Intersection-free Rigid Body Dynamics

ZACHARY FERGUSON, New York University
MINCHEN LI, University of California, Los Angeles and University of Pennsylvania
TESEO SCHNEIDER, New York University and University of Victoria
FRANCISCA GIL-URETA, New York University
TIMOTHY LANGLOIS, Adobe Research
CHENFANFU JIANG, University of California, Los Angeles and University of Pennsylvania
DENIS ZORIN, New York University
DANNY M. KAUFMAN, Adobe Research
DANIELE PANOZZO, New York University

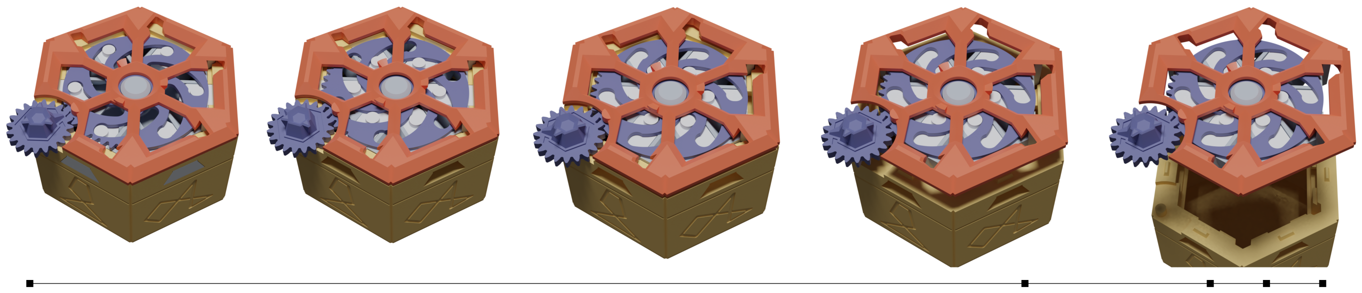

Fig. 1. **Expanding Lock Box.** An intricate locking mechanism designed for 3D printing can be directly simulated with our algorithm. As the “key” turns, the central spiral is rotated which in turn pulls in each of the five locking pins. When all pins have been retracted the bottom is able to freely fall. Our algorithm’s intersection-free guarantee enables the automatic testing of designs without the need to tune simulation parameters. ©Angus Deveson.

We introduce the first implicit time-stepping algorithm for rigid body dynamics, with contact and friction, that guarantees intersection-free configurations at every time step.

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 minimal separation. For time integration and contact response, we extend the recently proposed incremental potential contact framework to reduced coordinates and rigid body dynamics.

We introduce 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.

Authors’ addresses: Zachary Ferguson, New York University, zfergus@nyu.edu; Minchen Li, University of California, Los Angeles, University of Pennsylvania; Teseo Schneider, New York University, University of Victoria; Francisca Gil-Ureta, New York University; Timothy Langlois, Adobe Research; Chenfanfu Jiang, University of California, Los Angeles, University of Pennsylvania; Denis Zorin, New York University; Danny M. Kaufman, Adobe Research; Daniele Panozzo, New York University, panozzo@nyu.edu.

0730-0301/2021/8-ART183 $15.00
https://doi.org/10.1145/3450626.3459802

CCS Concepts: • **Computing methodologies** → **Physical simulation**; *Collision detection.*

Additional Key Words and Phrases: Rigid Body Simulation, Contact Mechanics, Continuous Collision Detection

ACM Reference Format:
Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. 2021. Intersection-free Rigid Body Dynamics. *ACM Trans. Graph.* 40, 4, Article 183 (August 2021), 16 pages. https://doi.org/10.1145/3450626.3459802

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 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 recently proposed IPC formulation [Li et al. 2020] for large deformation dynamics to rigid body dynamics. We rely on the same core ideas: model contacts via a set of barrier functions, 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 interlinked 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 minimal separation. For the broad phase, we propose to use interval arithmetic to compute conservative bounding boxes that can be used in a standard 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 2020]).

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.

2 RELATED WORK

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].

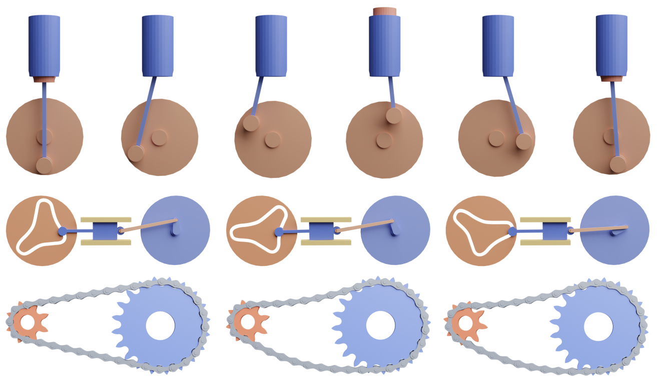
Fig. 2. **Mechanisms.** We demonstrate the robustness of our method on various mechanisms with tight conforming contact. Top: a piston is attached to a rotating disk and a static cylinder is used to constrain the motion of the piston. Middle: A wheel with complex geometry rotates smoothly, but results 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. ©Okan (bike chain), Hampus Andersson (sprocket) under CC BY.

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 2.2. The second being that because rigid bodies are infinitely stiff, applied forces and contact 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 [Baraff 1991] rigid body contact has especially focused on linearized complementarity programming (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 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 2.2 and Appendix B. 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.

2.2 Collision Detection

We restrict our overview to continuous collision detection (CCD) algorithms for curved trajectories, as we are interested in rigid motions, and to CCD algorithms for linear trajectories with minimal separation, as our algorithm needs to tackle this subproblem. We refer to Wang et al. [2020] for an overview of CCD methods for linear trajectories without minimal separation.

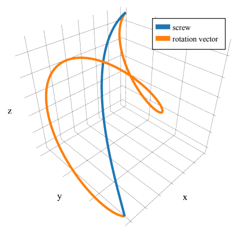

Fig. 3. Trajectories of interpolating rotation vectors can be wildly different form the traditional screw motion used by others.

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. 2020]. 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 B.

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. 2009], screw motion [Tang et al. 2011]. 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, 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, 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 4.3 and provide a comparison with our technique in Appendix B.

Minimal Separation Linear CCD.. Linear CCD with minimal separation [Harmon et al. 2011; Lu et al. 2019; Provot 1997; Stam 2009; Wang et al. 2020] 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 minimal separation. While any of the methods above could be used, we opt for [Wang et al. 2020], 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 minimal separation (Section 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.

3 IPC OVERVIEW

We briefly overview the Incremental Potential Contact solver introduced in [Li et al. 2020] to make our paper self-contained.

Li et al. [2020] 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:

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

where x^t is the set of nodal position, v^t the velocities, $E_d(x, x^t, v^t)$ is an Incremental Potential (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 Li et al. [2020].

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 (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:

$$B(x, \hat{d}) = \kappa \sum_{k \in C} b\big(d_k(x), \hat{d}\big), \tag{2}$$

where κ is the barrier stiffness, d_k is the mollified unsigned distance between the k pair of primitives (we refer to Li et al. [2020] 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

$$b(d, \hat{d}) = \begin{cases} -(d - \hat{d})^2 \ln\left(\frac{d}{\hat{d}}\right), & 0 < d < \hat{d} \\ 0 & d \geq \hat{d}. \end{cases} \tag{3}$$

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 (3) as the support of the barrier is local.

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

$$\phi_i(\boldsymbol{\theta}_i, q_i) = \mathcal{R}(\boldsymbol{\theta}_i)\hat{X}_i + q_i, \tag{4}$$

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 (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 Li et al. [2020] (briefly summarized in Section 3 above). Our first step requires us to formulate rigid body system time integrators as incremental potentials (IP) – these are not previously available. With rigid body IP in hand, we then can follow Li et al. [2020] 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 4.1) and then describe how we adapt line search, constraint set generation, and a Newton-type solver to the rigid body time step problem. As

[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.

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 query in Section 4.3. We provide an extensive comparison of our rigid body formulation and the original formulation in [Li et al. 2020] Section 6.1.

4.1 Rigid Body Incremental Potential

Following Li et al. [2020], 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 $\frac{1}{2}\operatorname{tr}(\dot{Q}_i J_i \dot{Q}_i^T)$ where

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

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 $Q_i^T Q_i - \mathrm{Id} = 0$ 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

$$Q_i^{t+1} = Q_i^t + h\dot{Q}_i^t - h^2 \nabla V(Q_i^{t+1}) J_i^{-1} + Q_i^{t+1}\Lambda J_i^{-1} + h^2[\tau_i]J_i^{-1}, \quad (5)$$

$${Q_i^{t+1}}^T Q_i^{t+1} - \mathrm{Id} = 0, \quad (6)$$

$$\dot{Q}_i^t = \frac{Q_i^t - Q_i^{t-1}}{h}, \quad (7)$$

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

$$\begin{aligned} \tilde{Q_i}^t &= Q_i^t + h\dot{Q}_i^t + h^2[\tau_i]J_i^{-1} \\ Q_i^{t+1} &= \operatorname*{argmin}_{Q} \tfrac{1}{2}\operatorname{tr}\left(QJ_iQ^T\right) + \operatorname{tr}\left(QJ_i(\tilde{Q}_i^t)^T\right) + h^2 V(Q), \\ &\text{s.t. } Q^TQ - \mathrm{Id} = 0. \end{aligned} \quad (8)$$

2

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

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

$$E_Q(Q) = \sum_{i=1}^{n} \left(\tfrac{1}{2}\operatorname{tr}(Q_iJ_iQ_i^T) - \operatorname{tr}(Q_iJ_i(\tilde{Q_i}^t)^T)\right), \quad (9)$$

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

$$\begin{aligned} \tilde{q_i}^t &= q_i^t + h\dot{q}_i^t + h^2(g + m^{-1}f_i) \\ E_q(q) &= \sum_{i=1}^{n} \left(\tfrac{1}{2}m_i q_i^T q_i - m_i q_i^T \tilde{q}_i^t\right), \end{aligned} \quad (10)$$

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

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

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 (1), when we wish to apply rigid body coordinates. This gives us the following *constrained* optimization problem to solve

$$(Q^{t+1}, q^{t+1}) = \operatorname*{argmin}_{Q,q} E(Q,q) + B(\phi(Q,q),\hat{d}) + D(\phi(Q,q)) \quad (11)$$

$$\text{s.t. } Q_i^TQ_i = \mathrm{Id},\ i = \{1,\ldots,n\}, \quad (12)$$

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

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

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 E.

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 [Li et al. 2020], 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

$$\mathcal{R}(\boldsymbol{\theta}) = \mathrm{Id} + \sin(\|\boldsymbol{\theta}\|)\left[\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}\right] + (1-\cos(\|\boldsymbol{\theta}\|))\left[\frac{\boldsymbol{\theta}}{\|\boldsymbol{\theta}\|}\right]^2, \quad (13)$$

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

$$\mathcal{R}(\boldsymbol{\theta}) = \mathrm{Id} + \mathrm{sinc}(\|\boldsymbol{\theta}\|)[\boldsymbol{\theta}] + 2\,\mathrm{sinc}^2\left(\frac{\|\boldsymbol{\theta}\|}{2}\right)[\boldsymbol{\theta}]^2, \quad (14)$$

where

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

Note, we compute values close to zero computed using a Taylor series expansion (see Appendix A.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 $\mathrm{sinc}(\|\boldsymbol{\theta}\|)$ is provided in Appendix A.2.

Additionally, when computing $\mathrm{sinc}(x)$ with interval arithmetic a naïve implementation using interval division can result in intervals far outside the range of $\mathrm{sinc}(x)$ (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 A.3.

Stabilization. Because of our transformation from axis-angle to rotation matrix, the Hessian $H(E_Q(Q))$ may not be positive semi-definite (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 (2)). In [Li et al. 2020], 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:

$$s_{ba} = \mathcal{R}(\boldsymbol{\theta}_b)^T(\mathcal{R}(\boldsymbol{\theta}_a)\hat{X}_a + q_a - q_b). \quad (15)$$

We can thus build a bounding volume hierarchy (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 (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 (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.

4.3 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

$$\phi_{a_i}(t) = \mathcal{R}(\boldsymbol{\theta}_i(t))a_i + q_i(t), \quad t \in [0,1]. \quad (16)$$

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 (15) to the time dependent case,

$$s_{ba}(t) = \mathcal{R}(\boldsymbol{\theta}_b(t))^T(\mathcal{R}(\boldsymbol{\theta}_a(t))\hat{X}_a + q_a(t) - q_b(t)). \quad (17)$$

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) intersect.

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

$$F_{\mathrm{vf}}(t,\alpha,\beta) = \boldsymbol{p}(t) - \big((1-\alpha-\beta)\boldsymbol{p}_1(t) + \alpha\boldsymbol{p}_2(t) + \beta\boldsymbol{p}_3(t)\big), \quad (18)$$

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 $\boldsymbol{p}_3, \boldsymbol{p}_4$

$$F_{\text{ee}}(t, \alpha, \beta) = \big((1-\alpha)\boldsymbol{p}_1(t) + \alpha \boldsymbol{p}_2(t)\big) - \big((1-\beta)\boldsymbol{p}_3(t) + \beta \boldsymbol{p}_4(t)\big) \quad (19)$$

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 (18) or (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 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. 2020] 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 B.

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 minimal separation of b. Let us consider the curved trajectory $\phi_{a_i}(t)$ (16) of a single vertex $a_i \in \hat{X}_i$. The time-dependent distance between the curved trajectory and the linear approximation is:

$$e_{a_i}(t) = \|\phi_{a_i}(t) - ((1-t)\boldsymbol{p}^0 + t\boldsymbol{p}^1)\|, \quad t \in [0,1]. \quad (20)$$

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 minimal separation CCD proposed by Wang et al. [2020], using the L^∞ metric for minimal separation. This idea is used in Algorithm 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 23). 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 5), we compute the initial distance between the two objects (Line 6) and an upper bound on the error of the linear approximation of the trajectory (Line 7). If the bound is larger than the initial distance (Line 8) 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 8) 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 C). To bound the cost of the linear CCD and prevent overrefinement, we set an upper bound $N^{\max}$ on the maximal number of subdivisions (we use 1000 in our experiments). The bound is however disabled when $t_0 = 0$, as we need to have a strictly positive time of impact (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 minimal separation, we use [Wang et al. 2020] with default parameters.

Algorithm 1 contact, toi = CurvedCCD($a_i, b_j, \delta, N^{\max}$)

```
1: $t_0 \leftarrow 0$
2: $ts \leftarrow \{1\}$
3: $N \leftarrow 1$
4: while $ts \neq \emptyset$ do
5:     $t_1 \leftarrow \text{top}(ts)$
6:     $d_{t_0} \leftarrow d(t_0, a_i, b_j)$ {$d$ is defined in [Li et al. 2020, (18)-(19)]}
7:     $b \leftarrow e_{a_i}([t_0, t_1]) + e_{b_j}([t_0, t_1])$
8:     if $b \geq \delta d_{t_0}$ and ($N < N^{\max}$ or $t_0 = 0$) then
9:         $ts \leftarrow ts \cup \{(t_1 + t_0)/2)\}$
10:        $N \leftarrow N + 1$
11:        continue
12:    end if
13:    impact, toi $\leftarrow$ lccd($\phi_{a_i}(t_0), \phi_{a_i}(t_1), \phi_{b_j}(t_0), \phi_{b_j}(t_1), b$)
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:    end if
19:    if impact then
20:        return true, $t_0 + \text{toi}(t_1 - t_0)$
21:    end if
22:    pop($ts$)
23:    $t_0 \leftarrow t_1$
24: end while
25: return false, $\infty$
```

Shared Earliest Time of Impact. As in [Li et al. 2020], 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 2 with $ts \leftarrow t^{\text{earliest}}$,

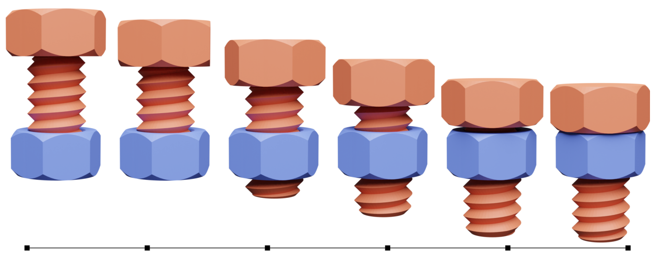
Fig. 4. **Bolt.** A bolt spins inside a static nut under gravity. Without friction, the bolt is quickly able to follow the threading and begins to rotate. ©YSoft be3D under CC BY-SA 3.0.

where t^{earliest} is the earliest time-of-impact for the current step (initially 1).

Minimal Separation. To extend our algorithm to guarantee a minimal separation we make three minor modifications to our formulation. First, we shift the input to our distance barrier (Equation (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 $b \leftarrow b + d_{\min}$).

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 of Li et al. [2020] to enforce Dirichlet boundary conditions. For each kinematic rigid body k, we construct the AL from the two terms,

$$\begin{aligned} E_{\text{A,q}}(q) &= \frac{\kappa_{\text{A},q}}{2} m_k \left\| q_k - \hat{q}_k^{t+1} \right\|^2 - \sqrt{m_k} \lambda_{\text{A},k}^T (q_k - \hat{q}_k^{t+1}) \\ E_{\text{A,Q}}(Q) &= \frac{\kappa_{\text{A},Q}}{2} \operatorname{tr}(Q_k - \hat{Q}_k^{t+1}) J_k (Q_k - \hat{Q}_k^{t+1})^T) \\ &\quad - \operatorname{tr}(\Lambda_{\text{A},k}^T (Q_k - \hat{Q}_k^{t+1}) J_k^{\frac{1}{2}}) \end{aligned}$$

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

Following the algorithm of Li et al. [2020], we initialize the Lagrange multipliers to $\lambda_{\text{A},k} = 0$ and $\Lambda_{\text{A},k} = 0$ and penalty stiffnesses to $\kappa_{\text{A},q} = 10^3$ and $\kappa_{\text{A},Q} = 10^3$. These potentials are then added to Equation (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 $\eta_Q > 0.999$ [Li et al. 2020].

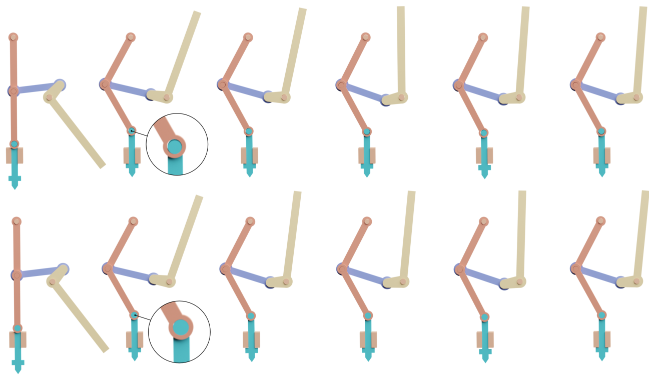
Fig. 5. **Punching Press.** We designed two variations of a punching press mechanism: one with loose joints (top row) and one with tight (bottom row). By applying a force to raise the punch, our use of full rigid DOF instead of articulated bodies allows us to model and test varying tolerance in joints.

If only stationarity is satisfied, we update the AL parameters. For brevity we only describe the update scheme for $\kappa_{\text{A},q}$, λ_{A} as the others follow closely. If $\eta_{\text{A},q} < 0.99$ and $\kappa_{\text{A}} < 10^8$, then

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

Otherwise, for each kinematic body k,

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

Additionally, whenever the AL convergence criteria are satisfied, we fix all prescribed DOF and remove the AL from Equation (12) for the remainder of the optimization. This helps by removing unnecessary stiffness in our objective function [Li et al. 2020].

5 RESULTS

Our algorithm is implemented in C++ and uses Eigen [Guennebaud et al. 2010] 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 6 and to a volumetric IPC formulation in Section 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).

We show a collection of more complex mechanisms in Figure 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

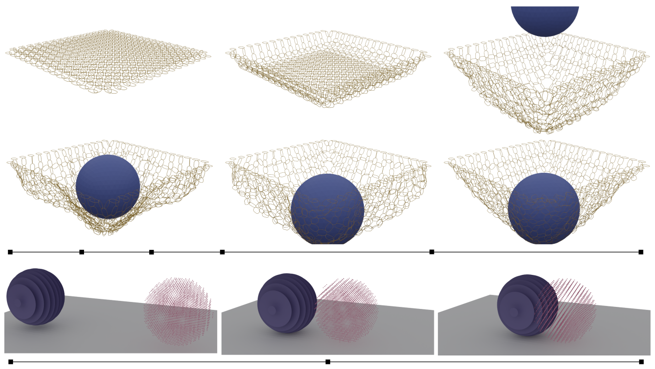
Fig. 6. **Codimensional bodies.** The IPC formulation allows us to easily simulate codimensional objects. Top: A ball is dropped onto a chain net composed of 1D codimensional edges. Bottom: A sphere of disconnected codimensional planes and a point cloud ball roll into each other. Upon contact, the geometry locks, and both spheres rock back and forth before coming to rest.

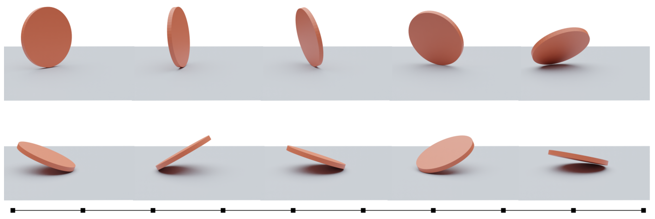
Fig. 7. **Spolling coin.** A coin spolls (spins while rolling) on a surface with friction ($\mu = 0.2$). As the coin falls it continues to rotate while only a single point touches the ground. To accurately capture these high-speed dynamics, we use a small timestep of $h = 10^{-4}$ s.

example by adding additional tolerance in the holes of a punching press (Figure 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, and directly use the STL files in our simulator (Figure 1). The mechanisms can be studied in our simulation, where it is easy to modify the 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 10. 1D co-dimensional objects are also supported and can be used, for example, to efficiently simulate a large chain net (Figure 6 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 6 Bottom).

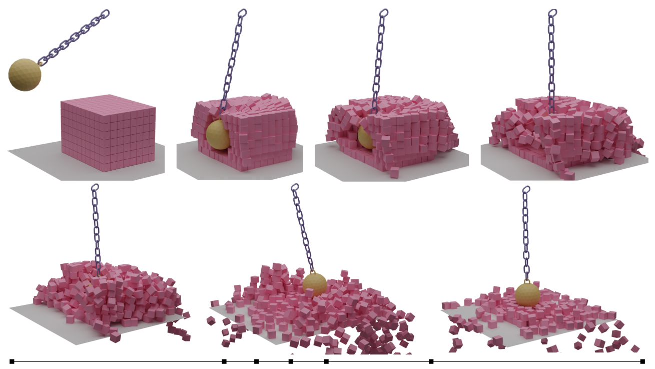
Fig. 8. **Wrecking ball.** A stack of 560 boxes is hit by a wrecking ball made from a chain of interlinked bodies.

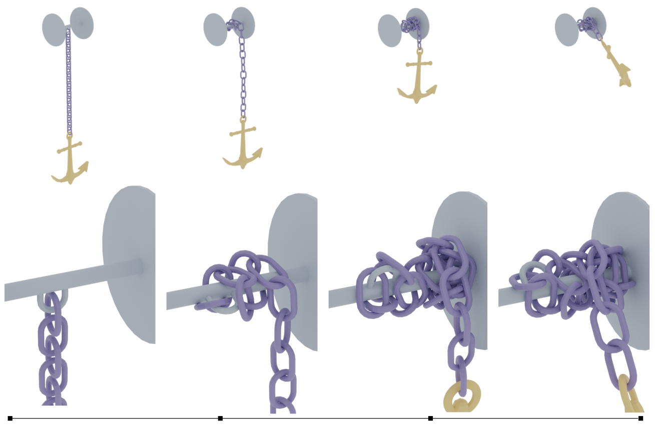
Fig. 9. **Anchor.** A heavy anchor attached to a chain briefly falls under gravity before being lifted by rolling the chain around an axle. Natural bunching and kinking behaviors are visible. ©Animation Anchor Line (anchor) under TurboSquid 3D Model License.

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 7), 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 8).

We can also stably simulate long chains of interlinked bodies. We show an example in Figure 9, where a heavy anchor is lifted by rolling up a chain composed of 21 individual links.

Friction. We repeat the arch scene experiment used to benchmark the friction model in [Li et al. 2020], replacing the deformable yet stiff blocks with rigid objects (Figure 11). The results are indistinguishable (see also Figure 19).

The Lewis is an interesting mechanism used to lift heavy bodies, relying on static friction (Figure 12). As a final friction experiment, we place a box on a spinning disk with four different coefficients

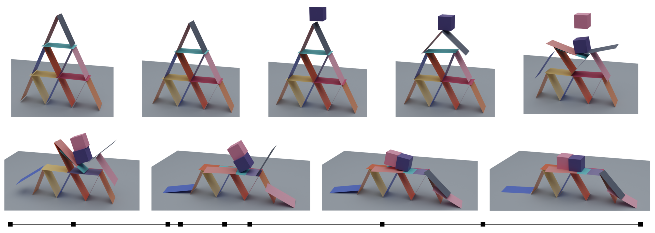
Fig. 10. **Codimensional card house.** We design a codimensional variant of the standard frictional benchmark of Kaufman et al. [2008], where each card is composed of only two triangles. The cards are briefly allowed to stably come to rest ($\mu = 0.9$), before being impacted by two cubes. The top two levels collapse, but the final floor is able to catch the cubes demonstrating our ability to quickly handle transitions between static and dynamic friction.

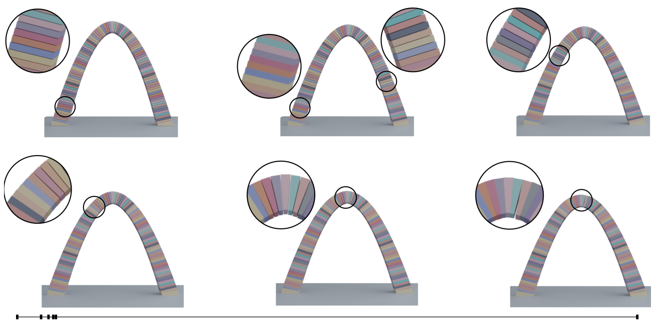
Fig. 11. **Arch.** An arch composed of 101 rigid blocks is in equilibrium under gravity due to friction forces.

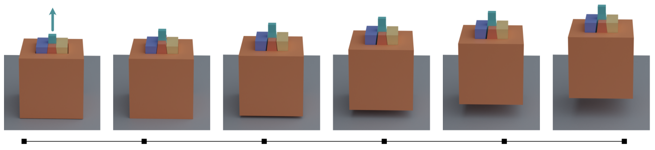
Fig. 12. **Lewis lifting mechanism.** Utilizing friction and geometry the Lewis is able to lift large weights. A pyramid-shaped piece is placed between wedge-shaped pieces. When the center piece is pulled up the surrounding pieces are pressed into the outer block. The center is moved kinematically at 0.5 m/s with $\mu = 0.3$ and is able to lift a block 10 times its mass.

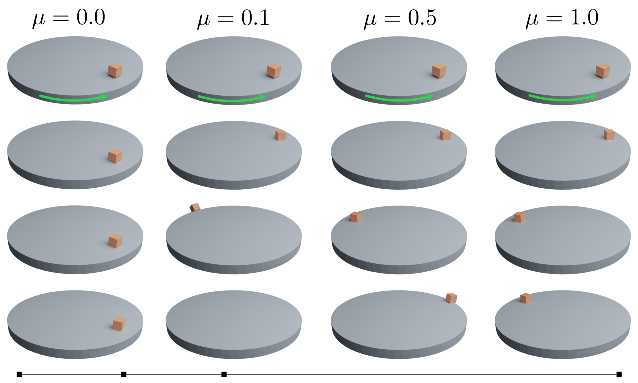

Fig. 13. **Turntable.** A block is dropped on an accelerating turntable with four different coefficients of friction ($\mu = 0, 0.1, 0.5, 1.0$). With $\mu = 0$, the block rests on top of the table, slowly drifting. With $\mu = 0.1$, the block quickly catches and is flung away by the table. With $\mu = 0.5$, 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.

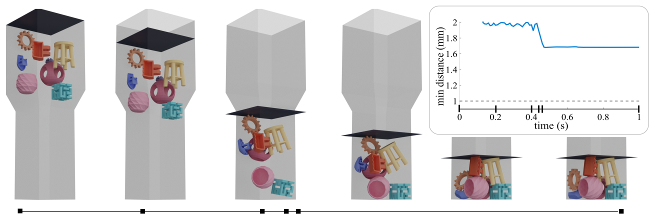

Fig. 14. **3D packing.** Based on the tolerance of Shapeway's PA11 material, we pack eight models into the bed of a 3D printer of size 290 × 290 × 600 mm, with a clearance of 1mm enforced by our minimum separation. (Inset) We plot the minimum distance throughout the simulation showing that we always maintain the desired minimum distance between objects. ©tjhowse, blecheimer, Kacie Hultgren, Creative Tools, Dustin Sallings, Brad Pitcher, Andy Lesniak, and Tony Buser under CC BY.

of friction. As the rotational velocity of the disk increases, the box loses contact and flies away (Figure 13) for $\mu \neq 1$.

Packing for 3D Printing with Minimal Separation. The stability of our algorithm over large time steps and the possibility to add controlled minimal 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 14): we 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. [2020] (and consequently of our curved CCD) increases considerably for large minimal 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 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 15).

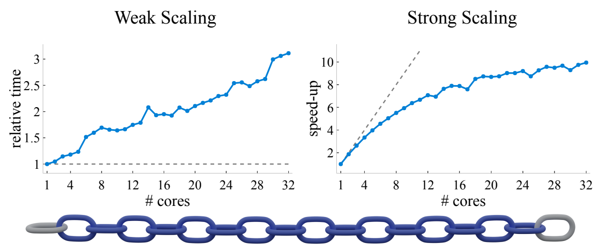

Fig. 15. **Scalability.** We test both the weak (left) and strong (right) scalability on a chain of densely meshed links (bottom). For weak scaling, we set the number of free links equal to the number of threads and plot the runtime divided by the single link time. For strong scaling, we use a chain 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.

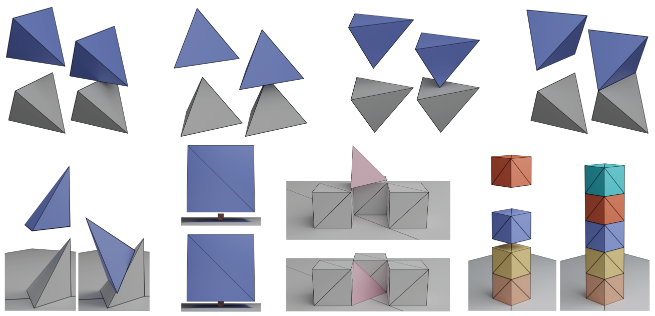

Fig. 16. **Unit tests.** A set of unit test scenes used to benchmark the accuracy and robustness of each method. We show the initial configuration and the resulting simulation using our method.

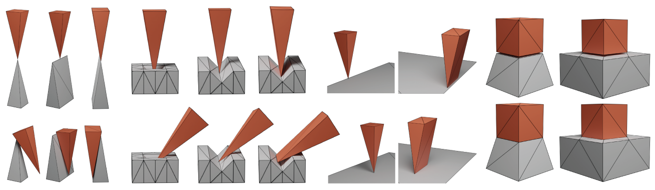

Fig. 17. **Erleben's degenerate test cases.** Our method can easily handle the challenging degenerate cases proposed by Erleben [2018].

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 16), degenerate test cases proposed by Erleben [2018] (Figure 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 simulate 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]. Handcrafted 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 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 contact 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.

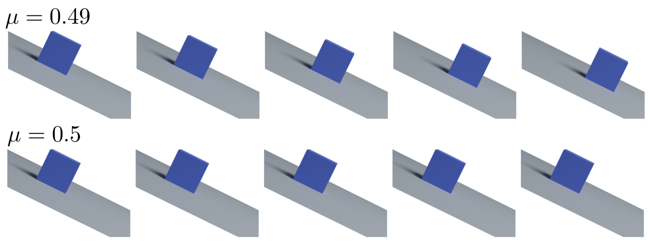

Fig. 18. **High school physics friction test.** We perform a simple test of high school physics by placing a block on an inclined plane with a slope of 26.565°. For a value of $\mu \geq \tan(26.565°) \approx 0.5$, the friction force will counter the acceleration due to gravity. We accurately replicate this by showing for $\mu = 0.49$ the block slides and for $\mu = 0.5$ the block does not slide.

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 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 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 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).

6.1 IPC

While not designed for rigid body simulations, the IPC algorithm [Li et al. 2020] 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 with tetrahedra, increasing the solve time, especially for complex geometries. We show three representative scenes in Figure 19: in the arch, there is no need to insert any internal vertices and IPC is actually faster than the rigid version (two times 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 times). 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.

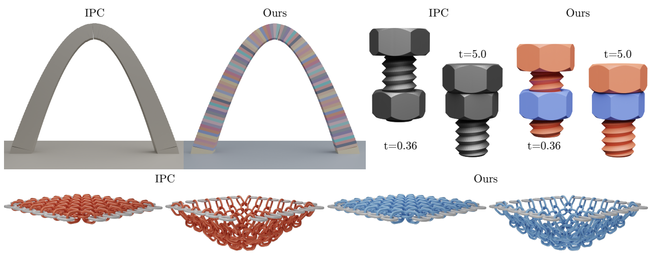

Fig. 19. **IPC comparison.** Comparison of the original volumetric, deformable IPC formulation (using material parameters for steel: Young's modulus $E = 200\,\text{GPa}$ and Poisson ratio Poisson's ratio $\nu = 0.3$) and our rigid body formulation. ©YSoft be3D (screw) under CC BY-SA 3.0.

7 LIMITATIONS AND CONCLUDING REMARKS

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.

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.

ACKNOWLEDGMENTS

We thank Erwin Coumans, Dan Negrut, Alessandro Tasora, Radu Serban, Simone Benatti, and Xun Tan for generous assistance in generating comparisons. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This work was partially supported by the NSF CAREER award 1652515 and IIS-1943199, the NSF grants IIS-1320635, DMS-1436591, DMS-1821334, OAC-1835712, OIA-1937043, CHS-1908767, CHS-1901091, CCF-1813624, ECCS-2023780, a gift from Adobe Research, a gift from nTopology, and a gift from Advanced Micro Devices, Inc.

REFERENCES

Mihai Anitescu and Gary D. Hart. 2004a. A Constraint-Stabilized Time-Stepping Approach for Rigid Multibody Dynamics with Joints, Contact and Friction. *Internat. J. Numer. Methods Engrg.* (2004).
Mihai Anitescu and Gary D. Hart. 2004b. A Fixed-Point Iteration Approach for Multibody Dynamics with Contact and Small Friction. *Mathematical Programming* 101, 1 (2004), 3–32.
Mihai Anitescu and Florian R. Potra. 1997. Formulating Dynamic Multirigid-Body Contact Problems with Friction as Solvable Linear Complementarity Problems. *ASME Nonlinear Dynamics* 14 (1997), 231–247.
Uri M. Ascher and Linda R. Petzold. 1998. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations* (1st ed.). Society for Industrial and Applied Mathematics, USA.
David Baraff. 1989. Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies. *Computer Graphics (Proceedings of SIGGRAPH)* 23, 3 (July 1989), 223–232.
David Baraff. 1991. Coping with Friction for Non-penetrating Rigid Body Simulation. *Computer Graphics (Proceedings of SIGGRAPH)* 25, 4 (July 1991), 31–41.
David Baraff. 1994. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, 23–34.
Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. 2012. Interactive Simulation of Rigid Body Dynamics in Computer Graphics. In *Eurographics 2012 - State of the Art Reports*, Marie-Paule Cani and Fabio Ganovelli (Eds.). The Eurographics Association.
Bernard Brogliato. 1999. *Nonsmooth Mechanics.* Springer-Verlag.
John Canny. 1986. Collision Detection for Moving Polyhedra. *IEEE Transactions on Pattern Analysis and Machine Intelligence* Pami-8, 2 (1986), 200–209.
Michael B. Cline and Dinesh K. Pai. 2003. Post-stabilization for rigid body simulation with contact and constraints. In *Proceedings of IEEE International Conference on Robotics and Automation.*
Eulalie Coevoet, Otman Benchekroun, and Paul G. Kry. 2020. Adaptive Merging for Rigid Body Simulation. *ACM Transactions on Graphics* (2020).
Erwin Coumans and Yunfei Bai. 2016–2019. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.
Kenny Erleben. 2007. Velocity-based shock propagation for multibody dynamics animation. *ACM Transactions on Graphics* (2007).
Kenny Erleben. 2017. Rigid Body Contact Problems using Proximal Operators. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17).* Association for Computing Machinery, New York, NY, Article 13, 12 pages.
Kenny Erleben. 2018. Methodology for Assessing Mesh-Based Contact Point Methods. *ACM Transactions on Graphics* 37, 3 (2018).
Michael Fogleman. 2017. Binary Packing for SLS printing. https://www.michaelfogleman.com/pack3d/.
F. Sebastin Grassia. 1998. Practical Parameterization of Rotations Using the Exponential Map. *Journal of Graphics Tools* 3, 3 (March 1998), 29–48.
Eran Guendelman, Robert Bridson, and Ronald Fedkiw. 2003. Nonconvex Rigid Bodies with Stacking. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* (2003).
Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3.
James K. Hahn. 1988. Realistic Animation of Rigid Bodies. *Computer Graphics (Proceedings of SIGGRAPH)* (Aug. 1988), 299–308.
Ernst Hairer, Christian Lubich, and Gerhard Wanner. 2006. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations.* Vol. 31. Springer.
David Harmon, Daniele Panozzo, Olga Sorkine, and Denis Zorin. 2011. Interference-Aware Geometric Modeling. *ACM Transactions on Graphics* 30, 6 (Dec. 2011), 1–10.
Shu-Wei Hsu and John Keyser. 2010. Piles of Objects. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* (2010).
Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/
Couro Kane, Jerrold E Marsden, Michael Ortiz, and Matthew West. 2000. Variational Integrators and the Newmark Algorithm for Conservative and Dissipative Mechanical Systems. *Internat. J. Numer. Methods Engrg.* 49, 10 (Dec. 2000).
Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. 2008. Staggered Projections for Frictional Contact in Multibody Systems. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 27, 5 (2008).

Fig. 20. **Simulation statistics** for all scenes presented in Section 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 1TB 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 Li et al. [2020] for full definition of these parameters.

Example	bodies	vertices	edges	faces	h (s)	d (m)	μ	ϵ_v (m/s)	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	1e-5		N/a		1e-2 ℓ	5K	9K	811	24.9	9.0
Bolt	2	3K	8K	5K	0.01	1e-4		N/a		1e-2 ℓ	332	759	86	8.3	3.5
Piston	4	6K	6K	2K	0.01	1e-3		N/a		1e-2 ℓ	370	1K	1323	9.9	1.5
Intermitten motion	5	2K	6K	4K	0.01	1e-4		N/a		1e-2 ℓ	21	498	875	5.7	0.2
Bike chain*	138	48K	143K	96K	0.01	1e-5		N/a		1e-2 ℓ	6K	11K	12403	42.0	21.6
Punch	6	2K	7K	5K	0.01	1e-4		N/a		1e-2 ℓ	57	136	591	9.2	1.3
Punch (loose)	6	2K	7K	5K	0.01	1e-4		N/a		1e-2 ℓ	47	78	903	12.5	1.3
Codim. house of cards	18	80	116	56	0.01	1e-3	0.9	1E-05	1	1e-3 ℓ	78	204	1045	161.9	3.2
Codm. chain net	673	23K	25K	1K	0.01	1e-3		N/a		1e-2 ℓ	2K	2K	3234	130.5	20.5
Disconnected components	3	5K	2K	850	0.01	1e-3	0.1	1E-03	1	1e-2 ℓ	7	10	41	1.9	0.1
Spolling coin	2	134	389	258	1e-4	1e-4	0.2	1E-05	*	1e-4 ℓ	12	84	229	3.8	0.01
Wrecking ball	575	8K	20K	13K	0.01	1e-3		N/a		1e-2 ℓ	4K	14K	1959	17.1	4.8
Anchor	23	9K	27K	18K	0.01	1e-3		N/a		1e-2 ℓ	267	1K	1178	21.4	3.0
Arch	102	812	2K	1K	0.01	1e-3	0.5	1E-03	1	1e-3 ℓ	649	695	1590	2.1	0.21
Lewis	7	64	149	98	0.01	1e-3	0.3	1E-03	1	1e-2 ℓ	50	55	26	2.7	0.02
Turntable (mu=0.0)	2	138	402	268	0.025	1e-3	0	1E-05	*	1e-4 ℓ	9	15	178	2.3	0.004
Turntable (mu=0.1)	2	138	402	268	0.025	1e-3	0.1	1E-05	*	1e-4 ℓ	2	13	284	2.9	0.003
Turntable (mu=0.5)	2	138	402	268	0.025	1e-3	0.5	1E-05	*	1e-4 ℓ	7	14	255	5.9	0.01
Turntable (mu=1.0)	2	138	402	268	0.025	1e-3	1	1E-05	*	1e-4 ℓ	10	10	246	9.1	0.01
3D packing*	10	7K	20K	13K	0.01	1e-3		N/a		1e-2 ℓ	184	456	8860	91.3	33.5

Michael Lerch, German Tischler, Jürgen Wolff Von Gudenberg, Werner Hofschuster, and Walter Krämer. 2006. FILIB++, a Fast Interval Library Supporting Containment Computations. *ACM Trans. Math. Software* 32, 2 (June 2006), 299–324.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Transactions on Graphics* 39, 4 (2020).

Per Lötstedt. 1982. Numerical Simulation of Time-Dependent Contact Friction Problems in Rigid Body Mechanics. *SIAM Journal of Scientific Statistical Computing* 5, 2 (1982), 370–393.

Libin Lu, Matthew J. Morse, Abtin Rahimian, Georg Stadler, and Denis Zorin. 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 '19)*. Association for Computing Machinery, New York, NY, Article 6, 30 pages.

M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and T. Y. Kim. 2020. Primal/Dual Descent Methods for Dynamics. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '20)*. Eurographics Association, Goslar, DEU, Article 9, 12 pages.

Khaled Mammou. 2020. *V-HACD*. https://github.com/kmammou/v-hacd

Jerrold E. Marsden and Tudor S. Ratiu. 2013. *Introduction to Mechanics and Symmetry*. Springer.

Hammad Mazhar, Toby Heyn, Dan Negrut, and Alessandro Tasora. 2015. Using Nesterov's Method to Accelerate Multibody Dynamics with Friction and Contact. 34, 3, Article 32 (May 2015), 14 pages.

Brian Mirtich. 2000. Timewarp Rigid Body Simulation. *Annual Conference Series (Proceedings of SIGGRAPH)*, 193–200.

Brian Mirtich and John F. Canny. 1995. Impulse-based dynamic simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*.

Brian Vincent Mirtich. 1996. *Impulse-Based Dynamic Simulation of Rigid Body Systems*. Ph.D. Dissertation.

Jean Jacques Moreau. 1966. Quadratic Programming in Mechanics: Dynamics of One-Sided Constraints. *SIAM Journal on Control* 4, 1 (1966), 153–158.

Jean Jacques Moreau. 1988. Unilateral Contact and Dry Friction in Finite Freedom Dynamics. *Nonsmooth Mechanics and Applications, CISM Courses and Lectures* 302 (1988), 1–82.

Jürgen Moser and Alexander P. Veselov. 1991. Discrete Versions of Some Classical Integrable Systems and Factorization of Matrix Polynomials. *Communications in Mathematical Physics* 139 (1991), 217–243.

Matthias Müller, Miles Macklin, Nuttapong Chentanez, Stefan Jeschke, and Tae-Yong Kim. 2020. Detailed Rigid Body Simulation with Extended Position Based Dynamics. *Computer Graphics Forum* 39, 8 (2020), 101–112.

B. Owren and B. Welfert. 2000. The Newton Iteration on Lie Groups. *BIT Numerical Mathematics* 40 (2000), 121–145.

Jia Pan, Liangjun Zhang, and Dinesh Manocha. 2012. Collision-Free and Smooth Trajectory Computation in Cluttered Environments. *The International Journal of Robotics Research* 31, 10 (2012), 1155–1175.

Xavier Provot. 1997. Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Computer Animation and Simulation*. Springer, 177–189.

Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. 2002a. Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum* 21, 3 (2002), 279–287.

Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. 2002b. Gauss' least constraints principle and rigid body simulations. In *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 1. 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* (1840), 380–440.

Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. 2005. Adaptive Dynamic Collision Checking for Single and Multiple Articulated Robots in Complex Environments. *IEEE Transactions on Robotics* 21 (July 2005), 338–353.

SideFX. 2020. *Houdini*. https://www.sidefx.com/products/houdini/

Breannan Smith, Danny M. Kaufman, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2012. Reflections on Simultaneous Impact. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31, 4 (2012), 106:1–106:12.
John M. Snyder. 1992. Interval Analysis for Computer Graphics. *Computer Graphics (Proceedings of SIGGRAPH)* 26, 2 (July 1992), 121–130.
John M. Snyder, Adam R. Woodbury, Kurt Fleischer, Bena Currin, and Alan H. Barr. 1993. Interval Methods for Multi-Point Collisions between Time-Dependent Curved Surfaces. *Annual Conference Series (Proceedings of SIGGRAPH)*, 321–334.
Jos Stam. 2009. Nucleus: Towards a Unified Dynamics Solver for Computer Graphics. *Proceedings of IEEE International Conference on Computer-Aided Design and Computer Graphics* (2009), 1–11.
David Stewart. 2000. Rigid-Body Dynamics with Friction and Impact. *SIAM Rev.* 42 (March 2000), 3–39.
David Stewart and J.C. Trinkle. 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.
Min Tang, Young J. Kim, and Dinesh Manocha. 2009. C^2A: Controlled Conservative Advancement for Continuous Collision Detection of Polygonal Models. In *Proceedings of IEEE International Conference on Robotics and Automation.* 849–854.
Min Tang, Young J. Kim, and Dinesh Manocha. 2011. *CCQ: Efficient Local Planning Using Connection Collision Query.* Springer Berlin Heidelberg, Berlin, Heidelberg, 229–247.
Alessandro Tasora, Radu Serban, Hammad Mazhar, Arman Pazouki, Daniel Melanz, Jonathan Fleischmann, Michael Taylor, Hiroyuki Sugiyama, and Dan Negrut. 2016. Chrono: An Open Source Multi-physics Dynamics Engine. Springer, 19–49.
Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.* 5026–5033.
Richard Tonge, Feodor Benevolenski, and Andrey Voroshilov. 2012. Mass Splitting for Jitter-Free Parallel Rigid Body Simulation. *ACM Transactions on Graphics* (2012).
Jeff Trinkle, Jong-Shi Pang, Sandra Sudarsky, and Grace Lo. 1995. *On Dynamic Multi-Rigid-Body Contact Problems with Coulomb Friction.* Technical Report. Texas A&M University, Department of Computer Science.
Warwick Tucker. 2011. *Validated Numerics: A Short Introduction to Rigorous Computations.* Princeton University Press, USA.
Etienne Vouga, Breannan Smith, Danny M. Kaufman, Rasmus Tamstorf, and Eitan Grinspun. 2017. All's Well That Ends Well: Guaranteed Resolution of Simultaneous Rigid Body Impact. *ACM Transactions on Graphics* 36, 4 (July 2017).
Bin Wang, François Faure, and Dinesh K. Pai. 2012. Adaptive Image-based Intersection Volume. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 31, 4 (July 2012).
Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2020. A Large Scale Benchmark and an Inclusion-Based Algorithm for Continuous Collision Detection. arXiv:2009.13349 [cs.GR]
J.M.P. van Waveren. 2005. Robust Continuous Collision Detection Between Arbitrary Polyhedra Using Trajectory Parameterization of Polyhedral Features. (March 2005).
Andrew Witkin and David Baraff. 2001. Physically Based Modeling. In *SIGGRAPH 2001 Course Notes.*
Hongyi Xu, Yili Zhao, and Jernej Barbič. 2014. Implicit Multibody Penalty-based Distributed Contact. *IEEE Transactions on Visualization and Computer Graphics* 20, 9 (2014).
Liangjun Zhang, Young J. Kim, and Dinesh Manocha. 2007a. C-DIST: Efficient Distance Computation for Rigid and Articulated Models in Configuration Space. In *Proceedings of ACM Symposium on Solid and Physical Modeling* (Beijing, China) *(SPM '07).* Association for Computing Machinery, New York, NY, 159–169.
Liangjun Zhang, Young J. Kim, Gokul Varadhan, and Dinesh Manocha. 2007b. Generalized Penetration Depth Computation. *Computer-Aided Design* 39, 8 (2007), 625–638.
Xinyu Zhang, Minkyoung Lee, and Young J Kim. 2006. Interactive Continuous Collision Detection for Non-convex Polyhedra. *The Visual Computer* 22, 9-11 (2006), 749–760.
Xinyu Zhang, Stephane Redon, Minkyoung Lee, and Young J. Kim. 2007c. Continuous Collision Detection for Articulated Models Using Taylor Models and Temporal Culling. *ACM Transactions on Graphics* 26, 3 (July 2007), 15–es.

A ROBUSTLY COMPUTING RODRIGUES' ROTATION FORMULA

A.1 Talyor Series Expansion of sinc

To avoid numerical issues when computing $\operatorname{sinc}(x)$ we instead use

$$\operatorname{sinc}(x) = \begin{cases} x^4/120 - x^2/6 + 1 & |x| \leq \epsilon \\ \frac{\sin(x)}{x} & \text{otherwise} \end{cases}$$

where $\operatorname{sinc}(|x| \leq \epsilon)$ is computed using a fifth-order Taylor series expansion around zero.

A.2 Rodrigues' Rotation Formula Derivatives

To avoid numerical issues in derivatives of Rodrigues' rotation formula (Equation (14)) we use a Taylor series expansion around 0. The gradient of $\operatorname{sinc}(\|\theta\|)$ is computed as

$$\nabla \operatorname{sinc}(\|\theta\|) = g(\|\theta\|)\theta$$

with

$$g(x) = \begin{cases} x^4/840 + x^2/30 - 1/3 & |x| \leq \epsilon_g \\ (x\cos(x) - \sin(x))/x^3 & \text{otherwise} \end{cases}$$

where $\epsilon_g = 10^{-4}$. The Hessian of $\operatorname{sinc}(\|\theta\|)$ is computed as

$$\nabla^2 \operatorname{sinc}(\|\theta\|) = h(\|\theta\|)\theta\theta^T + g(\|\theta\|)\mathrm{Id}$$

with

$$h(x) = \begin{cases} x^4/7560 - x^2/210 + 1/15 & |x| \leq \epsilon_H \\ (-x^2\sin(x) - 3x\cos(x) + 3\sin(x))/x^5 & \text{otherwise} \end{cases}$$

where $\epsilon_H = 0.1$.

A.3 Interval Computation of sinc

Given an interval $x = [a, b]$ we want to compute $\operatorname{sinc}(x)$ while avoiding exponentially large intervals around 0. We first start by exploiting the evenness of sinc to compute

$$y_{\text{neg}} = \operatorname{sinc}(x \cap [-\infty, 0]) = \operatorname{sinc}(-(x \cap (-\infty, 0])).$$

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 $\operatorname{sinc}(x \cap [0, m])$, we compute sinc as

$$\begin{aligned} y_{\text{monotonic}} = [\, & \operatorname{lower}(\operatorname{sinc}([\operatorname{upper}(x \cap [0, m])])), \\ & \operatorname{upper}(\operatorname{sinc}([\operatorname{lower}(x \cap [0, m])]))] \end{aligned}$$

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.

B 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 21 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).

C 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 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

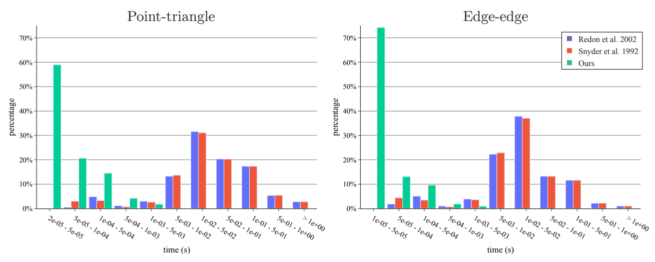

Fig. 21. **CCD comparison.** We compare our narrow-phase curved CCD with the methods of [Snyder 1992] and [Redon et al. 2002a]. We extracted 43K point-triangle and 240K edge-edge queries from the first ten steps of our bolt simulation (Figure 4) which has a good mix of linear and rotating contacts between close conforming geometry. Our method is several orders of magnitude faster than prior methods (x-axis is logarithmic).

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 COMPARISON WITH IPC

We compared against IPC on a set of nine scenes with varying geometric complexity and numbers of bodies. Figure 22 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.

E 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 of Li et al. [2020]).

An easy fix to this problem, should the need ever arise, is to substitute the resulting rotation vector $\boldsymbol{\theta} = \theta \boldsymbol{a}$ with $(\theta \mod 2\pi)\boldsymbol{a}$

Fig. 22. **IPC comparison.** We compare our method with the volumetric IPC [Li et al. 2020] 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	2.6x	10K	3K
Double pendulum	914.0	1559.9	0.6x	12K	4K
Arch (25 stones)	26.5	55.8	0.5x	2K	2K
Arch (101 stones)	238.3	487.8	0.5x	4K	5K
Wrecking ball	7179.8	5748.1	1.2x	9K	18K
Bolt	4031.0	1436.9	2.8x	24K	4K
Rolling cone	1184.2	150.9	7.8x	21K	16K
Chain net (4x4)	1369.9	99.8	13.7x	4K	3K
Chain net (8x8)	9950.5	1420.9	7.0x	5K	5K

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.