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 When all pins have been retracted the bottom is able to freely fall.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.

ACM Trans. Graph., Vol. 40, No. 4, Article 183. Publication date: August 2021.
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].

\[ 2007;\quad\text{Moreau 1988}\] at the cost of physical accuracy.

LCP and related contact models can also equivalently be formulated \textit{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 [Coveney 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 [Kauffman 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.

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 interpretation-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...
While any of the methods above could be used, we opt for \cite{Wang2020} which detects collisions when two primitives are at a distance \( x \). The later vanishes when primitives are further than a user-defined \( \alpha \). The barrier potential is then defined as:

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

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 \( \hat{l}_i \). 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\(^1\) \( \theta_i \in \mathbb{R}^3 \) and a translation \( q_i \in \mathbb{R}^3 \) that together map each body from its local frame to world coordinates with

\[
 \phi_i(\theta_i, q_i) = R(\theta_i) \hat{X}_i + q_i.
\]  

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 \( R \) (see Equation (14)) mapping from a rotation vector to a rotation matrix \cite{Grassia1998, Rodrigues1840}.

We initialize each simulation with a starting configuration of rotations \( \theta^0 \) 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 \( (\theta^{\text{end}}, q^{\text{end}}) \) obtained by time integrating the rigid body system, and the corresponding trajectory from \( (\theta^0, q^0) \) to \( (\theta^{\text{end}}, q^{\text{end}}) \) guaranteed free of intersections. The generated trajectory is piecewise linear in generalized coordinates, \((\theta, q)\), and is a curved trajectory in world coordinates.

Overview. Our approach follows the same high-level ideas as \cite{Li2020} (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 \( \phi \)) 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 \( \hat{d} \) is the mollified unsigned distance between the \( k \) pair of primitives (we refer to \cite{Li2020} 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
are any potential energies defined on
we derive here the IP formulation for a rigid body system integrated
1
2
straint,
Euler time stepper is then
can directly apply standard form, constrained time integrators with
time integrators on
𝐼
inertial frame
is the inertial matrix, and
rotation matrix velocities [Hairer et al. 2006] as \( \frac{1}{2} \text{tr}(\dot{Q}_i J_i Q_i^T) \)
where
\( J_i = \frac{1}{2} \text{diag}(-l_{1}^2 + l_{2}^2 + l_{3}^2, l_{1}^2 - l_{2}^2 + l_{3}^2, l_{1}^2 + l_{2}^2 - l_{3}^2) \)
is the inertia matrix, and \( l_{1}^2, l_{2}^2, \) and \( l_{3}^2 \) are components of the
inertial frame \( I_i \).

With the inertia matrix defined we now target flat equations of
motion that will allow us to compose IPs for arbitrary numerical time
integrators on \( Q \). To do so we simply apply constrained Lagrangian
dynamics with orthogonality \( Q_i^T Q_i - \text{Id} = 0 \) 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
\[
Q_i^{t+1} = Q_i^t + h Q_i^t + h^2 \nabla V(Q_i^{t+1}) J_i^{-1} + Q_i^{t+1} \Lambda J_i^{-1} + h^2 [\tau_i] J_i^{-1},
\]
\( Q_i^{t+1} T Q_i^T - \text{Id} = 0, \)
\( Q_i^T = Q_i^T, \)
where \( \Lambda \) is the symmetric Lagrange-multiplier matrix for our
constraint, \( \tau_i \) 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 \( [\cdot] \)
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
\[
\dot{Q}_i^t = Q_i^t + h Q_i^t + h^2 [\tau_i] J_i^{-1}
\]
\[
Q_i^{t+1} = \arg\min_{Q} \frac{1}{2} \text{tr}(Q J_i Q_i^T) + \text{tr}(Q J_i (\dot{Q}_i^t)^T) + h^2 V(Q),
\] ²
s.t. \( Q^T Q - \text{Id} = 0. \)

Then, for our entire rigid body system (presuming w.l.o.g. for
now no potentials) the implicit Euler IP for rotational coordinates is
\[
E_Q(Q) = \sum_{i=1}^{n} \left( \frac{1}{2} \text{tr}(Q_i J_i Q_i^T) - \text{tr}(Q_i J_i (\dot{Q}_i^t)^T) \right),
\]
and correspondingly for translational coordinates (directly from
standard implicit Euler) we have
\[
\dot{q}_i^t = q_i^t + h q_i^t + h^2 (g + m^{-1} f_i)
\]
\[
E_q(q) = \sum_{i=1}^{n} \left( \frac{1}{2} m_i q_i^T q_i - m_i q_i^T \dot{q}_i^t \right).
\]

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
\[
Q_i^t = \frac{1}{h} (Q_i^t - Q_i^{t-1}) \quad \text{and} \quad \dot{q}_i^t = \frac{1}{h} (q_i^t - q_i^{t-1}).
\]
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 \( E_2 \) in Equation (1), when
we wish to apply rigid body coordinates. This gives us the following
constrained optimization problem to solve
\[
(Q_i^{t+1}, q_i^{t+1}) = \arg\min_{Q, q} E(Q, q) + B(\phi(Q, q), \dot{q}) + D(\phi(Q, q)) \quad \text{subject to} \quad Q_i^T Q_i - \text{Id} = 0, \quad i = 1, \ldots, n,
\]
where the constraint is necessary to ensure that minimizer \( Q_i^{t+1} \)
gives rotation matrices.

Rotation Vector Parametrization. Our goal remains to use uncon-
strained optimization in order to apply as Newton-type solver with
line-search filtering and so robustly minimize the IP with guaran-
tees. To do so parameterizing rotations with the rotation vector, \( \theta_i \),
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
\[
(\theta_i^{t+1}, q_i^{t+1}) = \arg\min_{\theta, q} E(\mathcal{R}(\theta), q) + B(\phi(\mathcal{R}(\theta), q)) + D(\phi(\mathcal{R}(\theta), q)) \quad \text{subject to} \quad \theta_i^T \theta_i = 1
\]
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 addi-
tional 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 \( \text{SO}(3) \) [Öwren 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 in-
creased 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
\[
\mathcal{R}(\theta) = \text{Id} + \sin(\|\theta\|) \begin{bmatrix} \theta & (1 - \cos(\|\theta\|)) \end{bmatrix} \left( \begin{array}{cc} \theta & \|\theta\| \\ \|\theta\| & 0 \end{array} \right),
\]
where \(\mathcal{R}(0) = \text{Id}\). For numerical stability (around \(\theta = 0\)) we rewrite \(\mathcal{R}\) as
\[
\mathcal{R}(\theta) = \text{Id} + \sin(\|\theta\|)|\theta| + 2 \sin^2\left(\frac{\|\theta\|}{2}\right)|\theta|^2,
\]
where
\[
\sin(x) = \begin{cases} 1/\sin(x) & x = 0 \\ 0 & \text{otherwise}. \end{cases}
\]

Note, we compute values close to zero computed using a Taylor series expansion (see Appendix A.1) [Grassia 1998].

While \(\sin\) is \(C^\infty\), 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 \(\sin(\|\theta\|)\) is provided in Appendix A.2.

Additionally, when computing \(\sin(x)\) with interval arithmetic a naive implementation using interval division can result in intervals far outside the range of \(\sin(x)\) (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 \(\xi\) and solving. We continue the process, increasing \(\xi\) by a factor of two until either \(\xi > \xi_{\text{max}} = 1e12\) or the solve is successful. In practice, this offset is rarely needed, and we never reach \(\xi_{\text{max}}\) 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 \(\delta\), as they do not contribute to the barrier potential \(B\) (Equation (2)). In [Li et al. 2020], the pairs of primitives closer than \(\delta\) 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
\[
s_{ba} = \mathcal{R}(\theta_b)^T(\mathcal{R}(\theta_a) \hat{X}_a + q_a - q_b).
\]

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 \(\delta\), 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 \(X_i\) are mapped from a configuration \((\theta^0_i, q^0_i)\) to a configuration \((\theta_i, q_i)\), by
\[
\phi_{a_i}(t) = \mathcal{R}(\theta_i(t)) a_i + q_i(t), \quad t\in[0,1].
\]

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 projection in the reference system of one body extending Equation (15) to the time dependent case,
\[
s_{ba}(t) = \mathcal{R}(\theta_b(t))^T(\mathcal{R}(\theta_a(t)) \hat{X}_a + q_a(t) - q_b(t)).
\]

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-edge or edge-edge) intersect.

Consider the trajectory of a point \(p(t)\) and the trajectories of the three vertices of a triangle \(p_i(t), p_j(t), p_k(t)\). The most direct formulation of continuous collision detection is to explicitly look for the earliest root of the following non-linear system of equations
\[
F_{\mathcal{C}}(t, \alpha, \beta) = p(t) - ((1 - \alpha - \beta)p_i(t) + \alpha p_j(t) + \beta p_k(t)),
\]
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 \( P_1, P_2 \) and \( P_3, P_4 \)

\[
F(t, \alpha, \beta) = \left[ (1-\alpha)P_1(t) + \alpha P_2(t) \right] - \left[ (1-\beta)P_3(t) + \beta P_4(t) \right]
\]

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 \( \phi_{a_i}(t) \) of a single vertex \( a_i \in \mathcal{X}_t \). The time-dependent distance between the curved trajectory and the linear approximation is:

\[
e_{a_i}(t) = \| \phi_{a_i}(t) - ((1-t)P_0 + tP_1) \|, \quad t \in [0, 1].
\]

with \( P_0 = \phi_{a_i}(0) \) and \( P_1 = \phi_{a_i}(1) \). 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 \( L^\infty \) 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 \( t_0 = 0 \) 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.

**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 \( ts \leftarrow t_{earliest} \).
where $t_{\text{earliest}}$ 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 $b \leftarrow b + d_{\text{min}}$).

### 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,

$$E_{A,q}(q) = \frac{k_A}{2} m_k \| q_k - q_k^{t+1} \|^2 - \sqrt{m_k} \lambda_{A,k} (q_k - q_k^{t+1})$$

$$E_{A,Q}(Q) = \frac{k_{A,Q}}{2} \text{tr}(Q_k - Q_k^{t+1}) (Q_k - Q_k^{t+1})^T - \text{tr}(\Lambda_{A,k}^T (Q_k - Q_k^{t+1}) j_k^T),$$

where $(q_k^{t+1}, Q_k^{t+1})$ is the prescribed configuration at time $t + 1$.

Following the algorithm of Li et al. [2020], we initialize the Lagrange multipliers to $\lambda_{A,k} = 0$ and $\Lambda_{A,k} = 0$ and penalty stiffnesses to $k_{A,q} = 10^3$ and $k_{A,Q} = 10^3$. 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

$$\eta_q = 1 - \frac{\sum_k \| q_k^{t+1} - q_k \|^2}{\sum_k \| q_k^{t+1} - q_k^T \|^2}$$

and

$$\eta_Q = 1 - \frac{\sum_k \| Q_k^{t+1} - Q_k \|^2}{\sum_k \| Q_k^{t+1} - Q_k^T \|^2}$$

and converge iff the optimization’s stationarity criteria is satisfied with $\eta_q > 0.999$, and $\eta_Q > 0.999$ [Li et al. 2020].
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. 6. 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).

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.

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).
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 [Mamou 2020]. Hand-crafted custom decompositions are often employed instead which can provide better approximation of the geometry and so improved collision proxies. In the following experiments we use input meshes for convex geometries (all of the unit test and Erleben’s tests) or else, for concave geometries, an expert-constructed manual decomposition.

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

We also test Bullet’s experimental collision handling between arbitrary input triangle meshes directly (without convex decomposition proxies) and find it fails on almost all unit tests and the tests of Erleben [2018] using default parameters. We observe large amounts of energy injected into the system as an effect of position stabilization: once an intersection appears, the simulator quickly pushes 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 $0.1 \times$-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 $0.1 \times$-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 0.1x-scale much slower (more than 3x) 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 x 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 10x (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.

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 $\mu \geq \tan(26.565°) = 0.5$, 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.

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.
While our algorithm is slower than competing methods, our method with more code optimizations or the use of GPU accelerators, (2) 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 Xin Tan for generous assistance in generating comparisons. This work was supported in part through the NYUJT 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**


**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 $\ell$ 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.

<table>
<thead>
<tr>
<th>Example</th>
<th>bodies</th>
<th>vertices</th>
<th>edges</th>
<th>faces</th>
<th>$\mu$</th>
<th>$\epsilon_v$ (m/s)</th>
<th>friction iterations</th>
<th>newton iterations</th>
<th>contacts avg.</th>
<th>contacts max.</th>
<th>memory (MB)</th>
<th>iterations</th>
<th>timing (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expanding lock box</td>
<td>11</td>
<td>66K</td>
<td>22K</td>
<td></td>
<td>0.01</td>
<td>1e-5</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>5K</td>
<td>9K</td>
<td>811</td>
<td>24.9</td>
<td>9.0</td>
</tr>
<tr>
<td>Bolt</td>
<td>2</td>
<td>3K</td>
<td>8K</td>
<td>5K</td>
<td>0.01</td>
<td>1e-4</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>332</td>
<td>759</td>
<td>86</td>
<td>8.3</td>
<td>3.5</td>
</tr>
<tr>
<td>Piston</td>
<td>4</td>
<td>6K</td>
<td>8K</td>
<td>2K</td>
<td>0.01</td>
<td>1e-3</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>21</td>
<td>498</td>
<td>875</td>
<td>5.7</td>
<td>0.2</td>
</tr>
<tr>
<td>Intermittent motion</td>
<td>5</td>
<td>2K</td>
<td>6K</td>
<td>4K</td>
<td>0.01</td>
<td>1e-4</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>21</td>
<td>498</td>
<td>875</td>
<td>5.7</td>
<td>0.2</td>
</tr>
<tr>
<td>Bike chain*</td>
<td>13</td>
<td>48K</td>
<td>143K</td>
<td>96K</td>
<td>0.01</td>
<td>1e-5</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>6K</td>
<td>11K</td>
<td>12403</td>
<td>42.0</td>
<td>21.6</td>
</tr>
<tr>
<td>Punch</td>
<td>6</td>
<td>2K</td>
<td>7K</td>
<td>5K</td>
<td>0.01</td>
<td>1e-4</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>57</td>
<td>136</td>
<td>591</td>
<td>9.2</td>
<td>1.3</td>
</tr>
<tr>
<td>Punch (loose)</td>
<td>6</td>
<td>2K</td>
<td>7K</td>
<td>5K</td>
<td>0.01</td>
<td>1e-4</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>47</td>
<td>78</td>
<td>903</td>
<td>12.5</td>
<td>1.3</td>
</tr>
<tr>
<td>Codim. house of cards</td>
<td>18</td>
<td>40</td>
<td>116</td>
<td>56</td>
<td>0.01</td>
<td>1e-3</td>
<td>0.9</td>
<td>1E-05</td>
<td>1e-3 f</td>
<td>78</td>
<td>204</td>
<td>1045</td>
<td>161.9</td>
</tr>
<tr>
<td>Codim. chain net</td>
<td>672</td>
<td>23K</td>
<td>25K</td>
<td>1K</td>
<td>0.01</td>
<td>1e-3</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>2K</td>
<td>2K</td>
<td>3234</td>
<td>130.5</td>
<td>20.5</td>
</tr>
<tr>
<td>Disconnected components</td>
<td>5</td>
<td>9K</td>
<td>2K</td>
<td>850</td>
<td>0.01</td>
<td>1e-3</td>
<td>1e-03</td>
<td>1</td>
<td>1e-2 f</td>
<td>7</td>
<td>10</td>
<td>41</td>
<td>1.9</td>
</tr>
<tr>
<td>Spooling coin</td>
<td>2</td>
<td>134</td>
<td>389</td>
<td>258</td>
<td>1e-4</td>
<td>0.2</td>
<td>0.01</td>
<td>1</td>
<td>1e-4 f</td>
<td>84</td>
<td>229</td>
<td>3.8</td>
<td>1.7</td>
</tr>
<tr>
<td>Woolen net</td>
<td>267</td>
<td>7K</td>
<td>18K</td>
<td>1K</td>
<td>0.01</td>
<td>1e-3</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>267</td>
<td>1K</td>
<td>1178</td>
<td>21.4</td>
<td>3.0</td>
</tr>
<tr>
<td>Anchor</td>
<td>32</td>
<td>27K</td>
<td>18K</td>
<td>1K</td>
<td>0.01</td>
<td>1e-3</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>10</td>
<td>10</td>
<td>246</td>
<td>9.1</td>
<td>0.01</td>
</tr>
<tr>
<td>Arch</td>
<td>102</td>
<td>612</td>
<td>3K</td>
<td>1K</td>
<td>0.01</td>
<td>1e-3</td>
<td>0.5</td>
<td>1E-03</td>
<td>1e-3 f</td>
<td>469</td>
<td>695</td>
<td>1590</td>
<td>2.1</td>
</tr>
<tr>
<td>Lewis</td>
<td>7</td>
<td>64</td>
<td>149</td>
<td>98</td>
<td>0.01</td>
<td>1e-3</td>
<td>0.3</td>
<td>1E-03</td>
<td>1e-2 f</td>
<td>50</td>
<td>55</td>
<td>26</td>
<td>2.7</td>
</tr>
<tr>
<td>Turntable (mu=0.1)</td>
<td>2</td>
<td>138</td>
<td>402</td>
<td>268</td>
<td>0.025</td>
<td>1e-3</td>
<td>0.1</td>
<td>1E-05</td>
<td>1e-4 f</td>
<td>13</td>
<td>284</td>
<td>2.9</td>
<td>0.003</td>
</tr>
<tr>
<td>Turntable (mu=0.5)</td>
<td>2</td>
<td>138</td>
<td>402</td>
<td>268</td>
<td>0.025</td>
<td>1e-3</td>
<td>0.5</td>
<td>1E-05</td>
<td>1e-4 f</td>
<td>7</td>
<td>14</td>
<td>255</td>
<td>5.9</td>
</tr>
<tr>
<td>Turntable (mu=1.0)</td>
<td>2</td>
<td>138</td>
<td>402</td>
<td>268</td>
<td>0.025</td>
<td>1e-3</td>
<td>1</td>
<td>1E-05</td>
<td>1e-4 f</td>
<td>10</td>
<td>10</td>
<td>246</td>
<td>9.1</td>
</tr>
<tr>
<td>3D packing*</td>
<td>10</td>
<td>7K</td>
<td>29K</td>
<td>1K</td>
<td>0.01</td>
<td>1e-3</td>
<td>N/a</td>
<td>1e-2 f</td>
<td>184</td>
<td>456</td>
<td>3880</td>
<td>91.3</td>
<td>33.5</td>
</tr>
</tbody>
</table>


Hammad Mazah, Toby Heyn, Dan Neyruf, and Alessandro Tasora. 2015. Using Nesterov’s Method to Accelerate Multibody Dynamics with Friction and Contact. 34, 3 (May 2015), 14 pages.


A.1 Taylor Series Expansion of sinc

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

\[
\text{sinc}(x) = \frac{x^4/120 - x^2/6 + 1}{x} |x| \leq \epsilon
\]

otherwise

where \( |x| \leq \epsilon \) 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 \( \text{sinc}(\|\theta\|) \) is computed as

\[
\nabla \text{sinc}(\|\theta\|) = g(\|\theta\|)\theta
\]

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 \( \text{sinc}(\|\theta\|) \) is computed as

\[
\nabla^2 \text{sinc}(\|\theta\|) = h(\|\theta\|)\theta^T + g(\|\theta\|)I_d
\]

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 \( \text{sinc}(x) \) while avoiding exponentially large intervals around 0. We first start by exploiting the evenness of sinc to compute

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

Now that our domain is from \([0, \infty)\), we utilize the monotonicity of sinc to decompose \( x \) into \( x \cap [0, m] \) and \( x \cap (m, \infty) \) where \( m = 4.4934049579 \) is a conservative value lower value for the upper bound of the monotonic sub-domain. The latter case can be computed as normal using interval division because the values are not too small. For \( \text{sinc}(x \cap [0, m]) \), we compute sinc as

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

where \([ \cdot ]\) 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 \( \delta \)

The parameter \( \delta \) 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 \( \delta \): 0.1, 0.5, and 0.9. We do not consider a value outside of \((0, 1)\) because our distances are all unsigned and a
value $\delta > 1$ could result in the immediate termination of the linear CCD (the initial distance is less than the minimum distance $\delta$).

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 ($4 \times 4$ and $8 \times 8$) 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 $2\pi$, 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 $\theta = \theta a$ with $(\theta \mod 2\pi) a$. 

---

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).

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.