# I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation

MIN TANG, Zhejiang University
TONGTONG WANG, Zhejiang University
ZHONGYUAN LIU, Zhejiang University
RUOFENG TONG, Zhejiang University
DINESH MANOCHA, University of Maryland at College Park

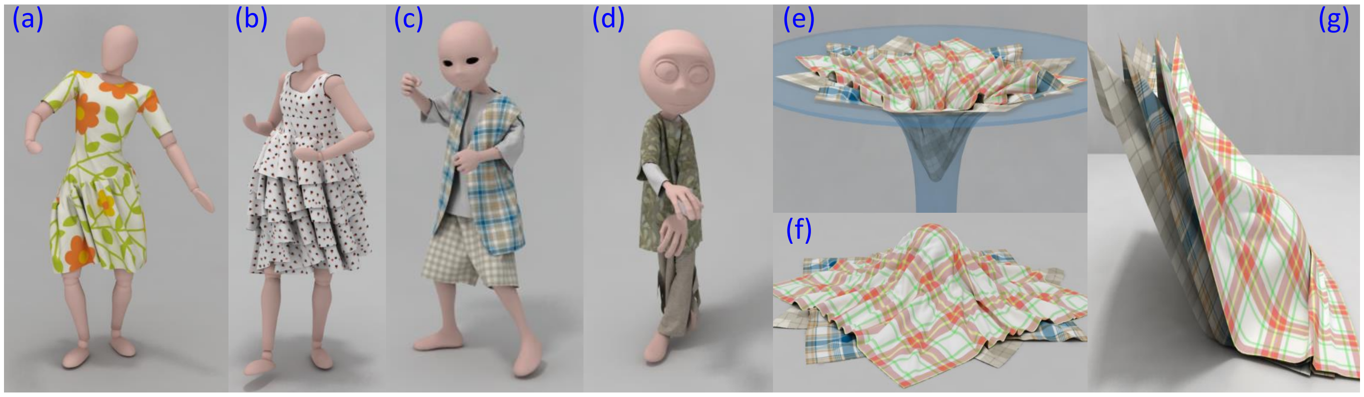


Fig. 1. **Benchmarks:** Our novel GPU-based collision handling algorithm is used to simulate complex cloth with irregular shape and multiple layers at $2-8$ fps on an NVIDIA GeForce GTX 1080. We observe $7-10$X speedup over prior algorithms.

We present an incremental collision handling algorithm for GPU-based interactive cloth simulation. Our approach exploits the spatial and temporal coherence between successive iterations of an optimization-based solver for collision response computation. We present an incremental continuous collision detection algorithm that keeps track of deforming vertices and combine it with spatial hashing. We use a non-linear GPU-based impact zone solver to resolve the penetrations. We combine our collision handling algorithm with implicit integration to use large time steps. Our overall algorithm, I-Cloth, can simulate complex cloth deformation with a few hundred thousand vertices at $2-8$ frames per second on a commodity GPU. We highlight its performance on different benchmarks and observe up to $7-10$X speedup over prior algorithms.

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

Additional Key Words and Phrases: collision handling, impact zone, cloth simulation, GPU

Authors' addresses: Min Tang, Zhejiang University, tang_m@zju.edu.cn; Tongtong Wang, Zhejiang University, wtt923@zju.edu.cn; Zhongyuan Liu, Zhejiang University, lzy_work@foxmail.com; Ruofeng Tong, Zhejiang University, trf@zju.edu.cn; Dinesh Manocha, University of Maryland at College Park, dm@cs.umd.edu.

**ACM Reference Format:**
Min Tang, Tongtong Wang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018. I-Cloth: Incremental Collision Handling for GPU-Based Interactive Cloth Simulation. *ACM Trans. Graph.* 37, 6, Article 204 (November 2018), 10 pages. https://doi.org/10.1145/3272127.3275005

## 1 INTRODUCTION

Cloth simulation has been an active area of research in computer graphics and physics-based modeling for more than three decades. Most of the work has focused on accurate simulation based on efficient techniques for time integration, collision detection, and response computation. Many cloth simulation algorithms have been implemented as part of computer-aided design and animation systems, though they are mainly used for offline or non-interactive applications. At the same time, applications such as computer games, virtual reality and virtual try-on systems need interactive simulation capabilities.

Many techniques have been proposed to accelerate the performance of cloth simulation. These include fast algorithms for collision detection and response [Bridson et al. 2002; Harmon et al. 2008; Otaduy et al. 2009; Tang et al. 2018]. Furthermore, many of these methods can be parallelized on commodity processors [Ni et al. 2015; Tang et al. 2016]. However, state of the art methods can take a few seconds per frame on a GPU for a high resolution cloth mesh. One of the major bottlenecks is collision handling, which can take up to $70-80\%$ of total frame time. This includes the detection of all possible contacts and self-penetrations along with response

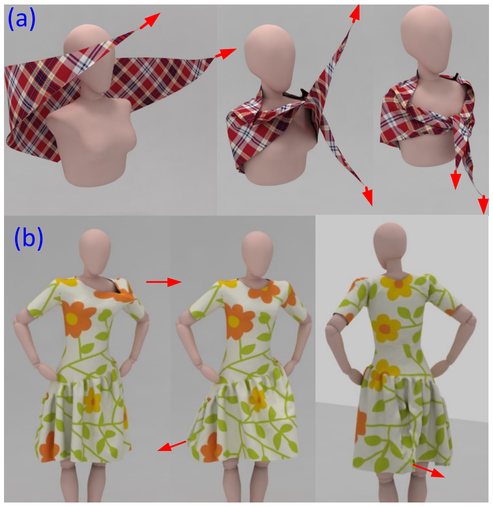


Fig. 2. **Interactive Cloth Manipulation by a User:** (a) Human dressing: We use the interactive capability of I-Cloth for tying a scarf by manipulating the two corners around the neck (left). (b) Cloth manipulation: A user applies forces by pulling the cloth at different locations. I-Cloth is able to compute new configurations of the cloth at $8-10$ fps on an NVIDIA GeForce GTX 1080 (see video).

force computation. Moreover, current methods tend to use small time steps to prevent deep penetrations and that reduces overall performance.

**Main Results:** We present an incremental collision handling method for cloth simulation. Our approach is based on impact zones and exploits the spatial and temporal coherence between successive iterations of the optimization algorithm. The major novel components of our approach are:

- *Incremental Continuous Collision Detection using Spatial Hashing:* Our approach exploits the fact that only a small set of mesh vertices undergoes deformation due to response forces. We keep track of deforming vertices and present high-level and low-level GPU culling algorithms for incremental CCD using spatial hashing (Section 3).
- *GPU-based Non-Linear Impact Zone Solver:* We present a novel non-linear optimization algorithm to compute a penetration-free state based on impact zones. Our approach is more robust than prior impact zone algorithms and takes fewer iterations. We also present an efficient scheme to parallelize on GPUs (Section 4).

These two techniques are integrated to perform collision handling. The improved accuracy of our non-linear solver allows us to choose larger time steps and results in faster GPU-based cloth simulator (I-Cloth). We have evaluated the performance on complex cloth meshes with tens or hundreds of thousands of triangles on an NVIDIA GeForce GTX 1080 (Section 5). Our algorithm can simulate all the contacts in highly-deforming cloth at $2-8$ frames per second. We observe up to $10X$ improvement in the performance of the collision handling. Moreover, I-Cloth offers up to $7X$ speedup over prior GPU-based cloth simulators.

## 2 PRIOR WORK AND BACKGROUND

In this section, we give an overview of the prior work in collision handling and cloth simulation.

### 2.1 Cloth Simulation

Many techniques have been proposed to improve the robustness and efficiency of cloth simulation based on implicit Euler integrators [Baraff and Witkin 1998], iterative optimization [Liu et al. 2013; Wang and Yang 2016], etc. Other methods use local and adaptive techniques to generate the dynamic detail of the simulated cloth [Lee et al. 2010; Narain et al. 2012] or use data-driven approaches [de Aguiar et al. 2010; Kim et al. 2013; Wang et al. 2010].

### 2.2 Collision Handling

Collision detection and response are regarded as major bottlenecks in cloth simulation. It is important to accurately detect all the collisions, as a single missed collision may result in an invalid simulation or noticeable artifacts [Bridson et al. 2002]. Most accurate methods for collision checking are based on exact CCD [Brochu et al. 2012; Provot 1997; Tang et al. 2014; Wang 2014]. Some of the commonly used techniques for collision response are based on impulse computation [Bridson et al. 2002; Sifakis et al. 2008], constraint solvers [Otaduy et al. 2009], and impact zone methods [Harmon et al. 2008; Provot 1997]. There is extensive work on fast GPU-based collision detection and proximity computation algorithms [Govindaraju et al. 2005; Sud et al. 2006; Tang et al. 2011]. An energy-based method has been proposed by Zheng and James [2012] for self-collision culling. Recently, elastoplastic friction models have been used [Guo et al. 2018; Jiang et al. 2017] to handle challenging contact scenarios, but they are slower. In this paper, we present improved methods for incremental CCD computation and impact zone computation and combine them with implicit integrators. Asynchronous contact mechanics methods [Ainsley et al. 2012; Vouga et al. 2011] have also been used for cloth simulation and these methods can provide robust handling at the cost of more computation.

### 2.3 Parallel Algorithms

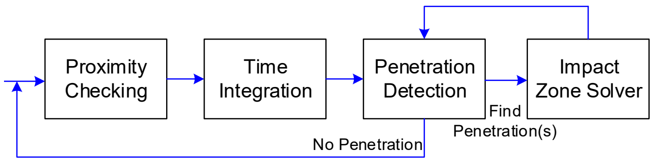
To accelerate the performance, many parallel algorithms have been proposed for faster collision detection on CPUs and GPUs [Pabst et al. 2010; Selle et al. 2009; Tang et al. 2018]. Other approaches exploit multiple CPU and/or GPU cores for faster time integration and cloth simulation [Cirio et al. 2014; Ni et al. 2015; Tang et al. 2013, 2016; Wang et al. 2016]. Our approach is designed to perform the entire simulation on a single GPU and exploits the high number of cores for collision handling.

### 2.4 Simulation Pipeline

**Notation:** We assume that the objects in the scene and the cloth mesh are composed of triangles. We use the symbols V, E and F to represent a vertex, an edge, and a face of a triangle, respectively. At any time instance, the entire cloth mesh $Q$ is represented as a point in a high-dimensional space: for an $n$-vertex mesh with vertices $X_1, X_2, ..., X_n \in R^3$, the *configuration space* is represented as $Q = R^{3n}$. The underlying mesh at time $t$ is represented as $Q_t$.

We use a triangle-mesh-based piece-wise linear elastic model to simulate cloth with non-linear anisotropic deformations [Tang et al. 2016; Wang et al. 2010]. We perform time integration using the backward Euler method [Baraff and Witkin 1998] and incorporate internal forces (stretching forces and bending forces) and external forces (gravity, wind forces, and repulsion/friction) into time integration. We use a GPU-based matrix assembly [Tang et al. 2016] to construct the stiffness matrix during each frame, and use a preconditioned conjugate gradient solver.

Our simulation algorithm consists of time integration, collision detection, and collision response computation. We use well-known implicit time integration methods along with contact forces [Bridson et al. 2002; Otaduy et al. 2009] and combine them with an integrated collision detection and response computation algorithm. We assume that the initial state of the cloth mesh is penetration-free. At each step, the overall simulation pipeline consists of the following stages:



(1) **Proximity Checking:** compute proximity VF/EE constraints;
(2) **Time Integration:** perform implicit time integration with internal/external forces and proximity constraints;
**REPEAT:**
(3) **Penetration Detection:** perform CCD to collect all the penetrations; (Section 3)
(4) **Impact Zone Solver:** run the solver on GPU and resolve all the detected penetrations; (Section 4)
**UNTIL** a new penetration-free state achieved.

These steps are performed at each time step. The impact zone constraint-enforcement is performed in step (4), having been decoupled from time integration, i.e. step (2), and is similar to prior collision response algorithms [Bridson et al. 2002; Harmon et al. 2008].

### 2.5 Proximity Constraints, Repulsive Forces, and Friction

For every VF/EE pair in proximity, we compute a repulsive force based on the proximity distance. The repulsive force is distributed among corresponding nodes [Bridson et al. 2002] and added to the stiffness matrix. The proximity constraints are integrated into implicit time integration [Otaduy et al. 2009; Tang et al. 2016]. We also model static/kinetic friction based on repulsive forces [Bridson et al. 2002], which is based on the relative tangential velocity between a VF/EE pair. We compute the repulsive force and the friction force at the same time, and apply them to the stiffness matrix.

### 2.6 Collision Response

Without loss of generality, we assume that the mesh has no collisions, i.e. in a penetration-free state, at $t = 0$ ($Q_0$). We compute a new state ($Q_1$) at $t = 1$ using implicit time integration. Next, we check whether $Q_1$ is penetration-free. If there are collisions, we compute a penetration-free configuration $Q_1'$ with minimal change in post-response kinetic energy. This computation can be formulated as a constrained optimization problem. The minimization function is defined based on the vertices of $Q_1$ and $Q_1'$:

$$\min \frac{\sum_{i=1}^{i=n} (X_i' - X_i)^2 * m_i}{\sum_{i=1}^{i=n} m_i}, \tag{1}$$

where $m_i$ are the nodes' mass. The constraints corresponding to a non-penetration state of the mesh at a given time $t$ are computed based on the well-known vertex-face (VF) and edge-edge (EE) pairs used for CCD computation [Provot 1997]:

$$C_{vf} = N \circ [X_4 - (\alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3], \tag{2}$$
$$C_{ee} = N \circ [(\alpha_3 X_3 + \alpha_4 X_4) - (\alpha_1 X_1 + \alpha_2 X_2)], . \tag{3}$$

For a VF pair, $N(t)$ represents the triangle normal, $X_1(t), X_2(t), X_3(t)$ are the vertices of the triangle and $X_4(t)$ represents the moving vertex. $\alpha_1$ $\alpha_2$ and $\alpha_3$ are the barycentric coordinates of the projection of X4(t) onto the plane spanned by the triangle. For an EE pair, $N(t)$ is the cross product of the two edges, where $X_1(t), X_2(t)$ are the vertices of one edge and $X_3(t), X_4(t)$ are the vertices of the other edge. $\alpha_1$, $\alpha_2$, $\alpha_3$ and $\alpha_4$ are the parametric values of corresponding closest points on the first and second edge, respectively.

**Impact Zone:** Our collision response algorithm is based on computing a penetration-free state using impact zones. An impact zone (IZ) corresponds to a set of intersecting VF/EE pairs with shared vertices. The underlying response algorithm groups all the intersecting VF/EE pairs of $Q_1$ into impact zones and computes a penetration-free state $Q_1'$ using an optimization algorithm. This optimization problem can be reduced to solving a linear system with inelastic projection [Harmon et al. 2008]. However, we observe that the resulting method can have stability issues and we propose an algorithm based on non-linear gradient descent solver (see Section 4). This process of collision checking and impact zone solving is repeated in an iterative manner until the mesh is penetration-free.

## 3 INCREMENTAL CONTINUOUS COLLISION DETECTION

A key aspect of cloth simulation is the accurate computation of CCD. Given the state of a mesh at $Q_0$ and $Q_1$, the CCD algorithms check for overlapping VF and EE pairs using linear interpolating vertex positions between $t = 0$ and $t = 1$. Current algorithms for CCD perform high-level culling using bounding volume hierarchies (BVHs) or spatial hashing, followed by low-level culling and reliable elementary tests between the VF and EE pairs [Tang et al. 2018, 2014; Wang 2014]. This CCD computation is performed repeatedly on the refined mesh during iterative impact zone based collision handling, till a penetration-free state is computed. In this section, we present an incremental CCD algorithm that exploits the spatial and temporal coherence between two successive iterations to reduce the run-time cost. Between two successive iterations of the impact-zone algorithm, only some vertices of $Q$ undergo deformation with a small motion. As opposed to checking the entire mesh for collisions, we restrict our tests to vertices of $Q$ that undergo some deformation and accelerate the computations using spatial hashing.

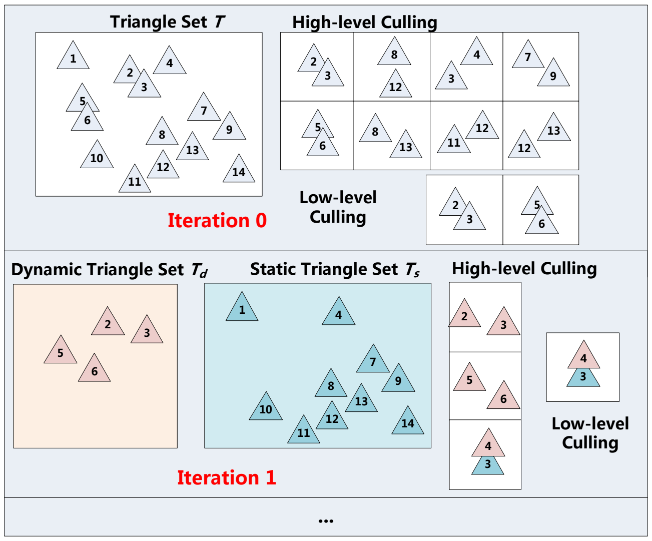


Fig. 3. **Incremental CCD with Spatial Hashing:** We decompose the triangle set $T$ into a Dynamic Triangle Set ($T_d$) and a Static Triangle Set ($T_S$) and store them separately using spatial hashing. Our algorithm can skip self-collision checking on $T_S$ and can be easily parallelized on GPUs.

In order to perform incremental CCD, we classify the set of triangles into various sets. The *Triangle Set* ($T$) corresponds to all the triangles in the scene. It includes the triangles of $Q$ and other objects in the scene. During each iteration, we partition $T$ into two subsets depending on the collision status: a *Dynamic Triangle Set* ($T_d$), which consists of triangles that have at least one vertex involved in colliding VF or EE pairs; and a *Static Triangle Set* ($T_s$), in which none of the vertices of its triangles is involved in the colliding pairs. We assume that the number of triangles in $T$ ($N_{all}$) is fixed throughout the simulation, while the sizes of $T_d$ and $T_s$ change depending on the number of collisions during each iteration. Let $N_d$ and $N_s$ be the number of triangles in $T_d$ and $T_s$, respectively. If a mesh is penetration-free, then $T_d = \emptyset$ and $T_s = T$.

The overall approach is described in Algorithm 1. We initially perform CCD computation on $T$ using high-level and low-level culling and dynamically compute the sets $T_d$ and $T_s$. This is illustrated in Fig. 3, where the resulting pairs of overlapping triangles are $(f_2, f_3)$ and $(f_5, f_6)$. In this case, $T_d = \{f_2, f_3, f_5, f_6\}$, $T_s$ contains all the other triangles. Our algorithm adjusts the vertices of the overlapping triangles in $T_d$ to eliminate the penetrations using impact zones, as described in Section 4. For the next iteration, we need to perform two kinds of collision checks:

- Inter-object collisions between the triangles in $T_d$ and $T_s$.
- Self-collisions between the triangles in $T_d$.

Mathematically, we reduce the worst-case intersection test complexity from $N_{all}^2$ to $N_d^2 + N_d * N_s$ and typically, $N_d << N_s$. We use the following lemma to perform fewer intersection tests.

**Incremental Collision Detection Lemma:** *Given the Triangle Set $T$, we let the set of vertices corresponding to its colliding VF/EE pairs be represented as $V_c$. $T$ can be partitioned into $T_d$ and $T_s$ based on $V_c$: If a triangle $f \in T$ has at least one vertex that belongs to $V_c$, then $f \in T_d$; otherwise, $f \in T_s$. If some vertices in $V_c$ move, all the triangles in $T_s$ can not collide with each other.*

Proof. Since the impact zone solver only adjusts the vertices in $V_c$, all the vertices of the triangles of $T_s$ remain unchanged. In other words, if there is a triangle pair in $T_s$ that collides, it must have a vertex belong to $V_c$, and the resulting triangle(s) would belong to $T_d$. As a result, we prove the lemma by contradiction. □

Our collision handling algorithm only recomputes the new positions of the vertices of $V_c$ with the impact zone solver and does not need to perform self-collisions on $T_s$.

**Algorithm 1** Incremental Collision Detection Algorithm

```
1: detect all the collisions among all the triangles $T$;
2: while collisions are detected from the last step using $T_d$ and $T_s$ do
3:     set flags for all the vertices in $V_c$, which is the set of all vertices in the colliding VF/EE pairs;
4:     based on flagged vertices, compute $T_d$ and $T_s$;
5:     perform CCD between $T_d$ and $T_s$;
6:     perform self-CCD among $T_d$;
7: end while
```

Our incremental CCD algorithm can be accelerated using BVHs or spatial hashing. We use the GPU-based spatial hashing algorithm [Pabst et al. 2010; Tang et al. 2018] for inter-object and intra-object collision culling and filter out redundant collision tests by maintaining the $T_d$ and $T_s$ at each iteration. We first check for all overlapping triangle pairs in $T$ by performing high-level culling with spatial hashing, followed by low-level culling, by checking each pair if its two triangles both belong to $T_s$. If so, this triangle pair will be culled. After performing the exact elementary tests, we update $V_c$ accordingly and set the flags for all the vertices in $V_c$ (Line 3 of Algorithm 1). Furthermore, we classify all the triangles in $T$ as $T_d$ or $T_s$ based on these flags (Line 4 of Algorithm 1). During the next iteration, the high and low-level culling techniques are applied to these updated sets.

## 4 GPU-BASED NON-LINEAR IMPACT ZONE SOLVER

At each iterative stage of the collision handling algorithm, a penetration-free state of $Q$ is computed using impact zones. In this section, we present a non-linear optimization algorithm for impact zones and parallelize it on GPU architectures.

### 4.1 Gradient Descent Solver

The constrained optimization problems defined by Equations(1-3) are transformed into an unconstrained problem by an augmented Lagrangian method [Narain et al. 2012; Nocedal and Wright 2006].

For conciseness, we denote the minimization function in Equation (1) as $g(X)$ and the inequality constraints in Equations (2-3) as $c^*(X)$, i.e., the set of all inequalities ($C_{vf} \geq 0$ and $C_{ee} \geq 0$) for all the penetrating VF/EE pairs. The constrained optimization problems in Section 2.3 can be expressed as:

$$\min g(X) \qquad s.t.\ c(X) \leq 0, \tag{4}$$

where, $c(X) = -c^*(X)$. The general augmented Lagrangian method replaces $c(X)$ with a combination of penalty functions $\hat{c}(X, s)$ and Lagrange multipliers $\lambda$, and changes our constrained optimization problem into:

$$\min \{g(X) + \lambda^T \hat{c}(X, s) + \frac{\mu}{2}||\hat{c}(X, s)||^2\}, \tag{5}$$

where $\hat{c}(X, s) = c(X) + s$, $s$ is a non-negative slack variable, and the update rule of $\lambda$ is given as:

$$\lambda \longleftarrow \lambda + \mu\hat{c}(X).$$

$\mu$ is a parameter and we set $\mu = 10^3$ in our benchmarks. We use the method described in [Narain et al. 2012] to simplify this function into

$$f(X) = g(X) + \frac{\mu}{2}||\tilde{c}(X)||^2 - \frac{||\lambda||^2}{2\mu}, \tag{6}$$

and the update rule is $\lambda \longleftarrow \mu\tilde{c}(X)$, where the new penalty function $\tilde{c}(X) = max(c(X) + \frac{\lambda}{\mu}, 0)$. $X$ and $\lambda$ are updated alternately while solving this unconstrained optimization problem. During each iteration, we first fix $\lambda$, then solve $X$ by minimizing $f(X)$ using an enhanced gradient descent method, and then fix $X$ to update $\lambda$.

The overall algorithm is described as Algorithm 2. For each impact zone vertex, the overall solver consists of three steps: compute the gradient descent direction, find an advancing time step to update the optimization variable, and update the vertex position. We use Jacobi preconditioning and Chebyshev acceleration [Wang and Yang 2016] to improve the convergence rate of gradient descent. The advancing time step is computed by the backtracking line search method [Nocedal and Wright 2006], which gradually reduces the step length, until the Wolfe's condition is satisfied:

$$f(X^{(k)} - s^{(k)} \cdot \vartriangle X^{(k)}) < f(X^{(k)}) - \alpha \cdot s^{(k)} \cdot || \vartriangle X^{(k)}||^2, \tag{7}$$

where $f(X)$ is the objective function and $\alpha$ is a control parameter. $X^{(k)}$, $s^{(k)}$, and $\vartriangle X^{(k)}$ are the variables, the step length, and the gradient descent direction during the $k$th iteration, respectively. We only update the vertices $X_i \in V_c$ and update $T_d$ accordingly. In our implementation, we use termination conditions based on the number of maximum iterations (100) and the prescribed step length threshold ($10^{-12}$).

## 4.2 GPU-based Solver

We exploit GPU parallelism to accelerate the performance of our non-linear solver described in Algorithm 2. We update all the vertices $X_i \in V_c$ in an iterative manner to compute a penetration-free state. We highlight the details of our parallel algorithm in Fig. 4 and Algorithm 3. We use a thread block to handle each impact zone, i.e. each optimization problem. Therefore, all the impact zones can be processed in parallel. For each impact zone, we use an iterative scheme until the VF/EE pairs for that impact zone are non-overlapping. During each iteration, the whole solving process consists of three steps (i.e. steps 1-3 in Fig. 4). We first use the threads in each block to compute the descent direction for each vertex in parallel. Next the thread with index 0 is used to update the step length. We use backtracking line search [Nocedal and Wright 2006] as the step adjustment scheme based on the values of objective functions in the unconstrained optimization problem computed during the previous step. We use atomic operations to ensure the accuracy and shared memory structures to store step lengths and other parameters. If Wolfe's conditions during the backtracking line search scheme [Nocedal and Wright 2006] are satisfied, then each vertex position is updated in the next step using Chebyshev acceleration on a thread. We use a scheme similar to [Wang 2015] to adjust the parameters in the Chebyshev method:

$$X_i = (X_i - X_i') * \omega + X_i'$$

, where $X_i$ is adjusted vertex position, $X_i'$ is its position at previous time step, and $\omega$ is updated by the following equations:

$$\omega = \begin{cases} 1 & \text{if } k < 10, \\ 2/(2-\rho^2) & \text{if } k = 10, \\ 4/(4-\rho^2\omega) & \text{if } k > 10, \end{cases} \tag{8}$$

where $k$ represent the number of iterations used in our impact zone solver and $\rho$ is 0.9992 in our benchmarks. Otherwise, we go back to the first step of the iterative scheme until Wolfe's condition is satisfied. Overall, our approach exploits GPU parallelism for all these computations.

**Algorithm 2** CPU-based Non-Linear Impact Zone Solver

```
1:  for each impact zone do
2:      for each iterative step of gradient descent do
3:          for each vertex in impact zone do
4:              compute gradient descent direction, $g_i$;
5:          end for
6:          compute an appropriate step of gradient descent;
7:          if step length is less than the prescribed threshold or
    the number of iterations reaches the maximum then
8:              break;
9:          end if
10:         evaluate the objective function values for vertex $o_i$;
11:         for each vertex $X_i$ in impact zone do
12:             update $X_i$ using the step length and gradient descent
    direction;
13:             update $X_i$ using Chebyshev acceleration method;
14:         end for
15:         update Lagrange multipliers $\lambda$ for all constraints;
16:     end for
17: end for
```

## 4.3 Stability Analysis

Compared to the impact zone optimization based on a linear solver [Harmon et al. 2008], our non-linear solver demonstrates higher stability and accuracy. For some benchmarks (e.g., Funnel, Twisting and Sphere), the linear solver fails when the cloth tangles. For other benchmarks (e.g., Dress, Andy, and Bishop), the linear solver takes many more iterations than our non-linear solver to reach a penetration-free state. During the frame #1100 of Benchmark Funnel (Fig. 10), our non-linear GPU solver takes 2.86ms and 34 iterations to resolve the penetrations. On the other hand, the linear solver [Harmon et al. 2008] takes 174 iterations and 12.82ms.

The linear solver is based on elastic projection and assumes that the post-response relative velocities are exactly zero. Based on this assumption, the optimization problem can be approximated by a

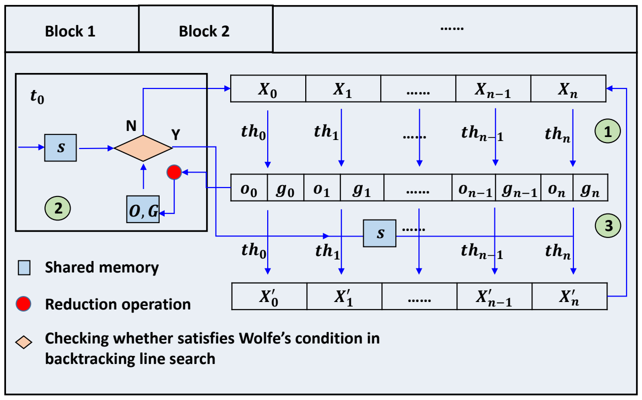


Fig. 4. **Impact Zone Solving Pipeline:** Each block is used to solve an impact zone optimization problem. The whole computation consists of three steps. $X_0 \ldots X_n$ and $X'_0 \ldots X'_n$ represent the positions of the vertices before and after updating. $th_i$ is the $i$th thread in a block. $s$ is the step length and $o_0 \ldots o_n$, $g_0 \ldots g_n$ are the objective function (based on Equation 1) values and the gradient descent directions of all the vertices, represented as $O$ and $G$, respectively $O$ is the total objective function value, and $G$ is the sum of the square of elements in all gradient descent directions. These two values are used to check Wolfe's condition.

linear system, but this can result in 'sticking' artifacts [Harmon et al. 2008]. Prior methods use many iterations of repulsive impulses [Bridson et al. 2002] to prevent such sticking behaviors and use an impact-zone solver at the end to reduce the impact of these sticking artifacts on the response computations. In our approach, we only use one pass of repulsive impulses, i.e. incorporate proximity constraints into the implicit time integration to prevent the sticking behavior, and mainly rely on the impact zone solver to resolve penetrations. Overall, our non-linear GPU solver demonstrates better accuracy and performance for collision response computation.

### 4.4 Timing and Convergence Analysis

Fig. 5 highlights the time spent and the number of iterations taken by our GPU solver on the Andy benchmark ((a) and (b), time step 1/200s) and on the Funnel benchmark ((c) and (d), time step 1/100s). As shown in the figure, the execution time is approximately proportional to the number of iterations. In practice, our GPU solver usually converges within 10 iterations for small time steps, even for complex contact configurations. However, our GPU solver may fail if the simulator chooses a large time step. A failure case is shown in Fig. 7(a). Our solver cannot converge to a non-penetration state when simulating the Dress benchmark with a large time step, i.e. 1/20s. Even after 200 iterations, there are still 1002 unresolved impacts. However, with a small time step, i.e. 1/50s, our method can solve all the impacts using 57 iterations (Fig. 7(b)). In this case, the number of impacts increases after 20 iterations. This can occur when the adjusted vertices trigger new penetrations with the surrounding triangles, and this can expand the size of impacts. Usually, the number of impacts decreases with more iterations. However, when new penetrations occur and new triangles are inserted into

**Algorithm 3** GPU-based Non-Linear Impact Zone Solver

```
1:  for each iteration do
2:      for each vertex in impact zone parallel do
3:          compute gradient descent direction;
4:      end for
5:      for each vertex in impact zone parallel do
6:          compute objective function;
7:      end for
8:      if the index of thread is 0 then
9:          adjust step length;
10:         if Wolfe's condition in line search is satisfied then
11:             continue;
12:         else go back to line 5;
13:         end if
14:     end if
15:     if step length is less than the prescribed threshold or the
        number of iterations reaches the maximum then
16:         break;
17:     end if
18:     for each vertex in impact zone parallel do
19:         corresponding thread updates the vertex using the step
        length and descent direction;
20:     end for
21:     for each vertex in impact zone parallel do
22:         corresponding thread updates the vertex using Cheby-
        shev acceleration;
23:     end for
24:     parallel update Lagrange multipliers $\lambda$ for each constraint;
25: end for
```

the impact zone, the number of impacts may increase and the response algorithm may fail to converge to a penetration-free state. Figure 6 highlights the number of iterations taken by our GPU solver on the Andy benchmark (frames 1000 – 2000). All these iterations are terminated when their step lengths are under our prescribed threshold ($10^{-12}$). Overall, our non-linear solver demonstrates better convergence than prior linear solvers, though it is not guaranteed to converge for large time steps.

## 5 IMPLEMENTATION AND PERFORMANCE

### 5.1 Implementation

We have implemented our collision handling algorithm as part of cloth simulation approach (I-Cloth) on an NVIDIA GeForce GTX 1080 (with 2560 cores at 1.6 GHz and 8G memory). Our implementation uses CUDA toolkit 9.1 and Visual Studio 2013 as the underlying development environment. We use a standard PC (Windows 7 Ultimate 64 bits/Intel I7 CPU@3.5G Hz/8G RAM) to evaluate performance. We perform single-precision floating-point arithmetic for all the computations on the GPU along with exact CCD computations.

### 5.2 Benchmarks

We use many regular and irregular-shaped cloth meshes to evaluate the performance (see video). These include three regular-shaped cloth benchmarks:

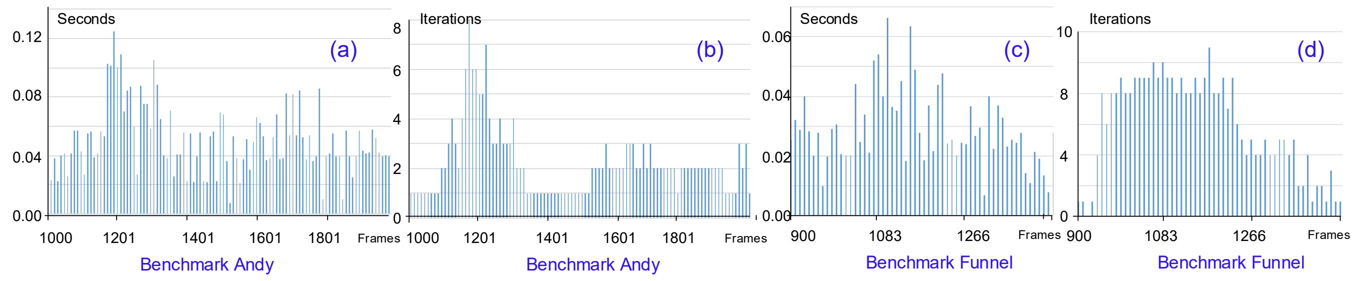


Fig. 5. **Timing and Converge Analysis:** This figure highlights the time spent and the number of iterations taken by our GPU solver on the Andy benchmark ((a) and (b)) for frames 1000 – 2000 and the Funnel benchmark ((c) and (d)) for frames 900 – 1450. For most frames, our solver converges in a few iterations.

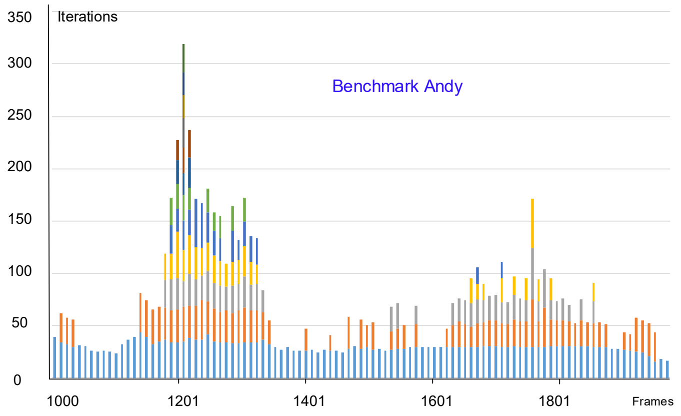


Fig. 6. **Gradient Descent Iterations:** We highlight the number of iterations taken by our GPU solver on the Andy benchmark (frames 1000 – 2000). All these iterations are terminated, when the their step lengths are under our prescribed threshold ($10^{-12}$). We use different colors to highlight the maximum number of gradient descent iterations used in terms of solving all impact zones (with Algorithm 3).

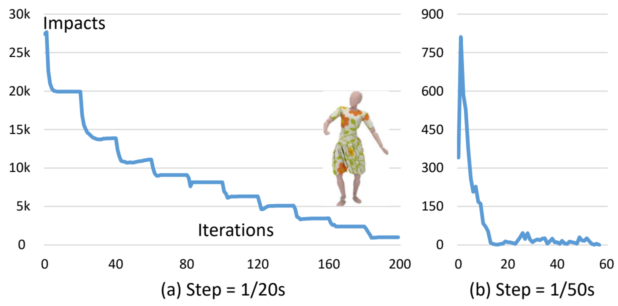


Fig. 7. **Failure Case Analysis:** (a) highlights a failure case where our GPU solver cannot converge to a non-penetration state when simulating the Dress benchmark with a large time step, i.e., 1/20s. Even after 200 iterations, there are still 1002 impacts unresolved. However, with a small time step, i.e., 1/50s, our method can resolve all the impacts within 57 iterations (b).

- **Funnel:** Three pieces of cloth with a total of 200K triangles fall into a funnel and fold to fit into the funnel (Fig. 1(e)).

| Resolution (triangles) | Bench-marks | Time Steps(s) | Cloth Simulation (fps) | | | Collsion Handling (s) | |
|---|---|---|---|---|---|---|---|
| | | | CAMA | PSCC | I-Cloth | PSCC | I-Cloth |
| 200K | Sphere | 1/200 | 0.35 | 1.06 | 1.95 | 0.790 | 0.285 |
| 200K | Twistin | 1/200 | 0.34 | 1.03 | 5.08 | 0.660 | 0.164 |
| 200K | Funnel | 1/100 | 0.60 | 2.33 | 8.47 | 0.353 | 0.066 |
| 127K | Andy | 1/200 | 0.27 | 1.19 | 3.73 | 0.689 | 0.177 |
| 90K | Dress | 1/100 | 0.28 | 0.92 | 6.06 | 0.774 | 0.073 |
| 215K | Tiered | 1/200 | 0.16 | 0.64 | 1.23 | 1.310 | 0.654 |
| 124K | Bishop | 1/100 | 0.31 | 1.06 | 4.46 | 0.743 | 0.148 |

Fig. 8. **Performance Comparison:** This table shows the average performance of different cloth simulators (the middle three columns) and collision handling algorithm (the right two columns) for various benchmarks on an NVIDIA GeForce GTX 1080. We observe considerable speedups over CAMA and PSCC: up to $22X$ and $6.6X$, respectively. Our incremental collision handling module is about $10.6X$ faster than PSCC.

- **Twisting:** Three pieces of cloth with a total of $200K$ triangles twist severely as the underlying ball rotates (Fig. 1(f)).
- **Sphere:** Three pieces of hanging cloth with a total of $200K$ triangles are hit by a forward/backward moving sphere (Fig. 1(g)).

These benchmarks contain many collisions. We also evaluate the performance on 5 complex benchmarks used for garment simulation:

- **Dress:** A dancing lady wearing a skirt (with $90K$ triangles) (Fig. 1(a)). We also manipulate this dress by a user applying a force (Fig. 2(b)).
- **Tiered:** A lady wearing a ruffled, layered skirt with $215K$ triangles (Fig. 1(b)).
- **Andy:** A boy wearing three pieces of clothing (with $127K$ triangles) practicing Kung-Fu (Fig. 1(c)).
- **Bishop:** A swing dancer wearing three pieces of clothing (with $124K$ triangles) (Fig. 1(d)).
- **Scarf:** A user ties a scarf (with $10K$ triangles) by applying the forces at the ends (Fig. 2(b)). I-Cloth can compute the new configurations at 8 – 10 fps.

## 5.3 Performance

Fig. 8 shows the mesh resolution, time step size and highlights the run-time performance of I-Cloth on these benchmarks. We also compare the performance of two recent GPU-based cloth simulation methods: CAMA [Tang et al. 2016] and PSCC [Tang et al. 2018]. As compared to PSCC, we observe up to $7X$ speedup using I-Cloth on

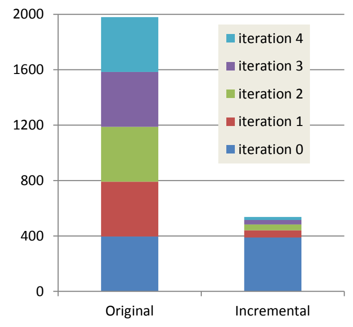


Fig. 9. **Benefits of Incremental CCD:** For a specific frame of Andy benchmark, four iterations are performed to compute a penetration-free state. The left figure shows the number of bounding box overlap tests for a prior CCD algorithm [Tang et al. 2018]. The right figure highlights the number of overlap tests performed by our incremental CCD algorithm (Section 3). As the number of iterations increases, the size of $T_d$ decreases and we observe 3.68X fewer overlap tests.

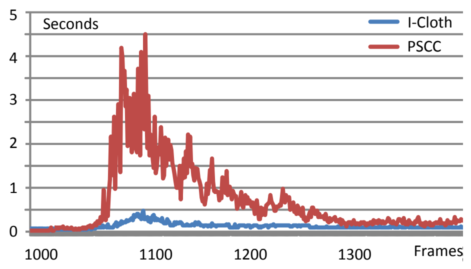


Fig. 10. **Benefits of a GPU-based Impact Zone Solver:** This figure compares the time spent in collision handling module for some frames (1000 – 1400) of the Funnel benchmark: our incremental collision handling algorithm vs. PSCC, which is based on [Harmon et al. 2008]. Our approach is faster and more robust in terms of handling deep penetrations.

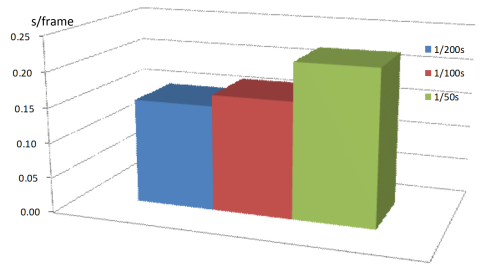


Fig. 11. **Performance with different time steps:** We evaluate the performance of I-Cloth on the Dress benchmark by increasing the time step size. Our non-linear solver can handle resulting deep penetrations and generate similar simulation results (see video) with about 2.5X performance improvement.

the same GPU. Collision handling in PSCC is performed using the solver described in [Harmon et al. 2008].

Fig. 9 highlights the benefits of incremental CCD over state of the art CCD algorithms. For a specific frame of Benchmark Andy, four iterations are performed to compute a penetration-free state of the cloth mesh. The left bar shows the number of bounding box overlap tests in full CCD algorithm that also uses high and low-level culling algorithms [Tang et al. 2018]. The right bar shows the number of overlap tests performed by our incremental CCD algorithm (Section 3).

Fig. 10 highlights the benefits of our GPU-based impact zone solver. Compared with the response algorithm used in PSCC, we observe significant speedups for the frames corresponding to tangled cloth with deep penetrations.

### 5.4 Large time steps

Prior cloth simulation algorithms tend to take smaller time steps to prevent deep penetrations. In contrast, our non-linear GPU impact zone solver allows us to take larger time steps and still compute reliable collision response. Fig. 11 highlights the performance of I-Cloth with different time steps for Benchmark Dress. With linearly increasing time steps, we observe almost equal simulation quality (see video for comparison), but the frame time only increases slightly. This implies that we can use higher time steps to achieve better overall performance. In Fig. 11, we obtain $1.76X$ and $2.73X$ speedups with $2X$ and $4X$ larger time steps, respectively.

## 6 COMPARISON AND ANALYSIS

In this section, we compare the performance of our approach with prior methods:

- *Comparison with PSCC:* Fig. 12 is a performance comparison for Benchmark Andy between PSCC and I-Cloth. Compared to PSCC, I-Cloth achieves $3.25X$ speedup on overall performance; the peak speedup for a frame is $6.6X$. The figure also shows the running time ratios of different computing stages of PSCC (left) and I-Cloth (right), respectively. These stages are: time integration, broad phase testing (high-level culling), narrow phase testing (low-level culling and exact elementary tests), and penetration handling. As shown in the figure, penetration handling is no longer the major efficiency bottleneck of I-Cloth. We have provided more comparison and breakup ratio figures for other benchmarks in the supplementary materials.
- *Other Collision Detection Algorithms:* Our incremental CCD algorithm is a simple extension of recent BVH-based [Tang et al. 2016] or Spatial Hashing-based [Fan et al. 2011; Pabst et al. 2010; Tang et al. 2018] collision detection algorithms. Besides impact zone-based penetration handling, our approach can also be used with adaptive remeshing [Narain et al. 2012] by marking all the inserted or modified triangles in $T_d$.

### 6.1 Convergence and Large Time Steps

Our non-linear impact zone solver enables use of larger time steps as compared to prior methods. We observe improved convergence in our benchmarks. However, for very large time steps our solver may not converge, as shown in Fig. 6.

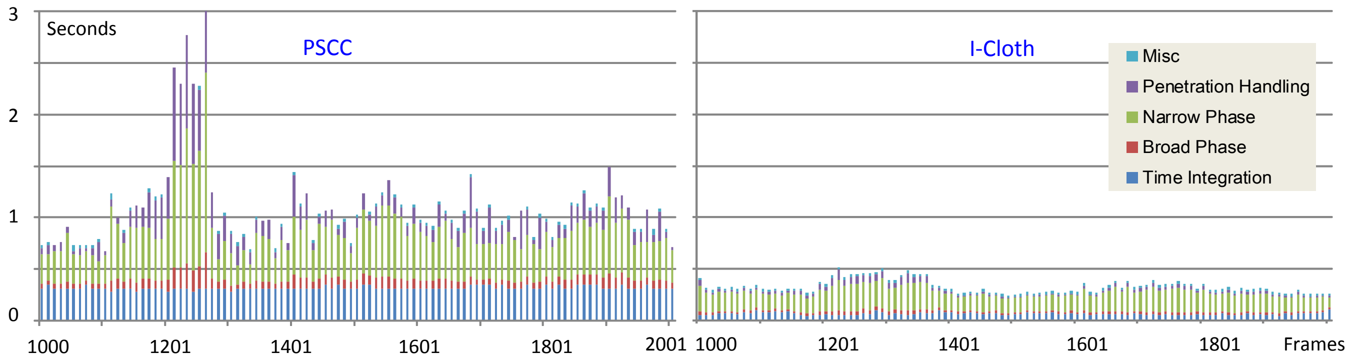


Fig. 12. **Performance Comparison on Andy benchmark:** We highlight the performance breakdown between time integration, CCD and collision response for each frame. For deeper penetrations (frames #1100-1300), our collision handling algorithm achieves high-speedups (up to $9X$), while the average speedup in cloth simulation is $3.25X$ over all frames.

## 6.2 Overall Performance

Our reported FPS is based on the execution time of each step. We have highlighted the time spent in each stage of the algorithm in Fig. 12 (and in more figures in the supplementary material). Currently, penetration detection and impact zone solving take a large fraction of the frame time.

## 6.3 Penetration Handling

As shown in Fig. 13, we compare the performance of the cloth simulation using three different schemes on a simplified version of benchmark Sphere:

- For case (a), implicit time integration without proximity constraints or an impact zone solver, we observe 1065 impacts (VF/EE penetrations).
- For case (b), implicit time integration with proximity constraints only (no impact zone solver), we observe 62 impacts or penetrations.
- For case (c), implicit time integration with proximity constraints and followed by our non-linear impact zone solver, a penetration-free state is achieved.

As highlighted in the figure, without the use of our impact solver, there are still penetrations or impacts after the integration step is performed. Overall, we need to use a combination of proximity constraints and our GPU-based non-linear impact zone solver to handle the inter-object and intra-object penetrations effectively and efficiently. This combination is more robust than prior collision response methods based on impact zones.

## 7 CONCLUSION, LIMITATIONS, AND FUTURE WORK

We present an incremental collision handling algorithm for GPU-based interactive cloth simulation. This includes an incremental CCD algorithm that keeps track of overlapping primitives using a dynamic data structure. and a non-linear impact zone solver. All these computations are parallelized on a GPU and combined with implicit time integration for interactive cloth simulation. Our resulting system, I-Cloth, can simulate complex cloth mesh at almost interactive rates, and we observe considerable speedups.

Our approach has some limitations. Our approach is mainly designed for cases with complex contacts or deep penetrations. The accuracy is governed by two main assumptions: linear interpolating motion for CCD and collision response computation based on impact zones. While our non-linear solver improves the accuracy, it is not guaranteed to converge with very deep penetrations or complex contact configurations. Furthermore, we assume that the proximity constraints in implicit time integration would prevent the sticking behavior. We have provided supplementary materials[1] (more benchmark timing data, video, API source code, etc.) to encourage future research in GPU-based cloth simulation.

There are many avenues for future research. In addition to overcoming the limitations, we feel that it is possible to further improve the performance by exploiting the memory hierarchy and caches to improve the performance of CCD and impact zone solvers. It would be useful to combine our method with accurate collision response methods based on elastoplastic methods and use our incremental CCD with other cloth simulation algorithms. It would be useful to achieve interactive performance using multiple GPUs (e.g., 20fps).

## ACKNOWLEDGEMENTS

This research is supported in part by NSFC (61732015, 61572423, 61572424, 61832016), the National Key R&D Program of China (2017YFB1002703), the Science and Technology Project of Zhejiang Province (2018C01080), and Zhejiang Provincial NSFC (LZ16F020003). Dinesh Manocha is supported in part by NSF Grant 1547106 and Intel. We would like to thank Xiaorui Chen for helping on the benchmarks, and the reviewers' constructive feedback and suggestions.

## REFERENCES

Samantha Ainsley, Etienne Vouga, Eitan Grinspun, and Rasmus Tamstorf. 2012. Speculative parallel asynchronous contact mechanics. *ACM Trans. Graph.* 31, 6, Article 151 (Nov. 2012), 8 pages.
David Baraff and Andrew Witkin. 1998. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques (SIGGRAPH '98).* ACM, New York, NY, USA, 43–54.
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.

[1]https://min-tang.github.io/home/ICloth/

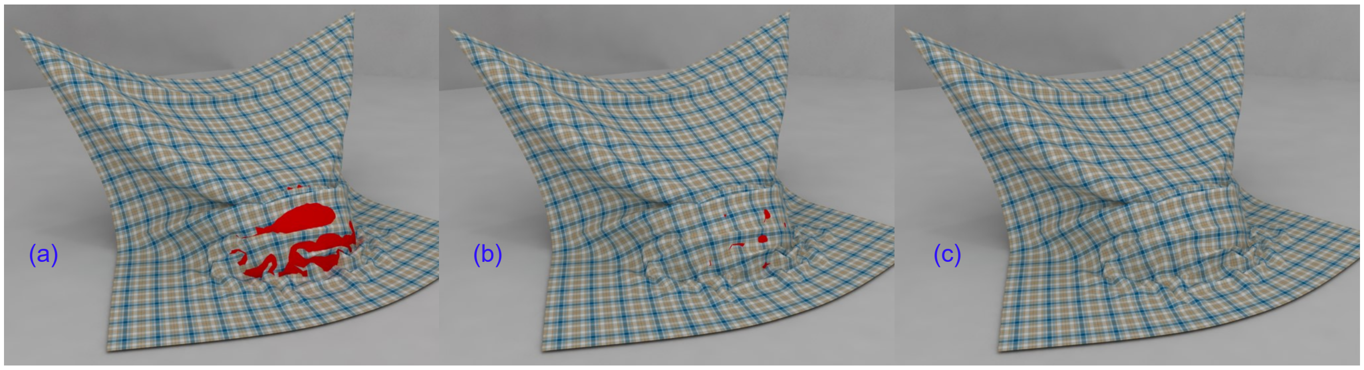


Fig. 13. **Impacts:** We compare the simulation results with implicit time integration without proximity constraints and impact zone solver (a), with proximity constraints only (b), and with both proximity constraints and impact zone solver (c). There are 1065 impacts in (a), 62 impacts in (b), and no impact in (c).

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

Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. 2014. Yarn-level Simulation of Woven Cloth. *ACM Trans. Graph. (SIGGRAPH Asia)* 33, 6, Article 207 (Nov. 2014), 11 pages.

Edilson de Aguiar, Leonid Sigal, Adrien Treuille, and Jessica K. Hodgins. 2010. Stable spaces for real-time clothing. *ACM Trans. Graph. (SIGGRAPH)* 29, Article 106 (July 2010), 9 pages. Issue 4.

Wenshan Fan, Bin Wang, Jean–Claude Paul, and Jiaguang Sun. 2011. A Hierarchical Grid Based Framework for Fast Collision Detection. *Computer Graphics Forum* 30, 5 (2011), 1451–1459.

Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. 2005. Quick-CULLIDE: Fast Inter- and Intra-Object Collision Culling Using Graphics Hardware. In *IEEE Virtual Reality Conference 2005, VR 2005, Bonn, Germany, March 12-16, 2005*. 59–66.

Qi Guo, Xuchen Han, Chuyuan Fu, Theodore Gast, Rasmus Tamstorf, and Joseph Teran. 2018. A Material Point Method for Thin Shells with Frictional Contact. *ACM Trans. Graph.* 37, 4 (2018), 147:1–147:15.

David Harmon, Etienne Vouga, Rasmus Tamstorf, and Eitan Grinspun. 2008. Robust Treatment of Simultaneous Collisions. *ACM Trans. Graph. (SIGGRAPH)* 27, 3, Article 23 (Aug. 2008), 4 pages.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. 2017. Anisotropic Elastoplasticity for Cloth, Knit and Hair Frictional Contact. *ACM Trans. Graph.* 36, 4, Article 152 (July 2017), 14 pages.

Doyub Kim, Woojong Koh, Rahul Narain, Kayvon Fatahalian, Adrien Treuille, and James F. O'Brien. 2013. Near-exhaustive Precomputation of Secondary Cloth Effects. *ACM Trans. Graph (SIGGRAPH)*. 32, 4, Article 87 (July 2013), 8 pages.

Yongjoon Lee, Sung-Eui Yoon, Seungwoo Oh, Duksu Kim, and Sunghee Choi. 2010. Multi-Resolution Cloth Simulation. *Comp. Graph. Forum (Pacific Graphics)* 29, 7 (2010), 2225–2232.

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), 209:1–7.

Rahul Narain, Armin Samii, and James F. O'Brien. 2012. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph. (SIGGRAPH Asia)* 31, 6, Article 152 (Nov. 2012), 10 pages.

Xiang Ni, L.V. Kale, and R. Tamstorf. 2015. Scalable Asynchronous Contact Mechanics Using Charm++. In *IEEE Parallel and Distributed Processing Symposium (IPDPS)*. 677–686.

Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (second ed.). Springer, New York, NY, USA.

Miguel A. Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. 2009. Implicit Contact Handling for Deformable Objects. *Computer Graphics Forum* 28, 2 (2009), 559–568.

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

Xavier Provot. 1997. Collision and Self-collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface*. 177–189.

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. Comp. Graph.* 15, 2 (March 2009), 339–350.

Eftychios Sifakis, Sebastian Marino, and Joseph Teran. 2008. Globally coupled collision handling using volume preserving impulses. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 147–153.

Avneesh Sud, Naga Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. 2006. Fast proximity computation among deformable models using discrete Voronoi diagrams. *ACM Trans. Graph. (SIGGRAPH)* 25, 3 (July 2006), 1144–1153.

Min Tang, Zhongyuan Liu, Ruofeng Tong, and Dinesh Manocha. 2018. PSCC: Parallel Self-Collision Culling with Spatial Hashing on GPUs. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (2018), 18:1–18.

Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong. 2011. Collision-Streams: Fast GPU-based collision detection for deformable models. In *Proceedings of I3D*. 63–70.

Min Tang, Ruofeng Tong, Rahul Narain, Chang Meng, and Dinesh Manocha. 2013. A GPU-based Streaming Algorithm for High-Resolution Cloth Simulation. *Comp. Graph. Forum (Pacific Graphics)* 32, 7 (2013), 21–30.

Min Tang, Ruofeng Tong, Zhendong Wang, and Dinesh Manocha. 2014. Fast and Exact Continuous Collision Detection with Bernstein Sign Classification. *ACM Trans. Graph. (SIGGRAPH Asia)* 33 (November 2014), 186:1–186:8. Issue 6.

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. *Computer Graphics Forum (Proceedings of Eurographics 2016)* 35, 2 (2016), 511–521.

Etienne Vouga, David Harmon, Rasmus Tamstorf, and Eitan Grinspun. 2011. Asynchronous variational contact mechanics. *Computer Methods in Applied Mechanics and Engineering* 200 (June 2011), 2181–2194.

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

Huamin Wang. 2015. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 246.

Huamin Wang, Florian Hecht, Ravi Ramamoorthi, and James O'Brien. 2010. Example-based wrinkle synthesis for clothing animation. *ACM Trans. Graph. (SIGGRAPH)* 29, 4, Article 107 (July 2010), 8 pages.

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

Zhendong Wang, Tongtong Wang, Min Tang, and Ruofeng Tong. 2016. Efficient and robust strain limiting and treatment of simultaneous collisions with semidefinite programming. *Computational Visual Media* 2, 2 (Jun 2016), 119–130.

Changxi Zheng and Doug L. James. 2012. Energy-based Self-Collision Culling for Arbitrary Mesh Deformations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)* 31, 4 (Aug. 2012), 98:1–98:12.