A Safe and Fast Repulsion Method for GPU-based Cloth Self Collisions

LONGHUA WU, The Ohio State University, USA
BOTAO WU, Frilly Inc., USA
YIN YANG, Clemson University, USA
HUAMIN WANG, The Ohio State University, USA

(a) A rest configuration (b) A lifted configuration (c) A folded configuration (d) A stretched configuration

Fig. 1. A shirt example with 56K vertices and 112K triangles. The front of this shirt contains five cloth layers. Thanks to a safe and fast repulsion method for cloth self collisions, our GPU-based cloth simulator animates this shirt at 38 to 72 FPS with a large time step (Δt = 1/100s). This is at least one order of magnitude faster than existing cloth simulators [Tang et al. 2018b].

Cloth dynamics and collision handling are the two most challenging topics in cloth simulation. While researchers have substantially improved the performances of cloth dynamics solvers recently, their success in fast collision detection and handling is rather limited. In this article, we focus our research on the safety, efficiency, and realism of the repulsion-based collision handling approach, which has demonstrated its potential in existing GPU-based simulators. Our first discovery is the necessary vertex distance conditions for cloth to enter self intersections, the negations of which can be viewed as vertex distance constraints continuous in time for sufficiently avoiding self collisions. Continuous constraints, however, cannot be enforced with ease. Our solution is to convert continuous constraints into three types of constraints: discrete edge length constraints, discrete vertex distance constraints, and vertex displacement constraints. Based on this solution, we develop a fast and safe collision handling process for enforcing constraints, a novel splitting method for integrating collision handling with dynamics solvers, and static and adaptive remeshing schemes to further improve the runtime performance. In summary, our cloth simulator is efficient, safe, robust, and parallelizable on a GPU. The experiment shows that it runs at least one order of magnitude faster than existing simulators.

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

Additional Key Words and Phrases: Vertex repulsion, the splitting method, cloth simulation, GPU acceleration, collision detection and handling

ACM Reference format:
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 (December 2020), 18 pages.
https://doi.org/10.1145/3430025

1 INTRODUCTION

Physics-based cloth simulation is a highly demanded computer graphics technique, and it is expected to revolutionize our lives through virtual design, marketing, and shopping applications in the near future. But there is a huge obstacle in the path of realizing these applications: the computational efficiency. Recent research [Bouaziz et al. 2014; Fratarcangeli et al. 2016; Liu et al. 2013; Narain et al. 2016; Wang 2015; Wang and Yang 2016; Wang et al. 2018] has successfully improved the efficiency of cloth dynamics solvers by formulating cloth dynamics as a nonlinear optimization

The authors would like to thank NVIDIA and Adobe for additional funding and equipment support.
Authors' addresses: L. Wu and H. Wang, The Ohio State University, Dreese Labs, 2015 Neil Avenue, Columbus, OH 43210-1277; emails: wu.2724@buckeyemail.osu.edu, whmin@cse.ohio-state.edu; B. Wu, Frilly Inc., 207 S Broadway Fl 7,Los Angeles, CA 90012-3609; email: botaow24@outlook.com; Y. Yang, Clemson University, 100 McAdams Hall, Clemson, S.C. 29634; email: yin5@clemson.edu.

0730-0301/2020/12-ART5 $15.00
https://doi.org/10.1145/3430025

problem and solving it with graphics hardware acceleration. This success depends heavily on the use of large time steps,[1] which lessens computational overheads and enhances the benefit of acceleration methods. Unfortunately, recent research progress on self collision handling of cloth is rather limited. Existing cloth collision handling algorithms, including both discrete and continuous ones [Baraff et al. 2003; Bridson et al. 2003; Volino and Magnenat-Thalmann 2006], are often unfriendly with graphics hardware due to their localized and sequentialized workloads. Parallelizing and accelerating these algorithms [Lauterbach et al. 2010; Tang et al. 2018a, 2016, 2018b] is not trivial, and the recent result is still nowhere close to real-time performance. In general, fast and robust self collision handling of cloth is a challenging problem in computer graphics.

A common way of dealing with cloth self collisions on a GPU is to apply vertex distance constraints, i.e., pushing vertices apart so cloth does not get entangled. While this practice has demonstrated its effectiveness and efficiency in previous systems [Fratarcangeli et al. 2016; Macklin et al. 2014; Stam 2009], it suffers from a critical limitation: A vertex can penetrate through a triangle without getting close to any triangle vertex. This issue can be lessened by adding interior triangle samples and increasing the repulsion distance threshold, but the risk remains, especially if triangles are overly stretched. The use of a large repulsion distance threshold also worsens early repulsion artifacts. For example, cloth sheets behave as if they float over each other without making real contact.

In this article, we focus our research on the development of a novel repulsion-based collision handling approach that is:

- *safe*, i.e., strictly free of self penetration at any time;
- *visually correct,* i.e., hardly visible repulsion artifacts caused by early responses, even though they still exist;
- and *efficient*, i.e., taking a small computational time.

We build the foundation of our research upon necessary conditions on vertex distances for a triangle mesh to enter self intersections. We then view the negations of these conditions as sufficient vertex distance constraints for avoiding self intersections, which must be satisfied continuously in time. To achieve this goal, we solve a series of technical problems and we make the following technical contributions.

- *Constraint enforcement.* Our first question is: How can we strictly satisfy the aforementioned constraints defined continuously in time? The answer to this question lies in the discovery that continuous constraints can be satisfied sufficiently by enforcing vertex distance and edge length constraints defined discretely at every vertex state, and vertex displacement constraints that restrict the distances the mesh vertices can travel during a collision step. Based on this discovery, we formulate our collision handling process as two phases: a soft phase that takes a small computational time to satisfy constraints most of the time; and a hard phase that enforces constraints strictly at the cost of more computational time and/or less accuracy.
- *Dynamics-collision integration.* To increase the success rate of the soft phase and to improve the performance of collision handling, we should limit the initial vertex displacement of each collision step. But here comes another question: How can we integrate collision handling with any cloth dynamics solver, especially those state-of-the-art ones demanding the use of large time steps? Simply synchronizing a collision step with a time step or one iteration of a dynamics solver would undermine the overall runtime performance. Our solution is a splitting method that treats cloth dynamics and collision handling as two independent yet coupled processes. Thanks to this method, the dynamics solver runs freely without any restriction on vertex displacement, while collision handling tries to catch up with the solver result at its own pace.
- *Uniform mesh sampling.* The analysis in the development of our collision handling process is based on the assumption that all of the edges are shorter than a global constant threshold. For fast and realistic simulation, we prefer to minimize the difference of edge lengths in both the reference configuration and the deformed configuration. To do so, we present a uniform mesh sampling scheme for the reference configuration and an adaptive mesh resampling scheme for the deformed configuration. Being developed for a GPU, the adaptive scheme is of particular importance, as it prevents edge length constraints from consuming too much computational time in simulation, especially when cloth gets overly stretched.

By integrating the proposed techniques into a novel cloth simulator and experimenting it on a GPU, we have successfully demonstrated its efficiency, safety in collision handling, and robustness against large time steps and deformations, as shown in Figure 1. Perhaps an even greater strength of our simulator is its freedom of adopting various cloth dynamics solvers. Since it treats cloth dynamics and collision handling as two independent processes, we can conveniently replace the current dynamics solver by others in the future.

2 RELATED WORK

Cloth collision handling. How to safely and efficiently handle cloth self collisions is known to be a challenging problem. Since the visual artifacts caused by cloth collisions are highly noticeable in cloth simulation, the early work on continuous collision handling [Bridson et al. 2002; Provot 1997] aimed at detecting and removing any self intersection that can happen within a time step. Brochu and colleagues [2012], Wang [2014], and Tang and collaborators [2014] later improved the accuracy of continuous collision detection tests with respect to floating point errors. Continuous collision handling techniques [Bridson et al. 2002; Harmon et al. 2008; Huh et al. 2001; Provot 1997] typically use geometric impulses and impact zones to remove detected collisions under the assumption that the time step is sufficiently small. When the time step is large, impulse methods are likely to fail, and impact zone methods are likely to suffer from locking and convergence issues. If self penetration artifacts are tolerable, cloth simulators can also use discrete collision handling techniques [Baraff et al. 2003; Buffet et al. 2019; Volino and Magnenat-Thalmann 2006; Wicke et al.

[1] When simulating cloth with its maximal velocity under 10 m/s, we consider the time step to be large if it is above 1/100 s or small if it is below 1/1000 s.

2006] to untangle cloth at the end of every time step. When the time step is large, these techniques still work, but they often fail to eliminate self penetrations even after many tries.

Since most continuous and discrete collision handling algorithms are naturally unfriendly with GPU acceleration, a popular and efficient collision handling practice on a GPU [Fratarcangeli et al. 2016; Macklin et al. 2014; Stam 2009] is to simply push vertices and samples apart from each other. Unfortunately, as mentioned earlier, this practice alone is not safe: It cannot prevent self collisions from happening, especially when the time step is large or cloth gets overly stretched.

In conclusion, the large time step is the common foe to most of the collision handling algorithms. As pointed out by Harmon and colleagues [2009], it is fundamentally the large vertex displacement happening within one time step that causes all of the problems. To make collision handling safe and robust, we should find a way to limit the vertex displacement.

Collision culling. One research topic relevant but orthogonal to this research is collision culling, i.e., avoiding unnecessary collision tests if collisions do not happen. Collision culling techniques can be roughly grouped into two categories: spatial partitioning and bounding volume hierarchy [Schvartzman et al. 2010; Tang et al. 2010, 2011; Zheng and James 2012]. GPU-based implementations of both categories have been investigated [Lauterbach et al. 2010; Pabst et al. 2010; Tang et al. 2018a, 2018b] previously, although grid-based spatial partitioning on a GPU is slightly more popular in our opinion, thanks to its simplicity and fast memory access.

Recently, Tang and collaborators [2018b] developed an incremental collision algorithm for iterative impact zone optimization. Their simulator is similar to ours in that both must run multiple collision handling processes in every time step. The difference is that each of their processes is essentially one post-processing iteration in which only a few vertices move. As a result, their simulator performs collision culling and detection only once per time step.

Dynamics-collision integration. How to integrate cloth dynamics solvers with collision handling algorithms is an important yet often overlooked problem. When the time step is small, e.g., when using explicit time integration, simulators [Bridson et al. 2003; Choi and Ko 2002] can simply perform collision handling after cloth dynamics as post-processing at the end of every time step. But when the time step is large, especially when we use modern cloth dynamics solvers, simulators must use many sub-steps if we still want to post-process collisions [Bridson et al. 2002; Selle et al. 2009]. To improve the simulation performance, Thomaszewski and collaborators [2008] proposed to update vertices asynchronously using different time steps, so the simulation does not get restricted uniformly by small time steps or sub-steps. Based on the fact that most cloth dynamics solvers are iterative, a more plausible approach is to handle collisions as projection after every dynamics iteration. This approach requires to use relatively inexpensive collision handling processes, such as vertex repulsion [Macklin et al. 2014; Stam 2009]. While this approach suffers less from the use of large time steps, it can still run into collision handling problems, especially in the first few iterations when vertex displacements are large. The situation can be worsened once we start to consider accelerated or multi-resolutional solvers [Aksoylu et al. 2005; Green et al. 2002; Jeon et al. 2013; Tamstorf et al. 2015; Wang et al. 2018], which try to achieve even larger vertex displacements for faster convergence.

A relatively unpopular idea of avoiding troubles in the integration of cloth dynamics and collision handling is to use two state vectors, one for cloth dynamics and one for collision handling. This idea was initially explored for flexible cloth simulation of nonconforming meshes [English and Bridson 2008] and embedded point samples [Sifakis et al. 2007], which are not directly suitable for collision handling. Later, Harmon and colleagues [2011] developed the phantom mesh method using two vertex state vectors, so the dynamics solver is unaffected by the use of small collision sub-steps [Harmon et al. 2009]. These existing techniques typically perform the coupling of the two state vectors only once at the end of every time step. This is an acceptable practice when the time step is small. But as the time step gets larger, decoupling artifacts can emerge and we must address the coupling issue differently, as discussed in Section 5.

3 DISTANCE CONDITIONS FOR SELF INTERSECTIONS

To begin with, we would like to derive necessary conditions on vertex distances for a triangle mesh to become self intersected. There are two possible ways to achieve self intersections: vertex-triangle intersection and edge-edge intersection.

3.1 Vertex-triangle Intersection

Let $\mathbf{x}_i \in \mathbb{R}^3$ be the 3D position of vertex i and $\mathbf{x}_j\mathbf{x}_k\mathbf{x}_l$ be the vertex positions of triangle jkl. When they are in intersection, we define $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ as the distance from $\mathbf{x}_i$ to the closest triangle corner:

$$D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l) = \min\left(\|\mathbf{x}_{ij}\|, \|\mathbf{x}_{ik}\|, \|\mathbf{x}_{il}\|\right), \tag{1}$$

in which $\mathbf{x}_{ij} = \mathbf{x}_i - \mathbf{x}_j$ is the relative position from $\mathbf{x}_j$ to $\mathbf{x}_i$. Since $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ is continuous and $\mathbf{x}_i$ is inside of a closed triangle, there must exist $\mathbf{x}_i^*$ that maximizes $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$, according to the extreme value theorem. If the triangle is acute, $\mathbf{x}_i^*$ must be its interior circumcenter, as Theorem A.1 shows; otherwise, $D(\mathbf{x}_i^*, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ cannot exceed half of the longest edge length, as Theorem A.2 shows. Together, we obtain an upper bound $B_{i,jkl}$ on $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$:

$$D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l) \le B_{i,jkl} = \begin{cases} r_{jkl} = \dfrac{\|\mathbf{x}_{jk}\| \cdot \|\mathbf{x}_{kl}\| \cdot \|\mathbf{x}_{lj}\|}{\sqrt{l_{jkl}(l_{jkl}-2\|\mathbf{x}_{jk}\|)(l_{jkl}-2\|\mathbf{x}_{kl}\|)(l_{jkl}-2\|\mathbf{x}_{lj}\|)}}, & \text{if acute,} \\ \frac{1}{2}\max(\|\mathbf{x}_{jk}\|, \|\mathbf{x}_{kl}\|, \|\mathbf{x}_{lj}\|), & \text{otherwise,} \end{cases} \tag{2}$$

in which $l_{jkl} = \|\mathbf{x}_{jk}\| + \|\mathbf{x}_{kl}\| + \|\mathbf{x}_{lj}\|$ is the sum of the triangle edge lengths and r_{jkl} is the circumradius. If the triangle is acute, Theorem A.3 proves that r_{jkl} is a monotonically increasing function of any edge length. Overall, $B_{i,jkl}$ is a monotonically increasing function of any edge length: It does not decrease as any edge length increases.

3.2 Edge-edge Intersection

Let $\mathbf{x}_i\mathbf{x}_j$ and $\mathbf{x}_k\mathbf{x}_l$ be the vertex positions of two intersecting edges. We define $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ as the shortest distance of the four vertex pairs:

$$D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l) = \min(\|\mathbf{x}_{ik}\|, \|\mathbf{x}_{il}\|, \|\mathbf{x}_{jk}\|, \|\mathbf{x}_{jl}\|). \tag{3}$$

Let $\|\mathbf{x}_{ij}\|$ and $\|\mathbf{x}_{kl}\|$ be the given lengths of the two edges. Theorem A.4 proves that $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ is maximized only if the edges intersect perpendicularly at their midpoints. In that case, we have:

$$D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l) \leq B_{ij,kl} = \frac{1}{2}\sqrt{\|\mathbf{x}_{ij}\|^2 + \|\mathbf{x}_{kl}\|^2}. \tag{4}$$

Here $B_{ij,kl}$ serves as an upper bound on $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$, which is also a monotonically increasing function of any edge length.

4 CONSTRAINT FORMULATION AND ENFORCEMENT

In this section, we will first formulate constraints for a triangle mesh to stay away from self intersections, based on the upper bounds on vertex distances provided in Section 3. We will then discuss how to enforce those constraints strictly and efficiently.

4.1 Continuous and Discrete Constraints

Since Equations (2) and (4) are necessary conditions for vertex-triangle and edge-edge pairs to intersect, we can treat their negations as sufficient conditions to avoid self intersections. A realistic issue though is that $B_{i,jkl}$ and $B_{ij,kl}$ are functions of edge lengths varying over time. To address this issue, we propose a constant upper bound L on all of the edge lengths. Since both $B_{i,jkl}$ and $B_{ij,kl}$ are monotonically increasing functions of edge lengths, we treat $B = \max(B_{i,jkl}, B_{ij,kl}) = \max(L/\sqrt{3}, L/\sqrt{2}) = L/\sqrt{2}$ as the global lower bound on the distance between any two non-adjacent vertices within a mesh. We then describe the intersection-free constraints as:

$$\begin{cases} \|\mathbf{x}_{ij}(t)\|^2 \leq L^2, & \text{if } \{i,j\} \text{ is an edge,} \\ \|\mathbf{x}_{ij}(t)\|^2 \geq B^2 = L^2/2, & \text{otherwise,} \end{cases} \tag{5}$$

for t being any time within the simulation period. In practice, we perform cloth simulation through a sequence of updates $\{\mathbf{x}^{[k]}\}$ and we typically assume that the trajectory between two consecutive updates is linear: $\mathbf{x}(t) = (1-t)\mathbf{x}^{[k]} + t\mathbf{x}^{[k+1]}$. In that case, $t \in [0,1]$ is an interpolant between two updates. We then convert Equation (5) into quadratic inequalities of t:

$$\begin{cases} \left\|(1-t)\mathbf{x}_{ij}^{[k]} + t\mathbf{x}_{ij}^{[k+1]}\right\|^2 \leq L^2, & \text{if } \{i,j\} \text{ is an edge,} \\ \left\|(1-t)\mathbf{x}_{ij}^{[k]} + t\mathbf{x}_{ij}^{[k+1]}\right\|^2 \geq B^2, & \text{otherwise,} \end{cases} \tag{6}$$

for any $t \in (0,1]$. We note that Equation (6) is satisfied at $t = 1$ during the last update, so it is automatically satisfied at $t = 0$ during the current update. To detect any constraint violation, we calculate t that minimizes/maximizes the left-hand sides of Equation (6) and then evaluate the inequalities directly. We can even eliminate t from Equation (6) by its closed form and formulate the constraints solely as functions of $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$. Unfortunately, we cannot easily enforce such constraints due to their complexity.

Instead of enforcing *continuous constraints*, i.e., those being defined continuously over time in Equation (6), we propose to derive and enforce *discrete constraints* being defined solely at every update $\mathbf{x}^{[k+1]}$ in a way that the satisfaction of discrete constraints guarantees the satisfaction of continuous constraints. When $\{i,j\}$ is an edge, it is straightforward to see that if $\|\mathbf{x}_{ij}^{[k]}\|^2 \leq L^2$ and $\|\mathbf{x}_{ij}^{[k+1]}\|^2 \leq L^2$, then $\|\mathbf{x}_{ij}(t)\|^2 \leq L^2$ as well for $t \in [0,1]$. This means we can simply turn continuous edge length constraints into discrete ones defined at every update $\mathbf{x}^{[k+1]}$. But when $\{i,j\}$ is not an edge, we may not be able to obtain the same conclusion. Let $t^* \in [0,1]$ be the interpolant that minimizes $\|\mathbf{x}_{ij}(t)\|^2$. There are three possibilities: $t^* = 0$, $t^* = 1$, or $t^* \in (0,1)$ when $\mathbf{x}_{ij}(t) \cdot (\mathbf{x}_{ij}^{[k+1]} - \mathbf{x}_{ij}^{[k]}) = 0$. In all of the three cases, we know:

$$\begin{aligned} &\|\mathbf{x}_{ij}(t)\|^2 \geq \|\mathbf{x}_{ij}(t^*)\|^2 \geq \\ &\quad \min\left(\left\|\mathbf{x}_{ij}^{[k+1]}\right\|^2, \left\|\mathbf{x}_{ij}^{[k]}\right\|^2\right) - \tfrac{1}{4}\left\|\mathbf{x}_{ij}^{[k+1]} - \mathbf{x}_{ij}^{[k]}\right\|^2. \end{aligned} \tag{7}$$

If we use a small constant B^{tight} to tighten every vertex distance constraint:

$$\left\|\mathbf{x}_{ij}^{[k]}\right\|^2, \left\|\mathbf{x}_{ij}^{[k+1]}\right\|^2 \geq (B + B^{\text{tight}})^2, \quad \text{if } \{i,j\} \text{ is not an edge,} \tag{8}$$

and set an upper limit on the displacement of each vertex:

$$\left\|\mathbf{x}_i^{[k+1]} - \mathbf{x}_i^{[k]}\right\|^2 \leq (B + B^{\text{tight}})^2 - B^2, \tag{9}$$

we then obtain:

$$\left\|\mathbf{x}_{ij}^{[k+1]} - \mathbf{x}_{ij}^{[k]}\right\|^2 \leq 4(B + B^{\text{tight}})^2 - 4B^2, \tag{10}$$

and we know $\|\mathbf{x}_{ij}(t)\|^2 \geq B^2$ for any $t \in [0,1]$. In summary, we have the following discrete constraints on $\mathbf{x}_{ij}^{[k+1]}$:

$$\begin{cases} \left\|\mathbf{x}_{ij}^{[k+1]}\right\|^2 \leq L^2, & \text{if } \{i,j\} \text{ is an edge,} \\ \left\|\mathbf{x}_{ij}^{[k+1]}\right\|^2 \geq (B + B^{\text{tight}})^2, & \text{otherwise,} \end{cases} \tag{11}$$

and $\|\mathbf{x}_i^{[k+1]} - \mathbf{x}_i^{[k]}\|^2 \leq (B + B^{\text{tight}})^2 - B^2$ for every vertex i. The satisfaction of discrete constraints in Equation (11) and vertex displacement constraints in Equation (9) ensures the satisfaction of continuous constraints in Equation (6).

Without loss of generality, we define a continuous constraint as $C_{ij}(\mathbf{x}^{[k]}, \mathbf{x}^{[k+1]}) \geq 0$ and a discrete constraint as $c_{ij}(\mathbf{x}^{[k+1]}) \geq 0$ in the rest of this article.

4.1.1 The Avoidance of Vertex Displacement Constraints. An important idea we will explore next in Section 4.2 is that we choose not to deal with vertex displacement constraints explicitly for two reasons. First, vertex displacement constraints are not always necessary, and enforcing them would slow down vertex movement. For instance, vertex distance constraints do not allow a vertex and a triangle to move at the same high velocity, even though they satisfy continuous constraints. Second, we can effectively reduce the collision handling cost, especially because vertex displacement constraints are not so compatible with others in a single GPU kernel, and they may need their own kernel.

We would like to emphasize that the above discussion does not suggest vertex displacement constraints are meaningless. Instead they provide theoretical guarantee on the efficiency and safety of our methods. In Section 4.2.1, they will justify the use of a small initial vertex displacement to improve the success rate of the soft phase; in Section 4.2.2, they will ensure the safe termination of the hard phase by finding a small step size.

4.2 Constraint Enforcement

Let $\mathbf{x}^{[k]}$ be the latest vertex state vector and $\mathbf{x}^{\text{init}}$ be the initial vertex state vector for the next update. Our goal is to find the next update $\mathbf{x}^{[k+1]}$, which is the closest to $\mathbf{x}^{\text{init}}$ and satisfies continuous constraints. According to Section 4.1, we choose to enforce

discrete constraints only and we formulate the constraint enforcement problem as:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \|\mathbf{x} - \mathbf{x}^{\text{init}}\|^2, \quad \text{s.t. } c_{ij}(\mathbf{x}) \geq 0 \text{ for any } \{i,j\}. \tag{12}$$

Since the original goal is to satisfy continuous constraints, we incorporate both continuous and discrete constraints into the termination condition of the constraint enforcement process.

The key problem is: *How can we achieve the satisfaction of continuous constraints and discrete constraints without even addressing vertex displacement constraints*? This problem is particularly important to the development of a real-time simulator, which requires a careful balance between the performance and the accuracy. Our solution is to divide constraint enforcement into two phases. In the soft phase, we try to find an acceptable $\mathbf{x}^{[k+1]}$ quickly first. Under a small initial displacement assumption, we aim at making most of the soft phase cases successful. But if the soft phase does fail, we switch to run a failsafe method in the hard phase, which can be safely terminated within a limited computational time, thanks to novel step size conditions.

4.2.1 The Soft Phase. In the soft phase, we calculate $\mathbf{x}^{[k+1]}$ as the projection of $\mathbf{x}^{\text{init}}$ to the feasible region: $c(\mathbf{x}) \geq 0$, as shown in Figure 3(a). This approach is fast and effective most of the time, since $\mathbf{x}^{\text{init}}$ is often close to the feasible region. Let ϵ^{slack} be a positive slack constant. We convert the original constrained optimization problem into the following unconstrained one:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \left\{ \|\mathbf{x} - \mathbf{x}^{\text{init}}\|^2 - \rho \sum_{\{i,j\}} \min\left(c_{ij}(\mathbf{x}) - \epsilon^{\text{slack}}, 0\right) \right\}, \tag{13}$$

in which ρ is a penalty strength constant and each constraint penalty term is active when $\mathbf{x}$ violates its corresponding discrete constraint: $c_{ij}(\mathbf{x}) \geq \epsilon^{\text{slack}}$. We treat $\mathbf{x}^{\text{init}}$ as the initialization and run a fixed number I^{soft} of gradient descent iterations to reduce the objective. After that, we test whether the result satisfies both continuous and discrete constraints. If it does, we declare a soft phase success and terminate the collision handling process. Otherwise, we abandon the soft phase and switch to the hard phase next.

There are three possible reasons why the soft phase can fail.

First, since the objective in Equation (13) is non-convex, the optimization can get stuck in a local optimum that fails to satisfy all of the constraints. For example, when a 2D square is stuffed with four alien vertices as shown in Figure 2, it cannot satisfy vertex distance and edge length constraints simultaneously, while the optimization can still reach an equilibrium state corresponding a local minimum. Fortunately, such local minimum cases are extremely uncommon in practice.

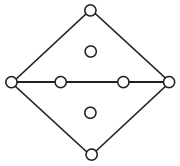


Fig. 2. A 2D square example. In this example, vertex distance constraints try to push vertices apart, while edge length constraints try to hold the square from being too expanded.

Second, the soft phase can fail, because discrete constraints are satisfied while continuous constraints are not. We can eliminate this possibility by enforcing vertex displacement constraints, but we prefer not to do so as discussed in Section 4.1.1. Instead, we assume that the initial displacement $\mathbf{x}^{\text{init}} - \mathbf{x}^{[k]}$ is sufficiently small. Ideally, if $\mathbf{x}^{[k+1]}$ is the closest[2] feasible point to $\mathbf{x}^{\text{init}}$, we have:

$$\begin{aligned} \left\|\mathbf{x}^{[k+1]} - \mathbf{x}^{[k]}\right\| &\leq \left\|\mathbf{x}^{\text{init}} - \mathbf{x}^{[k]}\right\| + \left\|\mathbf{x}^{\text{init}} - \mathbf{x}^{[k+1]}\right\| \\ &\leq 2\left\|\mathbf{x}^{\text{init}} - \mathbf{x}^{[k]}\right\|. \end{aligned} \tag{14}$$

Thus, the resulting displacement $\mathbf{x}^{[k+1]} - \mathbf{x}^{[k]}$ must also be small to eventually satisfy continuous constraints, as in Section 4.1.

Finally, the soft phase can fail due to an insufficient number of iterations, I^{soft}. As before, we rely on the use of a small initial displacement to address this issue. As the initial displacement decreases, the distance from $\mathbf{x}^{\text{init}}$ to the feasible region drops, which implies a lower demand of a large I^{soft}.

From the above analysis, we see that the use of a small initial displacement is essential to the reduction of soft phase failures. That being said, we cannot strictly prevent them from happening and we need the hard phase to terminate the collision process next.

4.2.2 The Hard Phase. The main purpose of the hard phase is to guarantee the safe termination of the collision handling process within a limited computational time. In this regard, its safety and efficiency are more important than its accuracy. Here, we model constraints as log barriers and apply the interior point method:

$$\mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \left\{ \|\mathbf{x} - \mathbf{x}^{\text{init}}\|^2 - \mu \sum_{\{i,j\}} \log\left(f(c_{ij}(\mathbf{x}), \epsilon^{\text{slack}})\right) \right\}, \tag{15}$$

where μ is a log barrier constant and $f(x,\epsilon)$ is a piecewise function:

$$f(x,\epsilon) = \begin{cases} x, & \text{if } x \leq 0, \\ a_3x^3 + a_2x^2 + a_1x + a_0, & \text{if } 0 < x \leq \epsilon, \\ \epsilon, & \text{if } \epsilon \leq x, \end{cases} \tag{16}$$

in which the coefficients are calculated for C^0 and C^1 continuity:

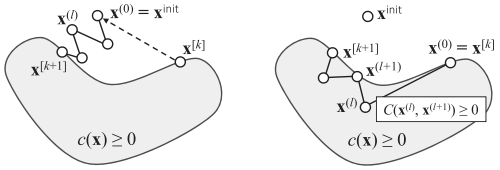
$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ \epsilon^3 & \epsilon^2 & \epsilon & 1 \\ 0 & 0 & 1 & 0 \\ 3\epsilon^2 & 2\epsilon & 1 & 0 \end{bmatrix} \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \epsilon \\ 1 \\ 0 \end{bmatrix}. \tag{17}$$

Intuitively, $f(x,\epsilon)$ limits the influence of the constraints within a range $x = c_{ij}(\mathbf{x}) \in (0, \epsilon^{\text{slack}}]$. We note that when $x \leq 0$, $f(x,\epsilon)$ is defined but Equation (15) is undefined. This is not an issue, because the interior point method forces $\mathbf{x}$ to be strictly feasible: $c_{ij}(\mathbf{x}) > 0$. To do so, we treat $\mathbf{x}^{[k]}$ as the initialization and we run a number of gradient descent iterations as shown in Figure 3(b) to reduce $F^{\text{hard}}(\mathbf{x})$, the objective of Equation (15):

$$\mathbf{x}^{(l+1)} = \mathbf{x}^{(l)} - \alpha_l^{\text{hard}} \nabla F^{\text{hard}}(\mathbf{x}^{(l)}), \tag{18}$$

in which α_l^{hard} is the step size. We choose a relatively small initial step size $\alpha_0^{\text{hard}} = 0.1$ and then apply backtracking line search to adjust it gradually until it achieves three conditions:

[2] In reality, the calculated result $\mathbf{x}^{[k+1]}$ may not be closer to $\mathbf{x}^{\text{init}}$ than $\mathbf{x}^{[k]}$ for many reasons, such as the non-convexity of the objective. Fortunately, this becomes unlikely as the initial displacement decreases, if we assume that the objective is convex within a small neighborhood of $\mathbf{x}^{\text{init}}$.



(a) The soft phase (b) The hard phase

Fig. 3. The iterations of the two phases. While the soft phase can reach its solution fast as shown in (a), the hard phase ensures that every $\mathbf{x}^{(l)}$ is acceptable: $c(\mathbf{x}^{(l)}) \geq 0$ and $C(\mathbf{x}^{(m)}, \mathbf{x}^{(m+1)}) \geq 0$ for $m = 0 \ldots l-1$, as shown in (b). Therefore, we can terminate the hard phase safely at any $\mathbf{x}^{(l)}$, regardless of the objective convergence.

- Every result $\mathbf{x}^{(l+1)}$ is strictly within the feasible region, i.e., $c_{ij}(\mathbf{x}^{(l+1)}) > 0$;
- The objective $F^{\text{hard}}(\mathbf{x})$ decreases adequately from $\mathbf{x}^{(l)}$ to $\mathbf{x}^{(l+1)}$, i.e., satisfying the Armijo-Goldstein condition;
- The displacement $\mathbf{x}^{(l+1)} - \mathbf{x}^{(l)}$, directly controlled by α_l^{hard}, is small enough to satisfy continuous constraints in Equation (6) from $\mathbf{x}^{(l)}$ to $\mathbf{x}^{(l+1)}$, according to Section 4.1.

Given the above description, we claim that every $\mathbf{x}^{(l)}$ is acceptable: It satisfies discrete constraints and it forms a piecewise linear and intersection-free trajectory from $\mathbf{x}^{(0)}$ to $\mathbf{x}^{(l)}$, although the continuous constraints here are different from the continuous constraints for linear motion. Therefore, we can safely terminate the hard phase at any $\mathbf{x}^{(l)}$. This is a crucial advantage compared with other optimization-based impact zone methods [Harmon et al. 2008; Tang et al. 2018b], which require the optimization to reach the convergence. Since their problems are often ill-conditioned, their methods can stagnate, especially if the time step is large.

The number of hard phase iterations imposes a tradeoff between the efficiency and the accuracy. The fewer the iterations are, the faster but the less accurate the hard phase becomes. Meanwhile, the hard phase typically needs more iterations than the soft phase to reach similar levels of accuracy, given the fact that the hard phase is initialized by $\mathbf{x}^{(0)} = \mathbf{x}^{[k]}$. Instead of using a fixed number of hard phase iterations, we choose to run I^{hard} iterations and then check the termination condition: $\|\nabla F^{\text{hard}}(\mathbf{x}^{(l)})\| \leq \eta$, in which η is the termination threshold. If this condition is not met, we run another I^{hard} iterations, check the condition again, and repeat, until either the condition is met or the total number of iterations reaches a maximal value.

One practical issue is that the optimization can be slowed down by an arbitrarily small step size if $\mathbf{x}^{[k]}$ is too close to the boundary of the feasible region: $c_{ij}(\mathbf{x}^{[k]}) = 0$. This is mostly due to the soft phase accepting any $\mathbf{x}^{[k]}$ that satisfies $c_{ij}(\mathbf{x}^{[k]}) \geq 0$. To solve this issue, we slightly adjust the success condition of the soft phase to: $c_{ij}(\mathbf{x}^{[k]}) \geq \epsilon^{\text{cond}}$. In our system, $\epsilon^{\text{cond}} = 0.01\text{mm}^2$.

Our experiment shows that a hard phase iteration alone is approximately 40% more expensive than a soft phase iteration. But due to step size adjustment and additional iterations, the hard phase can become significantly more expensive as the initial displacement $\mathbf{x}^{\text{init}} - \mathbf{x}^{[k]}$ grows. Ideally, we prefer the initial displacement to be small enough, so the soft phase is successful and the hard phase is not even needed, as discussed in Section 4.2.1. We will study how to achieve a tight upper limit on the initial displacement with efficiency in Section 5.

4.3 GPU-based Implementation

Implementing the collision handling process is mostly straightforward. To begin with, we collect all of the vertex pairs $\{i, j\}$ violating $c_{ij}(\mathbf{x}^{\text{init}}) \geq \epsilon^{\text{slack}}$ into an array. In every soft phase iteration, we launch a soft phase kernel to calculate their gradients in parallel and update $\mathbf{x}^{[k+1]}$ by atomic operations accordingly. After I^{soft} kernel launches, we test continuous and discrete constraints, label out the vertices involved in any constraint violation, and collect the constraints affected by those labeled vertices into a new array. We then process these remaining constraints by hard phase kernels, which calculate constraint gradients and update $\mathbf{x}^{[k+1]}$ in parallel by atomic operations again. At the end of the hard phase, we recheck continuous and discrete constraints. If they are satisfied, we end the collision handling process. Otherwise, we modify the step size and redo the hard phase.

4.3.1 Proximity Search. One computational challenge is how to efficiently find the nearby vertices of every vertex within a radius $\sqrt{(B + B^{\text{tight}})^2 + \epsilon^{\text{slack}}}$, the distance at which vertex distance constraints start to act. Similar to other GPU-based simulators [Pabst et al. 2010; Tang et al. 2018b], our simulator uses grid-based spatial partitioning to accelerate this proximity search. Let the grid cell size be equal to $\sqrt{(B + B^{\text{tight}})^2 + \epsilon^{\text{slack}}}$. For every vertex, we find its nearby vertices from those being stored in its surrounding 27 grid cells. In practice, we check only the 14 grid cells with lower or equal grid cell keys, since the proximity relationship between two vertices is mutual. When the simulator runs many collision handling steps, it can be computationally expensive to build the grid from scratch every time. The good news is the vertex displacement in a single step is made to be small, which means most of the vertices stay in the same grid cells from step to step. We explore such temporal coherence by maintaining two data structures: a dynamic grid that stores all of the vertices and a neighborhood cache defined at every vertex that stores the vertices in its surrounding cells. We use the neighborhood cache to reduce uncoalesced memory access specifically.

4.3.2 Missing Vertex Pairs. We choose to collect vertex pairs violating $c_{ij}(\mathbf{x}^{\text{init}}) \geq \epsilon^{\text{slack}}$ only once at the beginning of the process, because we want to reduce the proximity search cost. But doing this may ignore the pairs that start to violate $c_{ij}(\mathbf{x}^{\text{init}}) \geq \epsilon^{\text{slack}}$ later on. Fortunately, given a sufficiently large ϵ^{slack}, those missing pairs are unlikely to undermine the safety of the collision process, since they can still satisfy continuous and discrete constraints in the end. A safer yet more expensive solution is to expand the proximity search by collecting vertex pairs that violate $c_{ij}(\mathbf{x}^{\text{init}}) \geq \epsilon^{\text{SLACK}} > \epsilon^{\text{slack}}$, in which ϵ^{SLACK} is a greater constant. We have not found this to be necessary in our experiment so far.

4.3.3 The Choice of Parameters. Given the range of the reference edge lengths, we first determine the maximum edge length constant L and the vertex distance bound B. The greater L is, the more stretchability the mesh has before the collision handling process kicks in. However, an increase of L causes an increase

of B, which worsens early repulsion artifacts. As discussed later in Section 6.1, we resample the reference mesh with a desired edge length constant $L^{\text{ref}} = 4.2$mm. In our experiment, we specify $L = 6\sqrt{2}$mm and $B = 6$mm, which provide a reasonable balance between stretchability and repulsion artifacts.

Our next job is to determine the constraint tightening constant B^{tight}. According to Equation (9) in Section 4.1, this is the parameter that controls the vertex displacement limit for satisfying continuous constraints. Since we do not enforce vertex displacement constraints explicitly as described in Section 4.2, we want this limit to be large enough, so that vertex displacement constraints can be automatically satisfied most of the time. In our experiment, we set $B^{\text{tight}} = 0.5$ mm. When $B = 6$ mm, the vertex displacement limit becomes $\sqrt{6.5^2 - 6^2} = 2.5$ mm. This is significantly greater than the initial vertex displacement limit we will specify later in Section 5.

Finally, we would like to decide the slack constant ϵ^{slack}. This parameter should be sufficiently large to facilitate the convergence of the two phases and to address the missing pair issue, as discussed in Section 4.3.2. But it should also be small to reduce the proximity search cost and to lessen early repulsion artifacts. In our experiment, we use $\epsilon^{\text{slack}} = 21.75\ \text{mm}^2$, which makes the proximity radius equal to $\sqrt{(B + B^{\text{tight}})^2 + \epsilon^{\text{slack}}} = 8.0$ mm.

4.3.4 A Rollback Approach for Reduction Operations. The last issue is the reduction operations needed for detecting any constraint or optimization failure, which triggers the hard phase, step size adjustment, and the use of more hard phase iterations, as described in Section 4.2. Performing too many reduction operations can be prohibitively expensive on a GPU, as shown in Wang and Yang [2016]. To address this problem, we run R collision steps with the hard phase being disabled and test if any violation or failure occurs only in the end. If so, we roll back the last R steps and re-run the simulation in the safe mode with both phases being enabled and reductions being performed whenever they are needed. Based on the assumption that constraint violations are rare, this approach decreases the reduction cost by nearly a factor of $(R \cdot I^{\text{soft}})$.

5 DYNAMICS-CONSTRAINT INTEGRATION

Let $\mathbf{x}^t$ and $\mathbf{v}^t$ be the vertex state and velocity vectors at time t and Δt be the time step. We define the goal of cloth simulation as finding the next vertex state vector $\mathbf{x}^{t+\Delta t}$, which solves the following constrained optimization problem:

$$\begin{aligned} \mathbf{x}^{t+\Delta t} &= \arg\min_{\mathbf{x}} E(\mathbf{x}, \mathbf{x}^t + \Delta t \mathbf{v}^t), \\ \text{s.t. } C_{ij}\left(\mathbf{x}^t, \mathbf{x}\right) &\geq 0 \text{ and } c_{ij}(\mathbf{x}) \geq 0, \text{ for any } \{i, j\}. \end{aligned} \tag{19}$$

Here, $E(\mathbf{x}, \mathbf{x}^t + \Delta t \mathbf{v}^t)$ stands for the cloth dynamics objective, which can be minimized by various GPU-based solvers [Fratarcangeli et al. 2016; Wang and Yang 2016], and $C_{ij}(\mathbf{x}^t, \mathbf{x})$ stands for continuous constraints defined on either a line segment from $\mathbf{x}^t$ to $\mathbf{x}$ or a piecewise linear curve passing through intermediate updates.

A crucial problem is: *How can we integrate the proposed collision handling process together with any cloth dynamics solver*? As discussed in Section 4.2, our collision handling process needs a small initial vertex displacement to function with efficiency. This is not an issue when the time step is small, since the vertex displacement should also be small and we can simply handle collisions as post-projection at the end of the time step. But when the time step is large, if we still want to post-process collisions, we would have to break the displacement into multiple sub-steps and handle collisions in each sub-step as Figure 4(a) shows. Such a practice is problematic, since the dynamics result can be very different from the final result $\mathbf{x}^{t+\Delta t}$ and it can lead to artifacts, as shown in Section 5.4. A better idea is to combine a cloth dynamics step and a collision handling step into a joint iterative step as Figure 4(b) shows, in which the displacement within a single joint step should be much smaller than the displacement within a whole time step. Unfortunately, this idea still suffers from the existence of large displacements, which can cause soft phase failures and performance drops especially in the first few steps, as Figure 5(a) shows.

Our key idea is to treat cloth dynamics and collision handling as two independent yet coupled processes as Figure 4(c) shows. Specifically, we split the vertex state vector into two: $\mathbf{x}$ for collision handling and $\mathbf{y}$ for cloth dynamics. We then formulate the original cloth simulation problem in Equation (19) into:

$$\begin{aligned} \left\{\mathbf{x}^{t+\Delta t}, \mathbf{y}^{t+\Delta t}\right\} &= \arg\min_{\{\mathbf{x},\mathbf{y}\}} \left\{ E(\mathbf{y}, \mathbf{y}^t + \Delta t \mathbf{v}^t) + \tfrac{\sigma}{2} \|\mathbf{x} - \mathbf{y}\|_{\mathbf{M}}^2 \right\}, \\ \text{s.t. } C_{ij}(\mathbf{x}^t, \mathbf{x}) &\geq 0 \text{ and } c_{ij}(\mathbf{x}) \geq 0, \text{ for any } \{i, j\}, \end{aligned} \tag{20}$$

in which $\mathbf{M}$ is the mass matrix, $\|\mathbf{x} - \mathbf{y}\|_{\mathbf{M}}^2 = (\mathbf{x} - \mathbf{y})^{\mathsf{T}}\mathbf{M}(\mathbf{x} - \mathbf{y})$, and σ is the positive coupling strength coefficient. Ideally, we want a large σ to achieve strong coupling between $\mathbf{x}$ and $\mathbf{y}$, but a too large σ can overly stiffen the system. In our experiment, we set $\sigma = 80{,}000\text{s}^{-2}$ as a constant. Because only the intersection-free state $\mathbf{x}$ is useful for display, we eliminate the difference from accumulation by setting $\mathbf{y}^{t+\Delta t} = \mathbf{x}^{t+\Delta t}$ at the end of every time step. Next we divide Equation (20) into two sub-problems:

$$\begin{cases} \mathbf{y}^{[k+1]} = \arg\min_{\mathbf{y}} \left\{ E(\mathbf{y}, \mathbf{y}^t + \Delta t \mathbf{v}^t) + \frac{\sigma}{2} \left\| \mathbf{x}^{[k]} - \mathbf{y} \right\|_{\mathbf{M}}^2 \right\}, \\ \mathbf{x}^{[k+1]} = \arg\min_{\mathbf{x}} \frac{\sigma}{2} \left\| \mathbf{x} - \mathbf{y}^{[k+1]} \right\|_{\mathbf{M}}^2, \\ \qquad \text{s.t. } C_{ij}(\mathbf{x}^t, \mathbf{x}) \geq 0 \text{ and } c_{ij}(\mathbf{x}) \geq 0, \text{ for any } \{i, j\}, \end{cases} \tag{21}$$

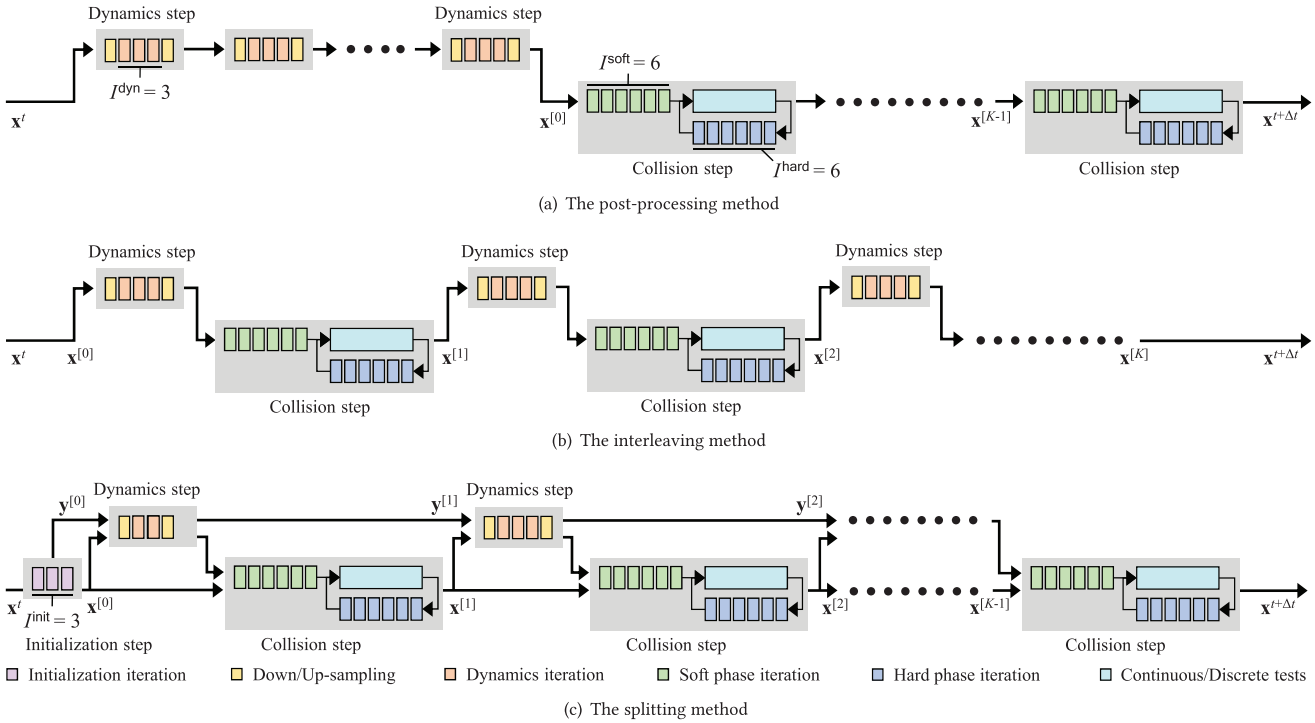
and solve them iteratively K times. Since we cannot afford solving the sub-problems exactly every time, we propose to solve them in an inexact fashion: In the dynamics step, we solve the first sub-problem by I^{dyn} cloth dynamics iterations; and then in the collision step, we solve the second sub-problem by the proposed collision handling process with a displacement limit. Specifically, if there is no constraint, we can solve the second sub-problem in the collision step exactly by a single Newton iteration:

$$\begin{aligned} \mathbf{x}^{[k+1]} &= \mathbf{x}^{[k]} + (\sigma\mathbf{M})^{-1}\sigma\mathbf{M}\left(\mathbf{y}^{[k+1]} - \mathbf{x}^{[k]}\right) \\ &= \mathbf{x}^{[k]} + \left(\mathbf{y}^{[k+1]} - \mathbf{x}^{[k]}\right). \end{aligned} \tag{22}$$

But, since constraints do exist in a collision step, we initialize vertex i for the collision handling process using an initial displacement limit D:

$$\mathbf{x}_i^{\text{init}} = \mathbf{x}_i^{[k]} + \min\left(\frac{D}{\left\|\mathbf{y}_i^{[k+1]} - \mathbf{x}_i^{[k]}\right\|}, 1\right)\left(\mathbf{y}_i^{[k+1]} - \mathbf{x}_i^{[k]}\right). \tag{23}$$

In our experiment, we set $D = 0.25$ mm, one-tenth of the actual vertex displacement limit as discussed in Section 4.3.3. This is sufficient to make more than 99.9% of the soft phases successful.



(a) The post-processing method

(b) The interleaving method

(c) The splitting method

Fig. 4. The pipelines of three different methods that integrate the collision handling process with an iterative cloth dynamics solver. Different from the post-processing method in (a) and the interleaving method in (b), the splitting method in (c) uses two state vectors, $\mathbf{y}$ and $\mathbf{x}$, to deal with cloth dynamics and collision constraints in a separate yet coupled fashion. The splitting method allows the simulator to avoid artifacts and performance drops caused by large displacements, commonly noticed in the use of other methods.

Intuitively, we can view the displacement limit as asynchronous stepping of every vertex. Given collisions being avoided between $\mathbf{x}^{[k]}$ and $\mathbf{x}^{[k+1]}$ by the collision step, the number of steps, K, affects only the simulation accuracy, not the safety. We note that we can also view the whole approach as the coordinate descent method with constraints, which suffers from the convergence issue on the boundary of the feasible region if both descent directions cause constraint violations. Fortunately, this does not happen in our simulator, since the constraints exist in the second sub-problem only.

Algorithm 1 provides the pseudo code of our simulator. For simplicity, it omits the details in collision handling, such as step size adjustment. Figure 5(a) compares the computational costs of our simulator per iterative step, with the interleaving method[3] and the splitting method. It confirms that the splitting method effectively avoids high computational costs in the first few steps, caused by large vertex displacements. Meanwhile, Figure 5(b) shows that the splitting method does have a negative impact on the convergence rate with respect to the number of steps, due to the quadratic penalty term. Overall, the simulator still benefits from the use of the splitting method as shown in Figure 5(c), especially when the time budget is tight.

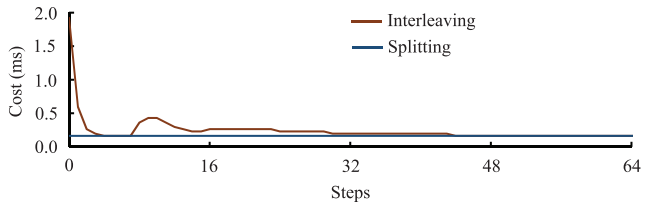
[3]To implement the interleaving method, we simply modify Algorithm 1 by setting $\mathbf{x}^{\text{init}} \leftarrow \mathbf{y}^{[k+1]}$ and $\mathbf{y}^{[k+1]} \leftarrow \mathbf{x}^{[k+1]}$ before and after every collision step. This essentially turns $\mathbf{y}$ into a duplicate variable of $\mathbf{x}$.

ALGORITHM 1: A GPU-based cloth simulator

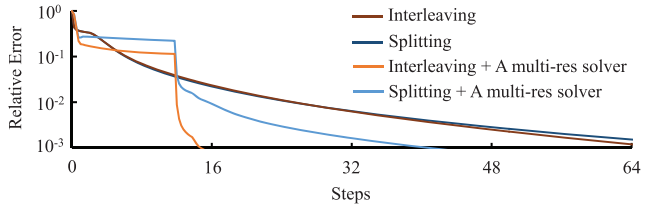
Input: $\mathbf{x}^t$, $\mathbf{y}^t$, $\mathbf{v}^t$, σ and Δt
Output: $\mathbf{x}^{t+\Delta t}$
$\mathbf{y}^{[0]} \leftarrow \mathbf{y}^t + \Delta t \mathbf{v}^t$;
$\mathbf{x}^{[0]} \leftarrow \text{Initialization}\left(\mathbf{x}^t, \mathbf{y}^{[0]}\right)$;
for $k = 0 \ldots K-1$ **do**
  $\mathbf{y}^{[k+1]} \leftarrow \text{Dynamics\_Solver}\left(\mathbf{x}^{[k]}, \mathbf{y}^{[k]}, \mathbf{y}^t + \Delta t \mathbf{v}^t, \sigma, k\right)$;
  for $i = 0 \ldots N-1$ **do**
    $\mathbf{x}_i^{\text{init}} \leftarrow \mathbf{x}_i^{[k]} + \min\left(\frac{D}{\left\|\mathbf{y}_i^{[k+1]} - \mathbf{x}_i^{[k]}\right\|}, 1\right)\left(\mathbf{y}_i^{[k+1]} - \mathbf{x}_i^{[k]}\right)$;
  end
  $\mathbf{x}^{[k+1]} \leftarrow \text{Soft\_Phase}(\mathbf{x}^{\text{init}})$;
  if $C_{ij}\left(\mathbf{x}^{[k]}, \mathbf{x}^{[k+1]}\right) < 0$ or $c_{ij}\left(\mathbf{x}^{[k+1]}\right) < \epsilon^{\text{cond}}$ **then**
    $\mathbf{x}^{[k+1]} \leftarrow \text{Hard\_Phase}(\mathbf{x}^{\text{init}}, \mathbf{x}^{[k]})$;
  end
end
$\mathbf{y}^{t+\Delta t} \leftarrow \mathbf{x}^{t+\Delta t} \leftarrow \mathbf{x}^{[K]}$;

5.1 Cloth Dynamics Solver

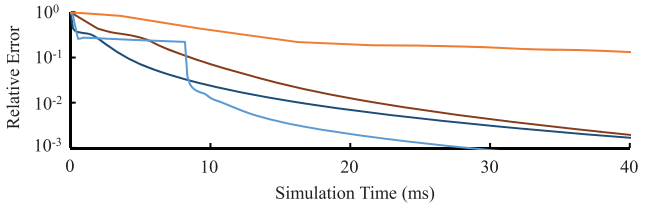
Thanks to the splitting method, our simulator can adopt a variety of GPU-based cloth dynamics solvers, including both descent ones [Fratarcangeli et al. 2016; Wang and Yang 2016; Wang et al. 2018] and non-descent ones. In this work, we choose to use the gradient descent solver with Jacobi preconditioning and Chebyshev



(a) The simulation cost per iterative step



(b) The convergence rates of different methods



(c) The convergence speeds of different methods

Fig. 5. The convergence of our simulator when it uses different methods in the dress example. The interleaving method causes the simulator to spend large computational time on collision handling in the first few steps as (a) shows, due to large vertex displacements. The splitting method avoids this issue and improves the convergence speed, as shown in (c). Here we define the relative error as: $(E(\mathbf{x}^{[k]}) - E(\mathbf{x}^*))/(E(\mathbf{x}^{[0]}) - E(\mathbf{x}^*))$, in which $E(\mathbf{x})$ is the cloth dynamics objective, and $\mathbf{x}^*$ stands for the exact solution, estimated by our simulator with the interleaving method after 1,024 iterative steps. We define one iterative step as the combination of one dynamics step and one collision step.

acceleration [Wang and Yang 2016] for its simplicity and efficiency. Currently, the solver supports two planar elasticity models: the mass-spring model and the triangular finite element model, and two bending elasticity models: the dihedral model [Bergou et al. 2006] and the meshless co-rotational model (in Appendix B). For experimental purposes, we also allow the simulator to solve cloth dynamics at different mesh resolutions in a hierarchy. Specifically as shown in Figure 4(c), a cloth dynamics step contains multiple cloth dynamics iterations solved at a given resolution level, preceded by down-sampling and followed by up-sampling operations. Doing this allows us to always handle collision steps at the finest level.

The optimal multi-resolutional strategy we found so far is to run a sufficient number of dynamics steps at the coarsest level first, then move on to a finer level, until eventually we solve dynamics steps at the finest level. Using this strategy, we can effectively apply Chebyshev acceleration across multiple dynamics steps solved at the same level. Figure 5 indicates that our multi-resolutional strategy magnifies both the negative impact on the convergence rate and the positive impact on the convergence speed. Interestingly, it causes the interleaving method to run even slower, as shown in Figure 5(c), due to very large displacements and excessive collision costs associated with coarse dynamics steps.

5.2 Initialization by Spatial Smoothing

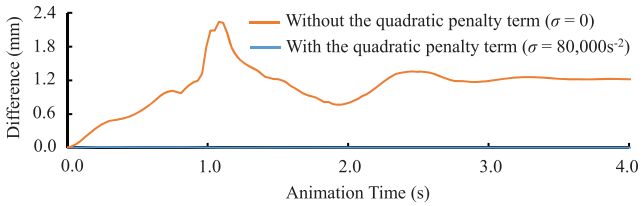
An interesting question is: *How should we initialize* $\mathbf{x}^{[0]}$ *for collision steps*? Unlike $\mathbf{y}^{[0]}$, which can be initialized as an instant prediction of $\mathbf{y}^{t+\Delta t}$, $\mathbf{x}^{[0]}$ cannot be initialized arbitrarily. Instead, it must be treated as the very first update of $\mathbf{x}$ and it must satisfy both continuous and discrete constraints like other updates.

A naive idea is to simply ignore this initialization and set $\mathbf{x}^{[0]} \leftarrow \mathbf{x}^t$. Doing this would create a tight upper bound on the speed a vertex can travel at, due to the initial displacement limit D in every collision step. When $D = 0.25$mm, $\Delta t = 1/100$ s, and $K = 72$, this upper bound is 1.8 m/s, about the same speed cloth reaches after 0.18 s in free fall. We can loose this upper bound by increasing the number of steps, K, but that also increases the computational cost. In simulation, this issue is often demonstrated as artificial damping artifacts, shown in Figure 12(b).

Mathematically, given $\mathbf{y}^{[0]}$ as the initialization of $\mathbf{y}$, we want a cheap way of calculating $\mathbf{x}^{[0]}$ as the initialization of $\mathbf{x}$, such that $\mathbf{x}^{[0]}$ is close to $\mathbf{y}^{[0]}$ and it satisfies continuous and discrete constraints from $\mathbf{x}^t$ to $\mathbf{x}^{[0]}$. To achieve this goal, we propose to apply a spatially smoothed version of the displacement $\mathbf{y}^{[0]} - \mathbf{x}^t$ on $\mathbf{x}^t$ as this initial update. We perform spatial smoothing by first converting the displacement $\mathbf{y}^{[0]} - \mathbf{x}^t$ from vertices to a coarse regular grid, and then we convert it back to vertices by trilinear interpolation. Once the displacement becomes spatially smooth, the relative displacement between two nearby vertices should be small and it should cause few constraint violations. But, since we cannot guarantee zero violation, we divide the smoothed displacement equally into a small number of pieces and apply them sequentially through I^{init} initialization iterations, as shown in Figure 4(c), each of which contains a standard collision step. In our experiment, $I^{\text{init}} = 6$. Figure 12 shows our initialization approach effectively reduces artificial damping. We note that spatial smoothness is important in keeping the initialization cost, especially the costs of initial collision steps, low. Without it, the initialization approach would spend large yet unnecessary costs to fix many constraint violations from $\mathbf{x}^t$ to $\mathbf{y}^{[0]}$.

5.3 No Lagrange Multipliers

A major difference between our method and other splitting methods, such as the alternating direction method of multipliers (ADMM), is that our method uses a relatively large coupling coefficient σ without using Lagrange multipliers. If it does use Lagrange multipliers, the simulator becomes unstable due to our inexact iterative steps. Trading the system stiffness for cheaper steps is a reasonable decision, because $\mathbf{x}^{[0]}$ and $\mathbf{y}^{[0]}$ are often far from their solutions and solving the steps exactly would be a waste of time, especially when the time step is large. Furthermore, we can address the system stiffness issue by running more cloth dynamics iterations per dynamics step. As shown in Section 8, dynamics steps are less expensive than collision steps and the impact of additional dynamics iterations on the overall runtime performance is limited.



(a) The mean vertex difference between $\mathbf{x}$ and $\mathbf{y}$ fluctuating over animation time

(b) Without the quadratic penalty term (c) With the quadratic penalty term

Fig. 6. A dress example. This example compares the simulation results without and with the quadratic penalty term. Without this term, the cloth dynamics solver is ignorant of rigid body collisions and causes $\mathbf{y}$ to be severely dragged down by user interaction. Eventually, the simulator overly stretches $\mathbf{x}$ as well and pulls it off the shoulders, as shown in (b).

The current design of our method is based on the assumption that we do not have enough computational time to get $\mathbf{x}$ and $\mathbf{y}$ strongly coupled at the end of every time step. If our goal is to solve the problem accurately with strongly coupled $\mathbf{x}$ and $\mathbf{y}$, we believe that exact iterative steps with Lagrange multipliers would be a better choice once $\mathbf{x}$ and $\mathbf{y}$ become close to their solutions. Before that happens, we can still apply our method as fast initialization.

5.4 Comparison to Collision Post-processing

A common practice performed by many existing physics-based cloth simulators is to post-process collisions at the end of every time step. Interestingly, since the collision step does not actually use σ in Equation (22) or (23), we can simply set $\sigma = 0$ to remove the quadratic penalty term in our simulator. In that case, the dynamics step becomes ignorant of collisions and our collision handling process is equivalent to a post-process. When the time step is large, this can cause a large difference between the cloth dynamics result $\mathbf{y}^{t+\Delta t}$ and the collision result $\mathbf{x}^{t+\Delta t}$. In Figure 6, this difference gets manifested as overly stretching artifacts when an external force tries to pull a dress down. In comparison, when $\sigma = 80,000\text{s}^{-2}$, the result is free of such artifacts, and it is consistent with the ground truth simulated by using a small time step.

(a) The phantom mesh method (b) Our method

Fig. 7. A comparison between the phantom mesh method and our method. To reduce the difference between $\mathbf{x}$ and $\mathbf{y}$, the phantom mesh method gradually corrects its Lagrange multiplier estimation over time. During this process, the estimation error can cause noticeable artifacts, as shown in (a), especially when the time step is large. Our method is free of this issue.

5.5 Comparison to Phantom Mesh

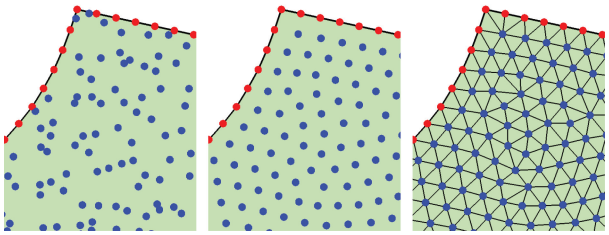
Similar to our method, the phantom mesh method [Harmon et al. 2011] also uses two vertex state vectors to protect cloth dynamics solvers from being affected by asynchronous collision stepping. But unlike our method that uses quadratic penalty within every time step and eliminates the difference in the end, the phantom mesh method estimates Lagrange multipliers once per time step and uses them to reduce the difference over time. While this method is effective when the time step is small, i.e., $\Delta t \leq 1/1000$ s, its original implementation suffers from severe oscillations when the time step is large, i.e., $\Delta t \geq 1/100$ s, a crucial condition desired by state-of-the-art cloth dynamics solvers for their performances. The oscillation issue can be lessened by reducing the update of Lagrange multipliers as discussed in Appendix C, but the result still contains noticeable artifacts. The reason is because the phantom mesh method needs sufficient time to fix Lagrange multipliers, during which the error can incorrectly modify the cloth shape. For example, when the dress gets released from a wrinkled configuration, temporarily incorrect Lagrange multipliers cause artifacts, as shown in Figure 7(a).

6 UNIFORM MESH SAMPLING

Our vertex distance constraints rely on the maximal edge length constant L. If the mesh contains too-long edges, L would be too large and the constraints would push vertices far away from each other, leaving large gaps in-between. However, if the mesh contains too-short edges, the constraints would apply excessive repulsions on vertices near each other, which can cause oscillations even if we simulate the mesh in the reference configuration. Therefore, for fast and realistic simulation, we must uniformly sample the mesh in both reference and deformed configurations.

6.1 Reference Mesh Sampling

During the precomputation stage, we would like to improve the uniformness of the reference mesh by making the ratio of the smallest edge length to the greatest edge length as close to one as possible. There are plenty of remeshing techniques [Alliez et al. 2002, 2008; Bossen and Heckbert 1998] available for achieving this goal. In this work, we choose to apply particle repulsion on



(a) After resampling (b) After repulsion (c) After triangulation

Fig. 8. The results of our reference mesh sampling process. This process consists of sampling, repulsion, triangulation, and optimization steps. In this example, the uniformness, i.e., the ratio of the smallest edge length to the greatest edge length, is improved to 0.656.

randomly sampled particles [Turk 1992], triangulate, and finally optimize the edge length ratio as shown in Figure 8. To begin with, we resample the mesh boundary by a desired mean edge length constant L^{ref}. We then add more particle samples inside the mesh by stratified sampling and apply particle repulsion to separate them evenly. In the end, we triangulate the samples and run an optimization to lengthen the shortest edge and shorten the longest edge:

$$\mathbf{x_I} = \arg\min_{\mathbf{x}} \left(\max_{\{i,j\}\in\mathcal{E}} \|\mathbf{x}_i - \mathbf{x}_j\| - \min_{\{i,j\}\in\mathcal{E}} \left\|\mathbf{x}_i - \mathbf{x}_j\right\| \right), \tag{24}$$

in which $\mathbf{x_I}$ is the positional vector of interior vertices and $\mathcal{E}$ is the edge set. We use the gradient descent method to solve Equation (24) with a fixed small step size, 0.01 mm in our experiment. This method improves the edge length ratio in Figure 8(c) even further, from 0.631 to 0.656. We note that, since the process does not modify boundary vertices, their fixed positions will affect the ideal edge length ratio. For example, if there exists a right boundary triangle, the edge length ratio cannot exceed $1/\sqrt{2}$.

6.2 Adaptive Mesh Resampling

When the mesh receives a large load, the edge length constraints in Section 4 will be severely violated. As a result, the collision handling process must spend more computational time to deal with those constraints in the hard phase.

To improve the success rate of the soft phase and lessen the computational burden, we propose to dynamically and proactively resample overly stretched edges and triangles, so fewer edge length constraints can become severely violated. Specifically, we predefine a set of sample candidates and then we activate the needed ones at the beginning of each collision step. Figure 9 shows two triangle subdivision levels used for generating those sample candidates. When the length of an edge is above an adaptive resampling threshold L^{ada}, we activate the mid-edge sample (in orange) and divide the edge into two sub-edges. When

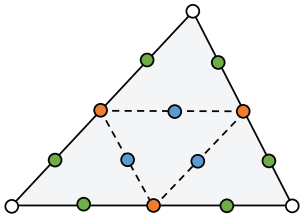
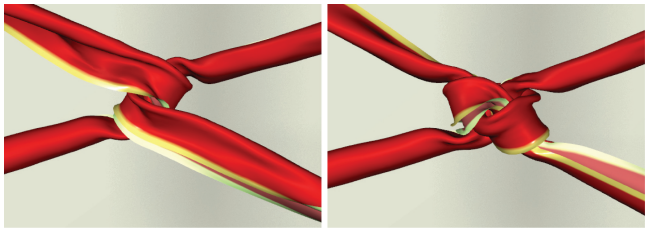


Fig. 9. Three types of sample candidates generated by two triangle subdivision levels.



(a) Without adaptive mesh resampling (b) With adaptive mesh resampling

Fig. 10. A tie example. In this example, cloth is severely stretched and squeezed. Without adaptive mesh resampling, our simulator fails to remove all of the self collisions in the soft phase, as shown in (a). Here, we intentionally disable the hard phase.

the length of a sub-edge is above L^{ada}, we further activate its own mid-edge sample (in green). Finally, if a sub-edge connecting two mid-edge samples gets stretched beyond L^{ada} as well, we activate its sample (in blue) inside of the triangle. Once we activate a sample, we keep it active until the end of the time step.

6.2.1 New Constraints. Activated samples introduce new vertex distance constraints into the collision handling process. Since we determine the sample position by barycentric interpolation of the three triangle vertex positions, we formulate and enforce new vertex distance constraints upon triangle vertex positions. For example, we define the continuous distance constraint between vertex i and an activated sample as:

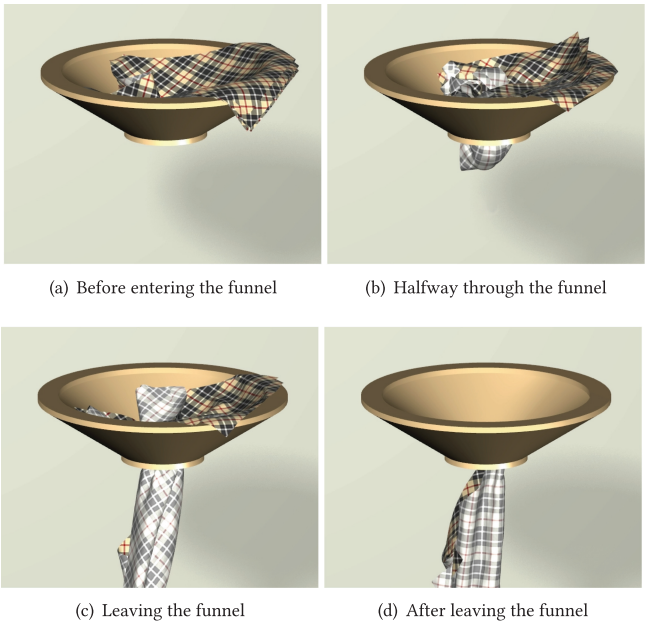
$$\left\|\mathbf{x}_i(t) - b_j\mathbf{x}_j(t) - b_k\mathbf{x}_k(t) - b_l\mathbf{x}_l(t)\right\|^2 \geq B^2, \tag{25}$$

in which $\mathbf{x}_j$, $\mathbf{x}_k$, and $\mathbf{x}_l$ are the vertices of the triangle embedding the sample, and b_j, b_k, and b_l are their barycentric coordinates.

Activated samples also introduce new sub-edge length constraints that can serve two purposes: to replace original constraints in the optimization objectives of Equations (13) and (15); and to replace original constraints in the success conditions of the two phases. Since the change of the optimization objectives modifies the elastic behavior of cloth as shown in Section 8.4.3 and costs the simulator more iterations to converge, we choose to replace original constraints in the success conditions only. In other words, we still enforce original edge length constraints by the same optimization objectives, while we accept the results as long as they satisfy looser sub-edge length constraints.

6.2.2 Discussions. Figure 10 reveals the effect of our adaptive mesh resampling approach, when we intentionally disable the hard phase. Without resampling, the simulator runs into self collision issues as Figure 10(a) shows; and with resampling, the simulator removes all of the self collisions by the soft phase as Figure 10(b) shows.

The key strength of our adaptive mesh resampling approach is its high compatibility with a GPU. Instead of performing adaptive remeshing [Koh et al. 2015; Narain et al. 2012], our approach uses predefined mesh samples and it does not modify mesh topology. Furthermore, since every edge or triangle generates three samples at most, the approach can balance thread workload well by storing and processing samples on per-edge or per-triangle basis. This



(a) Before entering the funnel (b) Halfway through the funnel

(c) Leaving the funnel (d) After leaving the funnel

Fig. 11. A funnel example. In this example, four square cloth sheets experience severe collisions when they are dragged through a funnel. Our simulator robustly simulates this example at 29 to 72 FPS.

advantage disappears once we consider more samples generated by higher-level subdivisions, which is unnecessary so far.

7 VELOCITY, DAMPING, AND FRICTIONAL CONTACTS

Similar to other variational simulators, our simulator calculates the velocity at the end of every time step as: $\mathbf{v}^{t+\Delta t} = \frac{1}{\Delta t}(\mathbf{x}^{t+\Delta t} - \mathbf{x}^t)$. After that, it reduces the velocity magnitude by a decay factor to account for air damping and by Laplacian smoothing to account for internal damping.

Although we can use our vertex repulsion approach to handle cloth-body collisions as well, we choose to use the level set method instead, thanks to its simplicity and efficiency. Specifically, at the beginning of every time step, we compute the signed distance function representing the body surface implicitly. We then treat cloth-body collisions as additional constraints and incorporate them as quadratic penalties (similar to the spring model) into soft and hard phase iterations. We choose not to use log barrier penalties [Wang 2018], since we want to reduce the nonlinearity of the whole collision handling process.

To simulate frictional contacts with the body, we create a spring connecting a vertex with its initial contact location after the first contact. We incorporate the spring potentials as additional terms into the objectives of soft and hard phase iterations. Meanwhile, we sum up the collision penalty force and the frictional spring force for every vertex, which is considered to be the total contact force. If the total force is inside of the Coulomb's friction cone at the end of the time step, we treat the friction to be static and maintain the spring to the next time step. Otherwise, for simulating dynamic friction, we destroy the old spring and create a new spring between the vertex and the new initial contact location in the next time step if it contacts the body again. We simulate self friction effects in a similar fashion, by sticking two vertices together after they collide and destroy their spring connection if the total contact force falls outside of the friction cone at the end of the time step. We use the average of the two vertex normals to define the cone.

The above frictional contact approach works fine if the Coulomb's frictional coefficient is relatively small, such as the funnel example shown in Figure 11. But when the coefficient becomes large, sticking artifacts can appear due to the inaccuracy of our contact forces. We plan to address this issue further by exploring recent frictional contact techniques [Brown et al. 2018; Li et al. 2018, 2020].

Table 1. The Statistics and the Runtime Performances of Our Examples

Name (#Verts., #Tri., Ref.)	Edge (mm)	FPS	
		2080 Ti	1080
Shirt (56K, 112K, Figure 1)	3.4 to 6.2	39 to 61	23 to 37
Dress (55K, 109K, Figure 6)	3.8 to 6.0	38 to 72	23 to 44
Tie (49K, 96K, Figure 10)	3.9 to 6.0	19 to 24	11 to 16
Funnel (58K, 113K, Figure 11)	3.8 to 5.7	29 to 72	17 to 42
Table (58K, 113K, Figure 14)	3.8 to 5.7	31 to 71	18 to 40
Gown (139K, 276K, Figure 12)	2.9 to 5.6	28 to 50	17 to 31
Layered dress (297K, 588K, Figure 15)	2.4 to 7.9	10 to 14	5 to 9

The second column provides the minimum and the maximum edge lengths of the reference mesh in each example.

Table 2. Parameters and Their Values Used by Our Simulator

Name	Definition	Location	Value
L	Maximal edge length	Eq. (5)	$6\sqrt{2}$mm
B^{tight}	Constraint tightening constant	Eq. (8)	0.5mm
ϵ^{slack}	Constraint slack constant	Eq. (13)	21.75mm^2
ϵ^{cond}	Success condition constant	Sec. 4.2	0.01mm^2
I^{soft}	# of soft phase iterations	Sec. 4.2	6 to 16
I^{hard}	# of hard phase iterations	Sec. 4.2	8
R	# of rollback steps for reduction	Sec. 4.3	12
K	# of iterative steps	Sec. 5	72
σ	Coupling coefficient	Eq. (20)	80,000s^{-2}
I^{dyn}	# of cloth dynamics iterations	Sec. 5	1 to 6
I^{init}	# of initialization iterations	Sec. 5.2	6
D	The initial displacement limit	Eq. (23)	0.25mm
L^{ref}	Reference sampling constant	Sec. 6.2	4.2mm
L^{ada}	Adaptive resampling constant	Sec. 6.2	6.5mm

8 RESULTS

The GPU implementation of our simulator uses CUDA 10.1. Our experiment runs on an Intel Core i7-6700 3.4 GHz CPU and an NVIDIA GeForce GTX 2080 Ti GPU, or an NVIDIA GeForce GTX 1080 GPU. Table 1 provides the statistics and the runtime performances of our examples. Table 2 summarizes the parameters and their values used by our simulator. We note that the performance can fluctuate dramatically over time due to the collision handling process being affected by the scene complexity. For all of the examples, we use three hierarchical mesh levels. The numbers of iterative steps spent by the solver at the three levels are: 48, 12, and 12,

(a) The initial configuration (b) $I^{\text{init}} = 0$ and $I^{\text{dyn}} = 6$ (c) $I^{\text{init}} = 6$ and $I^{\text{dyn}} = 1$ (d) $I^{\text{init}} = 6$ and $I^{\text{dyn}} = 6$ (e) $I^{\text{init}} = 6$ and $I^{\text{dyn}} = 12$ (f) Δt=1/1000s

Fig. 12. A gown example. This example compares the results being simulated from the same initial configuration in (a) for 0.3 s, using different parameter values. The use of the initialization approach and a sufficient number of coarse-level cloth dynamics iterations per step effectively reduces artificial damping artifacts, as shown from (b) to (e). We note that the simulation result always contains artificial damping caused by implicit Euler time integration.

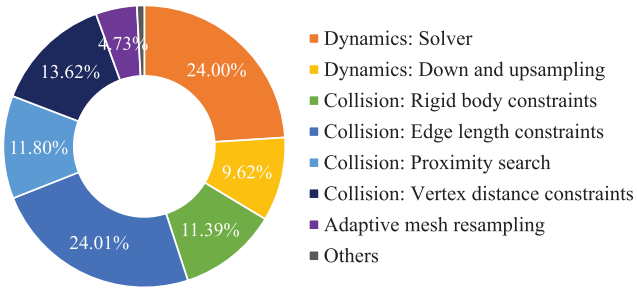
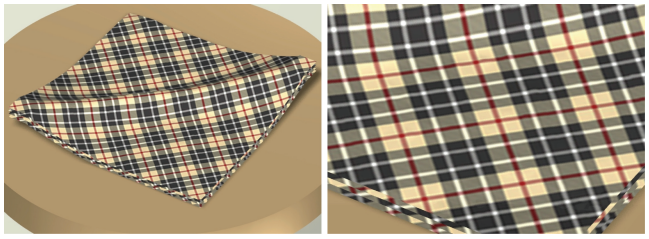


Fig. 13. A typical breakdown of the total computational cost. The collision handling process is notably more expensive than the cloth dynamics solver.



(a) An equilibrium configuration (b) A closeup view

Fig. 14. A table example. Similar to the square example, this example uses four identical 0.576 m×0.576 m cloth sheets. We hold two corners of each sheet and drape them upon each other as (a) shows.

respectively. The total number of iterative steps in one time step is 72.

8.1 Breakdown Analysis

Figure 13 provides a breakdown of the computational cost in the dress example. It shows that the collision handling process is more expensive than the cloth dynamics solver. It also shows that all of the major components in collision handling, including rigid body constraints, edge length constraints,[4] proximity search and vertex distance constraints, consume considerable costs. Thanks to the rollback approach as discussed in Section 4.3.4, constraint tests and objective evaluations are inexpensive, and we choose not to list their costs in Figure 13. Among all of the collision handling components, the use of edge length constraints contributes the most to the total cost, and their functions could be further optimized.

8.2 Speed-accuracy Tradeoffs

Given the collision safety being protected by the collision handling process, our simulator offers several options for trading the simulation speed with the simulation accuracy. The first option is to adjust the number of cloth dynamics iterations per dynamics step. As shown by the gown example in Figure 12, when the number I^{dyn} at the coarsest level grows from 1 to 12, the frame rate drops from 53 FPS to 43 FPS, while artificial damping artifacts decrease noticeably. From our experiment, we find that setting $I^{\text{dyn}} = 6$ at the coarsest level while $I^{\text{dyn}} = 1$ at other levels provides a good balance between the speed and the accuracy. The second option is to adjust the total number of steps, K. When K grows, both the total number of dynamics iterations and the allowed maximal vertex displacement within one time step increase. But because the computational cost is linearly proportional to K, we prefer to keep it as low as possible. Finally, the simulator suffers from artificial damping inherently caused by implicit Euler time integration. This issue can be addressed by using higher-order time integration or smaller time steps, as shown in Figure 12(f). We would like to note that artificial damping may not be entirely bad, as cloth motion would become too dynamic without any damping.

8.3 Artifact Evaluation

To evaluate the significance of the visual artifacts produced by our simulator, we design a table example in which four square cloth sheets stack on a round table as Figure 14 shows. According to the collision handling process in Section 4.2, the gap between two adjacent sheets should be under $\sqrt{(B + B^{\text{tight}})^2 + \epsilon^{\text{slack}}} = 8.0\,\text{mm}$. This gap is not so noticeable from a distance, but quite obvious in a closeup view, as Figure 14(b) shows. In the future, as the hardware becomes faster, we can reduce the gap by using higher-resolution meshes and more iterations in constraint enforcement.

Another potential artifact is the surface bumpiness caused by applying distance constraints among discrete vertex samples only.

[4] We define the cost of the constraints as the cost of the kernel function calculating their gradients. One issue is that we cannot separate the cost of edge length constraints from the cost of vertex distance constraints, since they are handled in parallel by the same function, as discussed in Section 4.3. Therefore, we estimate their costs according to their numbers and their gradient costs per thread.

Interestingly, Figure 14 shows that this artifact is hardly visible even in a closeup view. Assuming that cloth sheets are made of regular triangles with all of the edge lengths equal to $L^{\text{ref}} = 4.2$ mm, we know the distance from an upper sheet vertex to the lower sheet must be within $[\sqrt{8^2 - 4.2^2/3}\,\text{mm}, 8.0\,\text{mm}] \approx [7.6\,\text{mm}, 8.0\,\text{mm}]$. In other words, the bump is under 0.4 mm, too small to be noticeable as expected.

Finally, vertex distance constraints can cause artificial bending resistance when cloth is severely folded. In the current implementation of our system, we simply ignore a distance constraint if the two vertices are in each other's two-ring neighborhood. This solution also ignores self collisions within a two-ring neighborhood, which are fortunately uncommon and can be avoided by applying bending constraints if needed.

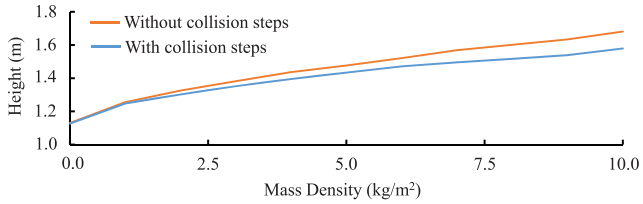
8.4 Stress Tests

In this subsection, we evaluate the performance and the robustness of our simulator through a series of stress tests.

8.4.1 Scalability. Our simulator demonstrates its nearly linear scalability with respect to the number of mesh elements in Table 1, when the collision complexity and the stiffness are roughly the same. As the collision complexity grows, the simulator must run more soft and/or hard phase iterations and the collision handling cost increases, as shown in the tie example. Meanwhile, to deal with high stiffness, the simulator must run more cloth dynamics iterations and the cloth dynamics cost increases. We note that the stiffness depends on not only the cloth material property, but also the mesh resolution. Since cloth dynamics contributes a relatively small portion of the total computational cost, we can afford spending more cloth dynamics iterations, if necessary.

8.4.2 Time Steps. A unique strength of our collision handling approach is its ability of handling very large time steps, such as Δt = 1/30 s. This is highly welcomed by state-of-the-art cloth dynamics solvers for achieving their high performances. In contrast, most previous collision handling approaches, including I-Cloth [Tang et al. 2018b], fail to resolve collisions with safety or efficiency once the time step becomes large. For larger time steps, our simulator must run more dynamics and collision steps per time step to maintain the simulation accuracy at a similar level. But in general, it is still more efficient to use larger time steps, thanks to the benefits gained by cloth dynamics solvers. This conclusion is consistent with the observations provided by many previous works [Bouaziz et al. 2014; Fratarcangeli et al. 2016; Wang and Yang 2016; Wang et al. 2018]. The issue of using larger time steps will worsen artificial damping. Given no better solution to this issue other than adopting higher-order time integration with more computational cost, we prefer to avoid very large time steps in practice.

8.4.3 Stretchability. To evaluate the influence of edge length constraints on cloth stretchability, we design a draping experiment using the layered dress example as shown in Figure 15, in which the mass density grows from 0.001 kg/m^2 to 10.0 kg/m^2. In comparison, the default mass density in other experiments is 0.276 kg/m^2. As the mass density grows, the stretching difference between the result without collision steps and that with collision steps increases.



(a) The height of a layered dress affected by the change of the mass density

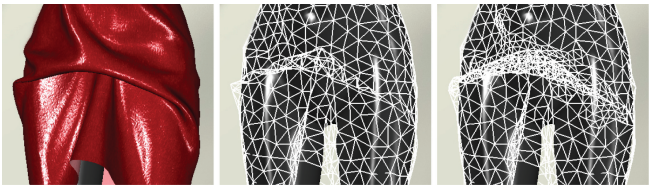
(b) Without collision steps (c) With collision steps

Fig. 15. A layered dress example. In this example, we test how a layered dress responds to the mass density change, in terms of its height as shown in (a) and its outer appearance as shown in (b) and (c).

According to Section 4, the collision handling process places an upper limit on edge lengths: $L = 6\sqrt{2}$ mm. This is about twice of the desired reference edge length: $L^{\text{ref}} = 4.2$ mm. But thanks to the slack constant ϵ^{slack}, the influence of constraint enforcement occurs before that. By definition, edge length constraints start to act when edge lengths reach $\sqrt{L^2 - \epsilon^{\text{slack}}} \approx 7.1$ mm. In other words, many edges can stretch up to 69% without being affected by constraint enforcement. Given the fact that most real-world cloth can hardly stretch beyond 20% , we believe the overall impact of edge length constraints on cloth stretchability is limited.

8.5 Adaptive Remeshing

In the next experiment, we evaluate the compatibility of our simulator with the adaptive remeshing concept. Given the original mesh hierarchy used by our simulator, we build an adaptively refined mesh by simply merging meshes at multiple hierarchical levels, as shown in Figure 16. We can simulate this mesh in full resolution directly for addressing the bending locking issues as done in Narain et al. [2012]. Alternatively, we also explore an idea of using an adaptively refined mesh hierarchy to accelerate the simulation of the original full-resolution mesh. In this new hierarchy, every mesh is an adaptively refined one, made of the original meshes at two original hierarchical levels. Unfortunately,



(a) The dress (b) The raw mesh (c) The refined mesh

Fig. 16. An adaptively remeshed dress example. Our simulator can simulate an adaptively refined mesh as shown in (c), which is updated in real time by merging the original meshes at two hierarchical levels.

we have not observed any benefit of this idea on the convergence speed yet. We plan to investigate this idea even further.

8.6 Performance Comparison to I-Cloth

To compare the performances of our simulator and I-Cloth [Tang et al. 2018b], we run the dress example at the same time step $\Delta t = 1/100$s by the same NVIDIA GeForce GTX 1080 GPU, i.e., the graphics hardware on which I-Cloth was originally tested. In this example, since I-Cloth fails to handle both cloth self collisions and cloth-body collisions at $\Delta t = 1/100$s, we fix two shoulder points and drape the dress under gravity without the body, as shown in Figure 17. The experiment shows that the average frame rate of our simulator in the first 200 frames is 50.16 FPS, while that of I-Cloth is 4.77 FPS. Given the fact that both our simulator and I-Cloth spend 45% to 60% of their computational costs on collision handling, our collision handling process is also approximately 10× faster than that of I-Cloth. As mentioned in Section 8.4.2, one strength of our simulator is its ability of handling very large time steps. If artificial damping is tolerable, we can improve the ratio of the simulation clock time to the animation scene time from 0.5016 to 0.9436 by increasing the time step from $\Delta t = 1/100$s to $\Delta t = 1/30$s. In comparison, I-Cloth fails to run this example at $\Delta t = 1/50$s. Our experiment also reveals that the performance of I-Cloth is much more sensitive to the moving speed of cloth: The performance decreases as the dress drops faster; and it increases as the dress gradually reaches quasistatic equilibrium. Finally, I-Cloth fails to run complex examples, such as the tie example, even at $\Delta t = 1/1000$s.

Fig. 17. A draping dress example without the human body.

8.7 Limitations

Based on the vertex repulsion concept, our collision handling approach requires vertices to be sufficiently away from each other. This means the simulator cannot handle close stitching of two vertices without using other techniques, such as embedded mesh. Our collision handling approach also cannot simulate overly stretching effects well, due to the use of edge length constraints as shown in Section 8.4.3. To achieve strong coupling of the two state vectors, our simulator needs a sufficiently large coupling coefficient σ in the quadratic penalty term. As a result, the simulator must spend more cloth dynamics iterations to overcome artificial damping artifacts caused by a stiff system, as discussed in Section 8.2. Currently, our simulator cannot handle frictional contacts in a proper fashion. Its runtime performance fluctuates over time due to the collision complexity of the scene. To improve the performance, we optimize our CUDA implementation mostly on the GTX architecture and we just started our research on the RTX architecture.

9 CONCLUSIONS AND FUTURE WORK

In this article, we demonstrate that we can eliminate the possibility of cloth self collisions by satisfying vertex distance and edge length constraints continuous in time or their discrete counterparts together with vertex displacement constraints. More importantly, we achieve strict and efficient enforcement of these constraints and their integration with cloth dynamics solvers in a GPU-based cloth simulator. The experiment reveals that we can now safely and robustly solve cloth dynamics and collision handling for millimeter-level cloth meshes in real time.

Our immediate plan next is to address the frictional issue, as it limits the ability of our simulator in producing realistic contact effects. We then would like to study the ways of decreasing distance thresholds to reduce early repulsion artifacts and to lessen the mesh quality requirement. Since the proximity search can be highly sensitive to the scene complexity, we are interested in further optimization of its GPU-based implementation. Finally, we will continue exploring performance improvement from an algorithmic perspective for real-time cloth simulation at even higher resolutions.

APPENDICES

A SUPPLEMENTAL THEOREMS

Theorem A.1. *Let $\mathbf{x}_i^*$ be the point within triangle $\mathbf{x}_j\mathbf{x}_k\mathbf{x}_l$ that maximizes $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$. If the triangle is acute, $\mathbf{x}_i^*$ must be its interior circumcenter.*

Proof. It is straightforward to see that for any $\mathbf{x}_i$ on the triangle boundary, $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ cannot exceed half of the greatest edge length, which is less than the circumradius. Since the triangle is acute, its circumcenter must be in the interior and $\mathbf{x}_i^*$ must be in the interior as well. Now if $\mathbf{x}_i^*$ is in the interior but not the circumcenter, then there are two possibilities: $\|\mathbf{x}_i^* - \mathbf{x}_j\| < \min(\|\mathbf{x}_i^* - \mathbf{x}_k\|, \|\mathbf{x}_i^* - \mathbf{x}_l\|)$, or $\|\mathbf{x}_i^* - \mathbf{x}_j\| = \|\mathbf{x}_i^* - \mathbf{x}_k\| < \|\mathbf{x}_i^* - \mathbf{x}_l\|$. In both cases, we can move $\mathbf{x}_i^*$ away to enlarge $\|\mathbf{x}_i^* - \mathbf{x}_j\|$ (and $||\mathbf{x}_i^* - \mathbf{x}_k||$), which increases $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ even further. This is against the assumption that $\mathbf{x}_i^*$ has already maximized $D(\mathbf{x}_i^*, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$. Therefore, $\mathbf{x}_i^*$ must be the circumcenter. □

Theorem A.2. *Let $\mathbf{x}_i^*$ be a point within triangle $\mathbf{x}_j\mathbf{x}_k\mathbf{x}_l$ that maximizes $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$. If the triangle is not acute, then $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l) \leq \frac{1}{2}\max(\|\mathbf{x}_j - \mathbf{x}_k\|, \|\mathbf{x}_k - \mathbf{x}_l\|, \|\mathbf{x}_l - \mathbf{x}_j\|)$.*

Proof. The proof of Theorem 1 suggests that if $\mathbf{x}_i^*$ is in the interior, then it must be the circumcenter. But, since the triangle is non-acute and it does not have an interior circumcenter, $\mathbf{x}_i^*$ must

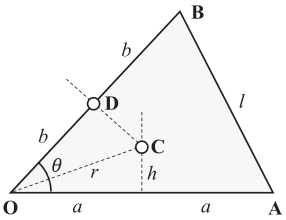


Fig. 18. An acute triangle with its interior circumcenter **C**. Theorem A.3 proves that its circumradius r is a monotonic function of the edge length l.

be on the triangle boundary. It is then straightforward to see that $D(\mathbf{x}_i, \mathbf{x}_j\mathbf{x}_k\mathbf{x}_l)$ cannot exceed half of the greatest edge length and the argument is true. □

Theorem A.3. *The circumradius of an acute triangle is a monotonic function of any triangle edge length.*

Proof. Without loss of generality, let $2a$ and $2b$ be two triangle edge lengths as Figure 18 shows. We first show that the distance h from circumcenter **C** to edge **OA** is a monotonic function of the angle θ between the two edges. Let **D** be the center of edge **OB**. Line **DC** is perpendicular to **OB** and we have:

$$\frac{a - b\cos\theta}{h - b\sin\theta} = -\frac{\sin\theta}{\cos\theta}, \tag{26}$$

which gives $h = (b - a\cos\theta)/\sin\theta$. When the triangle is acute, we have $\sin\theta > 0$ and $a > b\cos\theta$, so:

$$\frac{\mathrm{d}h}{\mathrm{d}\theta} = \frac{a - b\cos\theta + b\sin\theta}{\sin^2\theta} > 0. \tag{27}$$

Since $r = \sqrt{h^2 + a^2}$ is a monotonic function of h, it must also be a monotonic function of θ. Meanwhile, $\theta = \cos^{-1}(\frac{4a^2+4b^2-l^2}{8ab})$ is a monotonic function of l, the length of edge **AB**. Together, r is a monotonic function of l. □

Theorem A.4. *Let $\mathbf{x}_i\mathbf{x}_j$ and $\mathbf{x}_k\mathbf{x}_l$ be two intersecting edges with fixed lengths. The distance $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ is maximized when the edges are perpendicular and the intersection happens at their midpoints.*

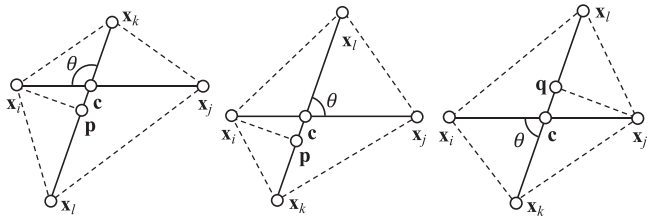
Proof. Without loss of generality, we assume that:

$$\begin{aligned}\max D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l) &= \|\mathbf{x}_i - \mathbf{x}_k\| \\ &\geq B_{ij,kl} = \tfrac{1}{2}\sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|^2 + \|\mathbf{x}_j - \mathbf{x}_k\|^2},\end{aligned} \tag{28}$$

where $B_{ij,kl}$ is the value of $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ when the edges are perpendicular and they meet at the midpoints. To begin with, we prove that $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ cannot be maximized if the edges are not perpendicular. Let $\mathbf{c}$ be the intersection point and θ be the angle between $\mathbf{x}_i\mathbf{c}$ and $\mathbf{x}_k\mathbf{c}$. There are three possibilities if $\theta \neq 90°$.

If $\theta > 90°$, then we must have $\|\mathbf{x}_i - \mathbf{c}\| \leq \|\mathbf{x}_j - \mathbf{c}\|$ and $\|\mathbf{x}_k - \mathbf{p}\| \leq \|\mathbf{x}_l - \mathbf{p}\|$, in which $\mathbf{p}$ is the projection of $\mathbf{x}_i$ on edge $\mathbf{x}_k\mathbf{x}_l$. In that case, $\mathbf{p}$ must be between $\mathbf{c}$ and $\mathbf{x}_l$, as Figure 19(a) shows. By turning $\mathbf{x}_k\mathbf{x}_l$ perpendicular to $\mathbf{x}_i\mathbf{x}_j$ and aligning $\mathbf{p}$ with $\mathbf{c}$, we elongate $\mathbf{x}_i\mathbf{x}_k$ while it is still the shortest of the four. Therefore, $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ cannot be maximized if $\theta > 90°$.

If $\theta < 90°$ and $\|\mathbf{x}_i - \mathbf{c}\| < \|\mathbf{x}_j - \mathbf{c}\|$, then $\mathbf{p}$ must be between $\mathbf{c}$ and $\mathbf{x}_k$, as Figure 19(b) shows. This is because if $\mathbf{x}_k$ is between $\mathbf{c}$ and $\mathbf{p}$ instead, then we would have $\|\mathbf{x}_i - \mathbf{x}_k\| \leq \|\mathbf{x}_i - \mathbf{c}\| < B_{ij,kl}$,



(a) Case 1 (b) Case 2 (c) Case 3

Fig. 19. Three possible cases when $\theta \neq 90°$. Theorem A.4 proves that $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ cannot be maximized in any of the three cases.

which is against Equation (28). Now, since $\|\mathbf{x}_i - \mathbf{x}_k\|$ is the shortest, we have $\|\mathbf{x}_k - \mathbf{p}\| \leq \|\mathbf{x}_l - \mathbf{p}\|$. By turning $\mathbf{x}_k\mathbf{x}_l$ perpendicular to $\mathbf{x}_i\mathbf{x}_j$ and aligning $\mathbf{p}$ with $\mathbf{c}$, we can elongate $\mathbf{x}_i\mathbf{x}_k$ while it is still the shortest. Therefore, $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ cannot be maximized if $\theta < 90°$ and $\|\mathbf{x}_i - \mathbf{c}\| < \|\mathbf{x}_j - \mathbf{c}\|$.

Finally, if $\theta < 90°$ and $\|\mathbf{x}_i - \mathbf{c}\| \geq \|\mathbf{x}_j - \mathbf{c}\|$, then we define $\mathbf{q}$ as the projection of $\mathbf{x}_j$ on edge $\mathbf{x}_k\mathbf{x}_l$ and we know $\mathbf{q}$ must be between $\mathbf{c}$ and $\mathbf{x}_l$, as Figure 19(c) shows. This is because if $\mathbf{x}_l$ is between $\mathbf{c}$ and $\mathbf{q}$ instead, then we would have $\|\mathbf{x}_i - \mathbf{x}_k\| \leq \|\mathbf{x}_j - \mathbf{x}_l\| \leq \|\mathbf{x}_j - \mathbf{c}\| < B_{ij,kl}$, which is against Equation (28). Now, by turning $\mathbf{x}_k\mathbf{x}_l$ perpendicular to $\mathbf{x}_i\mathbf{x}_j$ and aligning $\mathbf{q}$ with $\mathbf{c}$, we will increase $\|\mathbf{x}_i - \mathbf{x}_k\|$, $\|\mathbf{x}_j - \mathbf{x}_k\|$, and $\|\mathbf{x}_j - \mathbf{x}_l\|$, and we will have $\|\mathbf{x}_j - \mathbf{x}_l\| \leq \|\mathbf{x}_i - \mathbf{x}_l\|$. As a result, $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ will increase, so it cannot be maximized when $\theta < 90°$ and $\|\mathbf{x}_i - \mathbf{c}\| \geq \|\mathbf{x}_j - \mathbf{c}\|$.

From the above three cases, we see that $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ is maximized only when $\theta = 90°$. It is then straightforward to see that when $D(\mathbf{x}_i\mathbf{x}_j, \mathbf{x}_k\mathbf{x}_l)$ is maximized, the intersection point must be the midpoints of the two edges. □

B A MESHLESS BENDING MODEL

One issue associated with the dihedral bending model [Bergou et al. 2006] is that it requires a manifold triangle mesh. But in many practical cases, the mesh may not be manifold or even exist. Inspired by the shape matching method [Müller et al. 2005], we propose a simple meshless bending model to provide more flexibility in our cloth simulator.

Let $\mathbf{r}_i \in \mathbb{R}^3$ and $\mathbf{x}_i \in \mathbb{R}^3$ be the 3D positions of vertex i in the reference space and the deformed space. Let the neighborhood of vertex i in the reference space be $\mathcal{N}_i = \{j : \|\mathbf{r}_j - \mathbf{r}_i\| \leq r\}$, in which r is the neighborhood radius. We define the affine transformation $\mathbf{A}_i$ from the neighborhood in the reference space to that in the deformed space as the solution to the following minimization problem:

$$\mathbf{A}_i = \arg\min_{\mathbf{A}_i} \sum_{j\in\mathcal{N}_i} \left\|\mathbf{A}_i(\mathbf{r}_i - \mathbf{r}_j) - (\mathbf{x}_j - \mathbf{x}_i)\right\|^2, \tag{29}$$

whose solution is:

$$\mathbf{A}_i = \left(\sum_{j\in\mathcal{N}_i} (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{r}_i - \mathbf{r}_j)^\mathsf{T}\right)\left(\sum_{j\in\mathcal{N}_i} (\mathbf{r}_i - \mathbf{r}_j)(\mathbf{r}_i - \mathbf{r}_j)^\mathsf{T}\right)^{-1}. \tag{30}$$

We note that if the reference neighborhood is on a plane, the rightmost side of Equation (30) will be singular. In that case, it is more convenient to define $\mathbf{r}_i$ as a 2D vector and $\mathbf{A}_i$ as a 3×2 matrix. Under the assumption that the reference neighborhood is nearly planar and planar deformation is locally uniform, we treat

affine transformation as if it is solely caused by planar deformation. Therefore, we define the meshless bending energy at vertex i as:

$$E_i^{\text{bending}} = \frac{k^{\text{bending}}}{2} \sum_{j \in \mathcal{N}_i} \left\| \mathbf{A}_i^{-1}(\mathbf{x}_i - \mathbf{x}_j) - (\mathbf{r}_j - \mathbf{r}_i) \right\|^2, \tag{31}$$

in which k^{bending} is its stiffness coefficient. This model is not fully accurate, but it provides a plausible approximation to elastic bending deformation, especially when the aforementioned assumption holds.

C PHANTOM MESH IMPLEMENTATION

Without the quadratic penalty term, i.e., $\sigma = 0$, we reformulate the splitting problem into:

$$\begin{cases} \mathbf{y}^{t+\Delta t} = \arg\min_{\mathbf{y}} E(\mathbf{y}, \mathbf{y}^t + \Delta t \mathbf{v}^t), \\ \mathbf{x}^{t+\Delta t} = \arg\min_{\mathbf{x}} \frac{1}{2} \|\mathbf{x} - \mathbf{y}^{t+\Delta t}\|_{\mathbf{M}}^2, \end{cases} \tag{32}$$

subject to $C_{ij}(\mathbf{x}^t, \mathbf{x}) \geq 0$, $c_{ij}(\mathbf{x}) \geq 0$, and $\mathbf{x}^{t+\Delta t} = \mathbf{y}^{t+\Delta t}$. Let $\bar{\mathbf{y}} = \mathbf{y}^t + \Delta t \mathbf{v}^t$ and $\mathbf{M}$ be the mass matrix. We have $E(\mathbf{y}, \bar{\mathbf{y}}) = E_0(\mathbf{y}) + \frac{1}{2\Delta t^2} \|\mathbf{y} - \bar{\mathbf{y}}\|_{\mathbf{M}}^2$, in which $\|\mathbf{y} - \bar{\mathbf{y}}\|_{\mathbf{M}}^2 = (\mathbf{y} - \bar{\mathbf{y}})^{\mathsf{T}} \mathbf{M} (\mathbf{y} - \bar{\mathbf{y}})$ and $E_0(\mathbf{y})$ is the elastic potential [Wang and Yang 2016]. Meanwhile, since we model collision constraints as penalties in both the soft phase and the hard phase, we represent them by a single cost function: $E_1(\mathbf{x})$. With Lagrange multipliers $\boldsymbol{\lambda}$ for $\mathbf{x}^{t+\Delta t} = \mathbf{y}^{t+\Delta t}$, we aim at solving:

$$\begin{cases} \mathbf{y}^{t+\Delta t} = \arg\min \left\{ E_0(\mathbf{y}) + \frac{1}{2\Delta t^2} \|\mathbf{y} - \bar{\mathbf{y}}\|_{\mathbf{M}}^2 + \boldsymbol{\lambda}^{\mathsf{T}} (\mathbf{x}^{t+\Delta t} - \mathbf{y}) \right\}, \\ \mathbf{x}^{t+\Delta t} = \arg\min \left\{ E_1(\mathbf{x}) + \frac{1}{2} \|\mathbf{x} - \mathbf{y}^{t+\Delta t}\|_{\mathbf{M}}^2 + \boldsymbol{\lambda}^{\mathsf{T}} (\mathbf{x} - \mathbf{y}^{t+\Delta t}) \right\}. \end{cases} \tag{33}$$

The solutions to Equation (33) are:

$$\begin{cases} \mathbf{y}^{t+\Delta t} = \bar{\mathbf{y}} + \Delta t^2 \mathbf{M}^{-1} \boldsymbol{\lambda} - \Delta t^2 \mathbf{M}^{-1} \nabla E_0(\mathbf{y}^{t+\Delta t}), \\ \mathbf{x}^{t+\Delta t} = \mathbf{y}^{t+\Delta t} - \mathbf{M}^{-1} \boldsymbol{\lambda} - \mathbf{M}^{-1} \nabla E_1(\mathbf{x}^{t+\Delta t}). \end{cases} \tag{34}$$

Under the assumption that a small adjustment $\Delta\boldsymbol{\lambda}$ to $\boldsymbol{\lambda}$ has little impact on ∇E_0 or ∇E_1 [Harmon et al. 2011], we get:

$$\begin{cases} \mathbf{y}^{\text{new}} \approx \mathbf{y}^{t+\Delta t} + \Delta t^2 \mathbf{M}^{-1} \Delta\boldsymbol{\lambda}, \\ \mathbf{x}^{\text{new}} \approx \mathbf{y}^{\text{new}} - \mathbf{M}^{-1} \boldsymbol{\lambda} - \mathbf{M}^{-1} \Delta\boldsymbol{\lambda} - \mathbf{M}^{-1} \nabla E_1(\mathbf{x}^{t+\Delta t}) \\ \qquad = \mathbf{x}^{t+\Delta t} + \Delta t^2 \mathbf{M}^{-1} \Delta\boldsymbol{\lambda} - \mathbf{M}^{-1} \Delta\boldsymbol{\lambda}. \end{cases} \tag{35}$$

To achieve $\mathbf{x}^{\text{new}} = \mathbf{y}^{\text{new}}$, we then set $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$ at the end of the time step, for $\Delta\boldsymbol{\lambda} = \mathbf{M}(\mathbf{x}^{t+\Delta t} - \mathbf{y}^{t+\Delta t})$.

With the quadratic penalty term, we can also introduce Lagrange multipliers $\boldsymbol{\lambda}$ into the splitting problem:

$$\begin{cases} \mathbf{y}^{t+\Delta t} = \arg\min_{\mathbf{y}} \left\{ E(\mathbf{y}, \bar{\mathbf{y}}) + \frac{\sigma}{2} \|\mathbf{x}^{t+\Delta t} - \mathbf{y}\|_{\mathbf{M}}^2 + \boldsymbol{\lambda}^{\mathsf{T}} (\mathbf{x}^{t+\Delta t} - \mathbf{y}) \right\}, \\ \mathbf{x}^{t+\Delta t} = \arg\min_{\mathbf{x}} \left\{ E_1(\mathbf{x}) + \frac{\sigma}{2} \|\mathbf{x} - \mathbf{y}^{t+\Delta t}\|_{\mathbf{M}}^2 + \boldsymbol{\lambda}^{\mathsf{T}} (\mathbf{x} - \mathbf{y}^{t+\Delta t}) \right\}. \end{cases} \tag{36}$$

At the end of the time step, we obtain $\mathbf{x}^{t+\Delta t}$ and $\mathbf{y}^{t+\Delta t}$ that satisfy:

$$\begin{cases} \nabla E(\mathbf{y}^{t+\Delta t}) - \sigma \mathbf{M}(\mathbf{x}^{t+\Delta t} - \mathbf{y}^{t+\Delta t}) - \boldsymbol{\lambda} = \mathbf{0}, \\ \nabla E_1(\mathbf{x}^{t+\Delta t}) + \sigma \mathbf{M}(\mathbf{x}^{t+\Delta t} - \mathbf{y}^{t+\Delta t}) + \boldsymbol{\lambda} = \mathbf{0}. \end{cases} \tag{37}$$

Since the ideal solutions $\mathbf{x}^*$, $\mathbf{y}^*$, and $\boldsymbol{\lambda}^*$ satisfy $\nabla E(\mathbf{y}^*) - \boldsymbol{\lambda}^* = \mathbf{0}$ and $\nabla E_1(\mathbf{x}^*) + \boldsymbol{\lambda}^* = \mathbf{0}$, we define the update $\Delta\boldsymbol{\lambda}$ at the end of the time step as: $\Delta\boldsymbol{\lambda} = \sigma \mathbf{M}(\mathbf{x}^{t+\Delta t} - \mathbf{y}^{t+\Delta t})$. We note that this implementation is essentially the augmented Lagrangian method with $\boldsymbol{\lambda}$ being updated at the end of every time step only.

Unfortunately, both implementations suffer from severe oscillations due to the rapid change of $\boldsymbol{\lambda}$ when the time step is large. We can lessen this issue by introducing a step size coefficient s into the update: $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + s\Delta\boldsymbol{\lambda}$, for $s \in (0, 1)$. However, it can still cause temporary shape artifacts as shown in Figure 7(a).

REFERENCES

Burak Aksoylu, Andrei Khodakovsky, and Peter Schröder. 2005. Multilevel solvers for unstructured surface meshes. *SIAM J. Sci. Comput.* 26, 4 (Apr. 2005), 1146–1165.

Pierre Alliez, Mark Meyer, and Mathieu Desbrun. 2002. Interactive geometry remeshing. *ACM Trans. Graph. (SIGGRAPH)* 21, 3 (July 2002), 347–354.

Pierre Alliez, Giuliana Ucelli, Craig Gotsman, and Marco Attene. 2008. Recent advances in remeshing of surfaces. In *Shape Analysis and Structuring*. Springer, 53–82.

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

Miklos Bergou, Max Wardetzky, David Harmon, Denis Zorin, and Eitan Grinspun. 2006. A quadratic bending model for inextensible surfaces. In *Proceedings of the Symposium on Geometry Processing (SGP'06)*. 227–230.

Frank Bossen and Paul Heckbert. 1998. A pliant method for anisotropic mesh generation. In *Proceedings of the 5th International Meshing Roundtable*.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph. (SIGGRAPH)* 33, 4, (July 2014).

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph. (SIGGRAPH)* 21, 3 (July 2002), 594–603.

Robert Bridson, Sebastian Marino, and Ronald Fedkiw. 2003. Simulation of clothing with folds and wrinkles. In *Proceedings of the Symposium on Computer Animation (SCA'03)*. 28–36.

Tyson Brochu, Essex Edwards, and Robert Bridson. 2012. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (July 2012), 96:1–96:7.

George E. Brown, Matthew Overby, Zahra Forootaninia, and Rahul Narain. 2018. Accurate dissipative forces in optimization integrators. *ACM Trans. Graph. (SIGGRAPH Asia)* 37, 6 (2018).

Thomas Buffet, Damien Rohmer, Loïc Barthe, Laurence Boissieux, and Marie-Paule Cani. 2019. Implicit untangling: A robust solution for modeling layered clothing. *ACM Trans. Graph. (SIGGRAPH)* 38, 4 (2019).

Kwang-Jin Choi and Hyeong-Seok Ko. 2002. Stable but responsive cloth. *ACM Trans. Graph. (SIGGRAPH)* 21, 3 (July 2002), 604–611.

Elliot English and Robert Bridson. 2008. Animating developable surfaces using nonconforming elements. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (Aug. 2008), 1–5.

Marco Fratarcangeli, Valentina Tibaldo, and Fabio Pellacini. 2016. Vivace: A practical Gauss-Seidel method for stable soft body dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)* 35, 6 (Nov. 2016)s.

Seth Green, George Turkiyyah, and Duane Storti. 2002. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Proceedings of the International Conference on Smart Media and Applications*. 265–272.

David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. *ACM Trans. Graph. (SIGGRAPH)* 28, 3 (July 2009).

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2008. Robust treatment of simultaneous collisions. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (August 2008).

David Harmon, Qingnan Zhou, and Denis Zorin. 2011. Asynchronous integration with phantom meshes. In *Proceedings of the Symposium on Computer Animation (SCA'11)*. 247–256.

Suejung Huh, Dimitris N. Metaxas, and Norman I. Badler. 2001. Collision resolutions in cloth simulation. In *Proceedings of Computer Animation*. IEEE, 122–127.

Inyong Jeon, Kwang-Jin Choi, Tae-Yong Kim, Bong-Ouk Choi, and Hyeong-Seok Ko. 2013. Constrainable multigrid for cloth. *Comput. Graph. Forum (Pacific Graphics)* 32, 7 (2013), 31–39.

Woojong Koh, Rahul Narain, and James F. O'Brien. 2015. View-dependent adaptive cloth simulation with buckling compensation. *IEEE Trans. Vis. Comput. Graph.* 21, 10 (Oct. 2015), 1138–1145.

Christian Lauterbach, Qi Mo, and Dinesh Manocha. 2010. gProximity: Hierarchical GPU-based operations for collision and distance queries. In *Proceedings of the Eurographics Conference*, Vol. 29. 419–428.

Jie Li, Gilles Daviet, Rahul Narain, Florence Bertails-Descoubes, Matthew Overby, George E. Brown, and Laurence Boissieux. 2018. An implicit frictional contact

solver for adaptive cloth simulation. *ACM Trans. Graph. (SIGGRAPH)* 37, 4 (July 2018).

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4 (July 2020).

Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph. (SIGGRAPH Asia)* 32, 6 (Nov. 2013).

Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Trans. Graph. (SIGGRAPH)* 33, 4 (July 2014).

Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. (SIGGRAPH)* 24, 3 (July 2005), 471–478.

Rahul Narain, Matthew Overby, and George E. Brown. 2016. ADMM ⊇ projective dynamics: Fast simulation of general constitutive models. In *Proceedings of the Symposium on Computer Animation (SCA'16)*. 21–28.

Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 31, 6 (Nov. 2012).

Simon Pabst, Artur Koch, and Wolfgang Straßer. 2010. Fast and scalable CPU/GPU collision detection for rigid and deformable surfaces. *Comput. Graph. Forum* 29, 5 (2010), 1605–1612.

Xavier Provot. 1997. Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation*. Springer, 177–189.

Sara C. Schvartzman, Álvaro G. Pérez, and Miguel A. Otaduy. 2010. Star-contours for efficient hierarchical self-collision detection. *ACM Trans. Graph. (SIGGRAPH)* 29, 4 (July 2010).

Andrew Selle, Jonathan Su, Geoffrey Irving, and Ronald Fedkiw. 2009. Robust high-resolution cloth using parallelism, history-based collisions, and accurate friction. *IEEE Trans. Vis. Comput. Graph.* 15, 2 (Mar. 2009), 339–350.

Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. 2007. Hybrid simulation of deformable solids. In *Proceedings of the Symposium on Computer Animation (SCA'07)*. 81–90.

Jos Stam. 2009. Nucleus: Towards a unified dynamics solver for computer graphics. In *Proceedings of the 11th IEEE International Conference on Computer-aided Design and Computer Graphics*.

Rasmus Tamstorf, Toby Jones, and Stephen F. McCormick. 2015. Smoothed aggregation multigrid for cloth simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 34, 6 (Oct. 2015).

Min Tang, Young J. Kim, and Dinesh Manocha. 2010. Continuous collision detection for non-rigid contact computations using local advancement. In *Proceedings of the IEEE Conference on Robotics & and Automation*. 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 (July 2018), 18 pages.

Min Tang, Dinesh Manocha, Sung-Eui Yoon, Peng Du, Jae-Pil Heo, and Ruo-Feng Tong. 2011. VolCCD: Fast continuous collision culling between deforming volume meshes. *ACM Trans. Graph.* 30, 5 (Oct. 2011).

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 (Nov. 2014).

Min Tang, Huamin Wang, Le Tang, Ruofeng Tong, and Dinesh Manocha. 2016. CAMA: Contact-aware matrix assembly with unified collision handling for GPU-based cloth simulation. *Comput. Graph. Forum (Eurographics)* 35, 2 (May 2016), 511–521.

Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018b. I-Cloth: Incremental collision handling for GPU-based interactive cloth simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 37, 6 (Dec. 2018).

Bernhard Thomaszewski, Simon Pabst, and Wolfgang Straßer. 2008. Asynchronous cloth simulation. In *Proceedings of the Computer Graphics International Conference*.

Greg Turk. 1992. Re-tiling polygonal surfaces. In *Proceedings of the 19th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'92)*. 55–64.

Pascal Volino and Nadia Magnenat-Thalmann. 2006. Resolving surface collisions through intersection contour minimization. *ACM Trans. Graph. (SIGGRAPH)* 25, 3 (July 2006), 1154–1159.

Huamin Wang. 2014. Defending continuous collision detection against errors. *ACM Trans. Graph. (SIGGRAPH)* 33, 4 (July 2014).

Huamin Wang. 2015. A Chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Trans. Graph. (SIGGRAPH Asia)* 34, 6 (Oct. 2015).

Huamin Wang. 2018. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Trans. Graph. (SIGGRAPH)* 37, 4 (July 2018).

Huamin Wang and Yin Yang. 2016. Descent methods for elastic body simulation on the GPU. *ACM Trans. Graph. (SIGGRAPH Asia)* 35, 6 (Nov. 2016).

Zhendong Wang, Longhua Wu, Marco Fratarcangeli, Min Tang, and Huamin Wang. 2018. Parallel multigrid for nonlinear cloth simulation. *Comput. Graph. Forum (Pacific Graphics)* 37, 7 (2018), 131–141.

Martin Wicke, Hermes Lanker, and Markus Gross. 2006. Untangling cloth with boundaries. In *Proceedings of the Conference on Vision, Modeling, and Visualization*. 349–356.

Changxi Zheng and Doug L. James. 2012. Energy-based self-collision culling for arbitrary mesh deformations. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (July 2012).

Received April 2020; revised August 2020; accepted October 2020