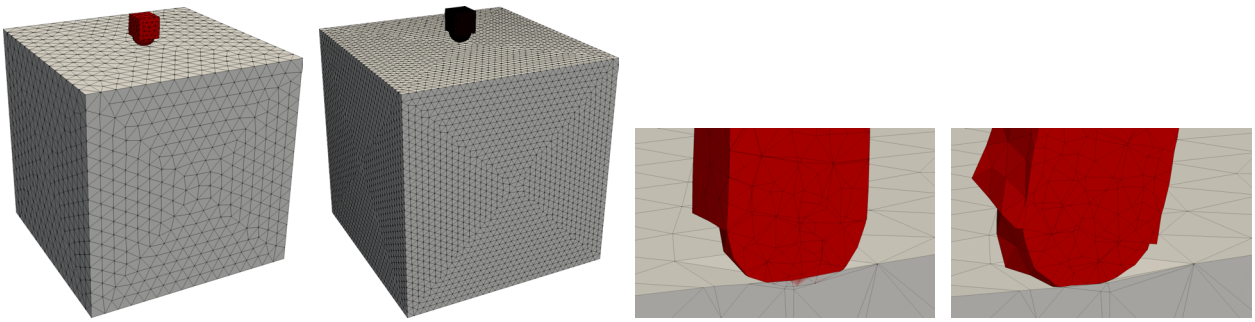


Figure B.3: Utopia comparison. Utopia setup coarse and fine mesh (first two image). Closeup on the two failures (third image has self-penetration because of large time step and fourth image has oscillations that prevent the solver the continue) for the coarse mesh.

Scene Setup. To investigate Utopia we set up a simulation with a large $10\,\text{mm} \times 10\,\text{mm} \times 10\,\text{mm}$ block with fixed displacement on the bottom and a small cube $1\,\text{mm} \times 1\,\text{mm} \times 1\,\text{mm}$ with a half-sphere on the bottom and prescribed $-1.5\,\text{N}$ force on the top flat face. See Figure B.3. Both objects have the same neo-Hookean material parameters: density $1\,\text{kg/m}^3$, $\lambda = 10\,\text{Pa}$, $\mu = 10\,\text{Pa}$. in Figure B.3 we show the initial set up and the resulting penetration and oscillations for the coarse mesh version. Note that oscillations disappear for the fine mesh and penetrations decrease as we go to smaller time steps. We tested time steps of 0.5, 0.05, 0.01, and 0.001 s.

Discussion. Utopia applies a parallel mortar algorithm for FE contact modeling. Like several other engineering FEM codes, Utopia requires a priori manual marking of all possible paired mesh

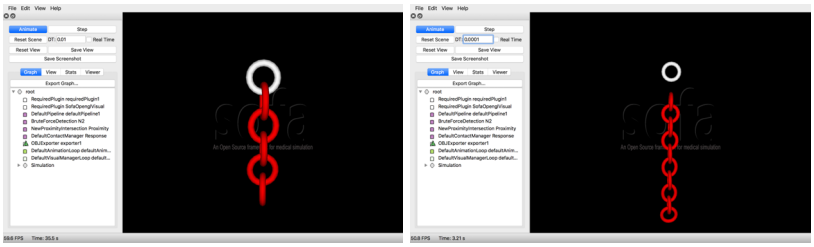


Figure B.4: SOFA comparison (varying chain-length). Left: SOFA is able to simulate a five link even with large timesteps. Right: As we add more links, decreasing time steps down to 10^{-4}s, we are unable to find a time step that will not break the chain.

Table B.1: SOFA FEM parameter values.

Elasticity Model	linear
Young's Modulus	1000
Poisson's Ratio	0.3
Mass	5.0

boundary regions that can be treated for contact resolution. Such marking could be automated, see e.g, Yang and Laursen [2008]). Note that even with automated pairing the mortar approach does not yet extend for more general collisions in regions that would involve more than two distinct surface regions. We tested the Utopia implementation [Zulian et al. 2016] with a simple didactic experiment in which we drop an elastic sphere under gravity upon an elastic cube with its bottom DOF fixed. At larger time steps, we observe large scale volume intersections. These contact errors then reduce but do not disappear as time step decreases. These issues are not surprising as mortar methods enforce contact constraints in a weak sense and so allow intersections.

B.4 SOFA

We modify the the chain example provided with SOFA by replacing the ring mesh with a low-resolution torus (~500 tetrahedra). Table B.1 shows the material settings used. Changing any of these parameters can result in different levels of failure (see Figure B.5 for an example of changing material stiffness).

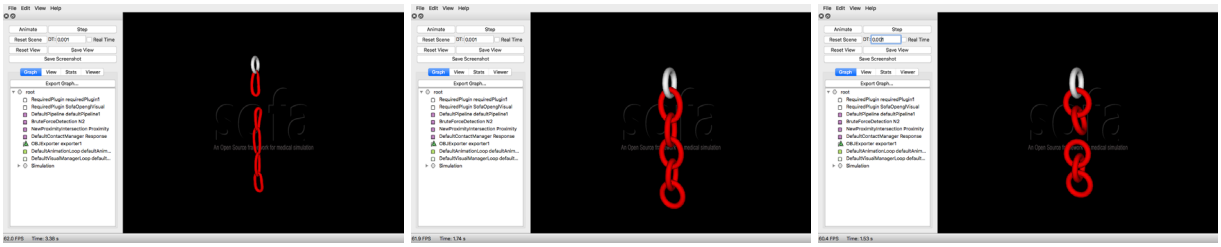


Figure B.5: SOFA comparison (varying stiffness). We test SOFA on a five link chain with three varying stiffnesses, all with a fixed timestep of 10^{-3} s. Left: Young's Modulus of 100 fails, Middle: Young's Modulus of 1000 works, Right: Young's Modulus of 10000 fails.

Table B.2: Houdini FEM parameter values.

Damping Ratio	0.5
Shape Stiffness	387
Volume Stiffness	384
Repulsion	1e11
Friction	0.1
Substeps	14
Collision Passes	16

B.5 Houdini

We test Houdini with the simple chain example. Each link is a torus with outer radius 1.25 and inner radius 0.28. The links are duplicated with a transform downwards of 1.62 and a rotation of 90°. One ring at the top is added as a static surface collider. Parameter values that were changed from default Houdini parameters for the FEM solver are shown in Table B.2 and values for the vellum solver are shown in Table B.3. The scene was set to 24 FPS.

Table B.3: Houdini Vellum parameter values that were changed from default.

Cloth Node		Struts Node		Solver Node	
Mass	Calculate Varying	Direction Jitter	2	Substeps	40
Thickness	Calculate Uniform	Constraints Per Point	80		
Stretch Stiffness	1000000	Stretch Stiffness	1e-9		
Bend Stiffness	1e-4	Compression Stiffness	10000		

B.6 SQP Benchmark

Figure B.8 summarizes the results of our benchmark comparison of different SQP-type contact handling methods. For each scene we vary the time step size, constraint offset, constraint type, and solve the problem as both a fully nonlinear problem at each time step and a quadratic approximation (Linearized elasticity) of the energy with nonlinear constraints. See our main paper for details.

Additionally, we also initially tested two active-set update strategies: the first being to clear the active set and rebuild it upon every iteration and the second, applied successfully in recent methods [Otaduy et al. 2009; Verschoor and Jalba 2019] incrementally adds newly detected collisions to the active set. We found that the latter performed significantly better across our benchmark so we restrict our results here to the latter.

In the table we document results of each simulation as either: 1) intersecting: the simulation fails with some amount of intersection (see Figure B.6 for examples of minor and major intersections); 2) blow-up: the simulation blows up due to rapid growth in energy and/or displacement during the optimization (see Figure B.7 for examples); 3) incomplete: the simulation is unable to complete in the given time limit of four hours, or 4) successful: the simulation successfully completes with none of the aforementioned failures. Note that IPC is the only method to succeed across all simulations.

B.6.1 Intersections

SQP-type optimizations can result in intersection for a variety of reasons, but they are commonly generated in cases where the time step is too large (leading to poor linearization of constraints) or the constraint offset is too small (allowing slight solver inaccuracies to generate drift and intersections). While possible in lucky cases, recovering from intersections is generally rare on it sown and of course challenging to fix algorithmically.

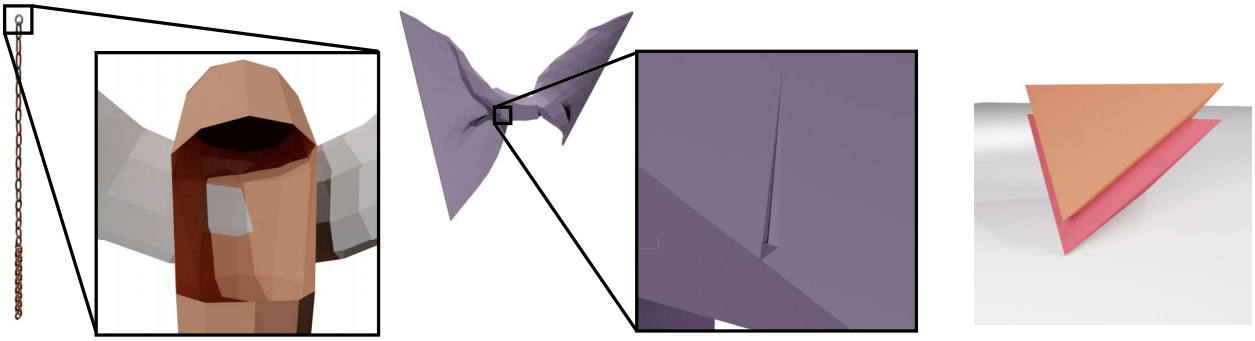


Figure B.6: SQP intersections. Severity of intersections affect usability of simulation results. Left: Examining a chain drop the results look correct until we look inside the top ring. It is clear the fixed ring has started to intersect the top FEM ring. These small intersections can build in to larger constraint drift and eventually pass-through of the rings. Middle: At first glance the results of this mat twist appear fine, but when we look closely there are several small intersections hidden in the folds of the mat (22 in total). Right: An extreme case of intersections where the two tetrahedron meshes have almost completely overlapped.

Not all intersections are alike, however. Minor intersections while not physical are commonly ignored as they do not affect visual quality. Figure B.6 shows examples of both minor and egregious intersections.

B.6.2 Optimization blow-up

Optimization "blow-up" occurs when the objective energy or search direction of the optimization increases at alarming rates due to numerical difficulties. Studying the results of our sweep, we find that blow-ups more often occur with large time steps where a large number of constraints may be activated at once. This can create a difficult, if not numerically infeasible, optimization problems and likewise instability due to lack of line search.

We also note the large amount of blow-ups when applying the single quadratic approximation per time step. While faster to optimize, this method is a poor approximation of the nonlinear elasticity energy and leads to increased energy values. Again, the problem is worsened in all cases by the lack of line-search to find a step length that sufficiently decreases the energy.

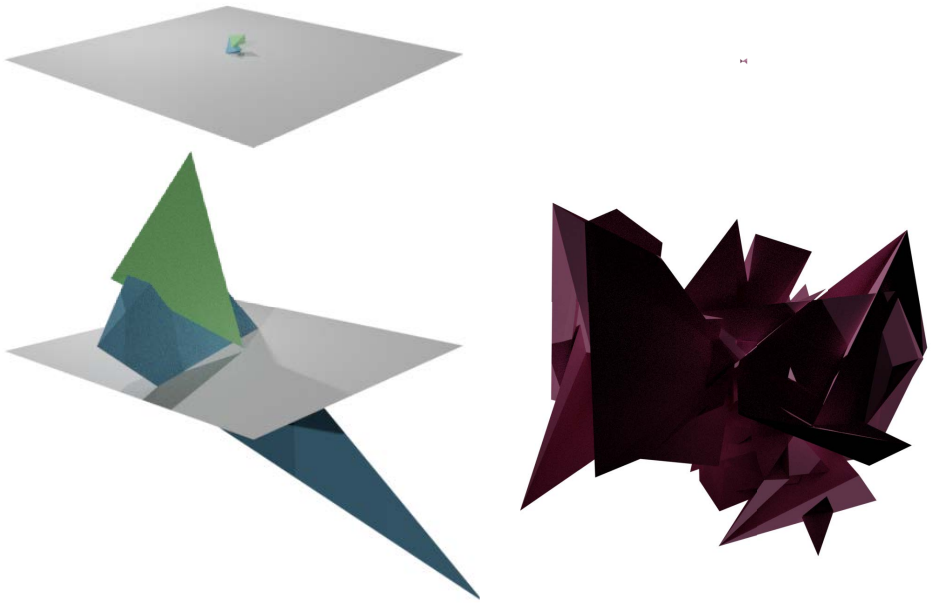


Figure B.7: SQP numerical explosions. Top: Two different examples (left and right) with their time steps right before the SQP optimization starts to fail with increasingly large displacements. Bottom: An intermediate state for each example during the SQP optimization "blow-up".

B.6.3 Poor convergence and subsequent timeout

In our benchmark, SQP-type methods struggle to handle moderately large scenes. We attribute this poor performance to 1) large numbers of constraints when the constraint offsets are enabled – this, in turn, can lead to infeasibility in the QP solve and so subsequent solve fails; and 2) oscillating optimization iterations largely resulting from lack of line-search. When the underlying QP solver fails, we choose to clear and build a new active set. While this helps in some cases, it often results in more QP fails and a stagnant optimization. Convergence could be improved with a line-search along the constrained search directions, but suitable merit-functions for line search here remain an open and ongoing area of research.

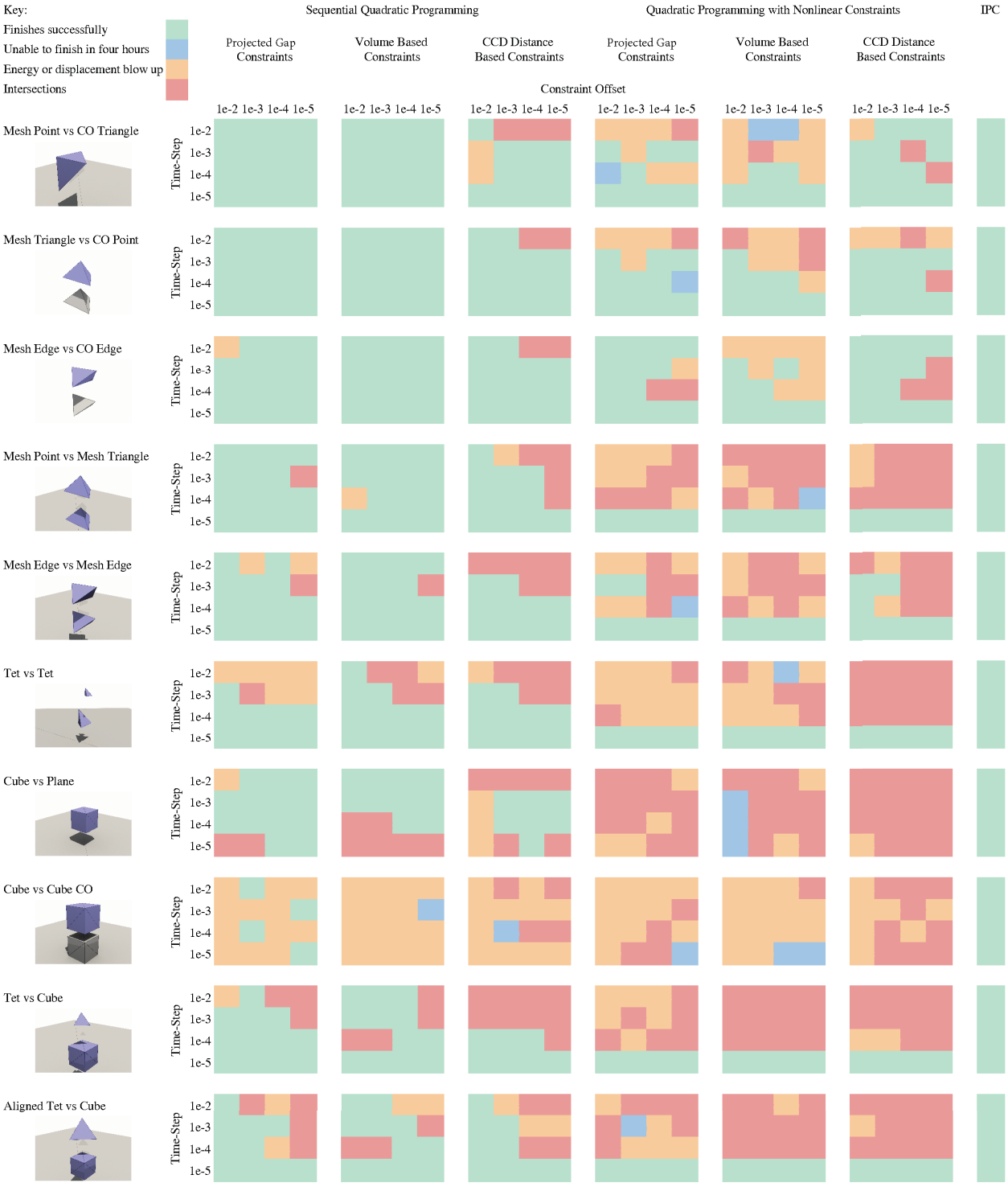


Figure B.8: SQP benchmark (part 1 of 3).

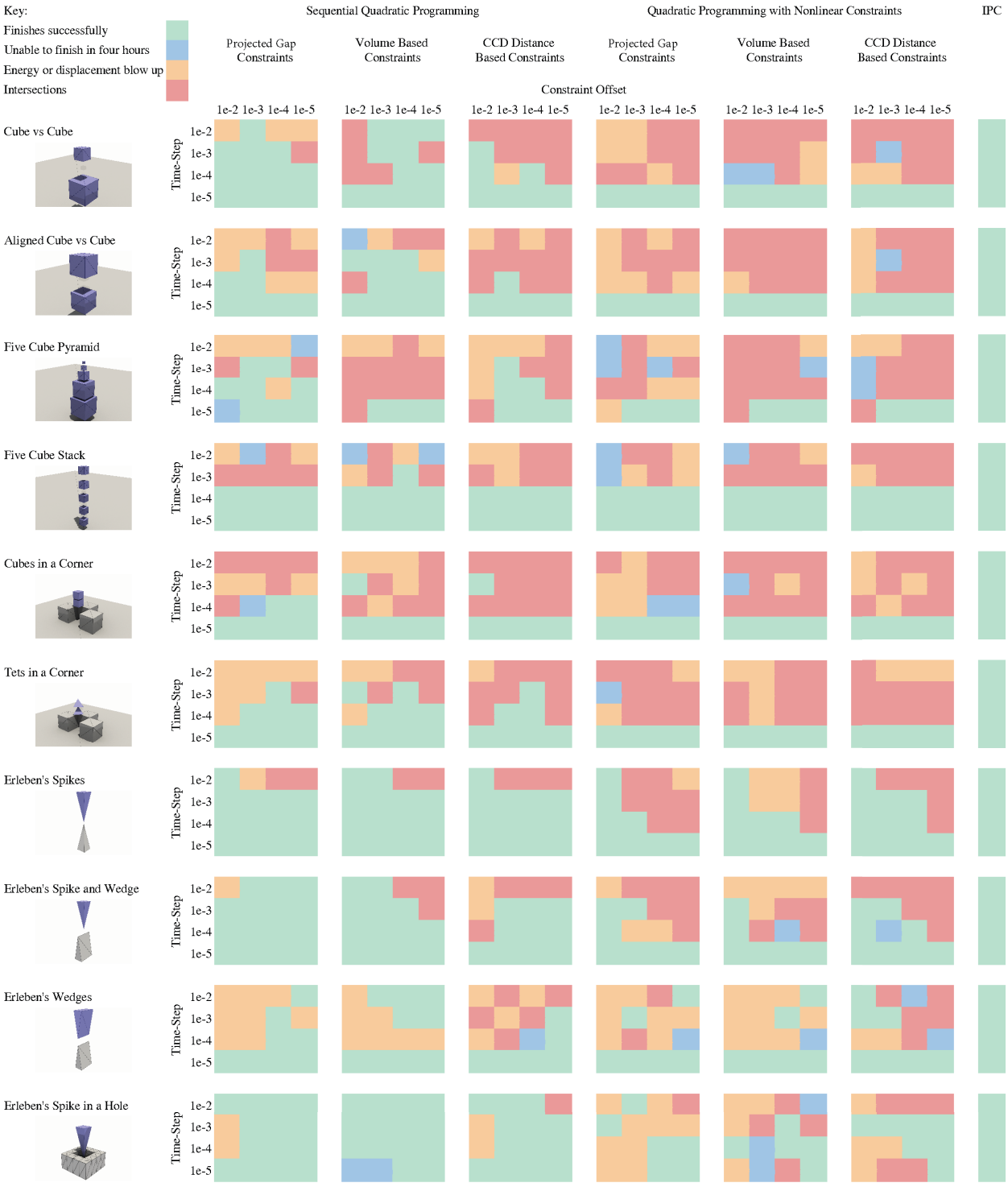


Figure B.8: SQP benchmark (part 2 of 3).

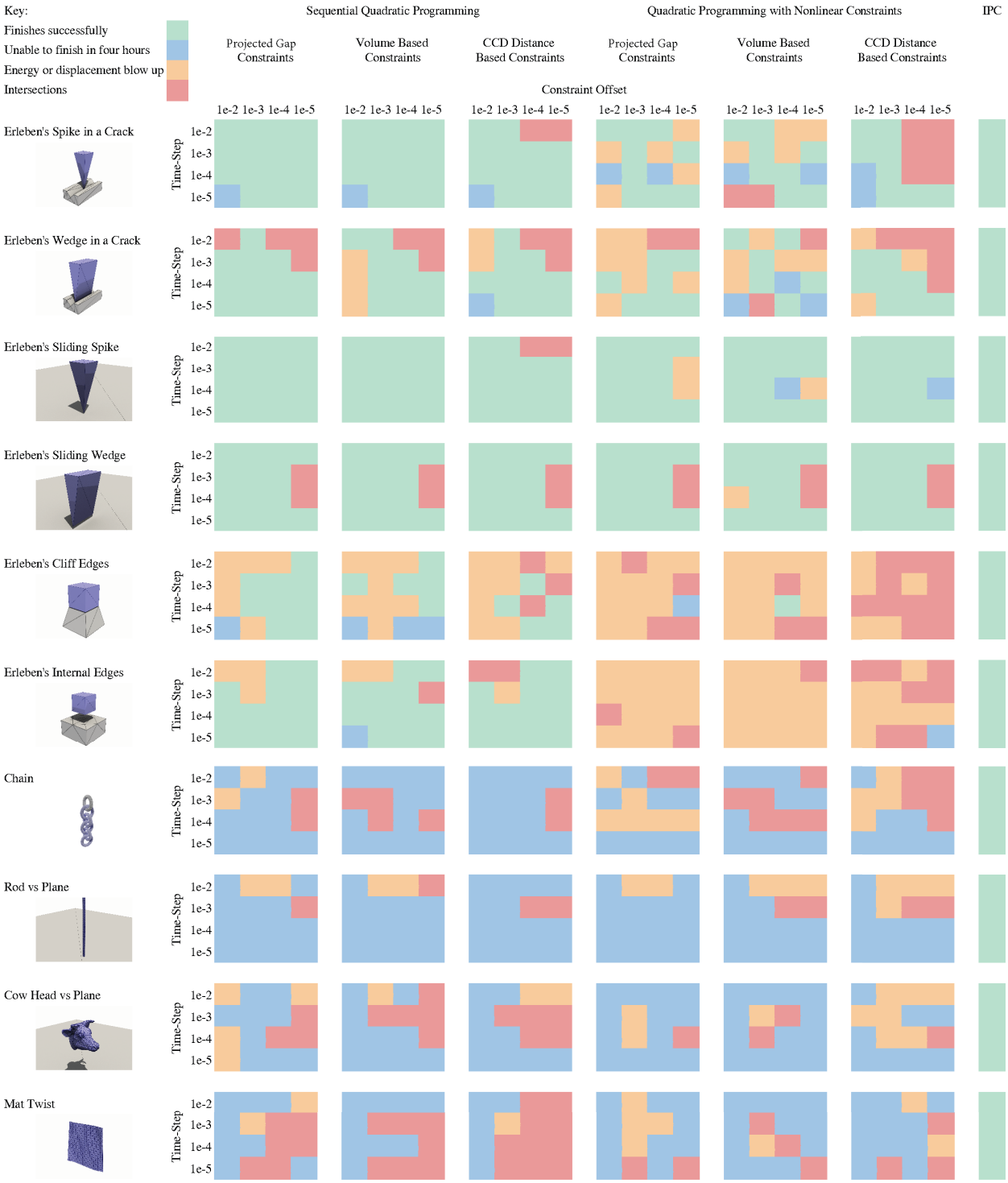


Figure B.8: SQP benchmark (part 3 of 3).

C | Incremental Potential Contact: Statistics

Table C.1: CPU names

Name	CPU	Memory
C1	4-core 2.9 GHz Intel Core i7	16 GB
C2	4-core 3.0 GHz Intel Xeon E-2690v2	32 GB
C3	4-core 3.6 GHz Intel Core i7	32 GB
C4	8-core 3.0 GHz Intel Xeon	32 GB
C5	8-core 3.6 GHz Intel Core i9	64 GB

Table C.2: IPC scene statistics: Tet & cube unit tests

Example	# nodes, # tets, # faces	h (s)	ρ (kg/m^3) E (Pa), ν	$\hat{d}$ (m)	Newton tol (m/s)	# contact avg (max)	CPU	memory (MB)	per timestep t (s), # iters
C-box	285, 812, 510	0.04	1e3, 1e5, 0.4	$10^{-5}\ell$	$10^{-2}\ell$	41 (381)	C1	80	0.2, 12.8
Tet-tet	8, 2, 8	0.025	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (6)	C1	70	1.4e-3, 2.3
Cube stack	40, 30, 60	0.025	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	8 (69)	C1	76	6e-3, 3.25
Tet corner	28, 19, 40	0.025	1e3, 1e5, 0.4	$10^{-6}\ell$	$10^{-2}\ell$	5 (21)	C1	70	2e-3, 3.9
Spikes[1]	5, 2, 6	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (4)	C2	11	9.4e-4, 2.05
Spike and Wedge[1]	5, 2, 6	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (3)	C2	11	1.0e-3, 2.01
Wedges[1]	6, 3, 8	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (19)	C2	11	8.1e-4, 2.06
Spike in a Hole[1]	5, 2, 6	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (5)	C2	11	1.0e-3, 2.10
Spike in a Crack[1]	5, 2, 6	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (2)	C2	11	7.7e-4, 2.01
Wedge in a Crack[1]	6, 3, 8	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (3)	C2	11	3.0e-3, 2.01
Sliding Spike[1]	5, 2, 6	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (1)	C2	11	9.5e-4, 2.01
Sliding Wedge[1]	6, 3, 8	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	0 (1)	C2	11	9.6e-4, 2.00
Cliff Edges[1]	8, 6, 12	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	7 (28)	C2	11	1.7e-3, 1.97
Internal Edges[1]	8, 6, 2	0.01	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	18 (28)	C2	11	1.3e-3, 2.03

[1]These scenes where created based on the benchmark proposed by Erleben [2018].

Table C.3: IPC scene statistics: Co-dimensional obstacles unit tests

Example	# nodes, # tets, # faces	h (s)	ρ (kg/m^3) E (Pa), ν	$\hat{d}$ (m)	Newton tol (m/s)	# contact avg (max)	CPU	memory (MB)	per timestep t (s), # iters
Ball on points	7K, 28K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	126 (182)	C1	229	2.8, 6.6
Ball on segments	7K, 28K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	439 (706)	C1	196	4.4, 11.3
Ball on squares	7K, 28K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	908 (1K)	C1	239	5.5, 13.4
Octocat on points	7K, 21K, 9K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	193 (846)	C1	272	5.0, 11.1
Octocat on segments	7K, 21K, 9K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	354 (1K)	C1	231	4.5, 10.4
Octocat on squares	7K, 21K, 9K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	808 (1.2K)	C1	241	5.0, 11.4
Mat on points	3K, 9K, 6K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	112 (250)	C1	159	0.64, 2.9
Mat on segments	3K, 9K, 6K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	326 (419)	C1	163	0.5, 2.5
Mat on squares	3K, 9K, 6K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	468 (780)	C1	141	1.5, 5.0

Table C.4: IPC scene statistics: Stress tests and general examples

Example	# nodes, # tets, # faces	h (s)	ρ (kg/m^3) E (Pa), ν	$\hat{d}$ (m)	Newton tol (m/s)	# contact avg (max)	CPU	memory (MB)	per timestep t (s), # iters
Mat on knives	3.2K, 9.1K, 6.4K	0.04	1e3, 2e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	291 (472)	C1	147	1.4, 5.5
5 chains	1K, 2.5K, 2K	0.04	1e3, 1e5, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	185 (260)	C1	97	0.2, 4.3
35 chains	7K, 17K, 14K	0.04	1e3, 1e7, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	1.7K (2.5K)	C1	191	1.1, 2.6
100 chains	20K, 49K, 40K	0.04	500, 1e7, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	40K (53K)	C1	450	4.0, 2.4
Ball on mat	5K, 16K, 9K	0.01	2e3, 1e8, 0.4 1e3, 1e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	327 (687)	C1	173	2.0 6.9
Cube on small cube	16, 12, 24	0.025	1e4, 1e8, 0.4 1e3, 1e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	14 (18)	C1	57	2e-3, 4.2
Dolphin funnel	8K, 36K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	7K (31K)	C1	357	27.9, 39.7
Dolphin funnel (FCR)	8K, 36K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	7K (35K)	C1	283	81.3, 129.7
Pin-cushion compress	9K, 28K, 10K	0.04	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	317 (496)	C1	233	3.7, 9.5
Golf ball (NM)	29K, 118K, 38K	2E-5	1150, 1e7, 0.45	$10^{-3}\ell$	$10^{-2}\ell$	1K (4K)	C1	861	12.1, 9.3
Bunny matrix	57K, 196K, 92K	0.04	1e3, 5e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	5.7K (7.0K)	C3	1338	123.0, 77.1
Tunneling test (10 m/s)	2.6K, 6.9K, 4K	0.02	1e3, 1e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	1 (62)	C1	201	4.7, 12.1
Tunneling test (100 m/s)	2.6K, 6.9K, 4K	0.02	1e3, 1e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	4 (297)	C1	235	6.2, 14.4
Tunneling test (1000 m/s)	2.6K, 6.9K, 4K	0.02	1e6, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	10 (876)	C1	306	8.2, 19.5
Mat twist (100s)	45K, 133K, 90K	0.04	1e3, 2e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	264K (439K)	C4	4546	776.2, 34.5
Rods twist (100s)	53K, 202K, 80K	0.025	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	243K (498K)	C4	2638	141.5 ,14.1
Trash compactor (octocat)	6K, 21K, 9K	0.01	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	793 (34K)	C4	447	12.8, 16.9
Trash compactor (ball, mat, & bunny)	15K, 56K, 22K	0.01	1e3, 1e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	6K (132K)	C4	638	61.9, 29.4
Squeeze out	45K, 181K, 60K	0.01	1e3, 5e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	37K (277K)	C4	1700	252, 42.5
Heavy ball hit	47K, 187K, 62K	0.01	1e3, 5e4, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	13K (59K)	C4	1498	263.9, 44.2

Table C.5: IPC scene statistics: Scalability Tests

Example	# nodes, # tets, # faces	h (s)	ρ (kg/m^3) E (Pa), ν	$\hat{d}$ (m)	Newton tol (m/s)	# contact avg (max)	CPU	memory (MB)	per timestep t (s), # iters
Armadillo twist 13K	13K, 55K, 17K	0.025	1e3, 5e3, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	832 (2.7K)	C3	453	4.3, 10.6
Armadillo twist 28K	28K, 121K, 35K	0.025	1e3, 5e3, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	2.5K (72K)	C3	890	11.8, 10.2
Armadillo twist 54K	54K, 227K, 69K	0.025	1e3, 5e3, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	5.5K (17K)	C3	1632	23.7, 10.0
Armadillo twist 122K	122K, 548K, 140K	0.025	1e3, 5e3, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	22K (54K)	C3	3744	64.0, 9.8
Armadillo twist 219K	219K, 928K, 277K	0.025	1e3, 5e3, 0.4	$10^{-3}\ell$	$10^{-2}\ell$	60.6K (133K)	C3	6547	126.2, 10.0
Mat twist (10s) 40x40	3K, 9K, 6K	0.04	1e3, 2e4 0.4	$10^{-3}\ell$	$10^{-2}\ell$	2K (4.6K)	C3	196	1.3, 7.6
Mat twist (10s) 100x100	20K, 59K, 40K	0.04	1e3, 2e4 0.4	$10^{-3}\ell$	$10^{-2}\ell$	31K (60K)	C3	792	9.6, 7.5
Mat twist (10s) 150x150	45K, 133K, 90K	0.04	1e3, 2e4 0.4	$10^{-3}\ell$	$10^{-2}\ell$	87K (162K)	C3	1815	33.1, 10.9
Mat twist (10s) 225x225	101K, 301K, 202K	0.04	1e3, 2e4 0.4	$10^{-3}\ell$	$10^{-2}\ell$	222K (408K)	C3	3895	107.2, 15.3
Squishy ball (w/ AMGCL)	688K, 2314K, 1064K	1E-3	1e3, 7e4, 0.4	$10^{-4}\ell$	$4 \times 10^{-2}\ell$	3.6K (105K)	C5	19463	328.3, 12.2

Table C.6: IPC scene statistics: Examples with friction

Example	# nodes, # tets, # faces	h (s)	ρ (kg/m^3) E (Pa), ν	$\hat{d}$ (m)	μ, ϵ_v (m/s), # friction iters	Newton tol (m/s)	# contact avg (max)	CPU	mem. (MB)	per timestep t (s), # iters
Ball mesh roller	7K, 28K, 11K	0.01	1e3, 1e4, 0.4	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	2.3K (5.6K)	C1	215	63.3, 58.6
Ball segment roller	7K, 28K, 11K	0.01	1e3, 1e4, 0.4	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	1.4K (4K)	C1	210	35.6, 51.8
Ball points roller	7K, 28K, 11K	0.01	1e3, 1e4, 0.4	$10^{-3}\ell$	1e-3, $10^{-3}\ell$, 1	$10^{-2}\ell$	305 (1.2K)	C1	210	9.5, 26.9
Hit board house	6K, 15K, 11K	0.025	1e3, 1e8, 0.4	$10^{-4}\ell$	1.0, $10^{-5}\ell$, 2	$10^{-2}\ell$	7K (13K)	C1	186	10.0, 16.6
Stable board house	5K, 15K, 11K	0.025	1e3, 1e8, 0.4	$10^{-3}\ell$	0.2, $10^{-5}\ell$, ∞	$10^{-2}\ell$	1.3K (1.4K)	C1	205	0.4, 1.2
Unstable board house	5K, 15K, 11K	0.025	1e3, 1e8, 0.4	$10^{-3}\ell$	0.1, $10^{-5}\ell$, ∞	$10^{-2}\ell$	7K (10K)	C1	299	2.4, 8.9
Stick-slip pen (NM)	630, 2K, 1K	1e-3	1e3, 1e5, 0.4	$10^{-3}\ell$	0.35, $10^{-6}\ell$, 1	$10^{-4}\ell$	0 (16)	C1	71	0.5, 30.8
Mat on board	6K, 18K, 12K	0.025	1e3, 1e4, 0.4	$10^{-3}\ell$	0.1, $10^{-3}\ell$, 1	$10^{-2}\ell$	1.7K (8,2K)	C1	243	7.7, 15.24
Cement Arch	216, 150, 324	0.01	2300, 2e10, 0.2	10^{-6}	0.5, $10^{-5}\ell$, ∞	$10^{-4}\ell$	101 (118)	C3	54	0.05, 5.7
Toy arch fall	216, 150, 324	0.01	1e3, 1e6, 0.4	$10^{-3}\ell$	0.2, $10^{-5}\ell$, ∞	$10^{-2}\ell$	66 (272)	C3	55	0.15, 17.3
Block on slope 1	8, 6, 24	0.025	1e3, 1e9, 0.4	$10^{-3}\ell$	0.5, $10^{-5}\ell$, ∞	$10^{-4}\ell$	4 (4)	C3	50	9e-4, 1.27
Block on slope 2	8, 6, 24	0.025	1e3, 1e9, 0.4	$10^{-3}\ell$	0.49, $10^{-5}\ell$, ∞	$10^{-4}\ell$	4 (4)	C3	51	3e-3, 6.9
Stick-slip Armadillo roller	67K, 386K, 24K	0.025	1e3, 5e5, 0.2	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 2	$10^{-2}\ell$	7.8K (16.2K)	C4	3542	1281.4, 113.9
Stick-slip Armadillo roller (FCR)	67K, 386K, 24K	0.025	1e3, 5e5, 0.2	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	8K (33K)	C3	3651	346, 66.8
Softer Armadillo roller (FCR)	67K, 386K, 24K	0.025	1e3, 1e5, 0.2	$10^{-3}\ell$	0.5, $10^{-3}\ell$, 1	$10^{-2}\ell$	11K (27K)	C4	3646	664,8, 60.3

D | Rigid IPC Technical Details

D.1 Robustly Computing Rodrigues' Rotation Formula

D.1.1 Talyor Series Expansion of sinc

To avoid numerical issues when computing $\text{sinc}(x)$ we instead use

$$\text{sinc}(x) = \begin{cases} x^4/120 - x^2/6 + 1 & |x| \leq \epsilon \\ \frac{\sin(x)}{x} & \text{otherwise} \end{cases}$$

where $\text{sinc}(|x| \leq \epsilon)$ is computed using a fifth-order Taylor series expansion around zero.

D.1.2 Rodrigues' Rotation Formula Derivatives

To avoid numerical issues in derivatives of Rodrigues' rotation formula (Equation (4.14)) we use a Taylor series expansion around 0. The gradient of $\text{sinc}(\|\boldsymbol{\theta}\|)$ is computed as

$$\nabla \text{sinc}(\|\boldsymbol{\theta}\|) = g(\|\boldsymbol{\theta}\|)\boldsymbol{\theta}$$

with

$$g(x) = \begin{cases} x^4/840 + x^2/30 - 1/3 & |x| \leq \epsilon_\nabla \\ (x\cos(x) - \sin(x))/x^3 & \text{otherwise} \end{cases}$$

where $\epsilon_\nabla = 10^{-4}$. The Hessian of $\text{sinc}(\|\boldsymbol{\theta}\|)$ is computed as

$$\nabla^2 \text{sinc}(\|\boldsymbol{\theta}\|) = h(\|\boldsymbol{\theta}\|)\boldsymbol{\theta}\boldsymbol{\theta}^\top + g(\|\boldsymbol{\theta}\|)\boldsymbol{I}$$

with

$$h(x) = \begin{cases} x^4/7560 - x^2/210 + 1/15 & |x| \leq \epsilon_{\nabla^2} \\ (-x^2\sin(x) - 3x\cos(x) + 3\sin(x))/x^5 & \text{otherwise} \end{cases}$$

where $\epsilon_{\nabla^2} = 0.1$.

D.1.3 Interval Computation of sinc

Given an interval $x = [a, b]$ we want to compute $\text{sinc}(x)$ while avoiding exponentially large intervals around 0. We first start by exploiting the evenness of sinc to compute

$$y_{\text{neg}} = \text{sinc}(x \cap [-\infty, 0]) = \text{sinc}(-(x \cap (-\infty, 0])).$$

Now that our domain is from $[0, \infty)$, we utilize the monotonicity of sinc to decompose x into $x \cap [0, m]$ and $x \cap [m, \infty)$ where $m = 4.4934094579$ is a conservative value lower value for the upper bound of the monotonic sub-domain. The latter case can be computed as normal using interval division because the values are not too small. For $\text{sinc}(x \cap [0, m])$, we compute sinc as

$$\begin{aligned} y_{\text{monotonic}} = [\,&\text{lower}(\text{sinc}([\text{upper}(x \cap [0, m])])), \\ &\text{upper}(\text{sinc}([\text{lower}(x \cap [0, m])]))] \end{aligned}$$

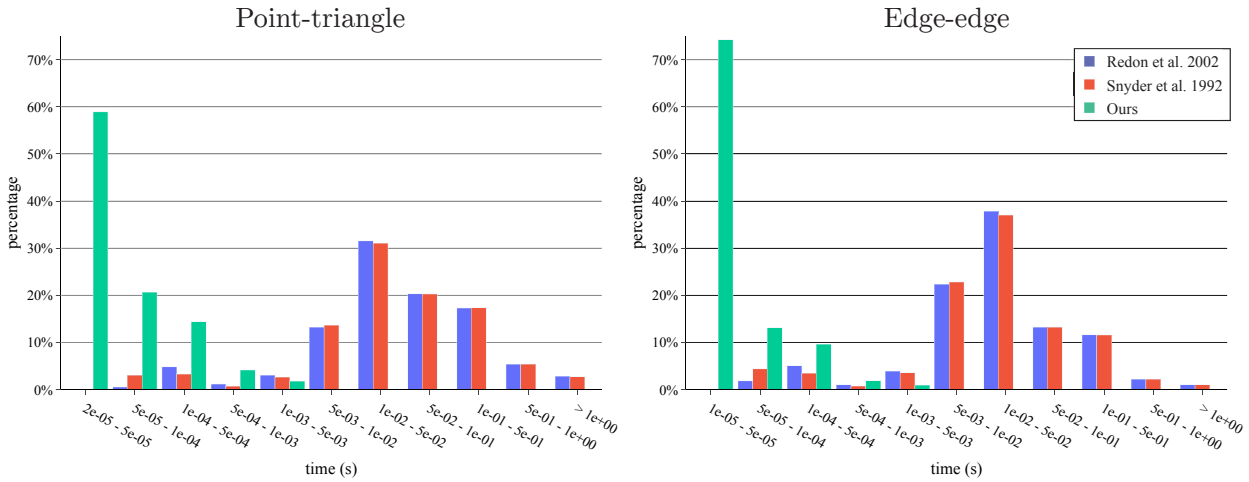


Figure D.1: Rigid CCD comparison. We compare our narrow-phase curved CCD with the methods of Snyder [1992] and [Redon et al. 2002a]. We extracted 43K point-triangle and 240K edge-edge queries from the first ten steps of our bolt simulation (Figure 4.4) which has a good mix of linear and rotating contacts between close conforming geometry. Our method is several orders of magnitude faster than prior methods (x-axis is logarithmic).

where [.] indicates computing an interval containing a single value to account for floating-point rounding. Finally, we combine all sub-domain results using the hull of all ranges.

D.2 Comparison for Curved CCD

We compared our curved narrow-phase CCD with the interval-based root-finding methods of Snyder [1992] and Redon et al. [2002a]. Figure D.1 contains a histogram of query timings, illustrating the orders of magnitude improvement of our method over previous works. This performance is due in part to the expensive nature of interval arithmetic but also the use of multivariate root-finder of in the case of Snyder [1992] and degeneracies in the univariate formulation of Redon et al. [2002a]. This results in queries that can take several seconds to process (our maximum time for point-triangle queries is 0.02s and for edge-edge is 0.3s).

D.3 Effect of δ

The parameter δ in our curved CCD controls the adaptive subdivision of our trajectories and in turn the accuracy and runtime of CCD. To demonstrate these effects we simulate the Piston (Figure 4.2) with three different values of δ: 0.1, 0.5, and 0.9. We do not consider a value outside of $(0, 1)$ because our distances are all unsigned and a value $\delta > 1$ could result in the immediate termination of the linear CCD (the initial distance is less than the minimum distance b).

For $\delta = 0.1$, the CCD is forced to do unnecessary refinement leading to a high runtime (611.6s with 227 Newton iterations).

For $\delta = 0.9$, the CCD requires less refinement and is, therefore, faster, but it is less accurate as the error b is not tightly bound. This inaccuracy results in a large number of Newton iterations (1162 iterations) which ultimately shifts the bottleneck and results in a large runtime (742.6s).

We, therefore, choose to use the Goldilocks value of $\delta = 0.5$ because it provides the best trade-off between runtime (130.6s) and iterations (211 iterations).

D.4 Comparison with IPC

We compared against IPC on a set of nine scenes with varying geometric complexity and numbers of bodies. Table D.1 provides a detailed summary of the total runtime and number of newton iterations. For scenes with simple geometry (Arch (25 and 101 stones) and Wrecking ball), our rigid formulation has little to no performance advantage over IPC because of its cheaper linear CCD. For more complex geometries (the chain net (4×4 and 8×8) and rolling cone) IPC suffers due to the large number of DOF.

Table D.1: IPC comparison. We compare our method with the volumetric IPC (Chapter 2) on a variety of scenes with varying geometric complexity and number of bodies. IPC performs well (in some cases better) than our method when the geometry is easily represented by only surface elements. When the geometry is complex, however, our reduced DOF allows us to get a performance gain.

Example	Runtime (s) (IPC)	Runtime (s) (Rigid)	Speed-up	Iterations (IPC)	Iterations (Rigid)
Pendulum	339.7	133.1	2.6×	10K	3K
Double pendulum	914.0	1559.9	0.6×	12K	4K
Arch (25 stones)	26.5	55.8	0.5×	2K	2K
Arch (101 stones)	238.3	487.8	0.5×	4K	5K
Wrecking ball	7179.8	5748.1	1.2×	9K	18K
Bolt	4031.0	1436.9	2.8×	24K	4K
Rolling cone	1184.2	150.9	7.8×	21K	16K
Chain net (4×4)	1369.9	99.8	13.7×	4K	3K
Chain net (8×8)	9950.5	1420.9	7.0×	5K	5K

D.5 Interpolating Large Rotation Vectors

Although rotation vectors are invariant to multiples of 2π, adding rotation vectors whose axes are not aligned is not. In fact, adding a small rotation update to a large rotation vector will result in a rotation axis close to the large rotation's axis. For example, $[0, 0, 0] + [0, 1, 0] = [0, 1, 0]$ results in a rotation of 1 radian around the y-axis, but $[2\pi, 0, 0] + [0, 1, 0] = [2\pi, 1, 0]$ results in a rotation of $\sqrt{4\pi^2 + 1}$ radians around an axis $\approx [0.988, 0.157, 0]$.

In our experiments, we find this property has little to no effect on the quality of simulation. In synthetic tests, however, this can lead to an increased number of Newton iterations (more updates necessary to move the axis of rotation) or small displacements that can trigger early convergence in our Newton optimization (using the same displacement-based convergence as in Chapter 2).

An easy fix to this problem, should the need ever arise, is to substitute the resulting rotation vector $\boldsymbol{\theta} = \theta \boldsymbol{a}$ with $(\theta \mod 2\pi)\boldsymbol{a}$ at the end of the timestep. It is important to only do this at the end of the timestep to avoid discontinuities in our potential during the optimization.

E | CCD Technical Details

E.1 Dataset Format

To avoid any loss of precision we convert every input floating-point coordinate in rationals using GNU Multiple Precision Arithmetic Library (GMP) [Granlund and the GMP Development Team 2012]. This conversion is exact since every floating point can be converted in a rational number, as long as the numerator and denominator are arbitrarily large integers. We then store the numerator and denominator as a `string` since the numerator and denominator can be larger than a `long` number. To retrieve the floating point number we allocate a GMP rational number with the two strings and convert it to `double`.

In summary, one CCD query is represented by a 8×7 matrix where every row is one of the 8 CCD input points, and the columns are the interleaved x, y, z coordinates of the point, represented as numerator and denominator. For convenience, we appended several such matrices in a common comma-separated values (CSV) file. The last column represents the result of the

ground truth. For instance a CC query between $p_1^0, p_2^0, p_3^0, p_4^0$ and $p_1^1, p_2^1, p_3^1, p_4^1$ is represented as

$$\begin{array}{cccccccc}
p^0_{1^x_n} & p^0_{1^x_d} & p^0_{1^y_n} & p^0_{1^y_d} & p^0_{1^z_n} & p^0_{1^z_d} & T \\
p^0_{2^x_n} & p^0_{2^x_d} & p^0_{2^y_n} & p^0_{2^y_d} & p^0_{2^z_n} & p^0_{2^z_d} & T \\
p^0_{3^x_n} & p^0_{3^x_d} & p^0_{3^y_n} & p^0_{3^y_d} & p^0_{3^z_n} & p^0_{3^z_d} & T \\
p^0_{4^x_n} & p^0_{4^x_d} & p^0_{4^y_n} & p^0_{4^y_d} & p^0_{4^z_n} & p^0_{4^z_d} & T \\
p^1_{1^x_n} & p^1_{1^x_d} & p^1_{1^y_n} & p^1_{1^y_d} & p^1_{1^z_n} & p^1_{1^z_d} & T \\
p^1_{2^x_n} & p^1_{2^x_d} & p^1_{2^y_n} & p^1_{2^y_d} & p^1_{2^z_n} & p^1_{2^z_d} & T \\
p^1_{3^x_n} & p^1_{3^x_d} & p^1_{3^y_n} & p^1_{3^y_d} & p^1_{3^z_n} & p^1_{3^z_d} & T \\
p^1_{4^x_n} & p^1_{4^x_d} & p^1_{4^y_n} & p^1_{4^y_d} & p^1_{4^z_n} & p^1_{4^z_d} & T,
\end{array}$$

where $p^t_{i^x_n}$ and $p^t_{i^x_d}$ are respectively the numerator and denominator of the x-coordinate of p, and T is the same ground truth. The dataset and a query viewer can be downloaded from the NYU Faculty Digital Archive[1].

E.2 Example of Degenerate Case not Properly Handled by Brochu et al. [2012]

Let

$$\begin{aligned}
p^0 &= [0.1, 0.1, 0.1], \quad v_1^0 = [0, 0, 1], \quad v_2^0 = [1, 0, 1], \quad v_3^0 = [0, 1, 1], \\
p^1 &= [0.1, 0.1, 0.1], \quad v_1^1 = [0, 0, 0], \quad v_2^1 = [0, 1, 0], \quad v_3^1 = [1, 0, 0]
\end{aligned} \tag{E.1}$$

be the input point and triangle. Checking if the point intersects the triangle is equivalent to check if the prism shown in Figure E.1 contains the origin. However, the prism contains a bilinear face that is degenerate (it looks like a "hourglass"). The algorithm proposed in [Brochu et al. 2012] does not consider this degenerate case and erroneously reports no collision.

[1] NYU Faculty Digital Archive: https://archive.nyu.edu/handle/2451/61518

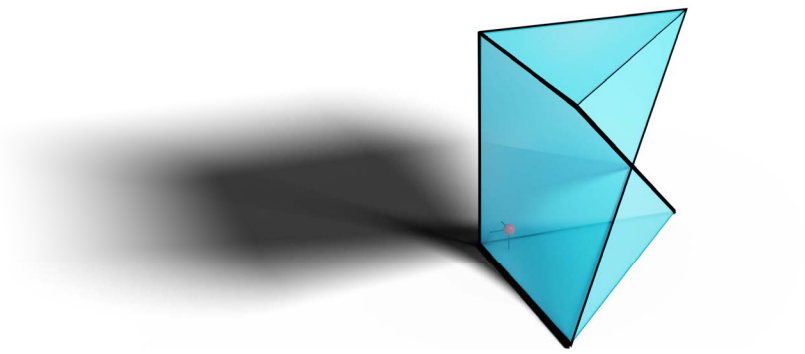


Figure E.1: Prism resulting from the input points and triangle in Equation (E.1). The origin is marked by the red dot.

E.3 Example of Inflection Point not Properly Handled by Tang et al. [2014]

Let

$$p^0 = [1,1,0], \quad v_1^0 = [0,0,5], \quad v_2^0 = [2,0,2], \quad v_3^0 = [0,1,0],$$
$$p^1 = [1,1,0], \quad v_1^1 = [0,0,-1], \quad v_2^1 = [0,0,-2], \quad v_3^1 = [0,7,0]$$

be the input point and triangle. Checking if they intersect at time t is equivalent to finding the roots of

$$-72t^3 + 120t^2 - 44t + 3.$$

To apply the method in [Tang et al. 2014] we need to rewrite the polynomial in form of Tang et al. [2014, Equation (1)]:

$$1B_0^3(t) - \frac{35}{3}B_1^3(t) + \frac{82}{3}B_2^3(t) + 14B_3^3(t).$$

Their algorithm assumes no inflection points in the Bezier curve. Thus it proposes to split the curve at the eventual inflection point (as in the case above). The formula proposed in [Tang et al.

2014, Section 4.1] contains a typo, by fixing it we obtain the inflection point at:

$$t = \frac{6k_0 - 4k_1 + k_2}{6k_0 - 6k_1 + 3k_2 - k_3} = \frac{5}{9}.$$

By using the incorrect formula we obtain $t = 155/312$, which is not an inflection point. In both cases, t cannot be computed exactly since it contains a division, and computing it approximately breaks the assumption of not having inflection points in the Bezier form. In the reference code, the authors detect the presence of an inflection point using predicates, but do not split the curve (the case is not handled). We modified the code (patch attached in the additional material) to conservatively return a collision in these cases.

Independently from this problem, their reference implementation returns false negative (i.e. misses collisions) for certain configurations, such as the following degenerate configuration:

$$\begin{aligned} p^0 &= [1, 0.5, 1], & v_1^0 &= [0, 0.57, 1], & v_2^0 &= [1, 0.57, 1], & v_3^0 &= [1, 1.57, 1], \\ p^1 &= [1, 0.5, 1], & v_1^1 &= [0, 0.28, 1], & v_2^1 &= [1, 0.28, 1], & v_3^1 &= [1, 1.28, 1]. \end{aligned}$$

We could not find out why this is happening, and we do not know if this is a theoretical or numerical problem, or a bug in the implementation.

E.4 Effect of δ on the interval-based methods

UIRF, IRF, and our method have a single parameter δ to control the size of the interval. Increasing δ will introduce more false positive, while making the algorithms faster (Figure E.2). Note that we limit the total running time to 24 h, thus UIRF does not have result for $\delta > 10^{-6}$ (for $\delta = 10^{-6}$ it takes 1 ms per query in average). δ has a similar effect on the number of false positives for the three interval based methods, while it has a more significant impact on the running time

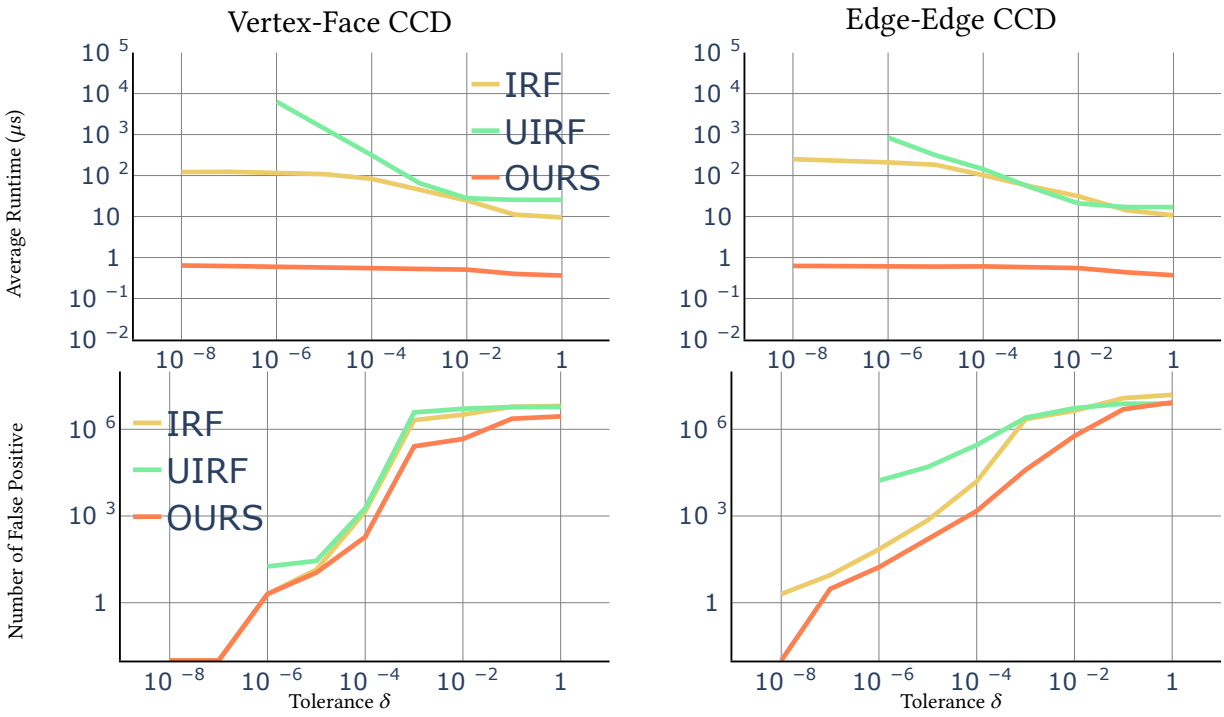


Figure E.2: Log plot of the effect of the tolerance δ on the running time (top) and false positives (bottom) for the three (Ours, UIRF, and IRF) interval based methods on the simulation dataset.

for UIRF and IRF.

E.5 Minimum separation with FPRF

In Table E.1, we compare our method with FPRF by changing the parameter η that mimics minimum separation.

Table E.1: FPRF MSCCD benchmark. Summary of the average runtime in µs (t), number of FP, and number of FN for FPRF and our method.

Handcrafted Dataset

	Vertex-Face MSCCD						Edge-Edge MSCCD					
	FPRF			Ours			FPRF			Ours		
d	t	FP	FN	t	FP	FN	t	FP	FN	t	FP	FN
10^{-2}	2.41	1.8K	4	18.86K	2.6K	0	1.16	3.3K	19	9.64K	4.8K	0
10^{-8}	4.53	83	3	1.60K	159	0	0.60	160	28	3.42K	309	0
10^{-16}	2.23	29	69	1.51K	108	0	0.55	45	145	2.92K	214	0
10^{-30}	2.24	9	70	1.39K	108	0	0.58	5	147	2.79K	214	0
10^{-100}	2.31	9	70	1.43K	108	0	0.80	5	147	2.82K	214	0

Simulation Dataset

	Vertex-Face MSCCD						Edge-Edge MSCCD					
	FPRF			Ours			FPRF			Ours		
d	t	FP	FN	t	FP	FN	t	FP	FN	t	FP	FN
10^{-2}	8.04	869.1K	1	12.04	8.1M	0	8.01	1.1M	0	19.12	8.3M	0
10^{-8}	8.00	4	2	0.72	8	0	0.77	16	0	0.73	40	0
10^{-16}	7.78	0	5.2K	0.71	2	0	0.25	0	2.3K	0.72	17	0
10^{-30}	7.77	0	5.2K	0.66	2	0	0.25	0	2.3K	0.67	17	0
10^{-100}	7.75	0	5.2K	0.66	2	0	0.25	0	2.3K	0.68	17	0

Bibliography

Alappat, C., Basermann, A., Bishop, A. R., Fehske, H., Hager, G., Schenk, O., Thies, J., and Wellein, G. (2020). A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. *ACM Transactions on Parallel Computing*, 7(3).

Alart, P. and Curnier, A. (1991). A mixed formulation for frictional contact problems prone to newton like solution methods. *Computer Methods in Applied Mechanics and Engineering*, 92(3):353–375.

Aldakheel, F., Hudobivnik, B., Artioli, E., Beirão da Veiga, L., and Wriggers, P. (2020). Curvilinear virtual elements for contact mechanics. *Computer Methods in Applied Mechanics and Engineering*, 372:113394.

Anitescu, M. and Hart, G. D. (2004a). A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. *International Journal for Numerical Methods in Engineering*, 60(14):2335–2371.

Anitescu, M. and Hart, G. D. (2004b). A fixed-point iteration approach for multibody dynamics with contact and small friction. *Mathematical Programming*, 101(1):3–32.

Anitescu, M. and Potra, F. R. (1997). Formulating dynamic multirigid-body contact problems with friction as solvable linear complementarity problems. *ASME Nonlinear Dynamics*, 14:231–247.

Ansys, Inc. (1971–2023). ANSYS. https://www.ansys.com/.

Ascher, U. M. and Petzold, L. R. (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, USA, 1st edition.

Attene, M. (2020). Indirect predicates for geometric constructions. *Computer-Aided Design*, 126:102856.

Babuska, I. and Guo, B. Q. (1988). The h-p version of the finite element method for domains with curved boundaries. *SIAM Journal on Numerical Analysis*, 25(4):837–861.

Babuška, I. and Guo, B. (1992). The h, p and h-p version of the finite element method; basis theory and applications. *Advances in Engineering Software*, 15(3):159–174.

Ball, J. M. (1981). Global invertibility of sobolev functions and the interpenetration of matter. In *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, volume 88, pages 315–328. Royal Society of Edinburgh Scotland Foundation.

Baraff, D. (1989). Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics (Proceedings of SIGGRAPH)*, 23(3):223–232.

Baraff, D. (1991). Coping with friction for non-penetrating rigid body simulation. *Computer Graphics (Proceedings of SIGGRAPH)*, 25(4):31–41.

Baraff, D. (1994). Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1994, pages 23–34, New York, NY, USA. Association for Computing Machinery.

Bargteil, A. and Shinar, T. (2018). An introduction to physics-based animation. In *ACM SIGGRAPH 2018 Courses*, New York, NY, USA. Association for Computing Machinery.

Bargteil, A. W. and Cohen, E. (2014). Animation of deformable bodies with quadratic bézier finite elements. *ACM Transactions on Graphics*, 33(3):27.

Bargteil, A. W., Wojtan, C., Hodgins, J. K., and Turk, G. (2007). A finite element method for animating large viscoplastic flow. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 26(3).

Bassi, F. and Rebay, S. (1997). High-order accurate discontinuous finite element solution of the 2d euler equations. *Journal of Computational Physics*, 138(2):251–285.

Belgacem, F., Hild, P., and Laborde, P. (1998). The mortar finite element method for contact problems. *Mathematical and Computer Modelling*, 28(4):263–271. Recent Advances in Contact Mechanics.

Belgrod, D., Wang, B., Ferguson, Z., Attene, M., Panozzo, D., and Schneider, T. (2022). Scalable and conservative continuous collision detection for parallel architectures.

Belytschko, T., Liu, W., and Moran, B. (2000). *Nonlinear Finite Elements for Continua and Structures*, volume 26. John Wiley & Sons, Ltd.

Bender, J. and Deul, C. (2013). Adaptive cloth simulation using corotational finite elements. *Computers & Graphics*, 37(7):820–829.

Bender, J., Erleben, K., Trinkle, J., and Coumans, E. (2012). Interactive simulation of rigid body dynamics in computer graphics. In Cani, M.-P. and Ganovelli, F., editors, *Eurographics 2012 - State of the Art Reports*, volume 33, pages 246–270. The Eurographics Association.

Bertails-Descoubes, F., Cadoux, F., Daviet, G., and Acary, V. (2011). A nonsmooth newton solver for capturing exact coulomb friction in fiber assemblies. *ACM Transactions on Graphics*, 30(1):1–14.

Bertsekas, D. P. (2016). *Nonlinear Programming*. Athena Scientific.

Bog, T., Zander, N., Kollmannsberger, S., and Rank, E. (2015). Normal contact with high order finite elements and a fictitious contact material. *Computers & Mathematics with Applications*, 70(7):1370–1390. High-Order Finite Element and Isogeometric Methods.

Bollhöfer, M., Eftekhari, A., Scheidegger, S., and Schenk, O. (2019). Large-scale sparse inverse covariance matrix estimation. *SIAM Journal on Scientific Computing*, 41(1):380–401.

Bollhöfer, M., Schenk, O., Janalik, R., Hamm, S., and Gullapalli, K. (2020). State-of-the-art sparse direct solvers. *Parallel Algorithms in Computational Science and Engineering*, pages 3–33.

Bouaziz, S., Martin, S., Liu, T., Kavan, L., and Pauly, M. (2014). Projective dynamics: Fusing constraint projections for fast simulation. *ACM Transactions on Graphics*, 33(4).

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, USA.

Bridson, R., Fedkiw, R., and Anderson, J. (2002). Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics*, 21(3):594–603.

Brochu, T. and Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing*, 31(4):2472–2493.

Brochu, T., Edwards, E., and Bridson, R. (2012). Efficient geometrically exact continuous collision detection. *ACM Transactions on Graphics*, 31(4):96:1–96:7.

Brogliato, B. (1999). *Nonsmooth Mechanics*. Springer-Verlag.

Budd, C. J., Huang, W., and Russell, R. D. (2009). Adaptivity with moving grids. *Acta Numerica*, 18:111–241.

Canny, J. (1986). Collision detection for moving polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Pami-8(2):200–209.

Cardoso, R. P. R. and Adetoro, O. B. (2017). On contact modelling in isogeometric analysis. *European Journal of Computational Mechanics*, 26(5-6):443–472.

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. (2008). Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35(3):1–14.

Chen, Y., Li, M., Lu, W., Fu, C., and Jiang, C. (2022). Midas: A multi-joint robotics simulator with intersection-free frictional contact.

Chentanez, N., Alterovitz, R., Ritchie, D., Cho, L., Hauser, K. K., Goldberg, K., Shewchuk, J. R., and O'Brien, J. F. (2009). Interactive simulation of surgical needle insertion and steering. *ACM Transactions on Graphics*, 28(3).

Choi, H., Crump, C., Duriez, C., Elmquist, A., Hager, G., Han, D., Hearl, F., Hodgins, J., Jain, A., Leve, F., Li, C., Meier, F., Negrut, D., Righetti, L., Rodriguez, A., Tan, J., and Trinkle, J. (2021). On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward. *Proceedings of the National Academy of Sciences*, 118(1).

Chouly, F., Hild, P., Lleras, V., and Renard, Y. (2022). Nitsche method for contact with coulomb friction: Existence results for the static and dynamic finite element formulations. *Journal of Computational and Applied Mathematics*, 416:114557.

Cline, M. B. and Pai, D. K. (2003). Post-stabilization for rigid body simulation with contact and constraints. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 3744–3751. IEEE.

Coevoet, E., Benchekroun, O., and Kry, P. G. (2020). Adaptive merging for rigid body simulation. *ACM Transactions on Graphics*, 39(4):35–1.

COMSOL Inc. (1998–2022). COMSOL Multiphysics. https://www.comsol.com/.

COMSOL Inc. (2019). *Structural Mechanics Module User's Guide*. COMSOL Inc., 5.5 edition. https://doc.comsol.com/5.5/doc/com.comsol.help.sme/StructuralMechanicsModuleUsersGuide.pdf.

Coumans, E. and Bai, Y. (2016–2019). Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Courant, R., Friedrichs, K., and Lewy, H. (1967). On the partial difference equations of mathematical physics. *IBM Journal of Research and Development*, 11(2):215–234.

Curtis, S., Gayle, R., Govindaraju, N., Kabul, I., Lin, M., Pabst, S., Redon, S., Sud, A., Tang, M., Yoon, S., Zhao, J., and Manocha, D. (2012). UNC Dynamic Scene Benchmarks. http://gamma.cs.unc.edu/DYNAMICB.

Curtis, S., Tamstorf, R., and Manocha, D. (2008). Fast collision detection for deformable models using representative-triangles. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 61–69.

Da, F., Batty, C., and Grinspun, E. (2014). Multimaterial mesh-based surface tracking. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*.

Daviet, G., Bertails-Descoubes, F., and Boissieux, L. (2011). A hybrid iterative solver for robustly capturing coulomb friction in hair dynamics. *ACM Transactions on Graphics*, 30:1–12.

Debunne, G., Desbrun, M., Cani, M.-P., and Barr, A. H. (2001). Dynamic real-time deformations using space and time adaptive sampling. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 31–36, New York, NY, USA. Association for Computing Machinery.

Demidov, D. (2019). Amgcl: An efficient, flexible, and extensible algebraic multigrid implementation. *Lobachevskii Journal of Math.*, 40(5):535–546.

Demkowicz, L. (2006). *Computing with hp-ADAPTIVE FINITE ELEMENTS*. Chapman and Hall/CRC.

Doyen, D., Ern, A., and Piperno, S. (2011). Time-integration schemes for the finite element dynamic signorini problem. *SIAM Journal on Scientific Computing*, 33:223–249.

Du, T., Wu, K., Ma, P., Wah, S., Spielberg, A., Rus, D., and Matusik, W. (2021). Diffpd: Differentiable projective dynamics. *ACM Transactions on Graphics*, 41(2).

Dunyach, M., Vanderhaeghe, D., Barthe, L., and Botsch, M. (2013). Adaptive Remeshing for Real-Time Mesh Deformation. In Otaduy, M.-A. and Sorkine, O., editors, *Eurographics 2013 - Short Papers*. The Eurographics Association.

Erhart, T., Wall, W. A., and Ramm, E. (2006). Robust adaptive remeshing strategy for large deformation, transient impact simulations. *International Journal for Numerical Methods in Engineering*, 65(13):2139–2166.

Erleben, K. (2007). Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics*, 26(2):12–es.

Erleben, K. (2017). Rigid body contact problems using proximal operators. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, New York, NY, USA. Association for Computing Machinery.

Erleben, K. (2018). Methodology for assessing mesh-based contact point methods. *ACM Transactions on Graphics*, 37(3).

Fang, Y., Li, M., Jiang, C., and Kaufman, D. M. (2021). Guaranteed globally injective 3d deformation processing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 40(4).

Faure, F., Duriez, C., Delingette, H., Allard, J., Gilles, B., Marchesseau, S., Talbot, H., Courtecuisse, H., Bousquet, G., Peterlik, I., et al. (2012). Sofa: A multi-model framework for interactive

physical simulation. In *Soft tissue biomechanical modeling for computer assisted surgery*, pages 283–321. Springer.

Faure, F., Gilles, B., Bousquet, G., and Pai, D. K. (2011). Sparse meshless models of complex deformable solids. *ACM Transactions on Graphics*, 30(4).

Ferguson, Z. et al. (2020). IPC Toolkit. https://ipc-sim.github.io/ipc-toolkit/.

Ferguson, Z., Li, M., Schneider, T., Gil-Ureta, F., Langlois, T., Jiang, C., Zorin, D., Kaufman, D. M., and Panozzo, D. (2021). Intersection-free rigid body dynamics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 40(4).

Fogleman, M. (2017). Binary packing for SLS printing. https://www.michaelfogleman.com/pack3d/.

Franke, D., Düster, A., Nübel, V., and Rank, E. (2010). A comparison of the h-, p-, hp-, and rp-version of the fem for the solution of the 2d hertzian contact problem. *Computational Mechanics*, 45(5):513–522.

Franke, D., Düster, A., and Rank, E. (2008). The p-version of the fem for computational contact mechanics. *Pamm*, 8(1):10271–10272.

Geilinger, M., Hahn, D., Zehnder, J., Bächer, M., Thomaszewski, B., and Coros, S. (2020). ADD: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics*, 39(6):1–15.

Gholamalizadeh, T., Moshfeghifar, F., Ferguson, Z., Schneider, T., Panozzo, D., Darkner, S., Makaremi, M., Chan, F., Søndergaard, P. L., and Erleben, K. (2022). Open-full-jaw: An open-access dataset and pipeline for finite element models of human jaw. *Computer Methods and Programs in Biomedicine*, 224:107009.

Govindaraju, N., Knott, D., Jain, N., Kabul, I., Tamstorf, R., Gayle, R., Lin, M., and Manocha, D. (2005). Collision detection between deformable models using chromatic decomposition. *ACM Transactions on Graphics*, 24:991–999.

Goyal, S., Ruina, A., and Papadopoulos, J. (1991). Planar sliding with dry friction part 2. dynamics of motion. *Wear*, 143(2):331–352.

Granlund, T. and the GMP Development Team (2012). *GNU MP: The GNU Multiple Precision Arithmetic Library*. GNU Project, 5.0.5 edition. http://gmplib.org/.

Grassia, F. S. (1998). Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48.

Grinspun, E., Krysl, P., and Schröder, P. (2002). Charms: A simple framework for adaptive simulation. *ACM Transactions on Graphics*, 21(3):281–290.

Guendelman, E., Bridson, R., and Fedkiw, R. (2003). Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 22(3):871–878.

Guennebaud, G., Jacob, B., et al. (2010a). Eigen v3. http://eigen.tuxfamily.org.

Guennebaud, G., Jacob, B., et al. (2010b). Eigen v3. http://eigen.tuxfamily.org.

Gurobi Optimization, LLC (2019). Gurobi optimizer reference manual. http://www.gurobi.com.

Gustafsson, T., Stenberg, R., and Videman, J. (2020). On nitsche's method for elastic contact problems. *SIAM Journal on Scientific Computing*, 42(2):B425–B446.

Hahn, D. and Wojtan, C. (2015). High-resolution brittle fracture simulation with boundary elements. *ACM Transactions on Graphics*, 34(4).

Hahn, J. K. (1988). Realistic animation of rigid bodies. *Computer Graphics (Proceedings of SIGGRAPH)*, 22(4):299–308.

Hairer, E., Lubich, C., and Wanner, G. (2006). *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31. Springer.

Harmon, D., Panozzo, D., Sorkine, O., and Zorin, D. (2011). Interference-aware geometric modeling. *ACM Transactions on Graphics*, 30(6):1–10.

Harmon, D., Vouga, E., Smith, B., Tamstorf, R., and Grinspun, E. (2009). Asynchronous contact mechanics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 28(3):1–12.

Harmon, D., Vouga, E., Tamstorf, R., and Grinspun, E. (2008). Robust treatment of simultaneous collisions. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(3):1–4.

Hormann, K., Greiner, G., and Campagna, S. (1998). Hierarchical parametrization of triangulated surfaces. *Proceedings of Vision, Modeling and Visualization*.

Hsu, S.-W. and Keyser, J. (2010). Piles of objects. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, page 155.

Hu, Y., Schneider, T., Wang, B., Zorin, D., and Panozzo, D. (2020). Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics*, 39(4).

Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., and Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Transactions on Graphics*, 37(4):60:1–60:14.

Huang, Z., Tozoni, D. C., Gjoka, A., Ferguson, Z., Schneider, T., Panozzo, D., and Zorin, D. (2022). Differentiable solver for time-dependent deformation problems with contact.

Hubbard, P. M. (1995). Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230.

Hüeber, S. and Wohlmuth, B. (2006). *Mortar methods for contact problems*, pages 39–47. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hughes, T., Cottrell, J., and Bazilevs, Y. (2005). Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135–4195.

Hutchinson, D., Preston, M., and Hewitt, T. (1996). Adaptive refinement for mass/spring simulations. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, pages 31–45, Berlin, Heidelberg. Springer-Verlag.

Hutter, M. and Fuhrmann, A. (2007). Optimized continuous collision detection for deformable triangle meshes. *Journal of WSCG*, 15:25–32.

Jacobson, A., Baran, I., Popović, J., and Sorkine, O. (2011). Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30(4):78:1–78:8.

Jacobson, A., Panozzo, D., et al. (2018). libigl: A simple C++ geometry processing library. https://libigl.github.io/.

Jameson, A., Alonso, J., and McMullen, M. (2002). Application of a non-linear frequency domain solver to the euler and navier-stokes equations. In *40th AIAA Aerospace Sciences Meeting & Exhibit*.

Jean, M. and Moreau, J. J. (1992). Unilaterality and dry friction in the dynamics of rigid body collections. In *Proceedings of Contact Mechanics International Symposium*, volume 1.

Jiang, Z., Dai, J., Hu, Y., Zhou, Y., Dumas, J., Zhou, Q., Bajwa, G. S., Zorin, D., Panozzo, D., and Schneider, T. (2022). Declarative specification for unstructured mesh editing algorithms. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 41(6).

Jiang, Z., Schaefer, S., and Panozzo, D. (2017). Simplicial complex augmentation framework for bijective maps. *ACM Transactions on Graphics*, 36(6):1–9.

Jiang, Z., Schneider, T., Zorin, D., and Panozzo, D. (2020). Bijective projection in a shell. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(6).

Jiang, Z., Zhang, Z., Hu, Y., Schneider, T., Zorin, D., and Panozzo, D. (2021). Bijective and coarse high-order tetrahedral meshes. *ACM Transactions on Graphics*, 40(4).

Johnen, A., Remacle, J.-F., and Geuzaine, C. (2013). Geometrical validity of curvilinear finite elements. *Journal of Computational Physics*, 233:359–372.

Jones, M. W., Baerentzen, J. A., and Sramek, M. (2006). 3d distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599.

Kane, C., Marsden, J. E., Ortiz, M., and West, M. (2000). Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for Numerical Methods in Engineering*, 49(10):1295–1325.

Kane, C., Repetto, E. A., Ortiz, M., and Marsden, J. E. (1999). Finite element analysis of nonsmooth contact. *Computer Methods in Applied Mechanics and Engineering*, 180(1-2):1–26.

Kaufman, D. M. and Pai, D. K. (2012). Geometric numerical integration of inequality constrained, nonsmooth Hamiltonian systems. *SIAM Journal on Scientific Computing*, 34(5):A2670–A2703.

Kaufman, D. M., Sueda, S., James, D. L., and Pai, D. K. (2008). Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 27(5):1–11.

Kaufman, D. M., Tamstorf, R., Smith, B., Aubry, J.-M., and Grinspun, E. (2014). Adaptive nonlinearity for collisions in complex rod assemblies. *ACM Transactions on Graphics*, 33(4):1–12.

Kikuchi, N. and Oden, J. T. (1988). *Contact Problems in Elasticity: A Study of Variational Inequalities and Finite Element Methods*, volume 8 of *SIAM Studies in App. and Numer. Math.* Society for Industrial and Applied Mathematics.

Kim, B. and Rossignac, J. (2003). Collision prediction for polyhedra under screw motions. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*, pages 4–10.

Kim, C. M., Danielczuk, M., Huang, I., and Goldberg, K. (2022). IPC-GraspSim: Reducing the Sim2Real gap for parallel-jaw grasping with the incremental potential contact model. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 6180–6187. IEEE Press.

Kim, T. and Eberle, D. (2022). Dynamic deformables: Implementation and production practicalities (now with code!). In *ACM SIGGRAPH 2022 Courses*, New York, NY, USA. Association for Computing Machinery.

Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F. (2006). Fluid animation with dynamic meshes. In *Proceedings of ACM SIGGRAPH 2006*, pages 820–825, New York, NY, USA. Association for Computing Machinery.

Koh, W., Narain, R., and O'Brien, J. F. (2015). View-dependent adaptive cloth simulation with buckling compensation. *IEEE Transactions on Visualization and Computer Graphics*, 21(10):1138–1145.

Konyukhov, A. and Schweizerhof, K. (2009). Incorporation of contact for high-order finite elements in covariant form. *Computer Methods in Applied Mechanics and Engineering*, 198(13):1213–1223.

Koschier, D., Deul, C., Brand, M., and Bender, J. (2017). An hp-adaptive discretization algorithm for signed distance field generation. *IEEE Transactions on Visualization and Computer Graphics*, 23(10):2208–2221.

Koschier, D., Lipponer, S., and Bender, J. (2015). Adaptive tetrahedral meshes for brittle fracture simulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, pages 57–66, Goslar, DEU. Eurographics Association.

Krause, R. and Zulian, P. (2016). A parallel approach to the variational transfer of discrete fields between arbitrarily distributed unstructured finite element meshes. *SIAM Journal on Scientific Computing*, 38(3).

Kry, P. G. and Pai, D. K. (2003). Continuous contact simulation for smooth surfaces. *ACM Transactions on Graphics*, 22(1):106–129.

Lan, L., Kaufman, D. M., Li, M., Jiang, C., and Yang, Y. (2022a). Affine body dynamics: Fast, stable and intersection-free simulation of stiff materials. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 41(4).

Lan, L., Luo, R., Fratarcangeli, M., Xu, W., Wang, H., Guo, X., Yao, J., and Yang, Y. (2020). Medial elastics: Efficient and collision-ready deformation via medial axis transform. *ACM Transactions on Graphics*, 39(3).

Lan, L., Ma, G., Yang, Y., Zheng, C., Li, M., and Jiang, C. (2022b). Penetration-free projective dynamics on the gpu. *ACM Transactions on Graphics*, 41(4).

Lan, L., Yang, Y., Kaufman, D., Yao, J., Li, M., and Jiang, C. (2021). Medial ipc: Accelerated incremental potential contact with medial elastics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 40(4).

Lerch, M., Tischler, G., Gudenberg, J. W. V., Hofschuster, W., and Krämer, W. (2006). Filib++, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software*, 32(2):299–324.

Li, J., Daviet, G., Narain, R., Bertails-Descoubes, F., Overby, M., Brown, G. E., and Boissieux, L. (2018a). An implicit frictional contact solver for adaptive cloth simulation. *ACM Transactions on Graphics*, 37(4).

Li, L. and Volkov, V. (2005). Cloth animation with adaptively refined meshes. In *Proceedings of the Twenty-Eighth Australasian Conference on Computer Science - Volume 38*, ACSC '05, pages 107–113, AUS. Australian Computer Society, Inc.

Li, M., Ferguson, Z., Schneider, T., Jiang, C., Zorin, D., Panozzo, D., and Kaufman, D. M. (2023). Convergent incremental potential contact. arXiv.

Li, M., Ferguson, Z., Schneider, T., Langlois, T., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. (2020). Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Transactions on Graphics*, 39(4).

Li, M., Kaufman, D. M., and Jiang, C. (2021). Codimensional incremental potential contact. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 40(4).

Li, M., Kaufman, D. M., Kim, V. G., Solomon, J., and Sheffer, A. (2018b). Optcuts: Joint optimization of surface cuts and parameterization. *ACM Transactions on Graphics*, 37(6).

Lipton, R., Rose, D., and Targan, R. (1979). Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358.

Logg, A., Wells, G., and Mardal, K.-A. (2011). *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, volume 84.

Longva, A., Löschner, F., Kugelstadt, T., Fernández-Fernández, J. A., and Bender, J. (2020). Higher-order finite elements for embedded simulation. *ACM Transactions on Graphics*, 39(6).

Lötstedt, P. (1982). Numerical simulation of time-dependent contact friction problems in rigid body mechanics. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–393.

Lu, L., Morse, M. J., Rahimian, A., Stadler, G., and Zorin, D. (2019). Scalable simulation of realistic volume fraction red blood cell flows through vascular networks. In *Proceedings of the*

International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, pages 1–30, New York, NY, USA. Association for Computing Machinery. https://github.com/Continuous-Collision-Detection/Minimum-Separation-Root-Finder.

Luo, X., Shephard, M. S., and Remacle, J.-F. (2001). The influence of geometric approximation on the accuracy of high order methods. *Rensselaer SCOREC report*, 1.

Léger, S., Fortin, A., Tibirna, C., and Fortin, M. (2014). An updated lagrangian method with error estimation and adaptive remeshing for very large deformation elasticity problems. *International Journal for Numerical Methods in Engineering*, 100(13):1006–1030.

Maas, S. A., Ellis, B. J., Ateshian, G. A., and Weiss, J. A. (2012). FEBio: Finite Elements for Biomechanics. *Journal of Biomechanical Engineering*, 134(1).

Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., and Kim, T. Y. (2020). Primal/dual descent methods for dynamics. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '20, Goslar, DEU. Eurographics Association.

Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., and Makoviychuk, V. (2019). Non-smooth newton methods for deformable multi-body dynamics. *ACM Transactions on Graphics*, 38(5):1–20.

Mammou, K. (2020). V-HACD. https://github.com/kmammou/v-hacd.

Manteaux, P.-L., Sun, W.-L., Faure, F., Cani, M.-P., and O'Brien, J. F. (2015). Interactive detailed cutting of thin sheets. In *Proceedings of ACM SIGGRAPH Motion in Games*, pages 1–8.

Manteaux, P.-L., Wojtan, C., Narain, R., Redon, S., Faure, F., and Cani, M.-P. (2017). Adaptive physically based models in computer graphics. *Computer Graphics Forum*, 36(6):312–337.

Marsden, J. E. and Ratiu, T. S. (2013). *Introduction to Mechanics and Symmetry*. Springer.

Martin, S., Kaufmann, P., Botsch, M., Grinspun, E., and Gross, M. (2010). Unified simulation of elastic rods, shells, and solids. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29(3):39:1–39:10.

Mazhar, H., Heyn, T., Negrut, D., and Tasora, A. (2015). Using nesterov's method to accelerate multibody dynamics with friction and contact. *ACM Transactions on Graphics*, 34(3).

Menon, S., Mooney, K. G., Stapf, K., and Schmidt, D. P. (2015). Parallel adaptive simplical remeshing for deforming domain cfd computations. *Journal of Computational Physics*, 298:62–78.

Mezger, J., Kimmerle, S., and Etzmuss, O. (2003). Hierarchical techniques in collision detection for cloth animation. In *Journal of WSCG*, volume 11, pages 322–329.

Mezger, J., Thomaszewski, B., Pabst, S., and Straśer, W. (2009). Interactive physically-based shape editing. *Computer Aided Geometric Design*, 26(6):680–694. Solid and Physical Modeling 2008.

Mirtich, B. (2000). Timewarp rigid body simulation. *Annual Conference Series (Proceedings of SIGGRAPH)*, pages 193–200.

Mirtich, B. and Canny, J. F. (1995). Impulse-based simulation of rigid bodies. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 181–ff., New York, NY, USA. Association for Computing Machinery.

Mirtich, B. V. (1996). *Impulse-Based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley.

Misztal, M., Erleben, K., Bargteil, A., Fursund, J., Christensen, B., Bærentzen, J., and Bridson, R. (2014). Multiphase flow of immiscible fluids on unstructured moving meshes. *IEEE Transactions on Visualization and Computer Graphics*, 20(1):4–16.

Misztal, M. K. and Bærentzen, J. A. (2012). Topology-adaptive interface tracking using the deformable simplicial complex. *ACM Transactions on Graphics*, 31(3):1–12.

Mitchell, W. F. (1991). Adaptive refinement for arbitrary finite-element spaces with hierarchical bases. *Journal of Computational and Applied Mathematics*, 36(1):65–78. Special Issue on Adaptive Methods.

Mitchell, W. F. and McClain, M. A. (2014). A comparison of hp-adaptive strategies for elliptic partial differential equations. *ACM Transactions on Mathematical Software*, 41(1).

Molinari, J. F. and Ortiz, M. (2002). Three-dimensional adaptive meshing by subdivision and edge-collapse in finite-deformation dynamic–plasticity problems with application to adiabatic shear banding. *International Journal for Numerical Methods in Engineering*, 53(5):1101–1126.

Moore, M. and Wilhelms, J. (1988). Collision detection and response for computer animation. *Computer Graphics (Proceedings of SIGGRAPH)*, 22(4):289–298.

Moreau, J. J. (1966). Quadratic programming in mechanics: Dynamics of one-sided constraints. *SIAM Journal on Control*, 4(1):153–158.

Moreau, J. J. (1973). On unilateral constraints, friction and plasticity. *New Variational Tech. in Math. Phys.*, pages 171–322.

Moreau, J. J. (1988). Unilateral contact and dry friction in finite freedom dynamics. *Nonsmooth Mechanics and Applications, CISM Courses and Lectures*, pages 1–82.

Moser, J. and Veselov, A. P. (1991). Discrete versions of some classical integrable systems and factorization of matrix polynomials. *Communications in Mathematical Physics*, 139:217–243.

Moshfeghifar, F., Gholamalizadeh, T., Ferguson, Z., Schneider, T., Nielsen, M. B., Panozzo, D., Darkner, S., and Erleben, K. (2022). Libhip: An open-access hip joint model repository suitable for finite element method simulation. *Computer Methods and Programs in Biomedicine*, 226:107140.

Mosler, J. and Ortiz, M. (2007). Variational h-adaption in finite deformation elasticity and plasticity. *International Journal for Numerical Methods in Engineering*, 72(5):505–523.

Müller, M., Chentanez, N., Kim, T.-Y., and Macklin, M. (2015). Air meshes for robust collision handling. *ACM Transactions on Graphics*, 34(4):1–9.

Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118.

Müller, M., Macklin, M., Chentanez, N., Jeschke, S., and Kim, T.-Y. (2020). Detailed rigid body simulation with extended position based dynamics. *Computer Graphics Forum*, 39(8):101–112.

Narain, R., Pfaff, T., and O'Brien, J. F. (2013). Folding and crumpling adaptive sheets. *ACM Transactions on Graphics*, 32(4).

Narain, R., Samii, A., and O'Brien, J. F. (2012). Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics*, 31(6).

Nelson, D. D. and Cohen, E. (1998). User interaction with cad models with nonholonomic parametric surface constraints. In *ASME International Mechanical Engineering Congress and Exposition*, volume 15861, pages 235–242. American Society of Mechanical Engineers.

Nelson, D. D., Johnson, D. E., and Cohen, E. (2005). Haptic rendering of surface-to-surface sculpted model interaction. In *ACM SIGGRAPH 2005 Courses*, pages 97–es, New York, NY, USA. Association for Computing Machinery.

Nitsche, J. C. C. (1971). Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36:9–15.

Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media, New York, NY, USA, second edition.

Nvidia Corporation (2004–2021). Physx. https://developer.nvidia.com/physx-sdk.

O'Brien, J. F., Bargteil, A. W., and Hodgins, J. K. (2002). Graphical modeling and animation of ductile fracture. *ACM Transactions on Graphics*, 21(3):291–294.

O'Brien, J. F. and Hodgins, J. K. (1999). Graphical modeling and animation of brittle fracture. In *Proceedings of ACM SIGGRAPH 1999*, pages 137–146. ACM Press/Addison-Wesley Publishing Co.

Oden, J. T. (1994). Optimal h-p finite element methods. *Computer Methods in Applied Mechanics and Engineering*, 112(1):309–331.

Ong, M. E. G. (1994). Uniform refinement of a tetrahedron. *SIAM Journal on Scientific Computing*, 15(5):1134–1144.

Ortiz, M. and Stainier, L. (1999). The variational formulation of viscoplastic constitutive updates. *Computer Methods in Applied Mechanics and Engineering*, 171(3-4):419–444.

Otaduy, M., Tamstorf, R., Steinemann, D., and Gross, M. (2009). Implicit contact handling for deformable objects. *Computer Graphics Forum*, 28:559–568.

Owren, B. and Welfert, B. (2000). The newton iteration on lie groups. *BIT Numerical Mathematics*, 40:121–145.

Pabst, S., Koch, A., and Straßer, W. (2010). Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. *Computer Graphics Forum*, 29:1605–1612.

Pagano, S. and Alart, P. (2008). Self-contact and fictitious domain using a difference convex approach. *International Journal for Numerical Methods in Engineering*, 75:29–42.

Pan, J., Zhang, L., and Manocha, D. (2012). Collision-free and smooth trajectory computation in cluttered environments. *The International Journal of Robotics Research*, 31(10):1155–1175.

Pandolfi, A., Kane, C., Marsden, J. E., and Ortiz, M. (2002). Time-discretized variational formulation of non-smooth frictional contact. *International Journal for Numerical Methods in Engineering*, 53(8):1801–1829.

Panetta, J., Zhou, Q., Malomo, L., Pietroni, N., Cignoni, P., and Zorin, D. (2015). Elastic textures for additive fabrication. *ACM Transactions on Graphics*, 34(4):135:1–135:12.

Pelteret, J.-P. (2016). The 'quasi-static finite-strain compressible elasticity' code gallery program. https://dealii.org/developer/doxygen/deal.II/code_gallery_Quasi_static_Finite_strain_Compressible_Elasticity.html. Accessed: 2023-04-24.

Pfaff, T., Narain, R., de Joya, J. M., and O'Brien, J. F. (2014). Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics*, 33(4).

Powell, D. (2021). Polyclipper. https://github.com/LLNL/PolyClipper.

Poya, R., Gil, A. J., Ortigosa, R., Sevilla, R., Bonet, J., and Wall, W. A. (2018). A curvilinear high order finite element framework for electromechanics: From linearised electro-elasticity to massively deformable dielectric elastomers. *Computer Methods in Applied Mechanics and Engineering*, 329:75–117.

Prévost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. (2013). Make It Stand: Balancing shapes for 3D fabrication. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 32(4):81:1–81:10.

Provot, X. (1997). Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation*, pages 177–189. Springer.

Puso, M. A. and Laursen, T. A. (2004). A mortar segment-to-segment contact method for large deformation solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 193(6-8):601–629.

Redon, S., Kheddar, A., and Coquillart, S. (2002a). Fast continuous collision detection between rigid bodies. *Computer Graphics Forum*, 21(3):279–287.

Redon, S., Kheddar, A., and Coquillart, S. (2002b). Gauss' least constraints principle and rigid body simulations. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 517–522.

Rodrigues (1840). Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. *Journal de Mathématiques Pures et Appliquées*, pages 380–440.

Schling, B. (2011). The boost c++ libraries.

Schmidt, A. and Siebert, K. G. (2000). A posteriori estimators for the h − p version of the finite element method in 1d. *Applied Numerical Mathematics*, 35(1):43–66.

Schneider, T., Dumas, J., Gao, X., Botsch, M., Panozzo, D., and Zorin, D. (2019a). Poly-spline finite-element method. *ACM Transactions on Graphics*, 38(3).

Schneider, T., Dumas, J., Gao, X., Zorin, D., and Panozzo, D. (2019b). PolyFEM. https://polyfem.github.io/.

Schneider, T., Hu, Y., Dumas, J., Gao, X., Panozzo, D., and Zorin, D. (2018). Decoupling simulation accuracy from mesh quality. *ACM Transactions on Graphics*, 37(6).

Schneider, T., Hu, Y., Gao, X., Dumas, J., Zorin, D., and Panozzo, D. (2022). A large-scale comparison of tetrahedral and hexahedral elements for solving elliptic pdes with the finite element method. *ACM Transactions on Graphics*, 41(3).

Schneider, T., Panozzo, D., and Zhou, X. (2021). Isogeometric high order mesh generation. *Computer Methods in Applied Mechanics and Engineering*, 386:114104.

Schüller, C., Kavan, L., Panozzo, D., and Sorkine-Hornung, O. (2013). Locally injective mappings. *Computer Graphics Forum (Proceedings of Eurographics/ACM SIGGRAPH Symposium on Geometry Processing)*, 32(5):125–135.

Schvartzman, S., Pérez, Á., and Otaduy, M. (2010). Star-contours for efficient hierarchical self-collision detection. *ACM Transactions on Graphics*, 29:80:1–80:8.

Schwarzer, F., Saha, M., and Latombe, J.-C. (2005). Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Transactions on Robotics*, 21:338–353.

Seitz, A., Farah, P., Kremheller, J., Wohlmuth, B. I., Wall, W. A., and Popp, A. (2016). Isogeometric dual mortar methods for computational contact mechanics. *Computer Methods in Applied Mechanics and Engineering*, 301:259–280.

Sevilla, R., Fernández-Méndez, S., and Huerta, A. (2011). Comparison of high-order curved finite elements. *International Journal for Numerical Methods in Engineering*, 87(8):719–734.

SideFX (1996–2023). Houdini. https://www.sidefx.com/products/houdini/.

Sifakis, E., Marino, S., and Teran, J. (2008). Globally coupled collision handling using volume preserving impulses. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA 2008, pages 147–153.

Simnett, T. J. R., Laycock, S. D., and Day, A. M. (2009). An Edge-based Approach to Adaptively Refining a Mesh for Cloth Deformation. In Tang, W. and Collomosse, J., editors, *Theory and Practice of Computer Graphics*. The Eurographics Association.

Simó, J. C. and Hughes, T. J. R. (1998). *Computational Inelasticity*. Springer.

Skouras, M., Thomaszewski, B., Kaufmann, P., Garg, A., Bickel, B., Grinspun, E., and Gross, M. (2014). Designing inflatable structures. *ACM Transactions on Graphics*, 33(4).

Smith, B., Kaufman, D. M., Vouga, E., Tamstorf, R., and Grinspun, E. (2012). Reflections on simultaneous impact. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31(4):106:1–106:12.

Smith, J. and Schaefer, S. (2015). Bijective parameterization with free boundaries. *ACM Transactions on Graphics*, 34(4):1–9.

Smith, M. (2009–2023). *ABAQUS/Standard User's Manual, Version 6.9*. Dassault Systèmes Simulia Corp, United States.

Snyder, J. M. (1992). Interval analysis for computer graphics. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, volume 26 of *SIGGRAPH 1992*, pages 121–130, New York, NY, USA. Association for Computing Machinery.

Snyder, J. M., Woodbury, A. R., Fleischer, K., Currin, B., and Barr, A. H. (1993). Interval methods for multi-point collisions between time-dependent curved surfaces. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1993, pages 321–334, New York, NY, USA. Association for Computing Machinery.

Spillmann, J. and Teschner, M. (2008). An adaptive contact model for the robust simulation of knots. *Computer Graphics Forum*, 27(2):497–506.

Stam, J. (2009). Nucleus: Towards a unified dynamics solver for computer graphics. In *Proceedings of IEEE International Conference on Computer-Aided Design and Computer Graphics*, pages 1–11.

Stein, K., Tezduyar, T. E., and Benney, R. (2004). Automatic mesh update with the solid-extension mesh moving technique. *Computer Methods in Applied Mechanics and Engineering*, 193(21):2019–2032. Flow Simulation and Modeling.

Stenberg, R. (1998). Mortaring by a method of J. A. Nitsche. *Computational Mechanics*.

Stewart, D. (2000). Rigid-body dynamics with friction and impact. *SIAM Review*, 42:3–39.

Stewart, D. and Trinkle, J. (2000). An implicit time-stepping scheme for rigid body dynamics with coulomb friction. *Proceedings of IEEE International Conference on Robotics and Automation*, 1:162–169.

Stewart, D. E. (2001). Finite-dimensional contact mechanics. *Philosophical Transactions of the Royal Society A*, 359:2467–2482.

Stuart, A. and Humphries, A. R. (1996). *Dynamical Systems and Numerical Analysis*. Cambridge University Press.

Suwelack, S., Lukarski, D., Heuveline, V., Dillmann, R., and Speidel, S. (2013). Accurate surface embedding for higher order finite elements. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '13, pages 187–192, New York, NY, USA. Association for Computing Machinery.

Tang, M., Curtis, S., Yoon, S.-e., and Manocha, D. (2009a). ICCD: Interactive continuous collision detection between deformable models using connectivity-based culling. *IEEE Transactions on Visualization and Computer Graphics*, 15:544–57.

Tang, M., Kim, Y. J., and Manocha, D. (2009b). C^2A: Controlled conservative advancement for continuous collision detection of polygonal models. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 849–854.

Tang, M., Kim, Y. J., and Manocha, D. (2011a). *CCQ: Efficient Local Planning Using Connection Collision Query*, pages 229–247. Springer Berlin Heidelberg, Berlin, Heidelberg.

Tang, M., Manocha, D., and Tong, R. (2010). Fast continuous collision detection using deforming non-penetration filters. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 7–13.

Tang, M., Manocha, D., Yoon, S.-e., du, P., Heo, J.-P., and Tong, R. (2011b). VolCCD: Fast continuous collision culling between deforming volume meshes. *ACM Transactions on Graphics*, 30:111:1–111:15.

Tang, M., Tong, R., Wang, Z., and Manocha, D. (2014). Fast and exact continuous collision detection with bernstein sign classification. *ACM Transactions on Graphics*, 33(6):186:1–186:8.

Tang, M., Yoon, S.-e., and Manocha, D. (2008). Adjacency-based culling for continuous collision detection. *The Visual Computer*, 24:545–553.

Tasora, A., Serban, R., Mazhar, H., Pazouki, A., Melanz, D., Fleischmann, J., Taylor, M., Sugiyama, H., and Negrut, D. (2016). Chrono: An open source multi-physics dynamics engine. In *High Performance Computing in Science and Engineering*, pages 19–49. Springer.

Temizer, I., Wriggers, P., and Hughes, T. (2011). Contact treatment in isogeometric analysis with nurbs. *Computer Methods in Applied Mechanics and Engineering*, 200(9):1100–1112.

Teran, J., Sifakis, E., Irving, G., and Fedkiw, R. (2005). Robust quasistatic finite elements and flesh simulation. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 181–190. ACM.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically deformable models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 205–214, New York, NY, USA. Association for Computing Machinery.

Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Tonge, R., Benevolenski, F., and Voroshilov, A. (2012). Mass splitting for jitter-free parallel rigid body simulation. *ACM Transactions on Graphics*, pages 1–8.

Trinkle, J., Pang, J.-S., Sudarsky, S., and Lo, G. (1995). On dynamic multi-rigid-body contact problems with Coulomb friction. Technical report, Texas A&M University, Department of Computer Science.

Tucker, W. (2011). *Validated Numerics: A Short Introduction to Rigorous Computations*. Princeton University Press, USA.

van Waveren, J. (2005). Robust continuous collision detection between arbitrary polyhedra using trajectory parameterization of polyhedral features. Technical report, id Software, Inc.

Vavourakis, V., Loukidis, D., Charmpis, D. C., and Papanastasiou, P. (2013). Assessment of remeshing and remapping strategies for large deformation elastoplastic finite element analysis. *Comput. Struct.*, 114–115:133–146.

Verschoor, M. and Jalba, A. C. (2019). Efficient and accurate collision response for elastically deformable models. *ACM Transactions on Graphics*, 38(2).

Villard, J. and Borouchaki, H. (2005). Adaptive meshing for cloth animation. *Engineering with Computers*, 20(4):333–341.

Volino, P. and Thalmann, N. M. (1994). Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum*, 13(3):155–166.

Von Herzen, B., Barr, A. H., and Zatz, H. R. (1990). Geometric collisions for time-dependent parametric surfaces. *Computer Graphics (Proceedings of SIGGRAPH)*, 24(4):39–48.

Vouga, E., Harmon, D., Tamstorf, R., and Grinspun, E. (2011). Asynchronous variational contact mechanics. *Computer Methods in Applied Mechanics and Engineering*, 200(25):2181–2194. https://github.com/evouga/collisiondetection.

Vouga, E., Smith, B., Kaufman, D. M., Tamstorf, R., and Grinspun, E. (2017). All's well that ends well: Guaranteed resolution of simultaneous rigid body impact. *ACM Transactions on Graphics*, 36(4).

Wang, B., Faure, F., and Pai, D. K. (2012). Adaptive image-based intersection volume. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31(4):1–9.

Wang, B., Ferguson, Z., Schneider, T., Jiang, X., Attene, M., and Panozzo, D. (2021). A large scale benchmark and an inclusion-based algorithm for continuous collision detection. *ACM Transactions on Graphics*, 40(5).

Wang, H. (2014). Defending continuous collision detection against errors. *ACM Transactions on Graphics*, 33(4):1–10.

Wang, Z., Tang, M., Tong, R., and Manocha, D. (2015). TightCCD: Efficient and robust continuous collision detection using tight error bounds. *Computer Graphics Forum*, 34:289–298.

Wicke, M., Ritchie, D., Klingner, B. M., Burke, S., Shewchuk, J. R., and O'Brien, J. F. (2010). Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics*, 29(4).

Witkin, A. and Baraff, D. (2001). Physically based modeling. In *SIGGRAPH 2001 Course Notes*, New York, NY, USA. Association for Computing Machinery.

Wojtan, C., Thürey, N., Gross, M., and Turk, G. (2009). Deforming meshes that split and merge. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, New York, NY, USA. Association for Computing Machinery.

Wolfram Research Inc. (2020). *Mathematica 12.0*. Wolfram Research Inc. http://www.wolfram.com.

Wong, W. S.-K. and Baciu, G. (2006). A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. In *Proceedings of the 2006 ACM International*

Conference on Virtual Reality Continuum and Its Applications, VRCIA '06, pages 181–188, New York, NY, USA. Association for Computing Machinery.

Wriggers, P. (1995). Finite element algorithms for contact problems. *Archives of Computational Methods in Engineering*, 2:1–49.

Wriggers, P., Schröder, J., and Schwarz, A. (2013). A finite element method for contact using a third medium. *Computational Mechanics*, 52(4):837–847.

Xu, H., Zhao, Y., and Barbič, J. (2014). Implicit multibody penalty-based distributed contact. *IEEE Transactions on Visualization and Computer Graphics*, 20(9).

Yang, B. and Laursen, T. A. (2008). A large deformation mortar formulation of self contact with finite sliding. *Computer Methods in Applied Mechanics and Engineering*, 197(6):756–772.

Zhang, E., Mischaikow, K., and Turk, G. (2005). Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics*, 24(1):1–27.

Zhang, J. E., Dumas, J., Fei, Y. R., Jacobson, A., James, D. L., and Kaufman, D. M. (2022). Progressive simulation for cloth quasistatics. *ACM Transactions on Graphics*, 41(6).

Zhang, L., Kim, Y. J., and Manocha, D. (2007a). C-DIST: Efficient distance computation for rigid and articulated models in configuration space. In *Proceedings of ACM Symposium on Solid and Physical Modeling*, SPM '07, pages 159–169, New York, NY, USA. Association for Computing Machinery.

Zhang, L., Kim, Y. J., Varadhan, G., and Manocha, D. (2007b). Generalized penetration depth computation. *Computer-Aided Design*, 39(8):625–638.

Zhang, X., Lee, M., and Kim, Y. J. (2006). Interactive continuous collision detection for non-convex polyhedra. *The Visual Computer*, 22(9-11):749–760.

Zhang, X., Redon, S., Lee, M., and Kim, Y. J. (2007c). Continuous collision detection for articulated models using taylor models and temporal culling. *ACM Transactions on Graphics*, 26(3):15–es.

Zhao, W., Queralta, J. P., and Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE.

Zheng, C. and James, D. L. (2012). Energy-based self-collision culling for arbitrary mesh deformations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31(4):1–12.

Zielonka, M. G., Ortiz, M., and Marsden, J. E. (2008). Variational r-adaption in elastodynamics. *International Journal for Numerical Methods in Engineering*, 74(7):1162–1197.

Zong, Z., Li, X., Ye, J., Wen, S., Yang, Y., Kaufman, D. M., Li, M., and Jiang, C. (2022). Topology optimization with frictional self-contact.

Zulian, P., Kopaničáková, A., Nestola, M. C. G., Fink, A., Fadel, N., Rigazzi, A., Magri, V., Schneider, T., Botter, E., Mankau, J., and Krause, R. (2016). Utopia: A C++ embedded domain specific language for scientific computing. https://bitbucket.org/zulianp/utopia.