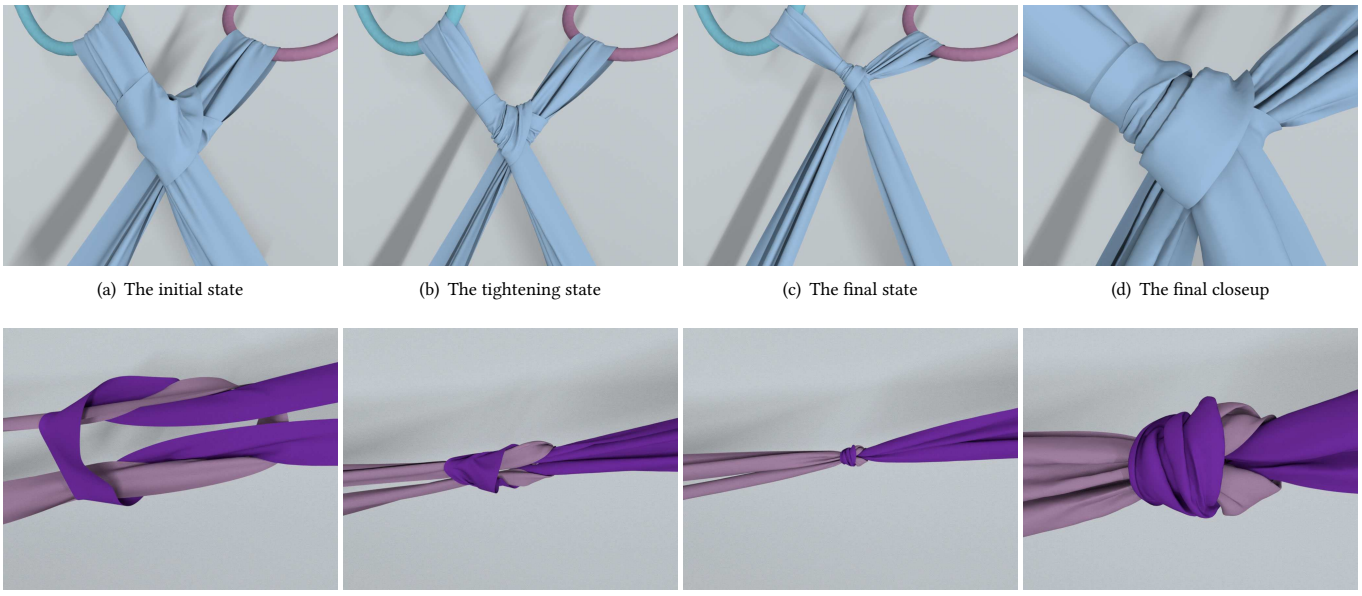
A Two-Way Interior Point Method for Collision Handling in Deformable Body Simulation

TIANYU WANG, State Key Lab of CAD&CG, Zhejiang University
DONGPING LI, XIAOWEI LIU, FaceUnity
JIONG CHEN, Inria Saclay / Telecom Paris
HUAMIN WANG, The Ohio State University / Style 3D
KUN ZHOU, State Key Lab of CAD&CG, Zhejiang University



(a) The initial state (b) The tightening state (c) The final state (d) The final closeup

(e) The initial state (f) The tightening state (g) The final state (h) The final closeup

Fig. 1. The bow knot example (on the top, with 142K triangles) and the reef knot example (on the bottom, with 71K triangles). In this work, we develop a two-way interior point method for safe, fast and accurate collision handling in deformable body simulation. Thanks to this method, our simulator can robustly handle complex collision contacts in these examples.

Impact zone optimization methods and interior point methods are popular choices for collision handling in deformable body simulation. Their key difference is in where the optimization process starts: an impact zone optimization method starts at the target state, while an interior point method starts at the previous state. Due to this difference, an impact zone optimization method can reduce its cost by focusing the optimization on colliding vertices only, but it can fail to succeed. Meanwhile, an interior point method is guaranteed to be safe, but it needs a considerable cost to repetitively advance all of the vertices during optimization. In this paper, we present a two-way interior point method for safe collision handling, which launches the optimization process at both ends by two interleaving steps. Specifically, in each backward step, the method solves a linear complementarity problem inexactly, to make the target state intersection-free; in each forward step, the method then uses a lightweight scheme to move vertices toward the modified target state, without expensive collision tests. Thanks to a set of unified volume-based contact constraints, our method flexibly handles a variety of codimensional deformable body examples, including volumetric bodies, cloth, hair and sand. Our experiment shows the method is safe, robust, efficient and easy to parallelize on GPUs. Compared with existing interior point methods, our method runs about two orders-of-magnitude faster, even in complex collision cases.

CCS Concepts: • **Computing methodologies** → **Physical simulation**;

Additional Key Words and Phrases: collision handling, interior point method, deformable body simulation, GPU computation, nonlinear optimization

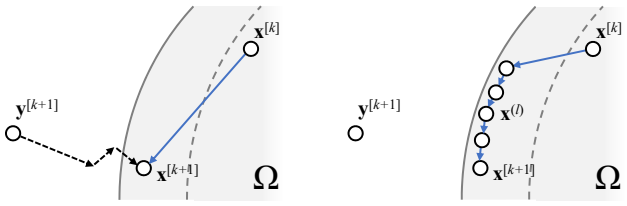
ACM Reference Format:
Tianyu Wang, Dongping Li, Xiaowei Liu, Jiong Chen, Huamin Wang, and Kun Zhou. 2021. A Two-Way Interior Point Method for Collision Handling in Deformable Body Simulation. *ACM Trans. Graph.* 1, 1 (December 2021), 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Tianyu Wang, State Key Lab of CAD&CG, Zhejiang University; Dongping Li, Xiaowei Liu, FaceUnity; Jiong Chen, Inria Saclay / Telecom Paris; Huamin Wang, The Ohio State University / Style 3D; Kun Zhou, State Key Lab of CAD&CG, Zhejiang University.

2021. 0730-0301/2021/12-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1. The strengths and the weaknesses of three methods. The question marks indicate the two mutually exclusive strengths of a method, between which the method must sacrifice one for the other.

Name	Safety	Efficiency	Accuracy
Impact zone optimizer	?	?	✓
Interior point method	✓	?	?
Our two-way method	✓	✓	✓



(a) An impact zone optimizer (b) An interior point method

Fig. 2. The optimization processes involved in an impact zone optimizer and an interior point method. While both methods try to find a feasible path from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$, neither of them can be safe, efficient and accurate at the same time.

1 INTRODUCTION AND BACKGROUND

How to safely, efficiently and accurately handle collisions is an important yet challenging problem in physics-based deformable body simulation. This problem becomes even more urgent recently, thanks to the substantial research progress on fast dynamics solvers. According to today's GPU-based simulators [Li et al. 2020b; Wu et al. 2020], collision handling can easily contribute 60 percent of the total computational cost. This percentage is even higher, when collisions are intense and frequent.

To understand the difficulty of collision handling, let us consider how a simulator works and the role of collision handling in it. Mathematically, the simulation of a deformable body is a sequence of updates $\left\{\mathbf{x}^{[k]}\right\}$ to vertex positions, in which $\mathbf{x}^{[k]} \in \mathbb{R}^{3N}$ is the stacked positional state of N vertices in the k-th update. Depending on the type of simulators, $\mathbf{x}^{[k]}$ corresponds to either the update of one time step [Bridson et al. 2003], or the update of one iteration involved in a time step [Macklin et al. 2014; Wu et al. 2020]. No matter what $\mathbf{x}^{[k]}$ means, the goal of collision handling is the same: *preventing the body from being intersecting at any time*. Let $\mathbf{x}^{[k]}$ be the previous state, $\mathbf{x}^{[k+1]}$ be the unknown next state and $\mathbf{y}^{[k+1]}$ be the target state, commonly predicted by dynamics solvers. We formulate collision handling as an optimization problem:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \left\|\mathbf{x} - \mathbf{y}^{[k+1]}\right\|^2, \quad \text{s.t. } \mathcal{X}\left(\mathbf{x}^{[k]}, \mathbf{x}\right) \subset \Omega, \quad (1)$$

in which Ω is the feasible (intersection-free) region and $\mathcal{X}\left(\mathbf{x}^{[k]}, \mathbf{x}\right)$ is a plausible path connecting $\mathbf{x}^{[k]}$ and $\mathbf{x}$. We typically assume that the path is linear, or piecewise linear through intermediate states $\mathbf{x}^{(0)}$, ..., $\mathbf{x}^{(l)}$, ..., $\mathbf{x}^{(L)}$, for $\mathbf{x}^{(0)} = \mathbf{x}^{[k]}$ and $\mathbf{x}^{(L)} = \mathbf{x}^{[k+1]}$.

An ideal collision handling method must satisfy three basic requirements:

- *safety*, i.e., the path from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$ being strictly intersection-free,
- *efficiency*, i.e., finding $\mathbf{x}^{[k+1]}$ fast,
- *accuracy*, i.e., finding $\mathbf{x}^{[k+1]}$ sufficiently close to $\mathbf{y}^{[k+1]}$.

Compared with the other requirements, accuracy is relatively less important and can be sacrificed. But we should not ignore it, or visual artifacts will appear. For example, the rigid impact zone technique [Bridson et al. 2002, 2003; Provot 1997] is safe and fast, but it causes locking artifacts due to eliminated relative motions in impact zones. Therefore we still need $\mathbf{x}^{[k+1]}$ to be close to $\mathbf{y}^{[k+1]}$.

Throughout history, researchers have discovered two types of approaches for safe collision handling: *impact zone optimizers* and *interior point methods*.

1.1 Impact Zone Optimizers

To address the locking artifacts caused by rigid impact zones, researchers [Harmon et al. 2008; Tang et al. 2016, 2018b] proposed to optimize the target state $\mathbf{y}^{[k+1]}$ until it becomes intersection-free. Intuitively, we can view this approach as the projection of $\mathbf{y}^{[k+1]}$ to Ω, as Fig. 2a shows. The strength of an impact zone optimizer is its efficiency: when most of the vertices are not involved in collision, the method just optimizes the vertices in small impact zones. One problem is an impact zone optimizer converges slowly or even fails to converge [Tang et al. 2018b], if $\mathbf{y}^{[k+1]}$ is too far away from Ω. Another problem is tunneling artifacts, due to the fact that impact zone optimization enforces only the state, rather than the whole path, to become intersection-free. We can fix both problems by using small steppings, but doing so inevitably introduces more computational cost.

1.2 Interior Point Methods

Alternatively, an interior point method launches its optimization at $\mathbf{x}^{[k]}$ and keeps the optimization strictly within Ω by small steppings, either synchronously [Li et al. 2020a] or asynchronously [Harmon et al. 2009; Wu et al. 2020]. Each optimization step provides an intermediate state and the path from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$ is piecewise linear, as Fig. 2b shows. An interior point method is strictly safe, but notoriously slow for two reasons.

- *Frequent tests.* The method needs to repetitively test the vertices in every optimization step, to make sure that no intersection occurs.
- *Barrier functions.* Barrier functions push $\mathbf{x}$ sufficiently against the feasible region boundary $\partial\Omega$, only when $\mathbf{x}$ gets close to $\partial\Omega$. As a result, small steppings are inevitable.

Note that we can view the rigid impact zone technique [Bridson et al. 2002, 2003; Provot 1997] as a trivial interior point method with one-step optimization.

1.3 A Quick Summary

As far as we know, no single method is able to achieve safety, efficiency and accuracy all at the same time. We can always sacrifice efficiency for safety or accuracy as Table 1 shows, but it is not ideal. Recently, Wu et al. [2020] proposed to speed up collision handling by a two-phase approach: a fast but unsafe soft phase based on

impact zone optimization and a slow but safe hard phase running an interior point method. Their assumption is that the approach is fast as long as the soft phase resolves most of the collision cases. But when collisions are intense and frequent, their approach falls back to the slow interior point phase.

Our goal in this work is simple: to develop a safe, efficient and accurate collision handling method. Instead of treating impact zone optimization and interior point optimization as two independent processes as in [Wu et al. 2020], we propose to combine them into a joint method that can receive benefits from both optimizations. Specifically, we make the following contributions.

- *A two-way framework.* In this framework, the optimization process starts at both the target state and the previous state by interleaving two steps: backward steps and forward steps. Our method based on this framework still works as an interior point method, but it avoids the efficiency issue by concentrating heavy computation in backward steps.
- *Inexact backward step.* To project the target state back to the intersection-free region, we propose to solve a linear complementarity problem inexactly in every backward step. Our implementation is based on the multi-color Gauss-Seidel method.
- *Lightweight forward step.* Thanks to the backward step, we develop a lightweight forward step that asynchronously advances vertices from the current state toward the target state. In this step, we use discrete distance evaluations, rather than continuous tests, to find safe steppings fast.

We implement our two-way interior point method on a GPU and incorporate it into our in-house GPU-based deformable body simulation engine. Coupled with volume-based contact constraints, our simulator is capable of simulating a variety of codimensional examples (as shown in Fig. 1 and 7), including volumetric bodies, cloth, hair and sand. Our experiment shows the method is safe, fast, friendly with GPU parallelization, and robust against large time steps and deformations.

2 OTHER RELATED WORKS

Discrete collision handling. Researchers [Baraff et al. 2003; Volino and Magnenat-Thalmann 2006; Wicke et al. 2006] developed discrete collision handling methods to remove intersections at the end of every time step. If a discrete collision handling method fails to eliminate all of the intersections, it can still repeat the process in the next time step, hopefully achieving the intersection-free state later. Therefore a discrete collision handling method is robust regardless of the time step. But as the time step increases, it becomes less likely to remove all of the intersections, which leads to long-lasting penetration artifacts in simulation.

Many physics-based simulators apply repulsion forces among proximity pairs to lessen the likelihood of collisions. Broadly speaking, this repulsion approach is a discrete collision handling method, as it calculates repulsive forces based on proximity distances discretely evaluated in time. Thanks to its simplicity, the repulsion approach is widely used in GPU-based simulation [Fratarcangeli et al. 2016; Macklin et al. 2014; Stam 2009] alone, without support from any other method. To help the repulsion approach achieve intersection-free guarantee, Wu et al. [2020] presented a fail-safe interior point repulsion phase, whose usage should be minimized due to a large computational overhead.

Continuous collision handling. The main difference between discrete collision handling and continuous collision handling is that discrete collision handling tries to eliminate intersections at the end of the time step, while continuous collision handling must resolve all of the intersections at any time. A typical continuous collision handling method contains two components: continuous collision detection (CCD) and applying collision responses. Continuous detection of a vertex-triangle or edge-edge collision involves solving a cubic equation, which is difficult, expensive and prone to errors [Ainsley et al. 2012; Provot 1997; Tang et al. 2014; Wang 2014]. But compared with CCD, obtaining the right collision responses is an even greater challenge. Bridson et al. [2002; 2003] initially used geometric impulses as responses and the rigid impact zone technique as a failsafe. To avoid the locking artifacts caused by the rigid impact zone technique, Harmon et al. [2008] and Tang et al. [2018b] proposed to calculate collision responses by impact zone optimization. Recently, interior point methods [Li et al. 2020a, 2021; Wu et al. 2020] have emerged to be popular choices for collision responses. The strength of an interior point method is its safety, no matter how large the time step is. But its performance is often unsatisfactory as discussed in Section 1.

We note that many continuous collision handling methods require considerable implementation efforts to get accelerated on GPUs [Lauterbach et al. 2010; Tang et al. 2018a, 2016], due to their high complexity and high dependency on sequential tasks.

Asynchronous steppings. Like other interior point methods, the conservative advancement approach [Mirtich and Canny 1995; Von Herzen et al. 1990] for rigid body collision handling suffers from the small stepping issue, as it requires all of the bodies to take the same step size. Mirtich [2000] addressed this issue by allowing rigid bodies to take different step sizes, while still respecting causality. Researchers [Harmon et al. 2009; Thomaszewski et al. 2008] later investigated this idea for asynchronous collision handling of cloth, and explored several speedup options [Ainsley et al. 2012; Harmon et al. 2011]. Our method is also asynchronous: vertices away from collisions can take large step sizes to reach their targets fast. More importantly, it avoids CCD tests and it is naturally free of performance or robustness issues associated with them.

Broad-phase collision culling. Broad-phase collision culling is important to collision handling methods, as it avoids unnecessary collision tests for collision-free primitive pairs. In general, collision culling techniques fall into two categories: those based on bounding volume hierarchy (BVH) [Lauterbach et al. 2010; Tang et al. 2010, 2011; Wang et al. 2017] and those based on spatial hashing [Barbič and James 2010; Pabst et al. 2010; Tang et al. 2018a; Teschner et al. 2003; Zheng and James 2012]. While GPU implementations of both categories have been investigated before, GPU-based spatial hashing is arguably more popular, thanks to its simplicity and parallelizability. Our method is orthogonal to collision culling techniques and it can adopt more advanced ones later.

ALGORITHM 1: A two-way interior point Method

Input: the current state $\mathbf{x}^{[k]}$, the target state $\mathbf{y}^{[k+1]}$, the proximity search bound $\left[D^{\min}, D^{\max}\right]$, the loop limit L and the termination condition ϵ.

$\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{[k]}$;
$\mathbf{y}^{(0)} \leftarrow \mathbf{y}^{[k+1]}$;
$D^{(0)} \leftarrow 0$;
$\mathscr{P} \leftarrow \emptyset$;
$\mathbf{r} \leftarrow \mathbf{1}$;
for $l = 0 \ldots L$ **do**
  if $D^{(l)} < D^{\min}$ **then**
    $\mathscr{P} \leftarrow$ `Proximity_Search`$(\mathbf{x}^{(l)}, D^{\max})$;
    $D^{(l)} \leftarrow D^{\max}$;
  end
  $\mathbf{y}^{(l+1)} \leftarrow$ `Backward_Step`$(\mathbf{y}^{(l)}, \mathbf{y}^{(0)}, \mathscr{P})$;
  $\mathbf{x}^{(l+1)} \leftarrow$ `Forward_Step`$(\mathbf{x}^{(l)}, \mathbf{y}^{(l+1)} - \mathbf{x}^{(l)}, \mathscr{P}, \mathbf{r})$;
  $D^{(l+1)} \leftarrow D^{(l)} - 2\max_i \left\|\mathbf{x}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right\|$;
  if $\max_i(r_i) < \epsilon$ **then**
    break;
  end
end
$\mathbf{x}^{[k+1]} \leftarrow \mathbf{x}^{(l+1)}$;

Frictional contacts. How to simulate frictional contacts, especially frictional self contacts, is another challenging problem in deformable body simulation. The popular velocity filtering approach [Bridson et al. 2002; Müller 2008] is simple, fast, but not so physically plausible, as it handles collisions and frictions in separate processes. Recently, researchers [Bertails-Descoubes et al. 2011; Daviet 2020; Li et al. 2020a; Ly et al. 2020; Macklin et al. 2019; Verschoor and Jalba 2019] are interested in handling collisions and frictions together through joint optimization. Although our work does not consider frictions yet, we plan to borrow their ideas for simulating plausible frictional contacts in the future.

3 A TWO-WAY FRAMEWORK

Our interior point method is based on a unique framework that launches its optimization at both ends, as Fig. 3 shows. Specifically, the method interleaves two steps: a backward step starting at the target state and a forward step starting at the previous state. Let $\mathbf{y}^{(l)}$ and $\mathbf{x}^{(l)}$ be the target state and the current state in the l-th step. In a new backward step, we first move the target state toward the intersection-free region by inexactly solving:

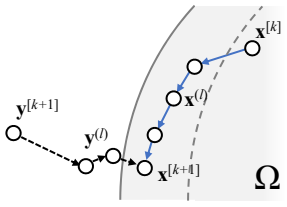


Fig. 3. The optimization process of our two-way interior point method.

$$\mathbf{y}^{(l+1)} = \arg\min_{\mathbf{y}} \left\|\mathbf{y} - \mathbf{y}^{(0)}\right\|^2, \quad \text{s.t. } \mathbf{c}(\mathbf{y}) \geq \mathbf{0}, \tag{2}$$

with $\mathbf{y}^{(l)}$ being treated as the initialization and $\mathbf{c}(\mathbf{y}) \geq \mathbf{0}$ being the constraints outlining the intersection-free region. In a new forward step, we then move the vertices from the current state $\mathbf{x}^{(l)}$ toward $\mathbf{y}^{(l+1)}$ asynchronously, by finding a safe step size $\alpha_i^{(l+1)}$ for every vertex i:

$$\mathbf{x}_i^{(l+1)} = \mathbf{x}_i^{(l)} + \alpha_i^{(l+1)}\left(\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right). \tag{3}$$

The method keeps running the two steps, until it finishes a fixed number of steps L, or $\max_i(r_i)$ is sufficiently small which means the optimized result is sufficiently close to the target position. Either way, the method can safely report any $\mathbf{x}^{(l+1)}$ as its result, no matter whether $\mathbf{y}^{(l)}$ reaches Ω or not. Alg. 1 outlines the pseudo code of our method. Section 4 and 5 provide the details of the two steps.

3.1 Proximity Search

An important component in our two-way framework is the proximity search. This component serves three purposes: to form the set of contact constraints in the backward step (in Subsection 4.1), to obtain proximity pair distances for safe steppings (in Section 5), and to calculate repulsive forces as part of dynamics (in Subsection 6.2). In our method, the proximity search uses the standard grid-based spatial hashing technique [Pabst et al. 2010; Tang et al. 2018a].

Since the proximity search has a non-negligible cost, one challenge is how to reuse search results as often as possible, rather than to redo the search in every step. Let $\mathscr{P}$ be the set containing all of the proximity pairs whose distances are below a certain bound $D^{(l)}$ in the l-th step:

$$p \in \mathscr{P}, \quad \forall p : \mathtt{dist}_p\left(\mathbf{x}^{(l)}\right) < D^{(l)}. \tag{4}$$

In the l+1-th step, $\mathscr{P}$ contains all of the pairs whose distances are below $D^{(l+1)} = D^{(l)} - 2\max_i \left\|\mathbf{x}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right\|$. If $D^{(l+1)} > D^{\min}$, we reuse $\mathscr{P}$. Otherwise, we think $\mathscr{P}$ is insufficient and we perform the proximity search to reset $D^{(l+1)} = D^{\max}$.

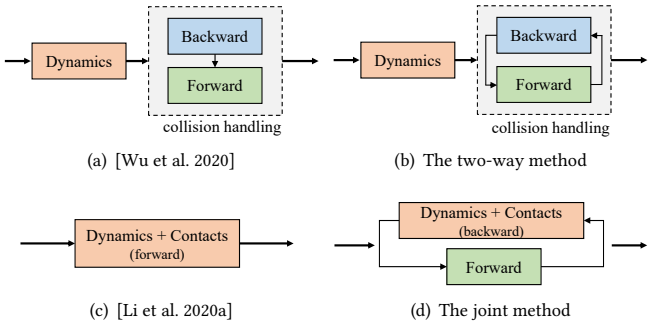
The computational cost of the proximity search depends on both $D^{\min}$ and $D^{\max}$. The cost decreases as $D^{\min}$ decreases, but $D^{\min}$ cannot be too small or $\mathscr{P}$ will become useless. Meanwhile, the cost decreases as $D^{\max}$ increases, but doing so introduces more memory usage and distance evaluations, due to an increased number of proximity pairs. In our experiment, we choose $D^{\min}$ = 2mm and $D^{\max}$ = 4mm, which results in approximately one proximity search every three steps.

3.2 Dynamics–Collision Integration

In the two-way framework, we assume that collision handling is a post-processing after running dynamics solvers. This assumption makes our method compatible with many existing simulators, including the one interleaving dynamics solvers and collision handling [Wu et al. 2020] shown in Fig. 4a.

One interesting question is whether we can combine dynamics solvers and collision handling as in [Li et al. 2020a], by integrating dynamics into a joint objective $E(\mathbf{x})$ to be minimized:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} E(\mathbf{x}), \quad \text{s.t. } \mathscr{X}\left(\mathbf{x}^{[k]}, \mathbf{x}\right) \subset \Omega, \tag{5}$$



(a) [Wu et al. 2020] (b) The two-way method

(c) [Li et al. 2020a] (d) The joint method

Fig. 4. The concept maps of four methods. Our method handles collisions as a post-processing after dynamics solvers shown in (b), so it is compatible with many existing simulators, such as the one proposed in [Wu et al. 2020]. Our method can also be modified to handle dynamics and contacts together as shown in (d), but it is not recommended.

Given Eq. 5, we can then define the backward step by inexactly solving:

$$\mathbf{y}^{(l+1)} = \arg\min_{\mathbf{y}} E(\mathbf{y}), \quad \text{s.t. } \mathbf{c}(\mathbf{y}) \geq \mathbf{0}. \tag{6}$$

Note that $\mathbf{y}^{(0)}$ is no longer the initial target state provided by dynamics solvers, but an initial guess.

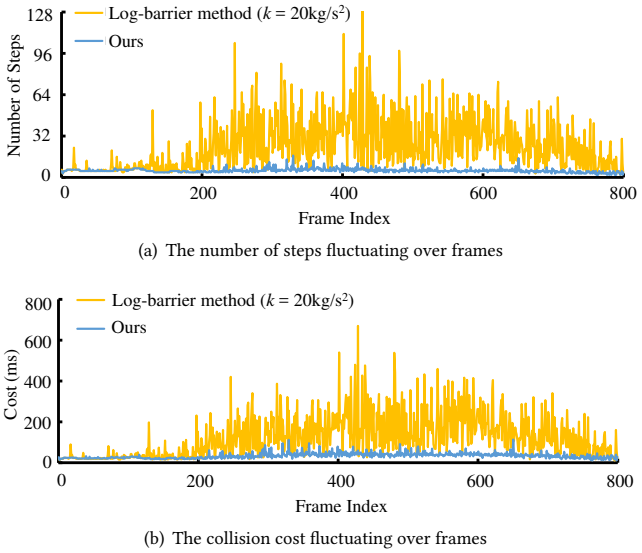
This joint method differs from the method proposed in [Li et al. 2020a] in that it treats interior point optimization as a separate forward step. As a result, it can apply different steppings to the two steps and achieve better runtime performance than the method in [Li et al. 2020a]. But since its backward step now handles both dynamics and contact constraints, the joint method still suffers from the incompatible stepping issue pointed out in [Wu et al. 2020]. Another issue is about the solver to the backward step. When the objective contains nonlinear dynamics, we can no longer convert the backward step into a linear complementarity problem and solve it efficiently as described in Section 4. Therefore we choose not to implement the joint method in practice.

3.3 Comparison to Impact Zone Optimizers

The main problem associated with impact zone optimization techniques [Harmon et al. 2008; Tang et al. 2018b] is that they can fail to converge, especially if the steppings are large. Tang et al. [2018b] found that their I-Cloth optimizer fails, when it simulates garments dressed on a dancing human body at Δt=1/20s. Our experiment shows I-Cloth fails to handle complex collision cases in knot examples, even at Δt=1/1000s. In comparison, our two-way method is safely terminable, regardless of the time step. Fig. 17 and 22 demonstrate our simulated examples at Δt=1/20s.

3.4 Comparison to a Log-Barrier Method

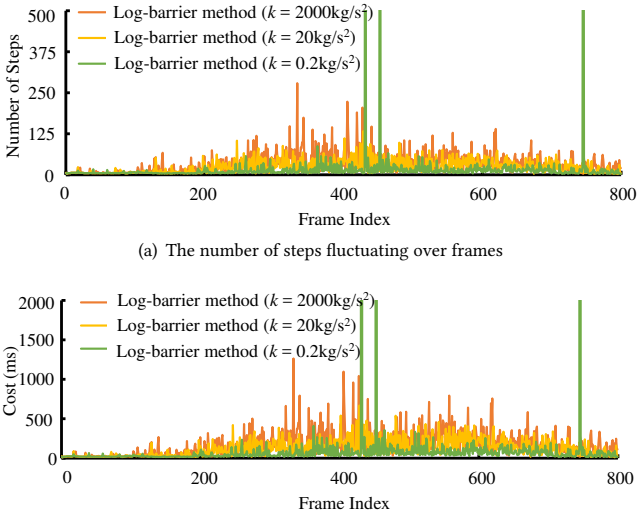
We cannot directly compare our simulator with the ones presented in [Li et al. 2020a; Wu et al. 2020], due to their many differences. Fortunately since their simulators are fundamentally based on log-barrier interior point methods, we implement one in our simulator as well and we compare its performance with that of our method. To implement the log-barrier method, we eliminate the backward



(a) The number of steps fluctuating over frames

(b) The collision cost fluctuating over frames

Fig. 5. The performances of our two-way method and the log-barrier method with its strength constant k = 20kg/s^2. While both methods are safe, our method uses larger step sizes and fewer steps to reach its convergence as (a) shows. As a result, our method runs approximately five times faster. By default, we use the bow knot example for evaluations in this paper.



(a) The number of steps fluctuating over frames

(b) The collision cost fluctuating over frames

Fig. 6. The influence of the strength constant k on the performance of the log-barrier method. Unlike our method, the log-barrier method needs a suitable value of k to achieve its optimal performance. In our experiment, we test several choices and decide to use k = 20kg/s^2 by default.

step and insert log-barrier functions into the forward objective:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \left\{ \left\| \mathbf{x} - \mathbf{y}^{[k+1]} \right\|^2 + \sum_{p} B_p(\mathbf{x}) \right\}, \tag{7}$$

in which $B_{\mathscr{p}}(\mathbf{x})$ is the barrier function of a proximity pair $\mathscr{p}$. Similar to [Li et al. 2020a; Wu et al. 2020], we limit $B_p(\mathbf{x})$'s influence by a range parameter d:

$$B_{\mathscr{p}}(\mathbf{x}) = k(\mathtt{dist}_{\mathscr{p}}(\mathbf{x}) - d)^2 \max\Big(-\log(\mathtt{dist}_{\mathscr{p}}(\mathbf{x})/d),\ 0\Big), \quad (8)$$

where k is the barrier strength constant and $\mathtt{dist}_{\mathscr{p}}(\mathbf{x})$ is the distance function of $\mathscr{p}$. Li et al. [2020a] proposed to solve Eq. 7 by Newton's method, which is second-order convergent if $\mathbf{x}$ is close to the solution. In practice, Newton's method is often outperformed by other descent methods at the beginning of the optimization process, due to its large per-iteration cost [Bouaziz et al. 2014; Wang 2015]. Therefore we choose to apply the gradient descent method as in [Wu et al. 2020] and we adopt the same step size scheme described in Section 5.

Fig. 5 compares the performance of our two-way method with that of the log-barrier method in the bow knot example. It shows that the two-way method needs considerably fewer steps to converge and it is approximately five times faster. We think this is mostly because the log-barrier method is forced to use many small step sizes as shown in Fig. 5a, when $\mathbf{x}$ needs to hug the boundary of Ω during optimization.

We note that the choice of the strength constant k plays an important role in the performance of the log-barrier method. If k is too small, the optimization will get too close to the boundary and cause the steppings to be small. On the other hand, if k is too large, the log-barrier repulsion component will dominate the gradient and slow down the progress. Fig. 6 compares the use of different k values, when d =1mm. In comparison, our two-way method is insensitive to parameter choices.

4 THE BACKWARD STEP

Similar to other impact zone optimizers, our backward step tries to project $\mathbf{y}^{(0)}$ back to Ω. But since we do not require it to succeed, we can more intuitively view it as the modification of the target state for guiding the forward step. Inspired by [Harmon et al. 2008; Otaduy et al. 2009], we linearize the constraints $\mathbf{c}(\mathbf{x}) \geq \mathbf{0}$ and convert Eq. 2 to the following problem in the l+1-th step:

$$\mathbf{y}^{(l+1)} = \arg\min_{\mathbf{y}} \frac{1}{2}\left\|\mathbf{y}-\mathbf{y}^{(0)}\right\|_2^2, \quad \text{s.t. } \mathbf{c}\left(\mathbf{x}^{(l)}\right)+\mathbf{J}^{(l)}\left(\mathbf{y}-\mathbf{x}^{(l)}\right) \geq \mathbf{0}, \quad (9)$$

for $\mathbf{y}^{(l)}$ being its initialization and $\mathbf{J}^{(l)} = \partial\mathbf{c}(\mathbf{x}^{(l)})/\partial\mathbf{x}$ being the Jacobian at $\mathbf{x}^{(l)}$. Here we perform the linearization at $\mathbf{x}^{(l)}$, not $\mathbf{y}^{(l)}$, for two reasons: first, $\mathbf{x}^{(l)}$ is closer to the colliding configuration and it is a candidate of the solution to the collision handling process; second, when the linearization is at $\mathbf{x}^{(l)}$, we can use the same proximity search result in both forward and backward steps. To solve Eq. 9, we formulate the following Lagrangian:

$$\mathscr{L}(\mathbf{y}, \boldsymbol{\lambda}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{y}^{(0)}\right\|_2^2 - \left(\mathbf{c}\left(\mathbf{x}^{(l)}\right) + \mathbf{J}^{(l)}\left(\mathbf{y} - \mathbf{x}^{(l)}\right)\right)^{\mathsf{T}}\boldsymbol{\lambda}, \quad (10)$$

whose minimizer satisfies the following KKT conditions:

$$\begin{cases} \nabla_{\mathbf{y}}\mathscr{L} = \mathbf{y} - \mathbf{y}^{(0)} - \left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} = \mathbf{0}, \\ \boldsymbol{\lambda} \geq 0 \perp \mathbf{c}\left(\mathbf{x}^{(l)}\right) + \mathbf{J}^{(l)}\left(\mathbf{y} - \mathbf{x}^{(l)}\right) \geq \mathbf{0}. \end{cases} \quad (11)$$

Multiplying the first condition in Eq. 11 with $\mathbf{J}^{(l)}$, we get:

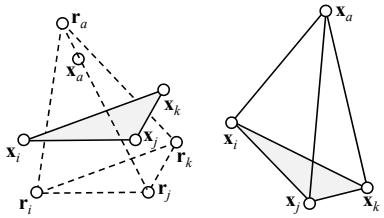
$$\mathbf{J}^{(l)}\mathbf{y} = \mathbf{J}^{(l)}\left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{J}^{(l)}\mathbf{y}^{(0)}, \quad (12)$$

and we obtain a linear complementarity problem (LCP) with only one unknown $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} \geq \mathbf{0} \perp \mathbf{c}\left(\mathbf{x}^{(l)}\right) + \mathbf{J}^{(l)}\left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{J}^{(l)}\left(\mathbf{y}^{(0)} - \mathbf{x}^{(l)}\right) \geq \mathbf{0}. \quad (13)$$

Once we get $\boldsymbol{\lambda}$, we apply the first condition of Eq. 11 to calculate the new target $\mathbf{y}^{(l+1)}$. Note that $\mathbf{y}^{(l)}$ is the initialization to the problem in Eq. 9 and it is calculated from the last $\boldsymbol{\lambda}$ in the l-th step, so we treat the last $\boldsymbol{\lambda}$ as the initialization to Eq. 13.

4.1 Contact Constraints



(a) A vertex-triangle pair (b) A deformed state

Fig. 8. A vertex-triangle pair. Our constraint enforces the volume of $\mathbf{x}_a\mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ to become the same as the volume of its projected version, in which the vertex and the triangle are separated by a desired distance δ.

An interesting question is how to define the contact constraints $\mathbf{c}(\mathbf{x}) \geq \mathbf{0}$ for a variety of primitive proximity pairs. In our method, we construct our contact constraints in a volume enforcement fashion.

To begin with, we consider a vertex-triangle proximity pair in Fig. 8a, whose distance is below a certain activation threshold δ. In our experiment, δ=1mm. Let $\{\mathbf{r}_a, \mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k\}$ be its projection, calculated by moving the vertex and the triangle in opposite normal directions until the distance becomes δ:

$$\begin{cases} \mathbf{r}_a &= \mathbf{x}_a &+ \frac{1}{2}\left(\delta - \left\|\mathbf{x}_a - b_i\mathbf{x}_i - b_j\mathbf{x}_j - b_k\mathbf{x}_k\right\|\right)\mathbf{n}, \\ \mathbf{r}_{i,j,k} &= \mathbf{x}_{i,j,k} &- \frac{1}{2}\left(\delta - \left\|\mathbf{x}_a - b_i\mathbf{x}_i - b_j\mathbf{x}_j - b_k\mathbf{x}_k\right\|\right)\mathbf{n}, \end{cases} \quad (14)$$

where b_i, b_j and b_k are the barycentric weights of the vertex on the triangle, and $\mathbf{n}$ is the constant triangle normal. Under the assumption that the triangle area is constant, we define the contact constraint by requiring the volume of $\mathbf{x}_a\mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ to be greater than or equal to the volume of $\mathbf{r}_a\mathbf{r}_i\mathbf{r}_j\mathbf{r}_k$:

$$c(\mathbf{x}_a, \mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) = \det(\partial\mathbf{x}/\partial\mathbf{r}) - 1 \geq 0, \quad (15)$$

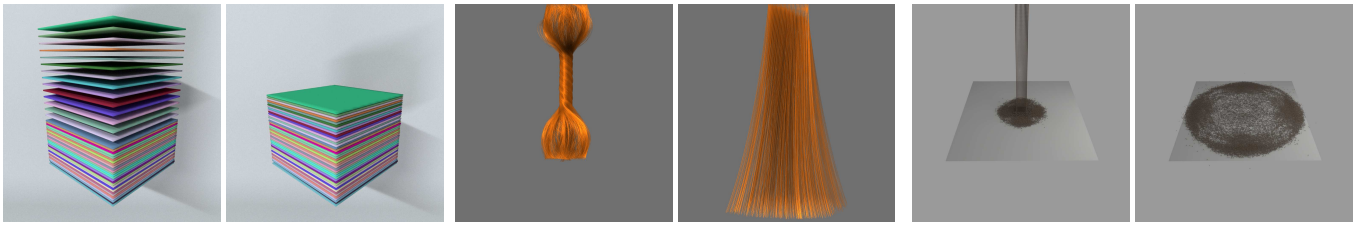
in which $\mathbf{r}_a$, $\mathbf{r}_i$, $\mathbf{r}_j$ and $\mathbf{r}_k$ are treated as constants.

Based on the same idea, we model the contact constraints for other simplex pairs, including edge-edge pairs, and vertex-vertex pairs. In the simplest case, the contact constraint for a vertex-vertex pair is:

$$c(\mathbf{x}_a, \mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{x}_a\| / \delta - 1 \geq 0. \quad (16)$$

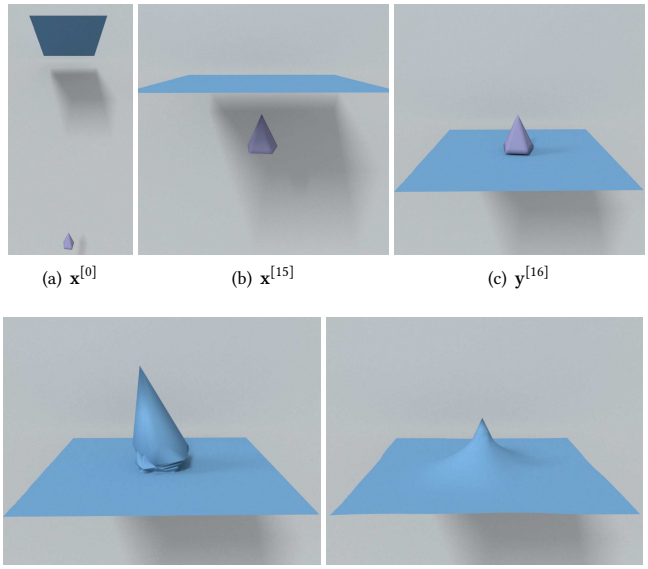
These constraints enable our method to handle contacts for a variety of codimensional deformable body examples, such as cloth, hair (in Fig. 7c and 7d) and sand (in Fig. 7e and 7f).

The key difference between our constraints and others [Harmon et al. 2008; Tang et al. 2018b] is that ours do not make strong assumptions on primitive position or orientation, which allows the backward step to achieve the projection goal more easily. Sifakis et al. [2008] explored a similar idea, but they chose to preserve the



(a) Falling mats (b) Stacked mats (c) Twisted hair (d) Loose hair (e) Falling sand (f) Piled sand

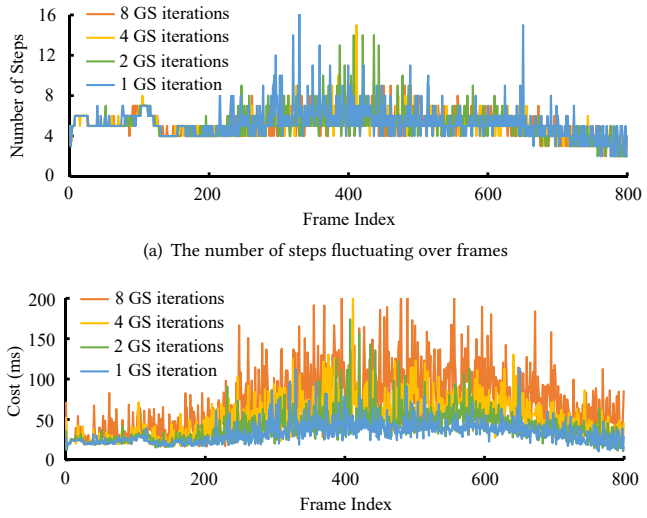
Fig. 7. Codimensional deformable body examples. Based on the volume enforcement idea, we develop contact constraints for a wide range of primitive proximity pairs. These contact constraints allows our method to simulate various codimensional examples, including elastic mats in (a) and (b), cloth, hair in (c) and (d), and sand in (e) and (f).



(a) $\mathbf{x}^{[0]}$ (b) $\mathbf{x}^{[15]}$ (c) $\mathbf{y}^{[16]}$

(d) $\mathbf{x}^{[16]}$ without edge length constraints (e) $\mathbf{x}^{[16]}$ with edge length constraints

Fig. 9. The results without and with edge length constraints. Without edge length constraints, the backward step is unable to prevent bodies from being severely deformed, such as cloth falling onto a needle shown in (d). By using edge length constraints, we eliminate this issue as shown in (e).



(a) The number of steps fluctuating over frames

(b) The collision cost fluctuating over frames

Fig. 10. The performances of our method with different numbers of Gauss-Seidel iterations per backward step. By spending more Gauss-Seidel iterations, we can solve the LCP problem more exactly in every backward step. But doing this does not necessarily decrease the number of steps, as (a) shows. Overall the most efficient practice is to use one Gauss-Seidel iteration per step.

volume, rather than enforce the volume to a desired value. In comparison, our constraints can keep pairs well separated, so that fewer collisions will occur in later updates.

4.1.1 Edge length constraints. When the objective of the backward step is to minimize the difference between $\mathbf{y}$ and $\mathbf{y}^{(0)}$ only, it is insufficient to prevent $\mathbf{y}$ from being overly deformed as shown in Fig. 9a, causing jittering artifacts in simulation. We can incorporate additional deformation resistance terms into the objective, but that destroys its simplicity and voids the aforementioned LCP formulation. Instead we introduce an additional edge length constraint to resist the deformation of each edge:

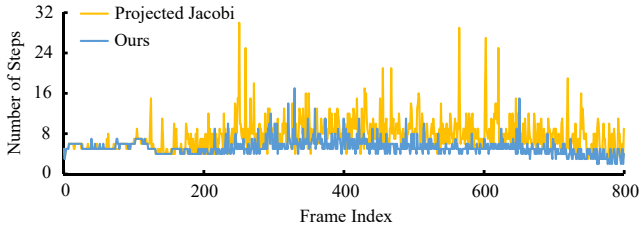
$$c(\mathbf{x}_i, \mathbf{x}_j) = \sigma - \left\|\mathbf{x}_i - \mathbf{x}_j\right\| \Big/ \left\|\mathbf{x}_i^{(0)} - \mathbf{x}_j^{(0)}\right\| \geq 0, \tag{17}$$

in which i and j are the two vertex indices and σ is the maximum stretching ratio. Fig. 9 compares the results without and with edge length constraints.

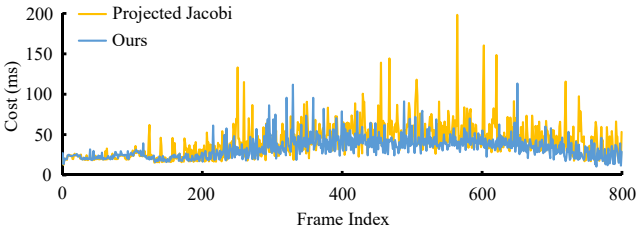
4.2 An Inexact GPU-Based Optimizer

A popular way of solving the LCP problem in Eq. 13 is to apply a projected iterative method [Erleben 2013], which enforces $\boldsymbol{\lambda} \geq \mathbf{0}$ after every iteration solving the linear system. In our simulator, we use multi-color Gauss-Seidel [Fratarcangeli et al. 2016] as our method, with colors being assigned to constraints.

Given the contact constraint set $\mathbf{c}_{\mathscr{C}}(\mathbf{x}) = \{c_i, i = 0, \cdots, M-1\}$ where M is the number of contact constraints, each $c_i(x)$ associated with a contact pair is regarded as a node on a graph $\mathscr{G}$, if two different contact constraints $c_i(x)$ and $c_j(x)$ share the same vertex of the input mesh, they are connected by an edge on the graph $\mathscr{G}$.



(a) The number of steps fluctuating over frames



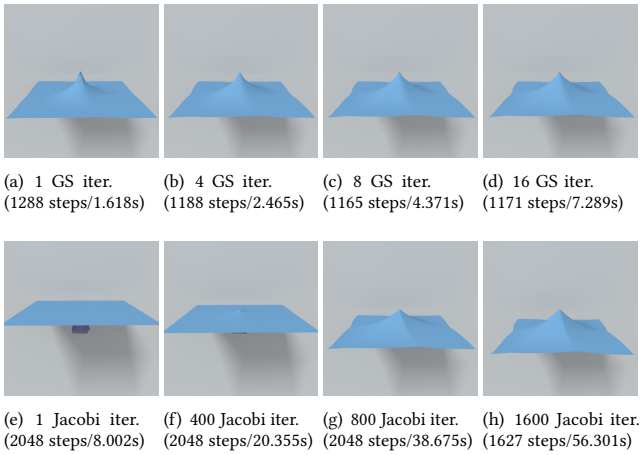
(b) The collision cost fluctuating over frames

Fig. 11. The performances of our method implemented with the projected Jacobi method and the projected Gauss-Seidel method (as ours). Compared with ours, the method implemented with projected Jacobi needs more steps to converge, resulting in more computational costs as shown in (b).

By the random graph coloring method [Fratarcangeli et al. 2016], the contact constraint set is partitioned into p colors, such that the arbitrary two different contact constraints $c_i(x)$ and $c_j(x)$ with the same color do not share the same vertex of the input mesh.

Because the edge length constraint set $\mathbf{c}_{\mathscr{E}}(\mathbf{x})$ is constant, we precompute the colors of this set for performance consideration in the graph-building and coloring step. When we use the colors for Gauss-Seidel, we append the colors for $\mathbf{c}_{\mathscr{C}}(\mathbf{x})$ after the precomputed colors for $\mathbf{c}_{\mathscr{E}}(\mathbf{x})$. In our experiment, the number of colors for for contact constraint set $\mathbf{c}_{\mathscr{C}}(\mathbf{x})$ is typically between 50 to 300 and the number of colors for edge length constraint set $\mathbf{c}_{\mathscr{E}}(\mathbf{x})$ is typically between 9 to 15. We note that position-based dynamics [Macklin et al. 2014; Müller 2008] implemented in a Gauss-Seidel fashion also uses constraints for coloring and parallelization, but it treats vertices as variables. In contrast, our method is a standard multi-color Gauss-Seidel method, using constraints for both coloring and variables.

4.2.1 Inexactness. One interesting question is how many iterations should we spend on solving the LCP problem? The more iterations we use, the more exactly we get the problem solved. But given the fact that we face a new LCP problem in the next backward step, it will be a waste if we spend too much computational cost on a single problem. Ultimately, our choice should be based on the total collision cost, fundamentally determined by two factors: the total number of steps for reaching the convergence and the costs associated with backward and forward steps. According to Fig. 10, increasing the number of Gauss-Seidel iterations negatively affects the overall performance. Therefore, we choose to use a single iteration per backward step by default.



(a) 1 GS iter. (1288 steps/1.618s) (b) 4 GS iter. (1188 steps/2.465s) (c) 8 GS iter. (1165 steps/4.371s) (d) 16 GS iter. (1171 steps/7.289s)

(e) 1 Jacobi iter. (2048 steps/8.002s) (f) 400 Jacobi iter. (2048 steps/20.355s) (g) 800 Jacobi iter. (2048 steps/38.675s) (h) 1600 Jacobi iter. (1627 steps/56.301s)

Fig. 12. In the same setting as Fig. 9, we find much more projected Jacobi iterations needed to get a comparable result as projected Gauss-Seidel. We set L=2048 and we find even 800 projected Jacobi iterations in each backward step can not get the result of convergence.

4.2.2 A projected Jacobi implementation. The analysis in Subsection 4.2.1 motivates us to consider an even more inexact implementation, i.e., replacing one projected Gauss-Seidel iteration by one projected Jacobi iteration. After testing this implementation, we conclude that it is not a suitable choice for two reasons. First, projected Jacobi needs an under-relaxation factor to ensure its convergence, which further lowers the convergence rate. Second, the nonsmooth nature of our method makes inter-step Chebyshev acceleration [Wang 2015] ineffective. Overall, the method with projected Jacobi needs more steps and more collision costs as Fig. 11 shows. For more, under some extreme conditions, we need run much more projected Jacobi iterations to get a comparable result as projected Gauss-Seidel, at least one order slower as shown in Fig. 12.

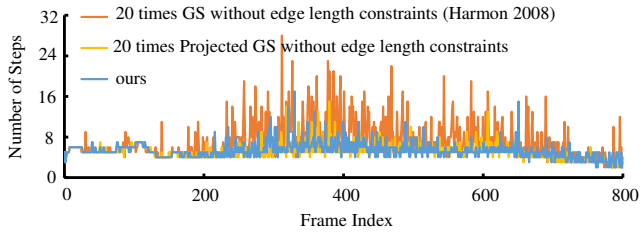
4.3 Comparison to Other Impact Zone Optimizers

We can adopt other impact zone optimizers [Harmon et al. 2008; Tang et al. 2018b] to perform the projection task in our backward step as well. In this subsection, we would like to evaluate their performances in our two-way method, with everything else being the same. We note that we cannot use these optimizers for collision handling alone, or they can run into convergence issues as discussed in Subsection 3.3.

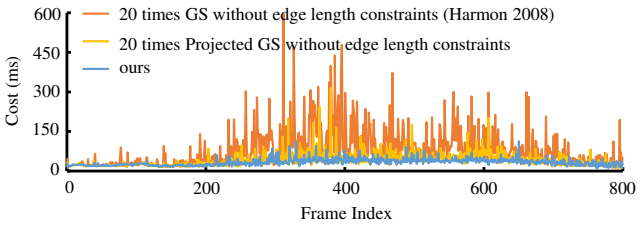
4.3.1 Comparison to [Harmon et al. 2008]. Both our optimizer and the one presented in [Harmon et al. 2008] solve Lagrangian multipliers to achieve a collision-free state. The main difference is that:

Our optimizer solves the LCP problem by the projected Gauss-Seidel method. In contrast, Harmon et al. [2008] chose to ignore $\boldsymbol{\lambda} \geq \mathbf{0}$ and solve the rest of the problem as a linear system. This can potentially cause sticking artifacts.

Other differences include: a. they apply their formulation on velocity field but ours is applied to location field; b. they only consider the contact and friction constraints but ours also consider the edge

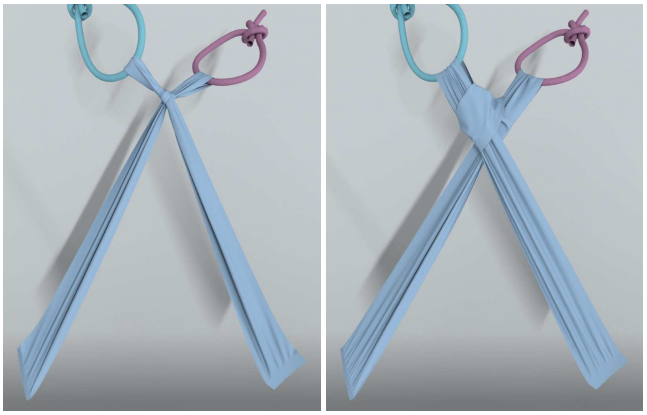


(a) The number of steps fluctuating over frames



(b) The collision cost fluctuating over frames

Fig. 13. The performances of our method with our own optimizer and the method proposed in [Harmon et al. 2008]. Compared with our method, it is less efficient.

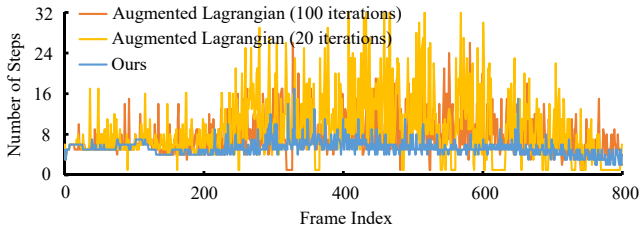


(a) our normal result (b) Harmon's method to enforce the edge length constraints as equality constraints

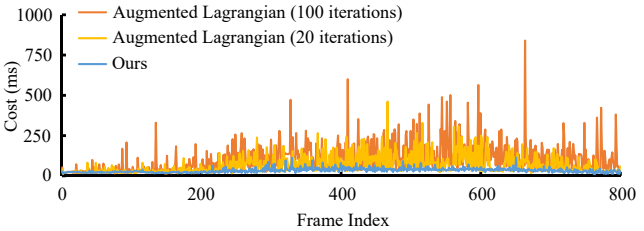
Fig. 14. Unlike our method, Harmon's approach is not compatible with the additional edge length conservation constraint which can cause stagnation artifacts.

length conservation constraints except for the contact constraints; c. they solve the linear system fully but ours only use one time coloring Projected Gauss-Seidel to evaluate $\boldsymbol{\lambda}$ quickly.

So a relative fair comparison can be done by fully solving the linear system in Eq. 13's right part without enforcing the complementarity condition. We should point out that this is not exactly the original method in [Harmon et al. 2008] because this modification is applied to location field, considering the additional edge length constraints as mentioned above but inheriting the full pure bilateral projection which we consider as the main differences. But a problem is that solving the linear system Eq. 13's right part will make



(a) The number of steps fluctuating over frames



(b) The collision cost fluctuating over frames

Fig. 15. The performances of our method with our own optimizer and the augmented Lagrangian optimizer proposed in [Tang et al. 2018b]. Since their optimizer converges significantly slower than ours, our method needs to run multiple iterations per backward step to reduce the total number of steps. Overall, our optimizer is more effective.

the edges' length almost constant which make the optimization stagnate as shown in Fig. 14. So we ignore additional edge length conservation constraints when implementing their method. In this setting, we compare our method with [Harmon et al. 2008].

Here we reuse the coloring Gauss-Seidel module to solve the linear system on GPU without enforcing $\boldsymbol{\lambda}$ to satisfy the complementarity condition. As expected, this will take a considerable amount of computational time because of a large number of linear systems to be solved for Alg. 1. There we fix the iteration number of Gauss-Seidel as 20 to avoid unnecessary computation and we find the iteration number can not be one as our method, otherwise the stagnation problem will happen similar as Fig. 14. We also consider a much simple modification by projecting $\boldsymbol{\lambda}$ to satisfy the complementarity condition. But even this tiny modification at the code level will make the optimization converge faster than the pure coloring Gauss-Seidel as shown in Fig. 13. So we underline that maintaining the complementarity condition correctly is much important for improving the convergence rate but in [Harmon et al. 2008] this condition is not being treated right.

4.3.2 Comparison to [Tang et al. 2018b]. Tang et al. [2018b] suggested the use of the augmented Lagrangian method together with gradient descent for their optimizer. The strength of their augmented Lagrangian optimizer is its simplicity: it does not need to solve any system for primal or dual variable, and the only major computational components are gradient and constraint evaluations. But since their optimizer converges considerably slower, our method with their optimizer must run multiple iterations per backward step to reduce the total number of steps. In our experiment, we implement their optimizer with two options: one running 20 iterations per step

and one running 100 iterations per step. Fig. 15 shows even when using 100 iterations per step, the method with their optimizer still needs a large number of steps. Overall our optimizer is the optimal choice for the method.

5 THE FORWARD STEP

As shown in Eq. 3, the forward step is a simple process advancing $\mathbf{x}$ from $\mathbf{x}^{(l)}$ toward $\mathbf{y}^{(l+1)}$. The key question is how to find the safe step size $\alpha_i^{(l+1)}$ for every vertex i, so that $\mathscr{X}\left(\mathbf{x}^{(l)}, \mathbf{x}^{(l+1)}\right) \subset \Omega$. One way of obtaining a safe step size is to use continuous collision detection (CCD). CCD tests calculate exact moments when proximity pairs intersect, using which we then determine how far $\mathbf{x}_i$ can travel. Unfortunately, previous works show that CCD tests are expensive, unreliable and difficult to implement on GPUs.

In our method, we propose to use an inexpensive yet reliable scheme for calculating the safe step size. Our key idea is based on the simple fact that a proximity pair cannot intersect, if none of its vertices moves more than half of its distance. To do so, the method needs a set of proximity pairs $\mathscr{P}$, in which each pair contains two non-adjacent simplices whose distance is below a uniform certain threshold $D^{(l)}$. Given $\mathscr{P}$, the method calculates D_i which is the shortest distance of the proximity pairs involving vertex i:

$$D_i = \min_{\{a,\, b\}} \mathtt{dist}\left(\mathbf{x}_a^{(l)}, \mathbf{x}_b^{(l)}\right) \le D^{(l)}, \forall a, b : a \neq b \text{ and } i \in a \cup b. \quad (18)$$

in which a and b are the two non-adjacent simplices and $\mathtt{dist}\left(\mathbf{x}_a^{(l)}, \mathbf{x}_b^{(l)}\right)$ is the distance function between them. We treat $D_i/2$ as an upper bound on the displacement of vertex i to ensure the intersection-free condition:

$$\forall i : \left\|\mathbf{x}_i^{(l)} - \mathbf{x}_i^{(l+1)}\right\| < D_i/2 \Rightarrow (1-t)\mathbf{x}^{(l)} + t\mathbf{x}^{(l+1)} \in \Omega, \quad (19)$$

for any $t \in [0, 1]$. Therefore our forward step updates the position of vertex i as:

$$\begin{aligned} \alpha_i^{(l+1)} &= \min\left(0.5\gamma D_i \Big/ \left\|\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right\|, 1\right), \\ \mathbf{x}_i^{(l+1)} &= \mathbf{x}_i^{(l)} + \alpha_i^{(l+1)}\left(\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right), \\ r_i &= r_i(1 - \alpha_i^{(l+1)}) \end{aligned} \quad (20)$$

in which γ is a damping factor preventing proximity pairs from getting too close in a single forward step. Using this step size calculated for every vertex, $\mathbf{x}^{(l+1)}$ must be an acceptable intermediate state, regardless of the search direction. We assign a scalar r_i to each vertex which works as a "clock" to measure how far it has to go to reach its destination so far. At the beginning of optimization, they are all set to 1 which means 100 percent of path needed to go and then they asynchronously decrease to be sufficiently small under the given threshold $\epsilon = 1e-4$ altogether which means the optimized result is sufficiently close to the target position.

Compared with CCD tests, distance evaluations used by our scheme are computationally inexpensive, reliable against floating-point errors and easy to parallelize on GPUs.

5.1 Comparison to CCD-Based Schemes

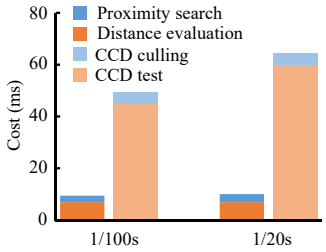


Fig. 16. The total costs of CCD-free and CCD-based step size schemes spent in the collision handling process. Overall, the CCD-free scheme is five to seven times faster.

To compare our CCD-based step size scheme with CCD-based schemes, we implement a CCD-based scheme by the tests provided by I-Cloth [Tang et al. 2018b], which is one of the fastest CCD implementations on a GPU. We also adjust both schemes to use the same proximity search tool provided by I-Cloth, so that we can eliminate the difference in proximity search implementations. On the same NVIDIA GeForce GTX 2080 Ti GPU, we test both schemes in several examples with two time steps: Δt=1/100s and Δt=1/20s.

In the examples with Δt=1/100s, the experiment shows our proximity search cost is about 60 percent of the broad-phase CCD culling cost, while the distance evaluation cost is about 15 percent of the narrow-phase CCD test cost. Together our CCD-free scheme is five times faster than the CCD-based scheme.

In the examples with Δt=1/20s, the narrow-phase CCD test cost increases significantly while the distance evaluation cost increases marginally. As a result, our CCD-free scheme is about seven times faster than the CCD-based scheme.

We note that the computational costs of the two step size schemes alone do not provide the full picture of their difference. In general, our CCD-free scheme provides smaller step sizes and causes 10 to 20 percent more steps needed for convergence. This further increases the cost of the backward step by 10 to 20 percent. But overall, it is still beneficial to use the CCD-free scheme, given the large computational cost needed for CCD tests.

5.2 Comparison to the Scheme in [Wu et al. 2020]

The step size scheme used by the hard phase in [Wu et al. 2020] is also based on CCD tests. Compared with other CCD-based schemes, their scheme is unique in that it handles vertex-vertex contacts only. Because of this, their CCD tests are inexpensive and their scheme adopts backtracking line search. If their scheme handles vertex-triangle or edge-edge contact, it should be expensive just like other CCD-based schemes.

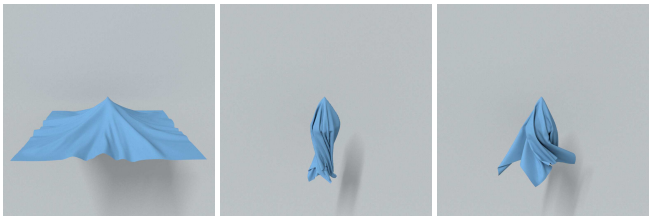
6 IMPLEMENTATION DETAILS

In this section, we would like to discuss the issues relevant to the implementation of our two-way interior point method in a GPU-based deformable body simulator.

6.1 Dynamics Solvers

Since our method works as a standalone collision handling process, typically as post-processing, it is naturally compatible with most of the dynamics solvers. In our simulator, we use Newton's method to solve the nonlinear optimization problem stemmed from deformable body dynamics [Bouaziz et al. 2014; Martin et al. 2011], and we apply the conjugate gradient method with a block Jacobian preconditioner

(a) The colliding state (one Newton iteration) (b) The swinging state (one Newn iteration) (c) The draping state (one Newton iteration)



(d) The colliding state (four Newton iterations) (e) The swinging state (four Newton iterations) (f) The draping state (four Newton iterations)

Fig. 17. The simulation results with different numbers of Newton iterations. Our simulator solves deformable body dynamics more accurately when it runs more Newton iterations, as shown in (d)-(f). But even if it does not, our two-way method is still able to safely handle collisions.

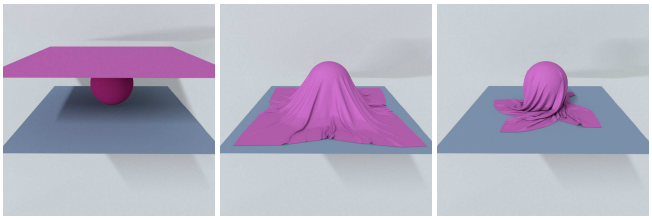
to solve the linear system in every Newton iteration. We treat the Newton iteration as an update and run our two-way interior point method for collision handling right afterwards. This practice is more physically plausible than performing collision handling only once after all of the Newton iterations. Currently, the implementation of our simulator uses CUDA 11.2 and the CUB library for reduction and sorting operations.

In our implementation, we fix the number of Newton iterations as a constant. Ideally, this number is related to the time step: the solver should run more Newton iterations as the time step increases, for more accurate dynamics results. But even if we choose to run a single Newton iteration when the time step is large, i.e., Δt=1/20s, our method still runs robustly as Fig. 17 shows.

6.2 Elastic and Repulsive Models

To simulate codimensional deformable body examples shown in Fig. 7, we provide a number of elastic models to our dynamics solver. This includes the StVK model for tetrahedral meshes, the co-rotational linear model and the quadratic bending model [Bergou et al. 2006] for triangular meshes, and the mass-spring model for hair strands.

Similar to many other simulators, our simulator incorporates a quadratic-energy-based repulsive model into deformable body dynamics, to reduce the collision complexity in simulation. This model uses the proximity search result calculated by our method in the collision handling process, so its computational overhead is small. Without this repulsive model, our method still handles collisions safely and accurately, but less efficiently.



(a) The initial state (b) The falling state (c) The sliding state

Fig. 18. A sphere example. When a square cloth patch falls onto a rotating sphere, it forms multiple folds and wrinkles due to its frictional contacts with the sphere and the ground floor.

Table 2. The statistics and the performances of our examples. The last column here lists the total collision cost spent by our two-way method in a single time step.

Name (#verts., #edg./#tri./#tet., ref.)	Time Step (s)	Avg. (Max.) # of Steps	Avg. (Max.) Collision Cost (s)
Needle (10k, 20k, Fig. 17a)	1/20	46 (1304)	0.036 (1.942)
Blade (59k, 116k, Fig. 22)	1/20	55 (779)	0.267 (4.463)
Funnel (49K, 98K, Fig. 19a)	1/20	27 (675)	0.087 (2.161)
Funnel (49K, 98K, Fig. 19b)	1/40	23 (437)	0.055 (1.198)
Funnel (49K, 98K, Fig. 19c)	1/80	12 (165)	0.040 (0.776)
Funnel (49K, 98K, Fig. 19d)	1/160	10 (95)	0.029 (0.296)
Sphere (50k, 100k, Fig. 18)	1/100	19 (219)	0.044 (0.475)
Dress (30K, 60K, Fig. 21a)	1/100	39 (142)	0.133 (0.511)
Gown (27K, 51K, Fig. 21c)	1/100	32 (150)	0.092 (0.443)
Bow knot (71K, 142K, Fig. 1a)	1/100	5.4 (17)	0.034 (0.113)
Reef knot (37K, 71K, Fig. 1e)	1/100	7 (23)	0.024 (0.097)
Tube (25k, 51K, Fig. 23)	1/100	19 (54)	0.308 (1.500)
Mat (33k, 88K, Fig. 7a)	1/100	9 (128)	0.020 (0.303)
Hair (63K, 62k, Fig. 7c)	1/100	16 (503)	0.252 (15.821)
Sand (30K, - , Fig. 7e)	1/100	51 (160)	0.223 (1.550)

6.3 Frictional Contacts

We adopt the velocity filtering approach proposed in [Bridson et al. 2002] to handle frictional contacts. First we run the dynamics solver to calculate the target state $\mathbf{y}^{[k+1]}$ with no friction. We then calculate penetration depths to estimate collision impulses and use them to determine corresponding frictional impulses by Coulomb's law. Finally we update $\mathbf{y}^{[k+1]}$ by both impulses and treat that as the new target state for collision handling. We note that the velocity filtering strategy is more suitable in deformable–rigid body contacts, than in self body contacts, since penetration depth estimations are inaccurate and irrelevant to the actual collision handling process, especially if the time step is large. Fig.18 demonstrates our current frictional effects of cloth in the sphere example.

7 RESULTS AND DISCUSSIONS

(Please watch the supplemental video for our animation examples.) We evaluate our simulator on an Intel Core i5-7500 3.4GHz CPU and an NVIDIA GeForce GTX 2080 Ti GPU. Table 2 summaries the statistics and the performances of our examples, and Table 3 provides the key variables and their values used by our simulator. In general, the collision cost of an example depends on two factors:

Table 3. The key variables and their values.

Symbol	Meaning	Value
L	The limit on the number of steps	512
$D^{\min}$	The proximity search lower bound	2mm
$D^{\max}$	The proximity search upper bound	4mm
δ	The constraint activation threshold	1mm
σ	The stretching ratio limit	1.1
γ	The damping factor on vertex movement	0.9

the number of colliding elements and the number of steps, both of which depend on two other hidden factors: the time step and the collision complexity. By the default, we set the maximum number of steps at L=512 with a time step 0.01s, and L=2048 with a larger time step 0.05s. In reality, our method is able to converge within 64 steps most of the time, as shown in Table 2.

7.1 Sensitivity to Time Steps

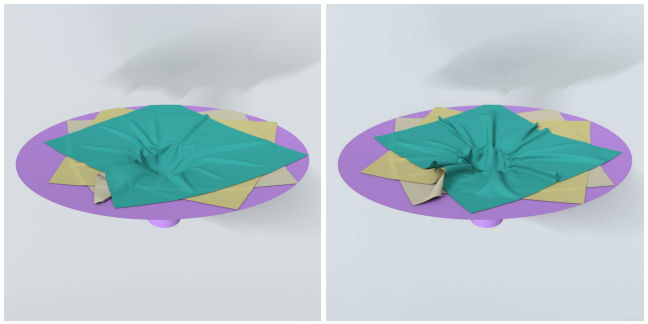
While our method is safe and robust regardless of the time step, its performance per time step drops as the time step increases. Fundamentally, this is due to severer collision cases and more steps needed for the method to converge. In our experiment, we test the funnel example with four time steps: Δt=1/160s, Δt=1/80s, Δt=1/40s and Δt=1/20s, and we intentionally set L=65536 so that we can see how many steps are needed for convergence. Fig. 19 shows our result at the 0.5s state simulated with different time steps and Table 2's second part shows the statistics and the performance of these four simulated results on the same funnel example. We note that as the time step decreases, there is less numerical damping as shown in Fig. 19, but fundamental phase of simulation is similar under different time steps. Furthermore, from Table. 2, we can note that the performance per time step drops as the time step increases, but from the computational efficiency of the whole simulation, the larger time step is still more efficient.

7.2 Breakdown Analysis

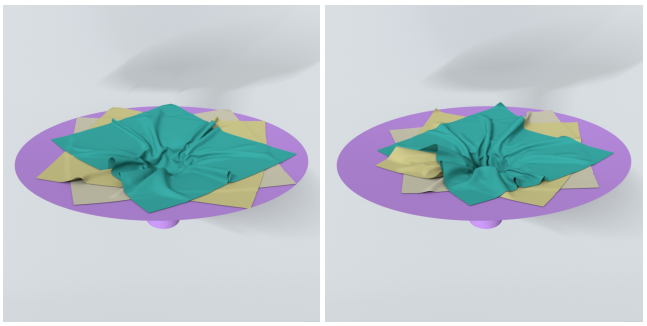
Fig. 20 provides a breakdown of the computational cost in the bow knot simulation during 10.0s with a time step 0.01s by implicit Euler time integration which is equivalent to running a single Newton iteration [Martin et al. 2011]. It show the proximity search to find contact constraints and to update its Jacobian matrix takes almost half of the total time cost. Thanks to our pre-computation of the edge constraints' colors, the graph building and coloring of the contact constraints do not consume too much cost. We would like to note that this breakdown can fluctuate during different benchmarks due to several factors like the coupling degree of contact constraints, mesh resolution, time step and boundary condition.

7.3 Limitations

The use of the threshold ϵ not only activates contact constraints, but also specifies the distance goals for contact constraints to reach. Doing this improves the convergence rate of our method, by keeping proximity pairs sufficiently apart. But that can also cause inexactness in collision handling, which often gets demonstrated as early collision artifacts if ϵ is too large. For solving the backward step



(a) Δt=1/20s (b) Δt=1/40s



(c) Δt=1/80s (d) Δt=1/160s

Fig. 19. Our results with different time steps.

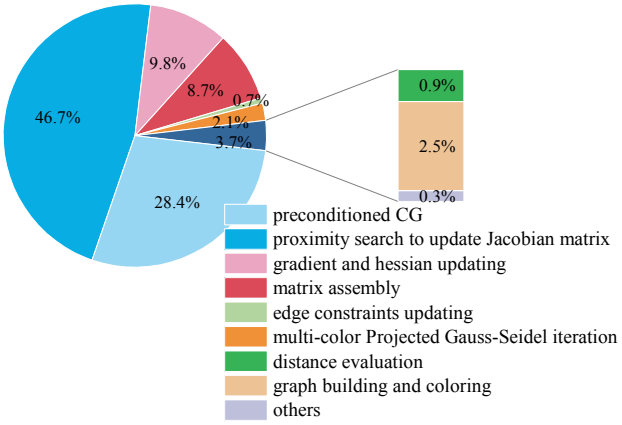
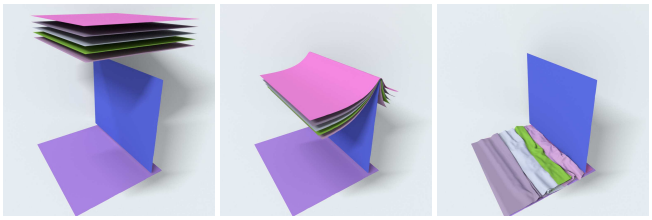


Fig. 20. A typical breakdown of the collision cost spent by our method in the bow knot example.

fast, our method assumes that the simulator handles dynamics and collisions in two separate processes. When the time step is large, the simulator must use sub-steps or interleave dynamics solvers and collision handling as shown in Fig. 17, so as to reduce inaccuracy in collision responses. Finally, our method does not consider friction and our simulator needs an additional velocity filtering process to produce frictional effects. For more plausible frictional contacts, the method should handle collisions and frictions jointly instead.

(a) A multi-layered dress (b) A multi-layered dress (c) A multi-layered gown (d) A multi-layered gown

Fig. 21. Dress and gown examples. Our method is capable of handling collisions among multiple layers of cloth, dressed by human characters performing various motions as shown in this figure.



(a) The falling state (b) The colliding state (c) The stacking state

Fig. 22. A blade example. In this example, five square cloth patches drop onto a sharp blade, then they drape, slide and stack under gravity. Our method robustly handles these collisions with a large time step Δt=1/20s.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we show that the ideas of impact zone optimization and interior point optimization can be combined into a novel two-way interior point method for safe, efficient and accurate collision handling. Instead of using barrier function or primal-dual formulation like other interior point methods, our method uses impact zone optimization to drive the interior point search. Instead of using CCD tests on GPU, our method uses the distance evaluation to get the asynchronous upper bound of displacement for the interior point guarantee. These differences enable our method to accomplish the search through safe yet simple and efficient forward steps.

Looking into the future, our immediate interest is to incorporate frictional contact models into our method as in [Li et al. 2020a]. We then would like to test our method running on multiple GPUs and study how this practice affects our implementation. Inspired by the splitting approach [Wu et al. 2020], we will investigate better ways for dynamics solvers and collision handling to be integrated. Finally, we plan to explore the use of our method in solving intersection-free geometric optimization problems, such as injective normal flow and animation repair [Fang et al. 2021].

REFERENCES

Samantha Ainsley, Etienne Vouga, Eitan Grinspun, and Rasmus Tamstorf. 2012. Speculative Parallel Asynchronous Contact Mechanics. *ACM Trans. Graph. (SIGGRAPH)* 31, 6, Article 151 (Nov. 2012), 8 pages.

David Baraff, Andrew Witkin, and Michael Kass. 2003. Untangling Cloth. *ACM Trans. Graph. (SIGGRAPH)* 22, 3 (July 2003), 862–870.

Jernej Barbič and Doug L. James. 2010. Subspace Self-Collision Culling. *ACM Trans. Graph. (SIGGRAPH)* 29, 4, Article 81 (July 2010), 9 pages.

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A Quadratic Bending Model for Inextensible Surfaces. In *Proceedings of SGP*. 227–230.

Florence Bertails-Descoubes, Florent Cadoux, Gilles Daviet, and Vincent Acary. 2011. A Nonsmooth Newton Solver for Capturing Exact Coulomb Friction in Fiber Assemblies. *ACM Trans. Graph.* 30, 1, Article 6 (Feb. 2011), 14 pages.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, Article 154 (July 2014), 11 pages.

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust Treatment of Collisions, Contact and Friction for Cloth Animation. *ACM Trans. Graph. (SIGGRAPH)* 21, 3 (July 2002), 594–603.

Robert Bridson, Sebastian Marino, and Ronald Fedkiw. 2003. Simulation of Clothing with Folds and Wrinkles. In *Proceedings of SCA*. 28–36.

Gilles Daviet. 2020. Simple and Scalable Frictional Contacts for Thin Nodal Objects. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 61 (July 2020), 16 pages.

Kenny Erleben. 2013. Numerical Methods for Linear Complementarity Problems in Physics-Based Animation. In *ACM SIGGRAPH 2013 Courses*. Association for Computing Machinery, New York, NY, USA.

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed Globally Injective 3D Deformation Processing. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 75 (July 2021), 13 pages.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: A Practical Gauss-Seidel Method for Stable Soft Body Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 35, 6, Article 214 (Nov. 2016), 9 pages.

David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous Contact Mechanics. *ACM Trans. Graph. (SIGGRAPH)* 28, 3, Article 87 (July 2009), 12 pages.

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2008. Robust Treatment of Simultaneous Collisions. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (Aug. 2008), 1–4.

David Harmon, Qingnan Zhou, and Denis Zorin. 2011. Asynchronous Integration with Phantom Meshes. In *Proceedings of SCA*. 247–256.

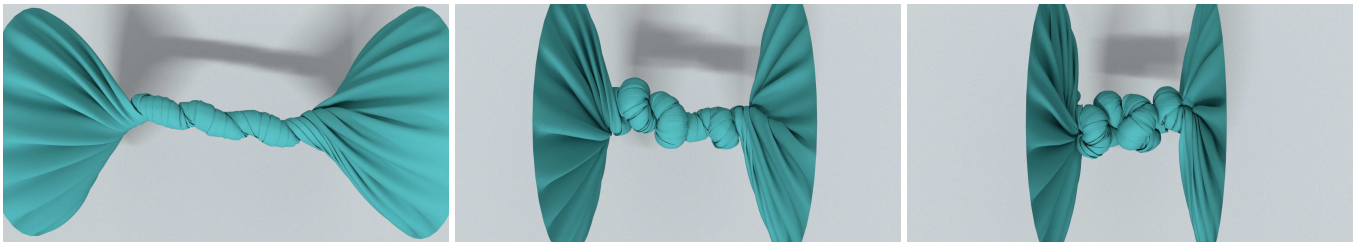
Christian Lauterbach, Qi Mo, and Dinesh Manocha. 2010. gProximity: Hierarchical GPU-Based Operations for Collision and Distance Queries. In *Proceedings of Eurographics*, Vol. 29. 419–428.

Cheng Li, Min Tang, Ruofeng Tong, Ming Cai, Jieyi Zhao, and Dinesh Manocha. 2020b. P-Cloth: Interactive Complex Cloth Simulation on Multi-GPU Systems Using Dynamic Matrix Assembly and Pipelined Implicit Integrators. *ACM Trans. Graph. (SIGGRAPH Asia)* 39, 6, Article 180 (Nov. 2020), 15 pages.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020a. Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 49 (July 2020), 20 pages.

Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 170 (July 2021), 24 pages.

Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. 2020. Projective Dynamics with Dry Frictional Contact. *ACM Trans. Graph. (SIGGRAPH)*



(a) The initial state (b) The twisted state (c) The final state

Fig. 23. A tube example. When a compliant cloth tube is severely squeezed, it experiences intense and frequent collisions in the middle as shown in (c). Our method robustly handles these collisions in this example with a Δt=1/100s time step.

39, 4, Article 57 (July 2020), 8 pages.

Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Viktor Makoviychuk. 2019. Non-Smooth Newton Methods for Deformable Multi-Body Dynamics. *ACM Trans. Graph.* 38, 5, Article 140 (Oct. 2019), 20 pages.

Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified Particle Physics for Real-Time Applications. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, Article 153 (July 2014), 12 pages.

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-Based Elastic Materials. *ACM Trans. Graph. (SIGGRAPH)* 30, 4, Article 72 (July 2011), 8 pages.

Brian Mirtich. 2000. Timewarp Rigid Body Simulation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 193–200.

Brian Mirtich and John Canny. 1995. Impulse-Based Dynamic Simulation. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*. A. K. Peters, Ltd., USA, 407–418.

Matthias Müller. 2008. Hierarchical Position Based Dynamics. In *Proceedings of Virtual Reality Interactions and Physical Simulations*. Grenoble.

Miguel A Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. 2009. Implicit Contact Handling for Deformable Objects. In *Comput. Graph. Forum (Eurographics)*, Vol. 28. Wiley Online Library, 559–568.

Simon Pabst, Artur Koch, and Wolfgang Straßer. 2010. Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces. *Comput. Graph. Forum* 29, 5 (2010), 1605–1612.

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

Eftychios Sifakis, Sebastian Marino, and Joseph Teran. 2008. Globally Coupled Collision Handling Using Volume Preserving Impulses. In *Proceedings of SCA*. 147–153.

Jos Stam. 2009. Nucleus: Towards a Unified Dynamics Solver for Computer Graphics. In *11th IEEE International Conference on Computer-Aided Design and Computer Graphics*.

Min Tang, Young J. Kim, and Dinesh Manocha. 2010. Continuous Collision Detection for Non-Rigid Contact Computations using Local Advancement. In *Proceedings of ICRA*. 4016–4021.

Min Tang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018a. PSCC: Parallel Self-Collision Culling with Spatial Hashing on GPUs. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 18 (July 2018), 18 pages.

Min Tang, Dinesh Manocha, Sung-Eui Yoon, Peng Du, Jae-Pil Heo, and Ruo-Feng Tong. 2011. VolCCD: Fast Continuous Collision Culling between Deforming Volume Meshes. *ACM Trans. Graph.* 30, 5, Article 111 (Oct. 2011), 15 pages.

Min Tang, Ruofeng Tong, Zhendong Wang, and Dinesh Manocha. 2014. Fast and Exact Continuous Collision Detection with Bernstein Sign Classification. *ACM Trans. Graph. (SIGGRAPH Asia)* 33, 6, Article 186 (Nov. 2014), 8 pages.

Min Tang, Huamin Wang, Le Tang, Ruofeng Tong, and Dinesh Manocha. 2016. CAMA: Contact-Aware Matrix Assembly with Unified Collision Handling for GPU-Based Cloth Simulation. *Comput. Graph. Forum (Eurographics)* 35, 2 (May 2016), 511–521.

Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018b. I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 37, 6, Article 204 (Dec. 2018), 10 pages.

Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. 2003. Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Proceedings of Vision, Modeling, Visualization*, Vol. 3. 47–54.

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous Cloth Simulation. In *Proceedings of Computer Graphics International*.

Mickeal Verschoor and Andrei C. Jalba. 2019. Efficient and Accurate Collision Response for Elastically Deformable Models. *ACM Trans. Graph.* 38, 2, Article 17 (March 2019), 20 pages.

Pascal Volino and Nadia Magnenat-Thalmann. 2006. Resolving Surface Collisions through Intersection Contour Minimization. *ACM Trans. Graph. (SIGGRAPH)* 25 (July 2006), 1154–1159. Issue 3.

Brian Von Herzen, Alan H. Barr, and Harold R. Zatz. 1990. Geometric Collisions for Time-Dependent Parametric Surfaces. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 39–48.

Huamin Wang. 2014. Defending Continuous Collision Detection against Errors. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, Article 122 (July 2014), 10 pages.

Huamin Wang. 2015. A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-Based Dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)* 34, 6, Article 246 (Oct. 2015), 9 pages.

Tongtong Wang, Zhihua Liu, Min Tang, Ruofeng Tong, and Dinesh Manocha. 2017. Efficient and Reliable Self-Collision Culling Using Unprojected Normal Cones. *Comput. Graph. Forum (Eurographics)* 36, 8 (2017), 487–498.

Martin Wicke, Hermes Lanker, and Markus Gross. 2006. Untangling Cloth with Boundaries. In *Proceedings of Vision, Modeling, and Visualization*. 349–356.

Longhua Wu, Botao Wu, Yin Yang, and Huamin Wang. 2020. A Safe and Fast Repulsion Method for GPU-Based Cloth Self Collisions. *ACM Trans. Graph.* 40, 1, Article 5 (Dec. 2020), 18 pages.

Changxi Zheng and Doug L. James. 2012. Energy-Based Self-Collision Culling for Arbitrary Mesh Deformations. *ACM Trans. Graph. (SIGGRAPH)* 31, 4, Article 98 (July 2012), 12 pages.