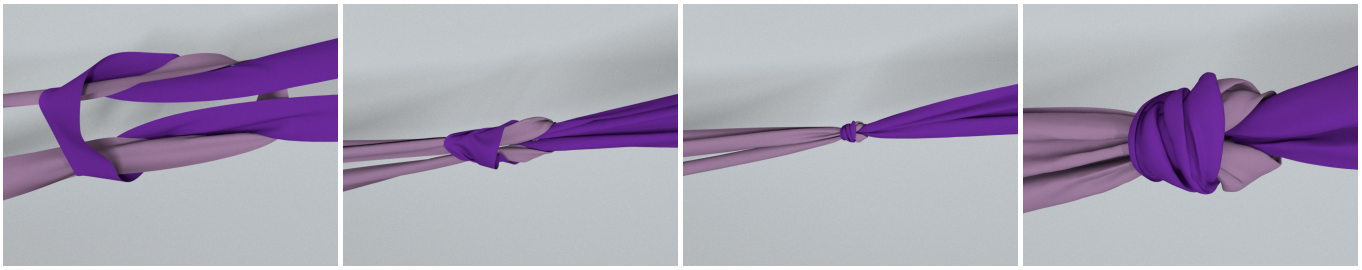
Fast GPU-Based Two-Way Continuous Collision Handling

TIANYU WANG*, FaceUnity, China
JIONG CHEN, LIX, Ecole Polytechnique, IP Paris, France
DONGPING LI, FaceUnity, China
XIAOWEI LIU, FaceUnity, China
HUAMIN WANG, Style3D, China
KUN ZHOU, State Key Lab of CAD&CG, Zhejiang University, China

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



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

Fig. 1. **Knotting.** The bow knot example (on the top, with 142K triangles) and the reef knot example (on the bottom, with 71K triangles) are presented. In this work, we develop a two-way method for *safe and fast* collision handling in deformable body simulation. Thanks to this method, our simulator can robustly handle complex collision contacts in these two examples at 4 to 17 FPS and 10 to 21 FPS respectively.

Step-and-project is a popular method to simulate non-penetrating deformable bodies in physically-based animation. The strategy is to first integrate the system in time without considering contacts and then resolve potential intersections, striking a good balance between plausibility and efficiency. However, existing methods can be defective and unsafe when using large time steps, taking risks of failure or demanding repetitive collision testing and resolving that severely degrade performance. In this paper, we propose a novel two-way method for fast and reliable continuous collision handling. Our method launches an optimization from both ends of the intermediate time-integrated state and the previous intersection-free state. It progressively generates a piecewise linear path and eventually obtains a feasible solution for the next time step. The algorithm efficiently alternates between a forward step and a backward step until the result is conditionally converged. Thanks to a set of unified volume-based contact constraints, our method offers flexible and reliable handling of various codimensional deformable bodies, including volumetric bodies, cloth, hair and sand. Experimental results demonstrate the safety, robustness, physical fidelity and numerical efficiency of our method, making it particularly suitable for scenarios involving large deformations or large time steps.

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

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

*Corresponding author

Authors' addresses: T. Wang, Independent researcher, China; J. Chen, LIX, Ecole Polytechnique (IP Paris), 1 rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France; D. Li, FaceUnity, 515 Yuhangtang Road, Gongshu District, Hangzhou, China; X. Liu, miHoYo, 519 Cangwu Road, Xuhui District, Shanghai, China; H. Wang, Style 3D, 99 Shuanglong Street, Xihu District, Hangzhou, China; K. Zhou, State Key Lab of CAD&CG, Zhejiang University, 866 Yuhangtang Road, Xihu District, Hangzhou, China.

0730-0301/2023/1-ART1 $15.00
https://doi.org/10.1145/3604551

ACM Reference Format:
Tianyu Wang, Jiong Chen, Dongping Li, Xiaowei Liu, Huamin Wang, and Kun Zhou. 2023. Fast GPU-Based Two-Way Continuous Collision Handling. *ACM Trans. Graph.* 1, 1, Article 1 (January 2023), 15 pages. https://doi.org/10.1145/3604551

1 INTRODUCTION

The simulation of intersection-free deformable body dynamics can be formulated as a constrained optimization problem [Kane et al. 1999; Martin et al. 2011]:

$$\mathbf{x}^{t+1} = \arg\min_{\mathbf{x}} E(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t), \quad \text{s.t. } \mathcal{X}(\mathbf{x}^t, \mathbf{x}) \subset \Omega, \tag{1}$$

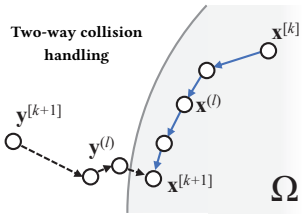
in which $\mathbf{x}^t, \mathbf{v}^t \in \mathbb{R}^{3N}$ are the stacked position and velocity vectors of N vertices at time t, $E(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t)$ is the dynamics objective, $\mathcal{X}(\mathbf{x}^t, \mathbf{x})$ is a sufficiently short linear or piecewise linear path from $\mathbf{x}^t$ to $\mathbf{x}$, and Ω is the feasible, intersection-free region. Li et al. [2020a; 2021] showed that globally convergent solutions could be obtained by augmenting the objective function with a smoothed Log-barrier-based contact energy term converting the original constrained problem into an unconstrained one and then solving it by Newton's method with a pre-filtered line search strategy based on continuous collision detection [Smith and Schaefer 2015].

However, their techniques are time-consuming, due to the frequent launching of the costly dynamics solver and truncated small step sizes for keeping the path within Ω. A common strategy [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] towards more efficient simulation is to divide the optimization into two steps:

$$\begin{cases} \mathbf{y}^{[k+1]} = \arg\min_{\mathbf{y}} Q_k(\mathbf{y}, \mathbf{x}^{[k]}, \mathbf{x}^t, \mathbf{v}^t), \\ \mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} D(\mathbf{x}, \mathbf{y}^{[k+1]}), \quad \text{s.t. } \mathcal{X}(\mathbf{x}^{[k]}, \mathbf{x}) \subset \Omega. \end{cases} \tag{2}$$

In each iteration, the dynamics solver first forms a quadratic model Q_k (or a linear model L_k [Wang and Yang 2016; Wu et al. 2020]) of $E(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t)$ at $\mathbf{x}^{[k]} \in \Omega$ to compute a target state $\mathbf{y}^{[k+1]}$, and then a collision handling module addresses all potential intersections to obtain a feasible state $\mathbf{x}^{[k+1]}$. Here, $D(\mathbf{x}, \mathbf{y}^{[k+1]})$ is a metric measuring the distance between $\mathbf{x}$ and $\mathbf{y}^{[k+1]}$, which is supposed to be significantly simpler than $E(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t)$. After several iterations of solving Eq. (2), the algorithm reports $\mathbf{x}^{[k+1]}$ as the solution $\mathbf{x}^{t+1}$ for Eq. (1). This strategy avoids frequent launching of high-cost solvers, e.g.,[Li et al. 2020a, 2021], but it sacrifices part of accuracy in exchange for performance since this alternating approach might not converge to a local minimum. Li et al. [2021] showed that such inaccuracy can manifest as undesirable wrinkling or jittering artifacts, which can be mitigated through parameter tuning.

Intuitively, the collision handling module should project $\mathbf{y}^{[k+1]}$ back to Ω, but it also has to avoid tunneling artifacts as shown in Fig. 3(a). In the past, researchers [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] parameterized this path as a line segment and evaluated $\mathcal{X}(\mathbf{x}^{[k]}, \mathbf{x}) \subset \Omega$ by continuous collision detection (CCD) tests. This approach typically produces plausible results when $\mathbf{y}^{[k+1]}$ is close to $\mathbf{x}^{[k]}$. However, as two states deviate from each other, finding $\mathbf{x}^{[k+1]}$ on the line segment that passes all of the CCD tests can become extremely difficult [Tang et al. 2018b]. Additionally, the metric to measure the distance for projection also plays a crucial role for achieving realistic simulation. Using mass-weighted L_2 norm is a straightforward approach that minimizes the change in post-response kinetic energy [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b]. However, this approach can introduce significant distortions in local elements when resolving all contacts (see Figs. 18 and 19), resulting in undesired artifacts such as over-stretching or oscillations over time.



In this paper, we develop a collision handling algorithm for the step-and-project method, capable of finding a high-quality solution $\mathbf{x}^{[k+1]}$ at a low cost. Our key idea is a two-way approach as illustrated in the inset. In this approach, we iteratively solve two steps: a *backward step* finding a sequence of targets $\{\mathbf{y}^{(l)}\}$ through impact zone optimization, serving as guidance for the evolution of $\mathbf{x}$, and a *forward step* that generates the actual path $\mathcal{X}(\mathbf{x}^{[k]}, \mathbf{x}) \subset \Omega$ by conservatively advancing vertices towards the guidance. As the forward step finds a sequence of states $\{\mathbf{x}^{(l)}\}$ satisfying $(1-t)\mathbf{x}^{(l)} + t\mathbf{x}^{(l+1)} \in \Omega$ for any $t \in [0, 1]$ and l, we actually construct $\mathcal{X}(\mathbf{x}^{[k]}, \mathbf{x})$ as a *piecewise linear* path rather than a linear one [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] relaxing the restriction on the search space. Aiming at both efficiency and quality for resolving contacts, we make the following technical contributions:

- *An inexact backward step.* We formulate the impact zone optimization as a linear complementary problem and solve it inexactly with a small number of iterations in each backward step. In addition, we introduce soft unilateral constraints on edge length to effectively mitigate oscillations resulting from large local element deformations.
- *A lightweight forward step.* Instead of relying on CCD tests, we use fast discrete distance evaluation to calculate safe asynchronous step sizes. This approach ensures that the path generated in each forward step remains safely within Ω.

We demonstrate that our two-way collision handling algorithm can be conveniently implemented on a GPU and integrated into our in-house GPU-based deformable body simulator. Coupled with volume-based contact constraints, our simulator is capable of simulating a variety of codimensional examples as depicted in Figs. 1 and 2, including volumetric bodies, cloth, hair and sand. The experiments validate that our system is safe, fast, GPU-friendly, and robust against large time steps and large deformations.

2 RELATED WORK

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, the process can be repeated in the next time step, hopefully reaching an 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; Stam 2009; Wang and Yang 2016], but it has no intersection-free guarantee if it is used alone, without support from any other method. To help the repulsion approach achieve intersection-free guarantee, Wu et al. [2020] presented a fail-safe Log-barrier repulsion phase, although its 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 about the timing: 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 could be prone to errors [Ainsley et al. 2012; Provot 1997; Tang et al. 2014; Wang 2014], especially in the single-precision floating-point computation environment. Recently, Yuksel [2022] presented an efficient and robust method for finding real roots of cubic and higher-order polynomials, and Lan et al. [2022] provided a re-designment of the CCD root finding procedure on GPU. For the robustness of CCD queries, we recommend [Wang et al. 2021] for a more detailed discussion. In addition to the possible robustness issue, the typical spatial acceleration structures used for CCD, such as bounding volume hierarchies (BVHs) and bounding volume traversal trees (BVTTs), could be hard to parallelize [Tang et al. 2011a, 2016].

But compared with CCD, obtaining the right collision responses is an even greater challenge. Bridson et al. [2002; 2005] 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] calculated collision responses by non-rigid impact zone optimization. ARCSim [Narain et al. 2012] used this method as its inner collision handling component, implemented with a combination of BVH-based CCD and an augmented Lagrangian solver for collision response. CAMA [Tang et al. 2016], PSCC [Tang et al. 2018a], I-Cloth [Tang et al. 2018b], and P-Cloth [Li et al. 2020b] further improved its performance on GPU(s) from two aspects.

- *Faster CCD on GPU(s).* In detail, this benefits from localized BVTT front propagation exploiting spatio-temporal coherence [Tang et al. 2016], parallel normal cone culling with spatial hashing [Tang et al. 2018a], incremental collision detection with spatial hashing exploiting spatio-temporal coherence [Tang et al. 2018b], and distributing the incremental collision detection on multiple GPUs [Li et al. 2020b].
- *More GPU-friendly non-rigid impact zone solver.* In detail, this benefits from assembling all of the impacts into one linear system to perform inelastic projection [Tang et al. 2016], parallelizing the augmented Lagrangian method of ARCSim in a gradient-descent manner [Tang et al. 2018b], and further parallelizing the augmented Lagrangian method on multiple GPUs [Li et al. 2020b].

We note that although the performance is continuously improved in these step-and-project methods, both the assumption of linear paths and simply measuring the projection distance via mass-weighted L_2 norm are fully inherited, as well as the specific choice of the augmented Lagrangian method for collision response in [Li et al. 2020b; Narain et al. 2012; Tang et al. 2018b].

Recently, Log-barrier-based methods [Li et al. 2020a, 2021; Wu et al. 2020] have emerged to be popular choices for collision handling in physical animation. They usually employ a smoothed [Li et al. 2020a, 2021] or unsmoothed [Wu et al. 2020] Log-barrier potential term to augment the objective, so that this term could be extremely large to "push" the state against the boundary of the feasible region. However, it is not sufficient for guaranteeing the state stays in the feasible region. Usually, CCD is employed to compute an upper bound of the safe distance to make sure that no intersection occurs. Clearly, the strength of these methods is their safety, but they could be notoriously slow for two reasons.

- *Frequent tests.* These methods have to repetitively test 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 $\partial\Omega$.

Moreover, we note that many continuous collision handling methods may require considerable implementation efforts to get accelerated on GPUs [Lauterbach et al. 2010; Li et al. 2020b; Tang et al. 2018a, 2016], due to their high dependency on sequential tasks [Li et al. 2021] and high complexity [Tang et al. 2018b].

Asynchronous steppings. 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 strategies [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, so 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 BVHs [Lauterbach et al. 2010; Tang et al. 2010, 2011b; Wang et al. 2017] and those based on spatial hashing [Barbič and James 2010;

ALGORITHM 1: A two-way 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 step number limit L and the termination condition ϵ.

1 $\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{[k]}$;
2 $\mathbf{y}^{(0)} \leftarrow \mathbf{y}^{[k+1]}$;
3 $D^{(0)} \leftarrow 0$;
4 $\mathbf{r}^{(0)} \leftarrow \mathbf{1}$;
5 $\mathscr{P} \leftarrow \emptyset$;
6 **for** $l = 0 \ldots L$ **do**
7 **if** $D^{(l)} < D^{\min}$ **then**
8 $\mathscr{P} \leftarrow \texttt{Proximity\_Search}\left(\mathbf{x}^{(l)}, D^{\max}\right)$;
9 $D^{(l)} \leftarrow D^{\max}$;
10 **end**
11 $\mathbf{y}^{(l+1)} \leftarrow \texttt{Backward}\left(\mathbf{y}^{(l)}, \mathbf{x}^{(l)}, \mathbf{y}^{[k+1]}, \mathscr{P}\right)$ // Sec. 4;
12 $\left\{\mathbf{x}^{(l+1)}, \mathbf{r}^{(l+1)}\right\} \leftarrow \texttt{Forward}\left(\mathbf{x}^{(l)}, \mathbf{r}^{(l)}, \mathbf{y}^{(l+1)} - \mathbf{x}^{(l)}, \mathscr{P}\right)$ // Sec. 5;
13 $D^{(l+1)} \leftarrow D^{(l)} - 2 \max_i \left\|\mathbf{x}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right\|$;
14 **if** $\left\|\mathbf{r}^{(l+1)}\right\|_\infty < \epsilon$ **then**
15 **break**;
16 **end**
17 **end**
18 $\mathbf{x}^{[k+1]} \leftarrow \mathbf{x}^{(l+1)}$;

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 contributions are orthogonal to collision culling techniques and we can adopt more advanced ones later.

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. While our work does not consider friction, we plan to leverage their ideas for simulating plausible frictional contacts in the future.

3 A TWO-WAY FRAMEWORK

As we discussed in Section 1, restricting to a linear path and simply taking mass-weighted L_2 norm as the distance function can be inappropriate for post-projection, especially when $\|\mathbf{y}^{[k+1]} - \mathbf{x}^{[k]}\|$ is large. Thus, we formulate the collision handling as the following optimization problem:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \frac{1}{2}\left\|\mathbf{x} - \mathbf{y}^{[k+1]}\right\|_{\mathbf{M}}^2, \tag{3a}$$

$$\text{s.t.} \begin{cases} \mathscr{X}\left(\mathbf{x}^{([k])}, \mathbf{x}\right) \subset \Omega, \\ \mathbf{c}(\mathbf{x}, \mathbf{y}^{[k+1]}) \geq \mathbf{0}, \end{cases} \tag{3b}$$

in which $\mathbf{M} \in \mathbb{R}^{3N \times 3N}$ is the scaled lumped mass matrix [Tang et al. 2018b]. The major differences between our formulation and the ones in [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] are twofold rooted in Eq. (3b): (i) the path $\mathscr{X}(\mathbf{x}^{([k])}, \mathbf{x})$ here should be piecewise linear and sufficiently short, while [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] restricts it as a linear segment; and (ii) additional edge length constraints $\mathbf{c}(\mathbf{x}, \mathbf{y}^{[k+1]})$ are introduced (see Section 4.2 for details) to avoid spuriously large deformations of local elements, which compensates for possible loss of shape preservation by only considering the mass-weighted Euclidean distance. In fact, the projection distance jointly defined by the mass-weighted L_2 norm plus edge length constraints is more consistently measured as the actual energy, in a sense that we are approximately minimizing the change of both kinetic energy and potential before and after projection. Numerically, such treatment does not complicate the objective function, allowing us to use existing fast iterative techniques as described below.

Our two-way method aims to find a good approximate solution subject to Eq. (3) at a low cost. In this method, two sets of intermediate variables $\{\mathbf{y}^{(l)}\}$ and $\{\mathbf{x}^{(l)}\}$ are introduced and updated alternately in two steps: a backward step starting at $\mathbf{y}^{(0)} = \mathbf{y}^{[k+1]}$, aiming to inexactly and progressively project $\mathbf{y}^{(l)}$ back to Ω as a target, and a forward step starting at $\mathbf{x}^{(0)} = \mathbf{x}^{[k]}$, aiming to move from the current state $\mathbf{x}^{(l)} \in \Omega$ towards $\mathbf{y}^{(l+1)}$ with a guarantee of $\mathscr{X}(\mathbf{x}^{[k]}, \mathbf{x})$ being inside Ω by conservative vertex advancement. Intuitively, $\{\mathbf{y}^{(l)}\}$ guide the evolution of $\{\mathbf{x}^{(l)}\}$ in the forward step, and in turn $\{\mathbf{x}^{(l)}\}$ explore and update the boundary of the feasible region Ω, which gives feedback to the updates of $\{\mathbf{y}^{(l)}\}$. The supplemental video illustrates our two-way optimization process.

Alg. 1 outlines the pseudo-code of our method: it keeps running the two steps alternately, until the termination metric $\mathbf{r}^{(l+1)}$ is small enough (Line 14) or it reaches the maximum number of iterations L (Line 6). We have $\mathbf{r}^{(l+1)}$ to make sure that the accumulated moving distance from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{(l+1)}$ should be close enough to the distance from $\mathbf{x}^{[k]}$ to $\mathbf{y}^{[k+1]}$. Either way, the method can safely report $\mathbf{x}^{(l+1)}$ as its result $\mathbf{x}^{[k+1]}$, no matter if $\mathbf{y}^{(l+1)}$ stays in Ω or not.

3.1 The Proximity Search

Before diving into details of our method, we first discuss about the proximity search. Functioning as broad-phase collision culling, the proximity search serves multi-fold 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 a part of dynamics (in Subsection 6.2). In our implementation, the proximity search is based on the standard grid-based spatial hashing technique [Pabst et al. 2010; Tang et al. 2018a].

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

$$\mathscr{P} \supset \mathscr{P}_{D^{\min}} = \{p \mid \forall p : \texttt{dist}_p\left(\mathbf{x}^{(l)}\right) < D^{\min}\}. \tag{4}$$

Assuming that in the l-th step, $\mathscr{P}$ is computed with a certain bound $D^{\max}$ ($\mathscr{P} = \mathscr{P}_{D^{\max}}$, $D^{(l)} = D^{\max} > D^{\min}$) so that all proximity pairs satisfying Eq. (4) are collected. To reuse this computed $\mathscr{P}$ in the (l+1)-th step, we point out that $\mathscr{P}$ still contains all of the pairs whose distances are below $D^{(l+1)} = D^{(l)} - 2\max_i \|\mathbf{x}_i^{(l+1)} - \mathbf{x}_i^{(l)}\|$. If $D^{(l+1)} \geq D^{\min}$, we can reuse $\mathscr{P}$ and only need to filter out some elements in $\mathscr{P}$ with recomputed distances. Otherwise, $\mathscr{P}$ is insufficient to fulfill Eq. (4). Thus, we have to reset $D^{(l+1)} = D^{\max}$ and perform the proximity search to avoid missing any necessary proximity pair.

The entire computational cost depends on values of both $D^{\min}$ and $D^{\max}$. On one hand, the cost decreases as $D^{\min}$ decreases, but $D^{\min}$ cannot tend to zero. To build a full set of contact constraints in the backward step, $\mathscr{P}_{D^{\min}}$ needs to collect all pairs whose distances are below a given activation threshold δ, which should not be too small in the discrete computing environment as suggested in [Li et al. 2020a]. Here we set δ = 1mm by default and $D^{\min} \geq \delta$. On the other hand, the cost decreases as $D^{\max}$ increases, but doing so requires more memory cost and distance evaluations as a result of an increasing number of proximity pairs. In our experiments, we find that setting $D^{\min} = 2\delta$ = 2mm and $D^{\max}$ = 4mm usually triggers one proximity search every three steps in average, which empirically keeps a reasonable balance between memory cost and computing time.

4 THE BACKWARD STEP

The backward step in our two-way approach is similar to the impact zone optimizations in [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b]. Instead of directly performing CCD to find a strictly intersection-free projection, we roughly optimize an intermediate target $\mathbf{y}^{(l)}$ almost intersection-free. Note that it is only used for guiding the vertex advancement in the forward step later, where the exact intersection-freeness is imposed as discussed in Section 5.

In the l-th iteration, the goal of the backward step is to project $\mathbf{y}^{(l)}$ back to Ω formulated as a constrained optimization:

$$\mathbf{y}^{(l+1)} = \arg\min_{\mathbf{y}} \frac{1}{2}\left\|\mathbf{y} - \mathbf{y}^{[k+1]}\right\|_{\mathbf{M}}^2, \quad \text{s.t. } \mathbf{c}(\mathbf{y}) \geq \mathbf{0}, \tag{5}$$

where $\mathbf{c}(\mathbf{y}) \geq \mathbf{0}$ contains the contact constraints and edge length constraints. Let $\mathbf{x}^{(l)}$ be the current state of the forward step in the l-th iteration. We linearize the contact constraints at $\mathbf{x}^{(l)}$: $\mathbf{c}(\mathbf{x}^{(l)}) + \mathbf{J}^{(l)}(\mathbf{y} - \mathbf{x}^{(l)}) \geq \mathbf{0}$, in which $\mathbf{J}^{(l)} = \partial\mathbf{c}(\mathbf{x}^{(l)})/\partial\mathbf{x}$ is the Jacobian matrix.

Linearization. Note that we perform assembly and linearization of contact constraints at $\mathbf{x}^{(l)}$ rather than $\mathbf{y}^{(l)}$. The reason is that we not only require $\mathbf{y}^{(l+1)}$ to be close to intersection-free, but also expect the path from $\mathbf{x}^{(l)}$ to $\mathbf{y}^{(l+1)}$ to stay inside Ω as much as possible. Since $\mathbf{x}^{(l)}$ is inside Ω in the first place, we are actually attempting to drive $\mathbf{y}^{(l+1)}$ to cross the boundary of Ω and stay on *the same side* as $\mathbf{x}^{(l)}$ rather than $\mathbf{y}^{(l)}$. Simply projecting the target back to Ω based on the constraints assembled around $\mathbf{y}^{(l)}$ can cause the stagnation issue as shown in Fig. 3(b), while performing the linearization at $\mathbf{x}^{(l)}$ effectively avoids such an issue and reaches the convergence with a more reasonable solution as shown in Fig. 3(c).

By introducing Lagrangian multipliers, we formulate the following Lagrangian:

$$\mathscr{L}(\mathbf{y}, \boldsymbol{\lambda}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{y}^{[k+1]}\right\|_{\mathbf{M}}^2 - \left(\mathbf{c}(\mathbf{x}^{(l)}) + \mathbf{J}^{(l)}(\mathbf{y} - \mathbf{x}^{(l)})\right)^{\mathsf{T}}\boldsymbol{\lambda}, \tag{6}$$

whose minimizer satisfies the KKT conditions:

$$\begin{cases} \nabla_{\mathbf{y}}\mathscr{L} = \mathbf{M}(\mathbf{y} - \mathbf{y}^{[k+1]}) - \left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} = \mathbf{0}, \\ \boldsymbol{\lambda} \geq 0 \perp \mathbf{c}(\mathbf{x}^{(l)}) + \mathbf{J}^{(l)}(\mathbf{y} - \mathbf{x}^{(l)}) \geq \mathbf{0}. \end{cases} \tag{7}$$

Multiplying the first condition in Eq. (7) with $\mathbf{J}^{(l)}$, we obtain:

$$\mathbf{J}^{(l)}\mathbf{y} = \mathbf{J}^{(l)}\mathbf{M}^{-1}\left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{J}^{(l)}\mathbf{y}^{[k+1]}. \tag{8}$$

Together with the second condition, we get a linear complementarity problem (LCP) with only one unknown $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda} \geq \mathbf{0} \perp \mathbf{c}(\mathbf{x}^{(l)}) + \mathbf{J}^{(l)}\mathbf{M}^{-1}\left(\mathbf{J}^{(l)}\right)^{\mathsf{T}}\boldsymbol{\lambda} + \mathbf{J}^{(l)}\left(\mathbf{y}^{[k+1]} - \mathbf{x}^{(l)}\right) \geq \mathbf{0}. \tag{9}$$

Once we solve $\boldsymbol{\lambda}$, we apply the first condition of Eq. (7) to calculate $\mathbf{y}^{(l+1)}$ for the next iteration. Note that $\mathbf{y}^{(l)}$ is the initialization to the problem in Eq. (5) and it is calculated from the last $\boldsymbol{\lambda}$ in the (l-1)-th step, so the cumulative effect of the previous l-1 iterations to $\mathbf{y}$ is retained.

4.1 Contact Constraints

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. 4(a), whose distance is below a certain activation threshold δ. In our experiment, δ=1mm by default. 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:

$$\begin{cases} \mathbf{r}_a &= \mathbf{x}_a \quad + \frac{1}{2}\left(\delta - \|\mathbf{x}_a - b_i\mathbf{x}_i - b_j\mathbf{x}_j - b_k\mathbf{x}_k\|\right)\mathbf{n}, \\ \mathbf{r}_{i,j,k} &= \mathbf{x}_{i,j,k} - \frac{1}{2}\left(\delta - \|\mathbf{x}_a - b_i\mathbf{x}_i - b_j\mathbf{x}_j - b_k\mathbf{x}_k\|\right)\mathbf{n}, \end{cases} \tag{10}$$

where b_i, b_j and b_k are the barycentric weights of vertex a 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 $\mathbf{r}_a\mathbf{r}_i\mathbf{r}_j\mathbf{r}_k$:

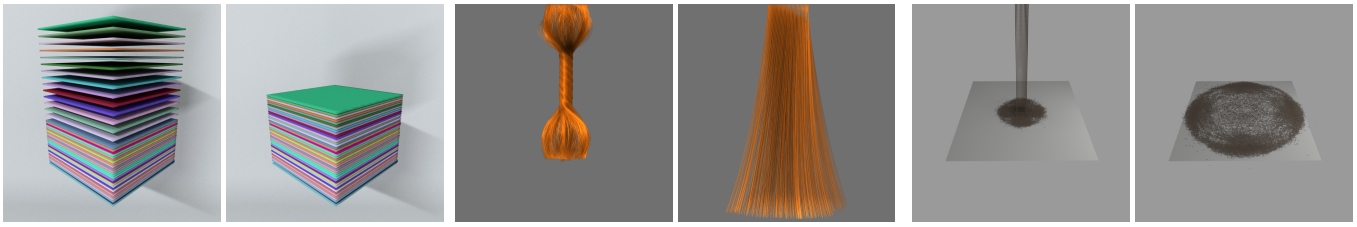
$$c(\mathbf{x}_a, \mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) = \det(\partial\mathbf{x}/\partial\mathbf{r}) - 1 \geq 0, \tag{11}$$

in which $\mathbf{r}_a$, $\mathbf{r}_i$, $\mathbf{r}_j$ and $\mathbf{r}_k$ are treated as constants, and $\partial\mathbf{x}/\partial\mathbf{r}$ is the artificial deformation gradient tensor, assuming that the computed $\mathbf{r}$ is the reference shape. Note that the contact constraint actually outlines the boundary of the feasible region at $\mathbf{x}^{(l)}$, so the sign of volume at $\mathbf{x}^{(l)}$ is positive. If $\mathbf{y}^{(l)}$ is on the other side of the boundary, the sign of volume at $\mathbf{y}^{(l)}$ would be negative and the inequality constraint would "pull" $\mathbf{y}^{(l)}$ to the same side as $\mathbf{x}^{(l)}$.

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:

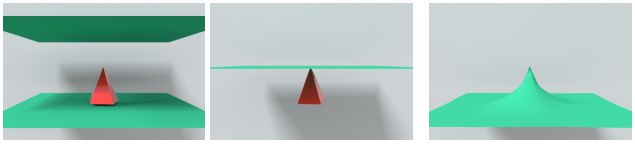
$$c(\mathbf{x}_a, \mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{x}_a\|/\delta - 1 \geq 0. \tag{12}$$

These constraints enable our method to handle contacts for a variety of codimensional deformable body examples, such as cloth, hair (in Fig. 2(c) and 2(d)) and sand (in Fig. 2(e) and 2(f)).



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

Fig. 2. **Codimensional deformable bodies.** Based on the volume enforcement idea, we develop contact constraints for a wide range of primitive proximity pairs. These contact constraints enable our method to simulate a variety of codimensional examples, including elastic mats in (a) and (b), cloth, hair in (c) and (d), and sand in (e) and (f).



(a) $\mathbf{x}^{[k]}$ (above) and $\mathbf{y}^{[k+1]}$ (below) (b) $\mathbf{x}^{[k+1]}$ using the contact constraints assembled around $\mathbf{y}^{(l)}$ (c) $\mathbf{x}^{[k+1]}$ using the contact constraints assembled around $\mathbf{x}^{(l)}$

Fig. 3. **Linearization choices.** The tunneling artifact occurs between two intersection-free states $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ (Fig. (a)). Projecting $\mathbf{y}^{[k+1]}$ back to Ω based on the constraints assembled around $\mathbf{y}^{(l)}$ stagnates $\mathbf{x}^{[k+1]}$ (Fig. (b)). Instead, if we assemble and linearize contact constraints around $\mathbf{x}^{(l)}$, $\mathbf{y}^{(l)}$ manages to cross the boundary of Ω and stay on the same side as $\mathbf{x}^{(l)}$, eventually advancing $\mathbf{x}^{(l)}$ to a more reasonable solution (Fig. (c)).

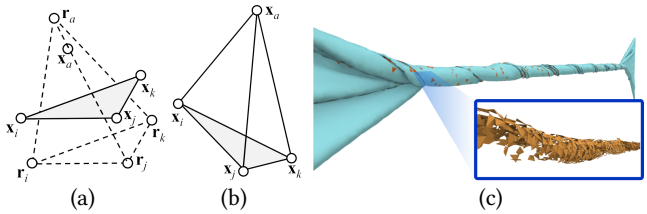
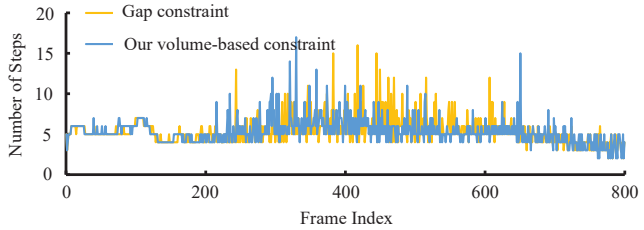
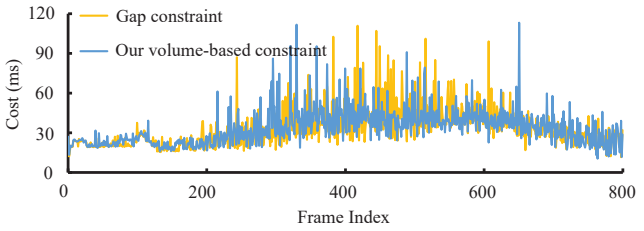


Fig. 4. **Vertex-triangle pair.** Our constraint enforces the volume of $\mathbf{x}_a\mathbf{x}_i\mathbf{x}_j\mathbf{x}_k$ to be the same as its reference $\mathbf{r}_a\mathbf{r}_i\mathbf{r}_j\mathbf{r}_k$ (Fig. (a)), in which the vertex and the triangle are separated by a threshold distance δ (Fig. (b)). Tessellated with small tetrahedra in crevices, the simulated cloth robustly prevents self-intersections (Fig. (c)).

We find that such volume-based constraints could make locking artifacts, which usually come from the edge-edge pairs, almost invisible. The possible reason is that if we renew the reference volume in every intermediate step as our method does, it does not cause the resistance of shearing or twisting, whereas if the reference shapes are constant, these artifacts would be observed as discussed in [Müller et al. 2015]. A potential problem with volumetric constraints is that they may increase the triangle area or edge length, which may influence the simulation quality or performance negatively. However, this issue is barely problematic in our experiments: we test the popular gap constraints [Andrews et al. 2022] as a replacement, and using both kinds of constraints has comparable performance as Fig. 5 shows, and our examples show that volumetric constraints can work well without obvious artifacts.



(a) The number of steps fluctuating over frames



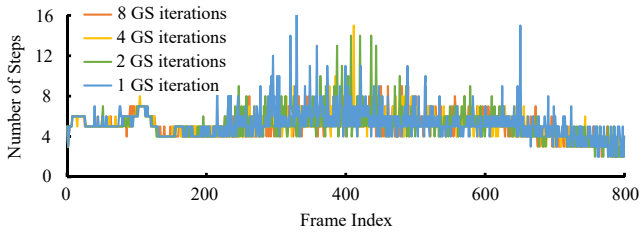
(b) The collision cost fluctuating over frames

Fig. 5. **Comparison with the gap constraints.** We compare the performance of our method implemented with the gap constraints and the volume-based constraints (as ours). Our volume-based constraints exhibit comparable performance to the gap constraints and do not have a negative impact on overall performance. By default, we use the bow knot example for performance evaluations in this paper.

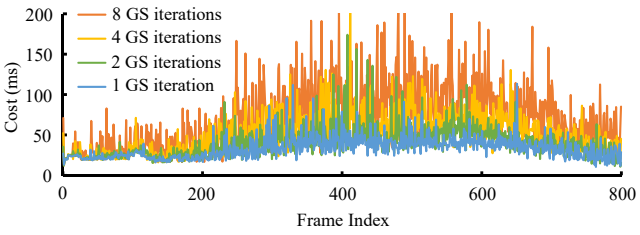
Sifakis et al. [2008] explored a similar idea, but they chose to preserve the volume, rather than enforce the volume to a desired value. In comparison, our constraints can keep pairs well separated, so that fewer collisions can occur in later updates.

4.2 Edge length Constraints

When the target $\mathbf{y}^{[k+1]}$ is close to $\mathbf{x}^{[k]}$, minimizing the mass-weighted Euclidean distance keeps a minimal change of kinetic energy after collision handling and the change of potential should not be significant either. However, as $\mathbf{y}^{[k+1]}$ starts to get far away from $\mathbf{x}^{[k]}$, simply using this kinetic energy norm to measure the projection distance could be inappropriate. It can cause spuriously large, local deformations (see Figs. 18 and 19) and even oscillations over time, as a result of a sharp change of potential before and after collision handling. We can incorporate additional deformation resistance terms into the objective function, but it breaks a simple

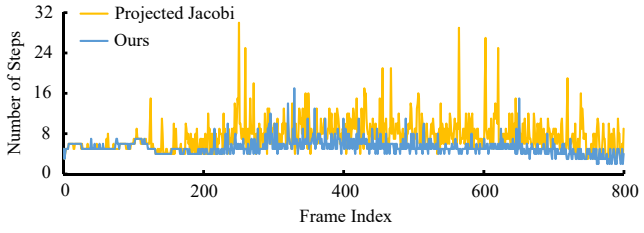


(a) The number of steps fluctuating over frames

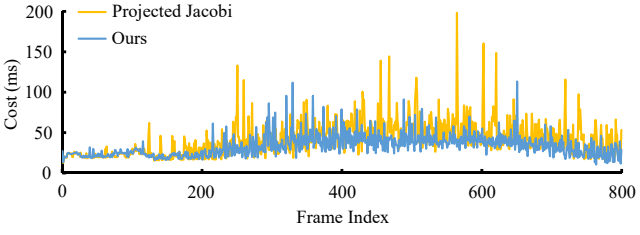


(b) The collision cost fluctuating over frames

Fig. 6. **Performance w.r.t. Gauss-Seidel iterations.** We compare the performance of our method using different numbers of Gauss-Seidel iterations per backward step. Applying more Gauss-Seidel iterations solves the LCP problem more accurately in every backward step, but it does not necessarily reduce the number of steps (Fig. (a)). Thus, we recommend using only one Gauss-Seidel iteration per step for the efficiency purpose (Fig. (b)).



(a) The number of steps fluctuating over frames



(b) The collision cost fluctuating over frames

Fig. 7. **Comparison with projected Jacobi method.** We compare the performance of our method implemented with the projected Jacobi method and the projected Gauss-Seidel method (as ours). Compared to ours, the use of the projected Jacobi method requires more steps (Fig. (a)) to conditionally converge under metric $\mathbf{r}$, resulting in higher computational costs (Fig. (b)).

LCP formulation and thus becomes more numerically involved. Given the fact that potentials are basically penalizing non-rigid deformations, we therefore attempt to preserve the shape of each element by preserving its edge lengths, so that each element mostly undergoes a rigid transformation and the change of potential stays at a low level after collision handling.

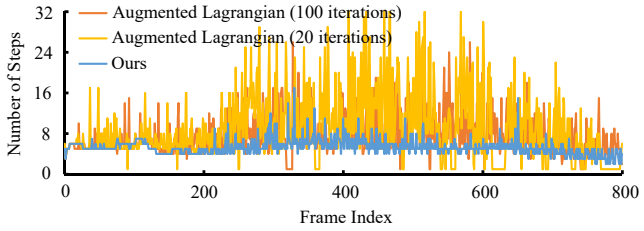
For keeping it as a simple LCP formulation where many fast iterative techniques can be used, we follow the strategy in [Macklin and Muller 2021] to suppress spurious distortions by incorporating constraints without complicating the objective function, and linearize it at $\mathbf{y}^{(l)}$:

$$c(\mathbf{x}_i, \mathbf{x}_j) = \sigma - \left\|\mathbf{x}_i - \mathbf{x}_j\right\| \Big/ \left\|\mathbf{y}_i^{[k+1]} - \mathbf{y}_j^{[k+1]}\right\| \geq 0, \qquad (13)$$

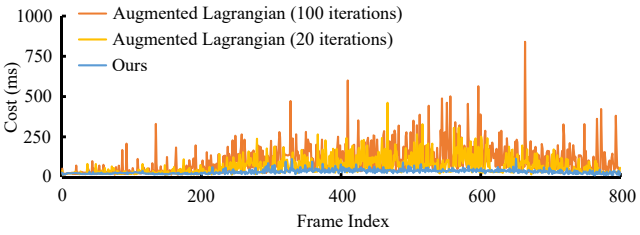
in which i and j are the two vertex indices of one edge and σ is the maximum violation ratio. Here we relax the strict bilateral constraint to the soft unilateral one and permit a certain level of violation for performance consideration. In our implementation, we use the squared versions of Eqs. (12) and (13) to avoid numerical errors and inefficiencies that can be introduced by taking squared roots as discussed in [Li et al. 2020a].

4.3 An Inexact GPU-Based Optimizer

A popular way of solving the LCP problem in Eq. (9) is to apply a projected iterative method [Erleben 2013], which enforces $\boldsymbol{\lambda} \geq \mathbf{0}$ after every iteration of solving the linear system. In our simulator, we adopt standard multi-color Gauss-Seidel as our solver. We use the randomized graph coloring method in [Fratarcangeli et al. 2016] while we assign color (graph node) to constraints instead of vertices. This is different from the strategy in [Fratarcangeli et al. 2016]. Please refer to [Fratarcangeli et al. 2016] for more details.



(a) The number of steps fluctuating over frames



(b) The collision cost fluctuating over frames

Fig. 8. **Comparison with augmented Lagrangian method.** We compare the performance of our own optimizer and the augmented Lagrangian optimizer in [Tang et al. 2018b]. Since their optimizer converges significantly slower than ours, it has to run multiple iterations per backward step to reduce the total number of steps. However, the improved convergence does not pay off the extra cost, which makes their method less efficient.

4.3.1 Inexactness. One interesting question is how many iterations should we spend on solving the LCP problem? The more iterations we use, the more accurately the problem is solved. But given the fact that the LCP problem is being constantly updated in each single backward step, it would be a waste if too much computational cost is spent 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 convergence and the costs associated with backward and forward steps. According to Fig. 6, 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.

4.3.2 A projected Jacobi implementation. The analysis in Subsection 4.3.1 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 Chebyshev acceleration [Wang 2015] ineffective across steps. Overall, the method with projected Jacobi needs more steps and more costs for resolving collisions as Fig. 7 shows.

4.4 Comparison to an Augmented Lagrangian Optimizer

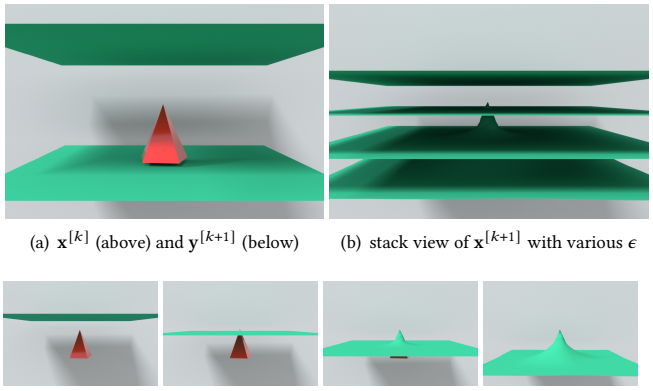
We can adopt other constrained optimization techniques to perform the task in our backward step as well. Specifically, we would like to evaluate the performance of an augmented Lagrangian optimizer with the gradient descent method, advocated by Tang et al. [2018b]. The strength of their optimizer is its simplicity: it does not have to solve any linear systems for primal or dual variables, and the major computational units are gradient and constraint evaluations. However, since their optimizer converges considerably slower, it has to 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. We find that when using fewer iterations, their optimizer degenerates very quickly in terms of changing $\mathbf{r}$. To avoid iterations with no revenue, we immediately terminate the optimization once we find the change of $\|\mathbf{r}\|_\infty$ is tiny ($\|\mathbf{r}^{(l)}\|_\infty - \|\mathbf{r}^{(l+1)}\|_\infty < 10^{-7}$) after one step. Fig. 8 shows that when using 20 iterations per step, the two-way method with their optimizer needs a large number of steps and often terminates early, especially when the knot is tightened. By increasing the iterations per step to 100, the early termination issue can be alleviated. However, the two-way method implemented with their optimizer still requires a large number of steps compared to our method, which consistently terminates within a short time under the metric $\mathbf{r}$. Overall our optimizer is a better choice.

5 THE FORWARD STEP

In each forward step, we move the vertices from the current state $\mathbf{x}^{(l)}$ toward $\mathbf{y}^{(l+1)}$ asynchronously:

$$\mathbf{x}_i^{(l+1)} = \mathbf{x}_i^{(l)} + \alpha_i^{(l+1)}\left(\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right). \tag{14}$$

The key question is how to find the safe step size $\alpha_i^{(l+1)}$ for every vertex i, so that $\mathcal{X}(\mathbf{x}^{(l)}, \mathbf{x}^{(l+1)}) \subset \Omega$. One way of obtaining a safe step size is to use continuous collision detection (CCD). CCD tests calculate exact moments when proximity pairs intersect, which tell how far $\mathbf{x}_i$ should be moved. However, previous works show that CCD tests could be prone to errors [Ainsley et al. 2012; Tang et al. 2014; Wang 2014] and require considerable efforts to get accelerated on GPUs [Tang et al. 2011a, 2016, 2018b].



(a) $\mathbf{x}^{[k]}$ (above) and $\mathbf{y}^{[k+1]}$ (below) (b) stack view of $\mathbf{x}^{[k+1]}$ with various ϵ

(c) $\mathbf{x}^{[k+1]}$ with ϵ = 0.75 (d) $\mathbf{x}^{[k+1]}$ with ϵ = 0.50 (e) $\mathbf{x}^{[k+1]}$ with ϵ = 0.25 (f) $\mathbf{x}^{[k+1]}$ with ϵ = 0.0001

Fig. 9. **Termination condition and threshold ϵ.** With the same $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$, the accumulated moving distance from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$ is getting close to the distance from $\mathbf{x}^{[k]}$ to $\mathbf{y}^{[k+1]}$ as ϵ approaching zero, reducing the risk of early termination.

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 use this idea, the method needs a set of proximity pairs $\mathscr{P}$, in which each pair contains two non-adjacent simplices with its distance below a global threshold $D^{(l)}$. Given $\mathscr{P}$, the method calculates D_i, the shortest distance of the proximity pairs involving vertex i:

$$D_i = \min_{\{a,b\}\in\mathscr{P}} \mathtt{dist}\left(\mathbf{x}_a^{(l)}, \mathbf{x}_b^{(l)}\right) \le D^{(l)},\ \forall a, b : a \neq b \text{ and } i \in a \cup b, \tag{15}$$

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

$$\forall i :\ \left\|\mathbf{x}_i^{(l)} - \mathbf{x}_i^{(l+1)}\right\| < D_i/2 \Rightarrow (1-t)\mathbf{x}^{(l)} + t\mathbf{x}^{(l+1)} \in \Omega, \tag{16}$$

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

$$\begin{cases} \alpha_i^{(l+1)} = \min\left(0.5\gamma D_i \Big/ \left\|\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right\|, 1\right), \\ \mathbf{x}_i^{(l+1)} = \mathbf{x}_i^{(l)} + \alpha_i^{(l+1)}\left(\mathbf{y}_i^{(l+1)} - \mathbf{x}_i^{(l)}\right), \\ r_i^{(l+1)} = r_i^{(l)}\left(1 - \alpha_i^{(l+1)}\right), \end{cases} \tag{17}$$

in which γ is a damping factor preventing proximity pairs from getting too close in a single forward step. Using the step size calculated for every vertex, we ensure that $\mathbf{x}^{(l+1)}$ is an acceptable intermediate state, regardless of the search direction.

A special feature we would like to mention in Eq. (17) is the termination metric r_i. The non-smooth nature of our optimization method makes it difficult to define the termination condition by the step size $\alpha_i^{(l+1)}$ directly, without potential early termination risks. To address this issue, we come up with a termination metric r_i in an accumulated fashion. Intuitively, it keeps track of the remaining

step size needed for vertex i to reach its target and we terminate the method once $\|\mathbf{r}^{(l+1)}\|_\infty$ drops below a certain threshold ϵ. As the threshold ϵ tends to zero, the accumulated moving distance from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$ tends to be close enough to the distance from $\mathbf{x}^{[k]}$ to $\mathbf{y}^{[k+1]}$, so the early termination risks of the optimization could be eliminated as shown in Fig. 9(f).

Compared to CCD tests, evaluating distance using our scheme is computationally inexpensive, reliable against floating-point errors and easy to parallelize on GPUs.

5.1 Comparison to CCD-Based Schemes

To compare our CCD-free stepping 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 factor out the difference in the implementation details. On the same NVIDIA GeForce GTX 2080 Ti GPU, we evaluate both schemes with a number of examples using two time steps: Δt=1/100s and Δt=1/20s.

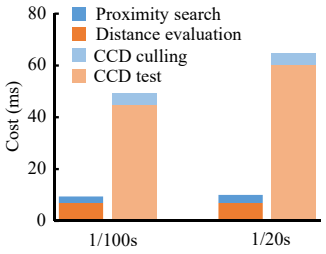


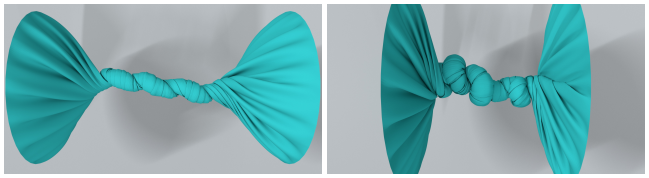
Fig. 10. **Comparison with CCD.** Overall, the CCD-free stepping schemes for collision handling is five to seven times faster than the CCD-based schemes within a single time step.

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. Overall, 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.

The Special CCD-Based Scheme in [Wu et al. 2020]. The step size scheme adopted by the hard phase in [Wu et al. 2020] is also based on CCD tests. Compared with other CCD-based schemes, their approach restricts all of the edge lengths to be less than a constant upper bound and accordingly derives a series of sufficient conditions to prevent intersections, which can be achieved by handling vertex-vertex contacts only. As a result, their CCD tests are less expensive, making their method suitable for fast simulation of virtual garments that are almost inextensible.



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

Fig. 11. **Twisting tube.** When a compliant cloth tube is severely squeezed, it experiences intense and frequent collisions in the middle part as shown in (b). Our method robustly handles these collisions in this example taking Δt=1/100s as the time step.

However, such a benefit comes at the cost of limited applicability. If we apply their scheme to simulate general shell deformations where this length restriction cannot be adopted, e.g., the sustaining inflation or contraction of the membrane in the normal flow example (Fig. 16), then CCD tests have to consider all necessary vertex-triangle and edge-edge contacts again. In such cases, their method would be less efficient.

6 IMPLEMENTATION DETAILS

In this section, we discuss the implementation details of our two-way approach embedded in a GPU-based deformable body simulator.

6.1 Dynamics Solvers

Since our method works as a standalone module for collision handling, it is naturally compatible with most of the dynamics solvers. In our simulator, we follow the pipeline in Eq. (2) to solve the nonlinear optimization problem stemming from deformable body dynamics. In detail, Q_k is a quadratic proxy of energies written as below:

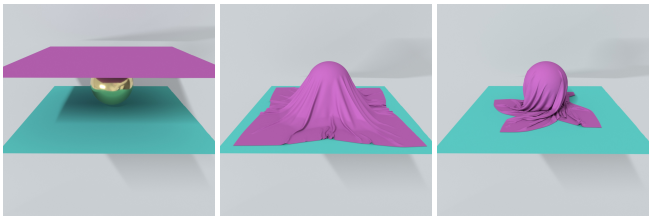
$$Q_k = E(\mathbf{x}^{[k]}, \mathbf{x}^t, \mathbf{v}^t) + \mathbf{b}^{[k]\intercal}(\mathbf{y}-\mathbf{x}^{[k]}) + \frac{1}{2}(\mathbf{y}-\mathbf{x}^{[k]})^\intercal \mathbf{G}^{[k]}(\mathbf{y}-\mathbf{x}^{[k]}), \tag{18}$$

in which $\mathbf{b}^{[k]}$ is the gradient and $\mathbf{G}^{[k]}$ is the modified Hessian of $E(\mathbf{x}, \mathbf{x}^t, \mathbf{v}^t)\big|_{\mathbf{x}=\mathbf{x}^{[k]}}$ after positive semi-definite (PSD) projection. We apply the conjugate gradient method with a block Jacobi preconditioner to solve the linear system emerging from the quadratic model in every Newton iteration, and run our two-way method for collision handling right after one single Newton iteration. Currently, 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 results. But even if we run a single Newton iteration for a large time step, i.e., Δt=1/10s, our method can still guarantee intersection-free with far fewer artifacts than existing collision handling algorithms, as shown in Fig. 19.

6.2 Elastic and Repulsive Models

To simulate the codimensional deformable body examples shown in Fig. 2, we provide a number of elastic models in our dynamics solver, including the St. Venant-Kirchhoff 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



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

Fig. 12. **Rotating sphere.** When a square cloth patch falls onto a rotating sphere, it generates multiple folds and wrinkles due to its frictional contacts with the sphere and the ground floor.

hair strands. The eigensystems of Hessian matrices for these energy models have been analyzed [Choi and Ko 2002; Etzmuß et al. 2003; Smith et al. 2019; Teran et al. 2005] and one only needs to compute, at most, 3×3 singular value decomposition and eigendecomposition for PSD projection which can be efficiently done on a GPU [Gao et al. 2018; Melax 2017]. Some other energy models can also be incorporated into this framework, such as the discrete shell model [Grinspun et al. 2003] which requires a larger (12×12) eigendecomposition for PSD projection [Chen et al. 2022; Li et al. 2021]. Instead, we simply apply the quadratic bending model [Bergou et al. 2006] that bypasses the PSD projection for efficiency. Similar to many other simulators [Li et al. 2020b; Narain et al. 2012; Tang et al. 2016, 2018b], our simulator incorporates a quadratic repulsive model into deformable body dynamics to reduce complex collisions in simulation.

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 compute a target state $\mathbf{y}^{[k+1]}$ with no friction. We then calculate penetration depths to estimate collision impulses and use them to compute associated frictional impulses by Coulomb's law as well. Finally, we update $\mathbf{y}^{[k+1]}$ based on both impulses and treat it 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, where such penetration depth estimations are more inaccurate, especially for large time steps. Fig. 12 demonstrates our current frictional effects achieved by the velocity filtering.

7 RESULTS AND DISCUSSIONS

We run our simulator on an Intel Core i5-7500 3.4GHz CPU and an NVIDIA GeForce GTX 2080 Ti GPU. Table 1 summarizes the statistics and performance of our examples, and Table 2 provides major parameters of our algorithm and their default values. Most results are computed with a default $\delta = 1$mm. Specifically, $\delta = 0.5$mm is used for the knotting examples in Fig. 1 and $\delta = 0.2$mm is used for the hair example in Fig. 2. Our examples include common benchmarks involving complex collisions (Figs. 1 and 11), and virtual garments of multiple layers dressed on human bodies (Fig. 20). Related animations are provided in the supplemental video. In general, the cost of handling contacts depends on the number of colliding

Table 1. **Statistics and performance.** This table lists the time step size, the maximum and mean numbers of steps, time cost spent by our two-way method and the number of Newton iterations within a single time step across different examples.

Name (#verts., #edg./#tri./#tet., ref.)	Time step (s)	Avg. (Max.) # of steps	Avg. (Max.) cost (s)	Newton Iter.
Needle (10k, 20k, Fig. 9(f))	1/20	46 (1304)	0.036 (1.942)	4
Blade (59k, 116k, Fig. 15)	1/20	55 (779)	0.267 (4.463)	4
Funnel (49K, 98K, Fig. 14a)	1/20	27 (675)	0.087 (2.161)	4
Funnel (49K, 98K, Fig. 14b)	1/40	23 (437)	0.055 (1.198)	4
Funnel (49K, 98K, Fig. 14c)	1/80	12 (165)	0.040 (0.776)	4
Funnel (49K, 98K, Fig. 14d)	1/160	10 (95)	0.029 (0.296)	4
Sphere (50k, 100k, Fig. 12)	1/100	19 (219)	0.044 (0.475)	2
Dress (30K, 60K, Fig. 20a)	1/100	39 (142)	0.133 (0.511)	1
Gown (27K, 51K, Fig. 20c)	1/100	32 (150)	0.092 (0.443)	1
Bow knot (71K, 142K, Fig. 1a)	1/100	5.4 (17)	0.034 (0.113)	1
Reef knot (37K, 71K, Fig. 1e)	1/100	7 (23)	0.024 (0.097)	1
Tube (25k, 51K, Fig. 11)	1/100	19 (54)	0.308 (1.500)	1
Mat (33k, 88K, Fig. 2a)	1/100	9 (128)	0.020 (0.303)	1
Hair (63K, 62k, Fig. 2c)	1/100	16 (503)	0.252 (15.821)	1
Sand (30K, - , Fig. 2e)	1/100	51 (160)	0.223 (1.550)	1

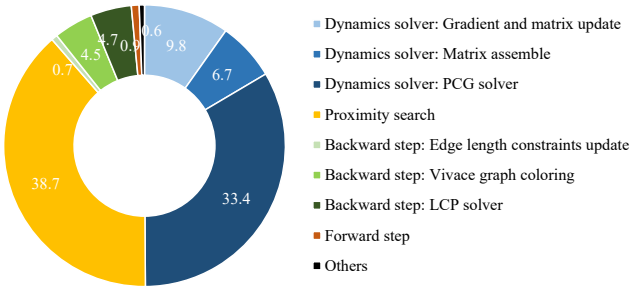


Fig. 13. **Breakdown analysis.** We visualize a typical breakdown of the cost spent in each stage for the bow knot example. As shown in this figure, the PCG solves and the proximity search are the two most computationally expensive parts in our simulator.

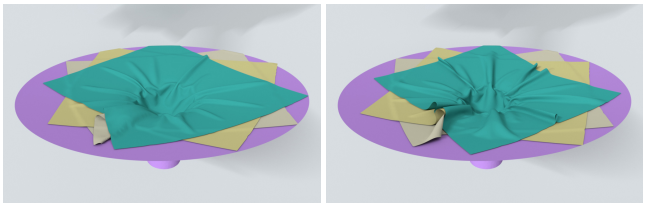
Table 2. **Parameters and their default values.**

Symbol	Meaning	Value
L	The limit on the number of steps	512 to 2,048
ϵ	The termination threshold	0.0001
$D^{\min}$	The proximity search lower bound	2mm
$D^{\max}$	The proximity search upper bound	4mm
δ	The constraint activation threshold	1mm
σ	The violation ratio limit	1.1
γ	The damping factor on movement	0.9

elements and the number of steps, which are in essence determined by the time step and the geometric complexity. By default, the step number limit L is set to 512 when Δt=1/100s, and 2,048 when Δt=1/20s. In practice, our method typically converges, i.e., $\|\mathbf{r}\|_\infty < \epsilon$, less than 64 steps, far before reaching the step number limit L as Table 1 shows.

7.1 Breakdown Analysis

Fig. 13 provides a breakdown of the computational cost spent in the bow knot example (Fig. 1). It shows that PCG solves and proximity



(a) $\Delta t = 1/20$s (b) $\Delta t = 1/40$s

(c) $\Delta t = 1/80$s (d) $\Delta t = 1/160$s

Fig. 14. **Funnel.** We simulate this example with various time step sizes. While our method reliably handles collisions at a large time step, i.e., Δt=1/20s, we recommend using a smaller time step in practice for more accurate results with less numerical damping.

search are the two most time-consuming components in our simulator. In comparison, the actual cost spent by the forward step and backward step is much lower, i.e., occupying only 10.8 percent of the total cost. It suggests that there is still a lot of room for further improving the performance of our method. For instance, we could leverage faster linear solvers and proximity search algorithms in the future.

7.2 Sensitivity to Time Steps

While our method is safe and robust regardless of time step size, its performance per step does drop as the time step increases. Fundamentally, this is due to more severe collisions and more steps required for convergence in our method. 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=65,536 so that we can know how many steps are needed for convergence. Fig. 14 compares our simulation results in this example and Table 1 provides their performance. From the performance perspective only, it is beneficial to use a larger time step to reduce the computational overhead associated with every time step. However, due to the existence of numerical damping, we should avoid very large time steps in actual applications.

7.3 Geometry Processing

Since our method works as a standalone module to resolve collisions, it can be seamlessly integrated into geometry processing applications for intersection-free results. For instance, our method can fix seriously self-intersecting animation frames robustly, such as the blade example shown in Fig. 15. We can also apply our method to enforce global injectivity in computing normal flows, such as the ones shown in Fig. 16.

For the globally injective normal flow application, we first compute $\mathbf{y}^{[k+1]}$ as below:

$$\begin{cases} \mathbf{y}_0^{[k+1]} = \mathbf{x}^{[k]} \pm \beta \mathbf{n}^{[k]}, \\ \mathbf{y}_{i+1}^{[k+1]}(v) = \mathbf{y}_i^{[k+1]}(v) + \alpha \triangle \mathbf{y}_i^{[k+1]}(v), \\ \mathbf{y}^{[k+1]} = \mathbf{y}_3^{[k+1]}, \end{cases} \tag{19}$$

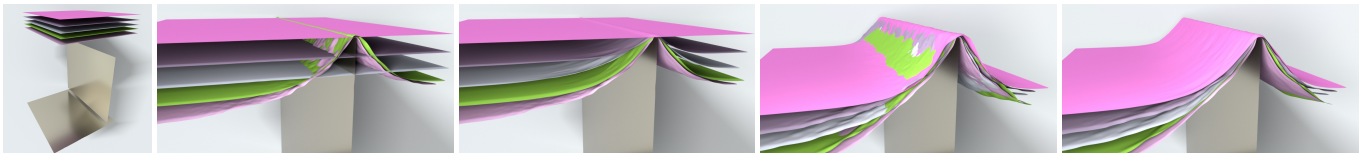
in which $\mathbf{n}^{[k]}$ is the normal vector field of the surface at $\mathbf{x}^{[k]}$, β is the flow speed, α is the smoothing intensity and $\triangle$ is the cotangent-form [Desbrun et al. 1999; Meyer et al. 2003; Pinkall and Polthier 1993] discretization of the Laplace-Beltrami operator. We apply a simple Jacobi iteration in parallel to smooth each vertex v, repeating the process three times to obtain $\mathbf{y}^{[k+1]}$. Then we run our two-way method for guaranteeing global injectivity. Finally, we get highly similar results compared with [Fang et al. 2021] on all examples taken from their benchmarks as shown in Fig. 16 within less than 1s.

7.4 Comparison to Existing Collision Handling Methods

As we argued above, the major problems of existing collision handling algorithms [Harmon et al. 2008; Li et al. 2020b; Narain et al. 2012; Tang et al. 2018a, 2016, 2018b] for the step-and-project method are their linear path assumption and inappropriate measurement of the projection distance. Such issues are exposed in a simple example as shown in Fig. 17: we initialize the angular velocity of various magnitudes in plane, leading to different targets $\mathbf{y}^{[k+1]}$ with increasing spin angles after the dynamics step. As the spin angle increases, the actual trajectory of every vertex should be close to a helix passing through the pyramid obstacle surface under gravity. Therefore, restricting to a linear path can make the algorithm difficult to find a valid path not colliding with the obstacle, which might become even impossible for complex shapes. Fig. 17(e) demonstrates such an increasing difficulty: as θ increases, the inelastic impact zone method [Harmon et al. 2008] implemented in ARCSim [Narain et al. 2012] requires drastically increasing cost to find a valid path. We note that when θ reaches 125°, the program cannot converge in two hours no matter how we tune the parameters.

Instead, our method runs inexpensive forward steps to form a piecewise linear path safely. In fact, this not only avoids expensive CCD tests, but also greatly increases the chances to find a valid path by exploring a larger space compared to a single line segment. Furthermore, with the help of the additional edge length constraints, the over-stretched artifacts arising from long-distance projection are pleasingly removed. Consequently, our two-way method can robustly remove all intersections along the path and reach a plausible and collision-free result (see Fig. 18 and the supplemental video) at a low cost (Fig. 17(e)). Theoretically, we note that taking a given linear path and "bending" it into a piecewise linear one in the feasible region is not unique to our approach. In fact, there exists one class of inequality-constrained nonlinear optimization methods that work in a similar way, i.e., the gradient (Newton) projection method [Conn et al. 1988; Lin and Moré 1999; Nocedal and Wright 2006]: given a gradient (Newton) direction (a linear path), a well-defined projection operator progressively projects the state into the feasible region and forms a piecewise linear path.

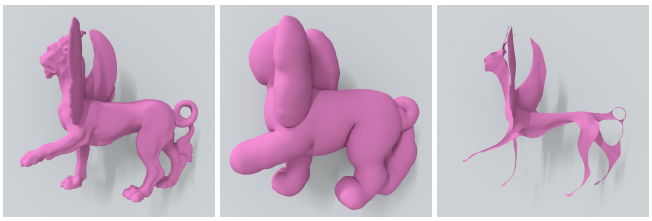
Thanks to the high-quality results of our two-way collision handling, the simulator using our method can generate more plausible



(a) The initial state (b) Before repair (c) After repair (d) Before repair (e) After repair

Fig. 15. **Cloth and blade colliding.** In this example, five square cloth patches are dropped onto a sharp metal blade, and then they drape, slide and stack under gravity. Our method reliably projects severely penetrated frames ((b) and (d)) to intersection-free states ((c) and (e)).

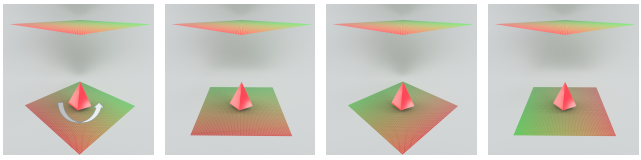
(a) The initial state (b) The positive flow (c) The negative flow



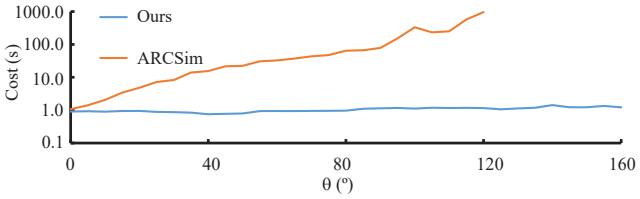
(d) The initial state (e) The positive flow (f) The negative flow

(g) The initial state (h) The positive flow (i) The negative flow

Fig. 16. **Globally injective normal flow.** Our method can also be applied to enforce global injectivity in both positive and negative normal flows.



(a) $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ when $\theta = 0°$ (b) $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ when $\theta = 45°$ (c) $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ when $\theta = 90°$ (d) $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ when $\theta = 135°$

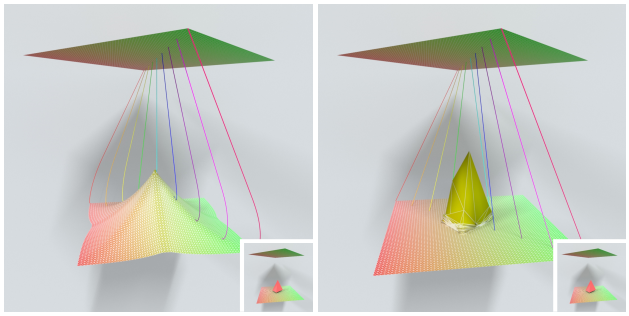


(e) Time cost comparison with ARCSim [Narain et al. 2012]

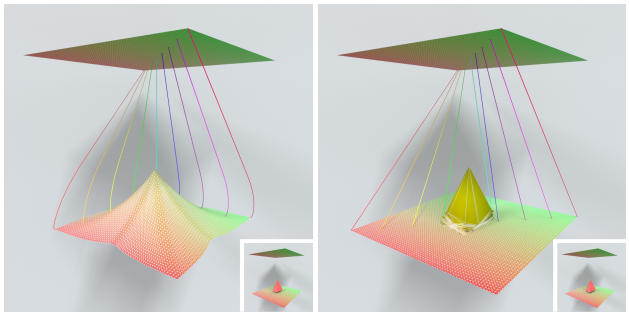
Fig. 17. **Spinning cloth patch.** From (a) to (d), the spin angle θ between $\mathbf{x}^{[k]}$ and $\mathbf{y}^{[k+1]}$ is increasing, and we use the color ramp to visualize the region correspondence. As θ increases, it becomes increasingly challenging for ARCSim to find a valid path, whereas our method is free of such drastic growth in time cost (Fig. (e)).

dynamics than existing collision handling methods when the time step is large. ARCSim [Narain et al. 2012] and CAMA [Tang et al. 2016], two implementations of non-rigid impact zone optimization [Harmon et al. 2008], are involved for evaluations. Note that other implementations, such as [Harmon et al. 2008; Li et al. 2020b; Tang et al. 2018a,b], should have similar behaviors due to similar methodology we introduced above, though with different performance. For fair comparisons, we run one Newton iteration starting from the same $\mathbf{x}^0$ (namely $\mathbf{x}^{[0]}$) with the same implementation and parameter setting to obtain $\mathbf{y}^{[1]}$ through the dynamics step, and then we pass both $\mathbf{x}^{[0]}$ and $\mathbf{y}^{[1]}$ to different collision handling modules to obtain $\mathbf{x}^{[1]}$ and update the velocity as $\mathbf{v}^1 = (\mathbf{x}^{[1]} - \mathbf{x}^{[0]})/\Delta t$. The output $\mathbf{x}^1$ (namely $\mathbf{x}^{[1]}$) and $\mathbf{v}^1$ will be used to initialize the dynamics step again which is followed by the collision step, and we repeat this process for the rest steps. This keeps the entire step-and-project processes all the same except for the switch between different projection methods for comparisons.

We test three collision-handling modules with a wide range of time steps to thoroughly observe the trend of their behaviors. When we use a large time step, such as Δt=1/10s, obvious over-stretched artifacts caused by long-distance projection are observed for both ARCSim and CAMA while our method is almost free of this issue and demonstrates better performance, as Figs. 19(a) to 19(c) and 19(g) and the supplemental video demonstrate. As the time step decreases, this problem should be gradually alleviated, as verified in the experiment: here, we decrease the time step by 1/1000s, from Δt=1/10s to Δt=1/1000s, to simulate the same scene repeatedly. However, existing non-rigid impact zone methods still suffer from visible artifacts when we use a moderate time step, such as Δt=25/1000s (see Figs. 19(j) to 19(o)). By comparing the mean vertex distance of their results with ours up to the same frame ($\frac{\sum_{t=0}^{t_s}\sum_{i=0}^{N-1}\|\mathbf{x}^t_{i,[\text{ARCSim|CAMA}]}-\mathbf{x}^t_{i,\text{ours}}\|}{N(t_s+1)}$, CLK = 1.2s, $t_s = \lfloor\frac{\text{CLK}}{\Delta t}\rceil$, Fig. 19(i)) and visually observing the simulated results, we find that ARCSim and CAMA become artifact-free and all three methods generate similar results when Δt decreases to about 1/125s (see



(a) $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$ by our method when $\theta = 45°$ (b) $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$ by ARCSim when $\theta = 45°$



(c) $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$ by our method when $\theta = 90°$ (d) $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$ by ARCSim when $\theta = 90°$

Fig. 18. **Trajectories avoiding collisions.** Here, we track the trajectories of nine vertices along one diagonal of the cloth patch, which are generated progressively using our two-way approach. Compared to the impact zone optimization implemented by ARCSim [Narain et al. 2012], our method enables a curved path from $\mathbf{x}^{[k]}$ to $\mathbf{x}^{[k+1]}$ to resolve collisions without the occurrence of intersections or spuriously stretched elements, while limiting the path to a linear trajectory may fail to find a faithful solution at a reasonable cost.

Figs. 19(d) to 19(f) and 19(h) and the supplemental video for details).

8 LIMITATIONS AND FUTURE WORK

While the experiments demonstrate the efficiency, robustness and good quality of our collision handling algorithm, there is no guarantee of a globally convergent approximate solution to Eq. (1). In fact, compared to the recently developed incremental potential contact (IPC) method [Li et al. 2021], all step-and-project methods seem to lack theoretical guarantee of global convergence in exchange for numerical efficiency. Therefore, we do not compare our method with IPC directly in terms of performance for fairness reasons. Although finding a convergent solution to Eq. (1) could be over-demanding and unnecessary for a faithful simulation in graphics, analyzing the compromise made by step-and-project methods and further improving its convergence is definitely valuable future work. Besides, our method does not consider friction modeling yet. Now it simply imitates frictional effects using an additional velocity filter. When aiming to replicate more realistic frictional contacts, it becomes necessary for the method to handle collisions and frictions simultaneously. Finally, we are interested in implementing our method on distributed systems comprising multiple GPUs to achieve real-time performance on large-scale scenes. To fully leverage the capabilities of distributed systems, we may have to thoroughly investigate the communication mechanisms between tasks, processes and threads, and thus devise efficient policies for parallelization and synchronization.

ACKNOWLEDGMENTS

The authors would like to thank all the anonymous reviewers for their valuable comments to improve this work. We also thank Min Tang for comparisons, Jin Huang and Chongyao Zhao for useful discussions, Cihui Xie, Peng Wang and Lingchen Yang for their help with rendering hair and garments. Finally, JC acknowledges the support from the Uber chair.

REFERENCES

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

Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022. Contact and Friction Simulation for Computer Graphics. In *ACM SIGGRAPH 2022 Courses (SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 3, 172 pages.

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

Jernej Barbič and Doug L. James. 2010. Subspace Self-Collision Culling. In *ACM SIGGRAPH 2010 Papers (SIGGRAPH '10)*. Association for Computing Machinery, New York, NY, USA, Article 81, 9 pages.

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A Quadratic Bending Model for Inextensible Surfaces. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing (SGP '06)*. Eurographics Association, Goslar, DEU, 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.

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

R. Bridson, S. Marino, and R. Fedkiw. 2005. Simulation of Clothing with Folds and Wrinkles. In *ACM SIGGRAPH 2005 Courses (SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 28–36.

Yunuo Chen, Minchen Li, Lei Lan, Hao Su, Yin Yang, and Chenfanfu Jiang. 2022. A Unified Newton Barrier Method for Multibody Dynamics. *ACM Trans. Graph.* 41, 4, Article 66 (jul 2022), 14 pages.

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but Responsive Cloth. *ACM Trans. Graph.* 21, 3 (jul 2002), 604–611.

Andrew R Conn, Nicholas IM Gould, and Ph L Toint. 1988. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM journal on numerical analysis* 25, 2 (1988), 433–460.

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

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. 1999. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 317–324.

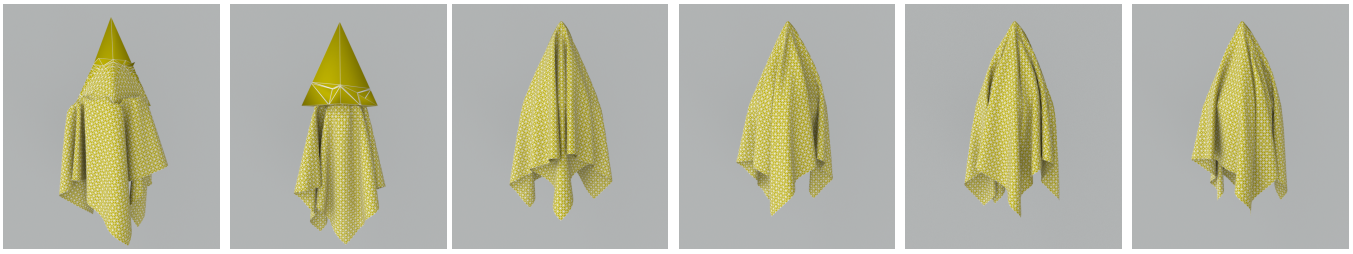
Kenny Erleben. 2013. Numerical Methods for Linear Complementarity Problems in Physics-Based Animation. In *ACM SIGGRAPH 2013 Courses (SIGGRAPH '13)*. Association for Computing Machinery, New York, NY, USA, Article 8, 42 pages.

Olaf Etzmuß, Michael Keckeisen, and Wolfgang Straßer. 2003. A fast finite element solution for cloth modelling. In *11th Pacific Conference on Computer Graphics and Applications, 2003. Proceedings.* IEEE, 244–251.

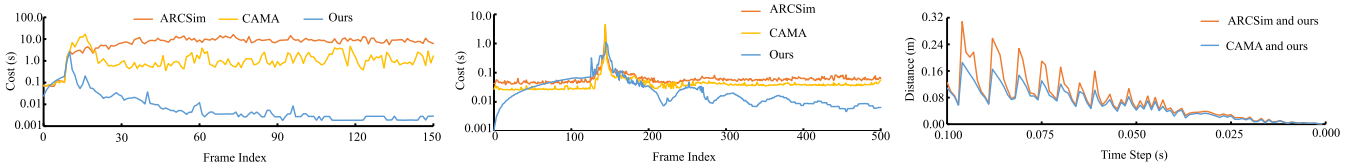
Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed Globally Injective 3D Deformation Processing. *ACM Trans. Graph.* 40, 4, Article 75 (jul 2021), 13 pages.

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

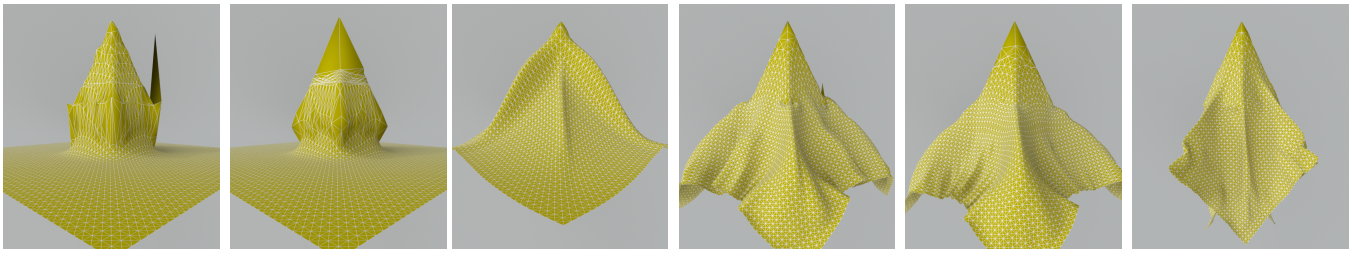
Ming Gao, Xinlei Wang, Kui Wu, Andre Pradhana, Eftychios Sifakis, Cem Yuksel, and Chenfanfu Jiang. 2018. GPU Optimization of Material Point Methods. *ACM Trans. Graph.* 37, 6, Article 254 (dec 2018), 12 pages.



(a) CAMA (Δt = 1/10s) (b) ARCSim (Δt = 1/10s) (c) Ours (Δt = 1/10s) (d) CAMA (Δt = 1/125s) (e) ARCSim (Δt = 1/125s) (f) Ours (Δt = 1/125s)



(g) Time cost comparison (Δt = 1/10s) (h) Time cost comparison (Δt = 1/125s) (i) Mean vertex distance curve



(j) CAMA (Δt = 25/1000s, CLK = 1.05s) (k) ARCSim (Δt = 25/1000s, CLK = 1.05s) (l) Ours (Δt = 25/1000s, CLK = 1.05s) (m) CAMA (Δt = 25/1000s, CLK = 1.2s) (n) ARCSim (Δt = 25/1000s, CLK = 1.2s) (o) Ours (Δt = 25/1000s, CLK = 1.2s)

Fig. 19. **Robustness to large time steps.** A piece of cloth (1m×1m) falling on a sharp cone is simulated with a decreasing time step, from Δt=1/10s to Δt=1/1000s. Compared to existing impact zone methods such as [Narain et al. 2012; Tang et al. 2016], our method produces nearly artifact-free dynamic simulation even with large time steps, exhibiting superior performance. In contrast, existing methods suffer from visible artifacts until the time step is reduced to a small enough value.

A multi-layered dress A multi-layered gown

Fig. 20. **Multi-layered clothing.** Our method reliably handles complex collisions that occur among multiple layers of clothing worn by these dancing characters.

Eitan Grinspun, Anil N. Hirani, Mathieu Desbrun, and Peter Schröder. 2003. Discrete Shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '03)*. Eurographics Association, Goslar, DEU, 62–67.

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

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

David Harmon, Qingnan Zhou, and Denis Zorin. 2011. Asynchronous Integration with Phantom Meshes. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '11)*. Association for Computing Machinery, New York, NY, USA, 247–256.

Couro Kane, Eduardo A Repetto, Michael Ortiz, and Jerrold E Marsden. 1999. Finite element analysis of nonsmooth contact. *Computer methods in applied mechanics and engineering* 180, 1-2 (1999), 1–26.

Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022. Penetration-Free Projective Dynamics on the GPU. *ACM Trans. Graph.* 41, 4, Article 69 (jul 2022), 16 pages.

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.* 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.* 39, 4, Article 49 (jul 2020), 20 pages.

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

Chih-Jen Lin and Jorge J Moré. 1999. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization* 9, 4 (1999), 1100–1127.

Mickaël Ly, Jean Jouve, Laurence Boissieux, and Florence Bertails-Descoubes. 2020. Projective Dynamics with Dry Frictional Contact. *ACM Trans. Graph.* 39, 4, Article 57 (jul 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 and Matthias Muller. 2021. A Constraint-Based Formulation of Stable Neo-Hookean Materials. In *Motion, Interaction and Games (MIG '21)*. Association for Computing Machinery, New York, NY, USA, Article 12, 7 pages.

Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. 2011. Example-Based Elastic Materials. In *ACM SIGGRAPH 2011 Papers (SIGGRAPH '11)*. Association for Computing Machinery, New York, NY, USA, Article 72, 8 pages.

Stan Melax. 2017. 3x3 Matrix Diagonalization. http://melax.github.io/diag.html

Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*. Springer, 35–57.

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.

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. 2015. Air Meshes for Robust Collision Handling. *ACM Trans. Graph.* 34, 4, Article 133 (jul 2015), 9 pages.

Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive Anisotropic Remeshing for Cloth Simulation. *ACM Trans. Graph.* 31, 6, Article 152 (nov 2012), 10 pages.

Jorge Nocedal and Stephen J Wright. 2006. *Numerical optimization*. Springer.

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.

Ulrich Pinkall and Konrad Polthier. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental mathematics* 2, 1 (1993), 15–36.

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 the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '08)*. Eurographics Association, Goslar, DEU, 147–153.

Breannan Smith, Fernando De Goes, and Theodore Kim. 2019. Analytic Eigensystems for Isotropic Distortion Energies. *ACM Trans. Graph.* 38, 1, Article 3 (feb 2019), 15 pages.

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (jul 2015), 9 pages.

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 (jul 2018), 18 pages.

Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong. 2011a. Collision-Streams: Fast GPU-Based Collision Detection for Deformable Models. In *Symposium on Interactive 3D Graphics and Games (I3D '11)*. Association for Computing Machinery, New York, NY, USA, 63–70.

Min Tang, Dinesh Manocha, Sung-Eui Yoon, Peng Du, Jae-Pil Heo, and Ruo-Feng Tong. 2011b. 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.* 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.* 37, 6, Article 204 (dec 2018), 10 pages.

Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. 2005. Robust Quasistatic Finite Elements and Flesh Simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '05)*. Association for Computing Machinery, New York, NY, USA, 181–190.

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 (mar 2019), 20 pages.

Pascal Volino and Nadia Magnenat-Thalmann. 2006. Resolving Surface Collisions through Intersection Contour Minimization. In *ACM SIGGRAPH 2006 Papers (SIGGRAPH '06)*. Association for Computing Machinery, New York, NY, USA, 1154–1159.

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

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2021. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Transactions on Graphics (TOG)* 40, 5 (2021), 1–16.

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

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

Huamin Wang and Yin Yang. 2016. Descent Methods for Elastic Body Simulation on the GPU. *ACM Trans. Graph.* 35, 6, Article 212 (nov 2016), 10 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.

Cem Yuksel. 2022. A Fast & Robust Solution for Cubic & Higher-Order Polynomials. In *ACM SIGGRAPH 2022 Talks (SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 28, 2 pages.

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