EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

Volume 44 (2025), Number 2

A Unified Discrete Collision Framework for Triangle Primitives

Submission ID: paper1071

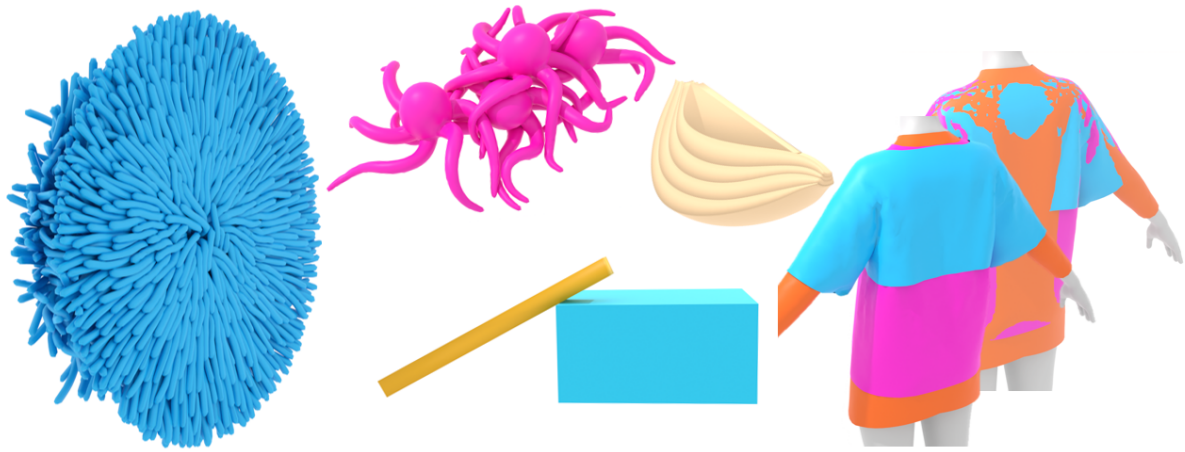

Figure 1: *Simulation results showcasing various types of contacts, generated solely by our method with XPBD.*

Abstract

We present a unified, primitive-first framework with DCD for collision response in physics-based simulations. Previous methods do not provide sufficient solutions on a framework that resolves edge-triangle and edge-edge collisions when handling self-collisions and inter-object collisions in a unified manner. We define a scalar function and its gradient, representing the distance between two triangles and the movement direction for collision response, respectively. The resulting method offers an effective solution for collisions with minor computational overhead and robustness for any type of deformable object, such as solids or cloth. The algorithm is conceptually simple and easy to implement. When using PBD/XPBD, it is straightforward to incorporate our method into a collision constraint.

CCS Concepts

• ***Computing methodologies*** → ***Collision detection;***

1. Introduction

Collision handling for deformable objects are challenging problems and active areas of research in computer graphics. Collision of objects assuming the real world involves various types of problems. Examples of possible collision scenarios are shown in Figure 1. For instance, a collision between two rectangular objects presents a simple problem with few meshes, but it is essential to ensure that edge intersections are detected. In the case of elastic object collisions, self-collision and inter-object collisions can occur simultaneously. Multi-layered objects, such as garments or onions, often already have intersections or contacts when the simulation begins. When considering collision handling, several factors must be taken into account: the type of collision (inter-collisions or self-collisions), the type of object (solid or surface), the initial state (intersection-free or not), and the collision location (at a point or on an edge).

Collision detection techniques are typically divided into Continuous Collision Detection (CCD) and Discrete Collision Detection (DCD). In recent years, Incremental Potential Contact [LFS*20], a widely used technique for collision handling, has emerged as a prominent method within CCD. CCD requires both starting with and maintaining an intersection-free state. In contrast, DCD allows for the handling of intersections that occur after collisions and resolving them through a three-step process: 1. Finding contact points for penetrated points, 2. Calculating the penetration measure, and 3. Computing the direction of movement for collision response.

Previous work on collision response can be broadly classified

into point-based and integral-based methods. Point-based methods [MEM*20; MZS*11; MASS15; FSG03; GBF03] generate the closest points on the surface to penetrating points. Most penetrating points are chosen from the vertices of the object, but in cases using Signed Distance Fields (SDFs), points on edges may also be included. Point-based methods are conceptually simple, and each proposed method addresses either a self-collision problem or an edge-edge intersection problem. However, it is not feasible to integrate existing specialized methods to address both issues. Only SDF-based methods handle arbitrary points on edges, but they struggle with self-collisions. On the other hand, integral-based methods [AFC*10; WFP12; ZMSL23] avoid these issues, but they can only be applied to solid objects or may exhibit low robustness depending on the initial states. Ultimately, no approach has been provided that can address all collision cases within the same framework.

The collision problems encountered in previous studies can fundamentally be addressed as triangle intersection problems. At the triangle primitives level, the following types of contacts can occur: point-triangle, edge-triangle, point-point, edge-edge, point-edge, and triangle-triangle contacts. Notably, all these contacts can be reduced to combinations of point-triangle and edge-edge interactions. In this paper, we propose a unified framework for addressing all issues at the triangle primitives level. While the geometric relationships of triangles have been a focus in collision detection, they have not, to our knowledge, been applied to handle collision response. However, we found that this geometric information is sufficient to address aforementioned collision problems. Our framework offers an effective solution for all the collisions shown in Figure 1. Our method requires no pre-computation and directly computes contact points, scalar functions representing signed distances between primitives, and their gradients representing the directions of movement for collision response. Furthermore, by formulating the distance as a function to enable gradient calculation, we achieve a streamlined algorithm that encompasses all three steps of the collision response process, resulting in high speed, simplicity, and ease of implementation. The entire process consists of no more than 185 lines of Python code.

2. Related Work

In this section, we review the most closely related previous work on collision response for deformable objects, focusing on approaches that leverage mesh-based information. For more comprehensive surveys on these topics, we refer the reader to [AEF22; WC21; NZXL20]. Regarding methods with DCD, Erleben [Erl18] summarizes various techniques, comparing eight different methods for finding contact points between two tetrahedra in rigid body simulations.

2.1. CCD for Deformable Objects

Several techniques focusing on triangle primitives have been proposed [BFA02; SSIF09; BEB12; Wan14; WCL*23], involving intersection tests based on primitive elements (vertices, edges, and faces). In particular, the method by Bridson et al. [BFA02] is considered one of the first truly robust approaches for handling collisions, contacts, and friction in cloth simulations. Another approach combines point sampling of objects with SDFs to determine when the trajectory of points intersects with the SDF isosurface [XB17]. CCD detects collisions before they occur; therefore, it requires ensuring an intersection-free state, which is computationally expensive. Moreover, even when starting from an intersection-free state, numerical issues can cause failures that result in objects merging.

2.2. Point-based Collisions

SDF is a popular shape representation for collision handling. It provides efficient queries and robust inside/outside information. A method that uses the SDF of an intersection-free pose of a character to solve self-intersections has been proposed [MZS*11], but Chen et al. [CDY23] have pointed out its inaccuracy with large deformations. An alternative approach is bifurcating the SDF nodes during construction when volumetric overlaps arise from self-intersections [MASS15]. For edge-edge collisions, several methods have been developed, generating contact points either at edge midpoints [FSG03] or at intersections of edges with the SDF isosurface [GBF03]. In particular, Fuhrmann et al. [FSG03] strongly recommend edge testing to enhance overall accuracy. Macklin et al. [MEM*20] proposed a method that generates contact points through local optimization on SDFs to accurately capture sharp point-face and edge-edge contacts. Although solutions for self-collisions and edge-edge collisions have been proposed individually, no unified approach addresses both aspects simultaneously. Methods that perform a sampling of the surface geometry at discrete points handle collisions between objects; however, Macklin et al. [MEM*20] highlight insufficiencies with sharp features and edge-edge collisions. Chen et al. [CDY23] proposed a method using tetrahedral ray traversal, which tests the validity of a given path and computes the closest surface points as contact points. This method is robust in providing accurate Euclidean shortest paths from points inside the mesh to the boundary, even when self-intersections are present. However, it does not sufficiently address edge-edge contacts beyond simply taking midpoints, leading to missed or unresolved edge-edge contacts (see Figure 4). Several methods for untangling cloth layers [BRB*19; SOTC22; YMJ*17] and for volumetric objects [FL01; HTK*04; ST05] have been proposed. Erleben [Erl18] extended the previous work to identify the most opposing surface triangles between two overlapping tetrahedra and calculate the contact normal based on those faces. Building on insights from these methods, we propose a more concise geometric approach that extends to include volumetric and surface objects together, without requiring a global search or history for collision response.

2.3. Integral-based Collisions

Unlike point-based methods, methods using integral values over a certain range can handle self-intersections and edge-edge collisions. The volume contact model [AFC*10; WFP12] and the integral penalty method [HFS*01] both identify overlapping volumes in shapes and introduce constraints to eliminate or minimize these penetrations. However, these models are only applicable to collisions between volumetric objects and require significant computational updates during deformation. Zesch et al. [ZMSL23] proposed a method using neural collision fields for intersecting triangle pairs, where integral values are pre-learned by a neural network, enabling

rapid collision computation during simulation. Since it's based on triangle primitives, it applies to arbitrary triangle meshes after a single network training. However, this method does not consider triangle normals or velocity vectors, as it assumes pre-penetration or minimal penetration initiation. This can lead to potential issues, such as incorrect collision directions and reduced stability, particularly when the initial conditions are not met (see Figure 4).

3. Method

3.1. Triangle-Triangle Intersection Test

To introduce our technique based on the triangle intersection detection method presented by Möller [Möl97], we briefly describe the algorithm. The core of the method is to treat triangle intersection detection as an interval overlap test along the line where the planes containing the triangles intersect. Figure 2 illustrates the geometric situation.

Let us denote the vertices of two triangles, T_1 and T_2, by $\boldsymbol{V}_0^1, \boldsymbol{V}_1^1, \boldsymbol{V}_2^1$, and $\boldsymbol{V}_0^2, \boldsymbol{V}_1^2, \boldsymbol{V}_2^2$, respectively; and the planes π_1 and π_2 in which the triangles lie. For example, the plane equation π_2: $\boldsymbol{N}_2 \cdot \boldsymbol{X} + d_2 = 0$ (where $\boldsymbol{X}$ is any point on the plane) is computed:

$$\begin{aligned} \boldsymbol{N}_2 &= \text{normalize}((\boldsymbol{V}_1^2 - \boldsymbol{V}_0^2) \times (\boldsymbol{V}_2^2 - \boldsymbol{V}_0^2)), \\ d_2 &= -\boldsymbol{N}_2 \cdot \boldsymbol{V}_0^2. \end{aligned} \tag{1}$$

The signed distance $d_{\mathrm{V}_i^1}$ from each vertex of T_1 to the plane π_2 is calculated by substituting the vertex coordinates into the plane equation:

$$d_{\mathrm{V}_i^1} = \boldsymbol{N}_2 \cdot \boldsymbol{V}_i^1 + d_2, \quad i = 0, 1, 2. \tag{2}$$

If all $d_{\mathrm{V}_i^1}$ ($i = 0$, 1 and 2) have the same sign, there is no intersection state between the triangles. Otherwise, the intersection of π_1 and π_2 is a line, $\boldsymbol{L} = \boldsymbol{O} + t\boldsymbol{D}$, where $\boldsymbol{D} = \boldsymbol{N}_1 \times \boldsymbol{N}_2$ is the direction of the line. If the intervals of both triangles on $\boldsymbol{L}$ overlap, the triangles are determined to intersect. To compute the scalar interval $[t_1, t_2]$ on $\boldsymbol{L}$, the vertices of T_1 are projected onto $\boldsymbol{L}$:

$$\begin{aligned} p_{\mathrm{V}_i^1} &= \boldsymbol{D} \cdot (\boldsymbol{V}_i^1 - \boldsymbol{O}), \\ t_1 &= p_{\mathrm{V}_0^1} + (p_{\mathrm{V}_1^1} - p_{\mathrm{V}_0^1}) \frac{|d_{\mathrm{V}_0^1}|}{|d_{\mathrm{V}_0^1} - d_{\mathrm{V}_1^1}|}. \end{aligned} \tag{3}$$

The intersection of T_2 with π_1 is calculated in the same manner. The steps of the algorithm are summarized as follows:

1. Compute the plane equation of T_2.
2. Reject the intersection if T_1 is on the same side.
3. Compute the plane equation of T_1.
4. Reject the intersection if T_2 is on the same side.
5. Compute the intersection line and intervals for each triangle.
6. Test for overlaps in the intervals.

We apply the concept of the signed distance $d_{\mathrm{V}_i^j}$ (j =1 and 2) to the collision response, which will be discussed in the next subsection. During implementation, other efficient methods for triangle intersection detection [XMCX22] can be used to further enhance performance.

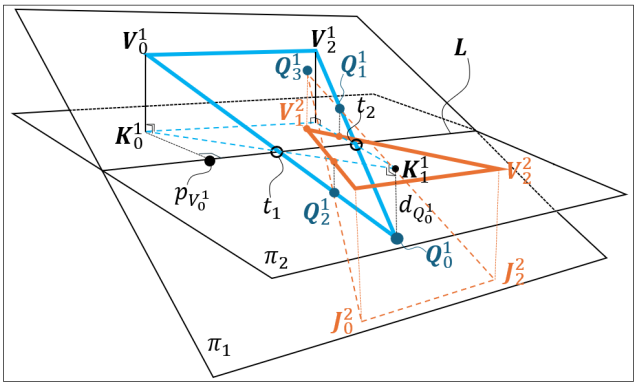

Figure 2: *The geometrical situation:* T_1 *(blue bold line) and* T_2 *(orange bold line) represent the original triangles. The blue dot line triangle is the projection of* T_1 *onto* π_2 *using the normal vector of* π_2*, and the orange dot line triangle is the projection of* T_2 *onto* π_1 *using the normal vector of* π_1*. Candidate points* $\boldsymbol{Q}_0^1 - \boldsymbol{Q}_3^1$ *on* T_1 *are derived from these projection processes.*

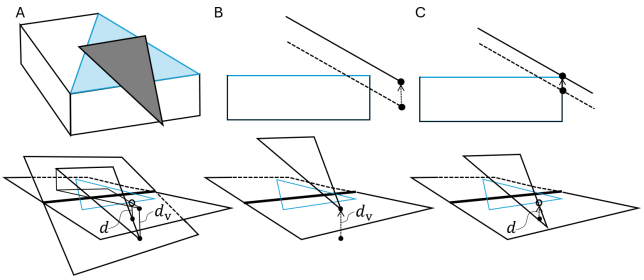

Figure 3: *Comparison of behaviors at the same frame between vertex-only calculations and calculations at intersection points after projection. (A) Consider the scenario where a rectangular prism and a triangle are in contact, focusing on finding the contact points between the blue triangle and the black triangle. (B) When corrected using the signed distance* d_V *at the vertex, the vertex moves excessively and fails to reach a contact state. (C) By utilizing the distance* d *computed by our proposed method, a collision is resolved at the minimum distance, resulting in edge-edge contacts. The upper part of (B) and (C) shows a side view.*

3.2. Computation of Scalar Functions and Gradients

In this subsection, we define scalar functions and their gradients, extending the technique proposed by Möller. While Möller's method focuses solely on fast triangle intersection detection, our approach additionally defines how to compute contact points, scalar distances, and collision responses. The idea behind our approach is that by using a global common function (signed distance) for collision response, we can ensure consistent handling of object collisions, even with a primitive-based method. $d_{\mathrm{V}_i^j}$ represents the signed Euclidean distance from a vertex $\boldsymbol{V}_i^j$ to a point on the plane of the intersecting triangle pair. We define the sign to be positive when the relationship with the other triangle is valid (e.g. outside the solid, or in the order of the predefined cloth layers), negative when it is violated, and zero

when triangles are in contact. Hence, when the sign is negative, the magnitude of $d_{\mathrm{V}_i^j}$ can be interpreted as the penetration depth relative to the plane. This penetration can be resolved by pushing the triangle back by the same magnitude.

To handle contact on edges, it is insufficient to compute d_{V^j} for each vertex, as the vertex-to-plane distance alone can overestimate or underestimate the penetration depth (see Figure 3). Therefore, we determine the appropriate contact points from the other triangle's range. The range is defined by the area enclosed by the following three types of points. Consider a triangle A paired with triangle B:

1. The vertices of triangle A, projected onto the plane of B using the normal vector of B, corresponding to the vertices of A determined to lie within the interior of B. (For example, point $\boldsymbol{Q}_0^1$ in Figure 2.)
2. The points on triangle A, projected onto the plane of A using the normal vector of B, corresponding to the vertices of B determined to lie within the interior of A. (For example, point $\boldsymbol{Q}_3^1$ in Figure 2.)
3. The points on the edges of triangle A, projected onto the plane of B using the normal vector of B, corresponding to the intersections of the projected edges of A with the edges of B. (For example, points $\boldsymbol{Q}_1^1$ and $\boldsymbol{Q}_2^1$ in Figure 2.)

The third type of points is equal to the intersections of the projected edges of B with the edges of A, after projecting triangle B onto the plane of A using the normal vector of B. Given that triangles are flat shapes, the points that define the area can be directly considered as candidates for determining the penetration depth. Among these candidate points, the point that can resolve the collision with the smallest distance is selected.

The computation of the depth from T_1 to T_2 proceeds as follows: First, T_1 is projected onto the plane π_2 using the normal vector $\boldsymbol{N}_2$ of T_2. The vertices $\boldsymbol{K}_0^1, \boldsymbol{K}_1^1, \boldsymbol{K}_2^1$ of the projected T_1 are computed:

$$\boldsymbol{K}_i^1 = \boldsymbol{V}_i^1 - (\boldsymbol{N}_2 \cdot (\boldsymbol{V}_i^1 - \boldsymbol{V}_0^2))\boldsymbol{N}_2. \tag{4}$$

At this point, T_2 and the projected T_1 are coplanar. Next, to compute the range in which the projected T_1 is contained within T_2, three vertex-triangle tests and nine edge-edge tests are performed. When edges intersect, the intersection points are also determined. Vertices judged to be within T_2 (corresponding to the first type mentioned above) and intersection points between intersecting edges (corresponding to the third type) are candidates for points with penetration depth. Similarly, T_2 is projected onto the plane π_1 of T_1 using the normal vector $\boldsymbol{N}_2$ of T_2. The vertices $\boldsymbol{J}_0^2, \boldsymbol{J}_1^2, \boldsymbol{J}_2^2$ of the projected T_2 are computed:

$$\boldsymbol{J}_i^2 = \boldsymbol{V}_i^2 + \frac{\boldsymbol{N}_1 \cdot (\boldsymbol{V}_0^1 - \boldsymbol{V}_i^2)}{\boldsymbol{N}_1 \cdot \boldsymbol{N}_2}\boldsymbol{N}_2. \tag{5}$$

Then, three vertex-triangle tests are performed to determine the vertices within T_1 (corresponding to the second type), which are added as candidates for points with penetration depth. The actual candidate points, which are on T_1, are denoted as $\boldsymbol{Q}_l^1$ $(l = 0, 1, \dots)$. Finally, the signed distance $d_{\mathrm{Q}_l^1}$ along the normal vector $\boldsymbol{N}_2$ is computed. The largest absolute value of the negative signed distances is determined to be the penetration depth from T_1 to T_2. The penetration depth from T_2 to T_1 is calculated using the normal vector $\boldsymbol{N}_1$ in the same manner as described above, then the candidates $\boldsymbol{Q}_m^2$ $(m = 0, 1, \dots)$ are obtained. The minimum of the two depths is used as the magnitude of the collision response in our method.

In summary, the magnitude of the collision response depends on the distance between T_1 and T_2 and is defined by the smaller of the penetration depths for each triangle. The scalar function f, where $d_{\mathrm{Q}_l^1}$ is the signed distance for the candidate points of T_1 and $d_{\mathrm{Q}_l^2}$ for T_2, is computed as:

$$\begin{aligned} f &= \min(d_{\mathrm{tri1}}, d_{\mathrm{tri2}}), \\ d_{\mathrm{tri1}} &= \max(0.0, -d_{\mathrm{Q}_l^1}), \\ d_{\mathrm{tri2}} &= \max(0.0, -d_{\mathrm{Q}_m^2}). \end{aligned} \tag{6}$$

In our method, the gradient $-\nabla \boldsymbol{f}$ of the function f corresponds to the normal vector of either one of the triangles, making the calculation straightforward. The derivation of the gradient $-\nabla \boldsymbol{f}$ from f is provided in the supplemental material.

3.3. Collision Response as a PBD Constraint

When using Position Based Dynamics (PBD) [MHHR07] for simulation, adding our scalar function as a PBD constraint enables effective collision response. The algorithm for collision response between two triangles in PBD is as follows, with Step 9 corresponding to the constraint C and its gradient $-\nabla \boldsymbol{C}$:

1. Project both triangles onto each other's planes using $\boldsymbol{N}_2$.
2. Identify candidates with vertex-triangle and edge-edge tests.
3. Compute signed distances of candidates on T_1 along $\boldsymbol{N}_2$.
4. Project both triangles onto each other's planes using $\boldsymbol{N}_1$.
5. Identify candidates with vertex-triangle and edge-edge tests.
6. Compute signed distances of candidates on T_2 along $\boldsymbol{N}_1$.
7. Compute the penetration depth d_{tri1} from T_1 to T_2.
8. Compute the penetration depth d_{tri2} from T_2 to T_1.
9. Set the PBD constraint $C = f$ (see Equation 6) and resolve the collision using the gradient $-\nabla \boldsymbol{C}$.

We provide the Python code of this method in the supplemental material as a reference to the reader.

4. Results

During each simulation step, we first use Axis-Aligned Bounding Box trees in the broad phase to roughly determine potential intersecting triangles. Next, we implement Möller's method [Möl97] to test detailed intersections in the narrow phase. Finally, our proposed method is processed to resolve the intersections. We only use DCD in our implementation. Our timing results are measured on an Intel Core i9-11900 CPU and an NVIDIA GeForce RTX 2080 Ti GPU. We integrated our proposed code into Gaia `https://github.com/AnkaChan/Gaia`, a physics simulation software based on XPBD [MMC16]. For the comparison with Zesch et al. [ZMSL23], we used the pre-trained model provided by the authors to add as a PBD constraint and its gradient. For the comparison with Chen et al. [CDY23], we used the built-in version in Gaia.

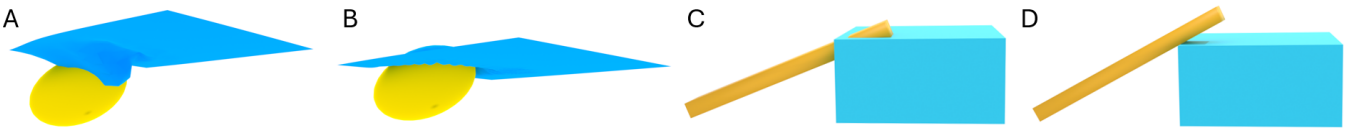

Figure 4: *Accuracy comparison with previous methods. (A) With the method of Zesch et al., collisions can cause the cloth to move in the wrong direction, resulting in penetration. (B) Our proposed method consistently corrects the mesh, allowing the cloth to fall smoothly. (C) The method of Chen et al. can miss collisions on edges, while (D) our method detects them and avoids penetration.*

Table 1: *Comparison with Zesch et al. [ZMSL23]: The runtime, measured in seconds, is the average of 10 executions when querying a batch of 256 triangle pairs on a CPU using PyTorch.*

	Zesch et al.	Our method
Runtime	3.21×10^{-3}	1.14×10^{-3}

Table 2: *Comparison with Chen et al. [CDY23]: The runtime, measured in seconds, is the average per frame on a CPU.*

	XPBD	Detection	Response
Chen et al.	6.40×10^{-1}	9.134×10^{-2}	5.274×10^{-3}
Ours	5.71×10^{-1}	9.414×10^{-4}	3.624×10^{-5}

4.1. Comparisons with Previous Work

We compare our approach with two methods: the triangle-primitive-based collision detection method using a neural network (NN) [ZMSL23] and the tetrahedral ray traversal method [CDY23], which effectively handles self-collisions in point-based methods. The performance comparison results are shown in Tables 1 and 2. For the comparison with the NN-based method, we followed the data generation approach described in [ZMSL23], preparing 256 normalized random triangle pairs. Both collision response algorithms were implemented in PyTorch. For the comparison with the point-based method, we prepared a scene consisting of four elastic objects dropping (totally, 11,628 vertices and 23,240 faces). As shown in Table 1, our method demonstrates comparable performance to that of the NN-based method. From Table 2, it can be seen that our method is more than 100 times faster than the point-based method in 'Response', which is related to each proposal.

For accuracy comparison, we present the simulation results of cloth dropping onto an ellipsoid in Figures 4 (A) and (B), comparing our method with that of Zesch et al. In their method, when a collision occurs, the cloth mesh moves in the wrong direction, resulting in deep penetration. This instability stems from the predicted value and the absence of normal information in the collision integration, which can result in overall inconsistency, especially when their DCD operates after penetration has occurred. In contrast, our proposed method uses scalar quantities in the normal direction for local triangle-triangle intersections, enabling appropriate corrections that maintain the overall consistency of the object. We also compare our method with Chen et al.'s approach, as shown in Figures 4 (C) and (D). Their method considers edge-edge collisions only at the midpoint between vertices, which can cause difficulties with collisions occurring at other points along the edge, especially when using a coarse mesh, resulting in penetration. In contrast, our proposed method handles collisions at arbitrary points on an edge, effectively managing edge-related issues regardless of mesh resolution.

4.2. Fundamental Test Cases

We conducted a fundamental collision test, including the edge case as proposed by Erleben [Erl18]. We successfully completed the test without causing breakdowns. The results are illustrated in the supplemental material.

Our method also handles cases where triangles intersect perpendicularly, which can lead to degeneration. One example of the collision handling steps for such a case is visualized in Figure 5. When degeneration occurs and the projected triangle becomes a line segment, the intersection points are back-projected onto the two overlapping edges. Although the corresponding pre-projection points yield two distinct points, the algorithm calculates the signed distances for each, ensuring accurate results. For a more detailed case-by-case validation, please refer to the supplemental material, which demonstrates that the corrections function correctly in all scenarios.

Note that when a triangle has already degenerated (e.g., its area becomes zero), our algorithm may fail. However, in such cases, most physics-based simulation methods would also not function correctly, which should be taken into account.

4.3. Self-Collision Handling

Our method can handle self-collisions within the same framework. A challenging example of self-collisions is shown in Figure 6, where a squishy ball collides with the ground. In simulations without self-collision handling, the ball gets tangled; on the other hand, when self-collision handling is activated, all self-intersections are appropriately resolved.

4.4. Layered Cloths Experiments

Our method can be applied to thin objects such as clothes, where the positional relationship between cloth layers along the direction of the normal vector is predefined. In the collision between the elastic octopus model and three-layered clothes, as shown in Figure 7, appropriate movements of both the cloth and elastic object are achieved. In this example, our framework can handle three types of collisions: cloth-to-cloth, octopus-to-cloth, and self-collisions of the octopus.

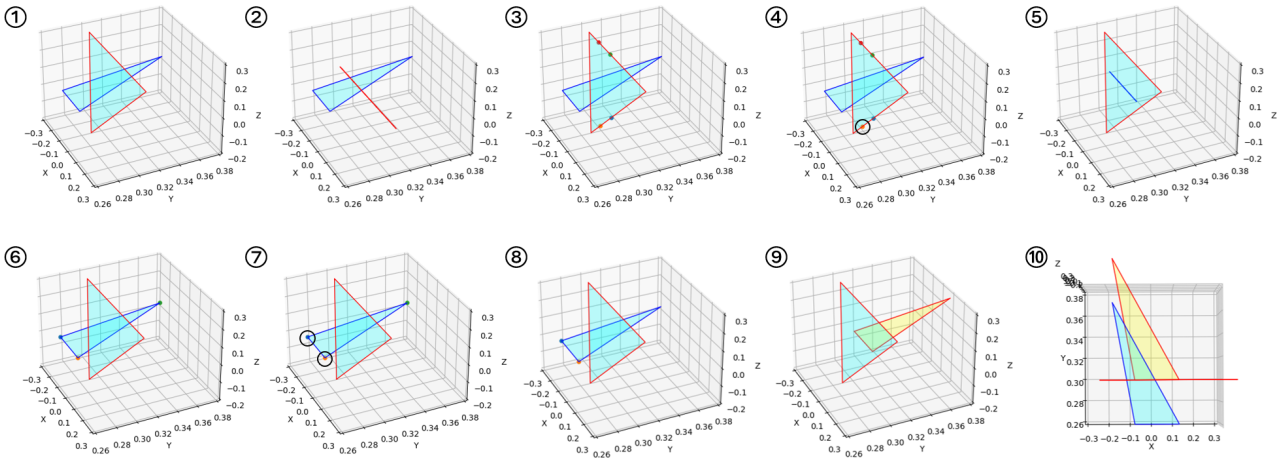

Figure 5: *Step-by-step collision handling process for an example where two triangles intersect perpendicularly. The normal vector of the triangle in the red frame is set to (0,1,0), and that of the triangle in the blue frame to (0,0,1). (1) Initial state. (2) Both triangles are projected using the normal vector of the second triangle (corresponding to step 1 in Section 3.3). (3) Candidate points are calculated (corresponding to step 2 in Section 3.3). (4) Signed distances are calculated at each point, determining the penetration depth for the first triangle (a point enclosed by a black circle, corresponding to steps 3 and 7 in Section 3.3). (5)-(7) The same process is applied for the second triangle (corresponding to steps 4-6 and 8 in Section 3.3). (8) The penetration depth calculated for both triangles in steps 7 and 8 in Section 3.3 is used to compute Equation 6. The candidate points with the smallest depth from the second triangle are chosen. (9) Since the gradient corresponds to the normal vector of the first triangle, it is applied to correct the collision (corresponding to step 9 in Section 3.3). (10) Top-down view: the blue triangle is before the correction and the yellow triangle is after the correction. Only a simple push-out correction applied to one of the triangles is shown.*

Figure 8 demonstrates that even when a simulation begins with intersecting garments, the method successfully resolves these collisions. This indicates the method's applicability to scenarios involving clothing layers, where intersections may occur before the physics-based simulation begins. Note that the layering order of the cloth is predefined, similar to the example in Figure 7.

4.5. Large-Scale Case

The robustness of our method was evaluated with large datasets. Figure 9 shows an example where 108 octopus models with self-collisions are falling down. Our method provides a robust collision handling solution, effectively managing complex, high-speed collisions involving numerous interactions.

4.6. Starting from a Penetrated or Contact State

Our method is capable of resolving pre-existing intersections. Besides the example shown in Figure 8, another demonstration is shown in Figure 10. In this example, all intersecting pairs are examined using the substep method outlined in Algorithm 1 of Macklin et al.'s paper [MSL*19], where the number of substeps n_{steps} is set to 60. Although the initial state contains penetrations, turning on collision handling progressively applies collision constraints to the intersecting pairs. In this case, all existing triangle intersections are resolved within 10 frames.

As shown in Figure 11, our method can also handle collisions for layered objects like an onion, starting from a state where the layers are already in contact before the simulation begins.

4.7. Performance

Performance metrics of the above experiments are shown in Table 3. Our method only incurs a small overhead in terms of the total calculation time for each frame. The material solver in the XPBD section was executed either on the CPU or GPU, and the last column on the right indicates which was chosen for each scene. All collision handling computations were performed on the CPU. 'Contact Type' indicates whether the collision was resolved by Point or Edge. A higher percentage of Edge suggests that many collisions were resolved at contact points on edges.

4.8. Limitations

Our method follows the inherent limitations of DCD. That is, when attempting to handle large time steps or fast movements using only DCD, deep penetrations or slipping through may occur, potentially leading to a failure to fully resolve the objects or causing them to penetrate each other again. By incorporating CCD, the limitations of DCD can be overcome, and the robustness of collision handling can be greatly improved. Our method focuses on the normal directions of two triangles, so when the two normal vectors face the same direction or when the correction direction cannot be determined based on the normal vector, incorrect corrections may occur, or the solution may fail to converge. However, if the overlapping relationships, such as in layered cloth, can be pre-determined, problems do not occur. Issues arise in situations like crumpling cloth, where the cloth self-collides or lacks a consistent overlapping pattern. To address these problems, it is necessary to combine methods that geometrically determine the normal direction for collision response, such

Table 3: *Performance results. Times are measured in seconds. The items labeled GPU in the far-right column indicate that the projection solver for constraints in XPBD was executed using GPU parallel processing. Items without labels were executed on the CPU. 'Response' refers to the time for our method.*

	Number of		Contact Type %		Frame Time		Breakdown of Average Frame Time				
	Vert.	Tri.	Point	Edge	Avrg.	Max.	XPBD	Broad Phase	Detection	Response	
Four octopuses falling (Table 2)	11,628	23,240	21.65 %	78.35 %	1.400	1.651	5.709×10^{-1}	7.318×10^{-1}	9.414×10^{-4}	3.624×10^{-5}	
Self-intersecting (Figure 6)	170,353	340,698	14.16 %	85.84 %	1.703	3.239	5.877×10^{-2}	6.765×10^{-1}	6.717×10^{-3}	4.567×10^{-4}	GPU
Layered cloth and an octopus (Figure 7)	6,174	11,954	20.21 %	79.79 %	1.705	2.932	1.587×10^{-1}	1.382	1.186×10^{-3}	2.028×10^{-4}	
Garment (Figure 8)	40,930	81,340	21.06 %	78.94 %	2.144	2.213	1.877×10^{-1}	3.803×10^{-2}	3.453×10^{-2}	1.688×10^{-3}	
Lots of octopuses falling (Figure 9)	104,652	209,160	24.86 %	75.14 %	$3.521\times10^{+1}$	$9.791\times10^{+1}$	4.839	$2.069\times10^{+1}$	1.481×10^{-1}	4.697×10^{-2}	GPU
Sharp contact (Figure 10)	86	134	31.94 %	68.06 %	5.575×10^{-3}	6.362×10^{-3}	4.185×10^{-4}	3.716×10^{-3}	3.300×10^{-6}	7.700×10^{-6}	
Triangle contact (Figure 11)	21,590	43,160	19.89 %	80.11 %	1.743	3.813	8.118×10^{-1}	3.712×10^{-2}	1.438×10^{-2}	3.730×10^{-5}	

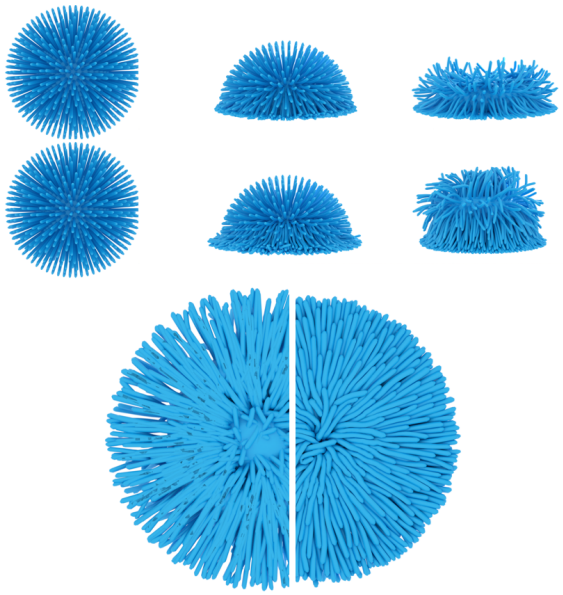

Figure 6: *Example of complex self-collisions. Without self-collision handling (top row, bottom left) and with self-collision handling (middle row, bottom right).*

as the approach proposed by Zhao et al. [ZYL13]. Alternatively, if a simulation begins with an intersection-free state, the sign of the normal direction can be determined by referring to the relationship between the two triangles when they are not intersecting.

5. Conclusion

We presented a novel formulation for handling collisions based on triangle intersection relationships and demonstrated its use as a unified collision handling solution for various simulation scenarios. Notably, we succeeded in generating contact points that do not miss edge-edge collisions and benefited from primitive-based self-collision handling, integrating them into the same framework. Moreover, as long as the normal direction for correction can be defined, the method resolves collisions quickly for both volumetric objects and thin objects like cloth.

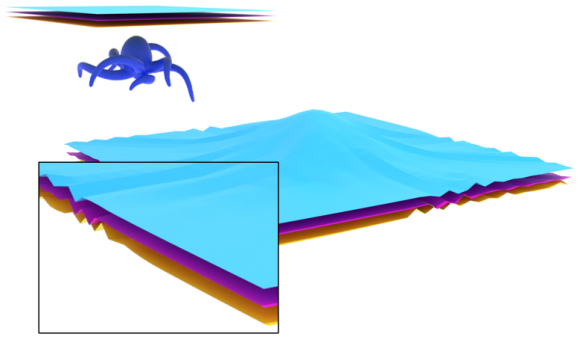

Figure 7: *Collision handling between layered cloths. Initial state and 120th frame (zoomed in). Collisions between thin objects are handled within the same framework used for volumetric object collisions.*

Figure 8: *Collision handling between layered cloths. The left image shows the first frame, while the right shows the 6th frame, where all intersections have been resolved.*

References

[AEF22] Andrews, Sheldon, Erleben, Kenny, and Ferguson, Zachary. "Contact and Friction Simulation for Computer Graphics". *ACM SIGGRAPH 2022 Courses*. ACM, New York, 2022. doi: 10.1145/3532720.3535640. 2.

[AFC*10] Allard, Jérémie, Faure, François, Courtecuisse, Hadrien, et al. "Volume Contact Constraints at Arbitrary Resolution". *ACM Trans. Graph.* 29.4 (July 2010). doi: 10.1145/1778765.1778819. 2.

Figure 9: *108 deformable objects are dropped. Left: 50th frame; Right: 120th frame. Walls are placed around the scene.*

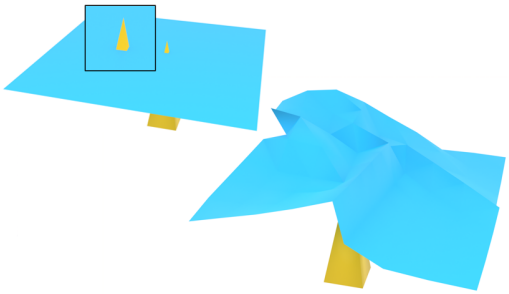

Figure 10: *Process of resolving an initial state where penetrations occur with sharp objects. Even with low-resolution cloth meshes, sharp vertices are accurately detected.*

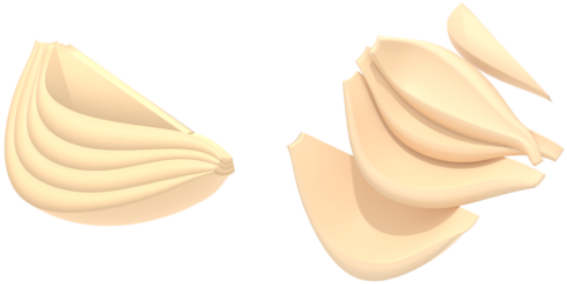

Figure 11: *It is also possible to separate layered objects, even if multiple triangle-triangle contacts have already occurred in the initial state before the simulation begins.*

[BEB12] Brochu, Tyson, Edwards, Essex, and Bridson, Robert. "Efficient Geometrically Exact Continuous Collision Detection". *ACM Trans. Graph.* 31.4 (July 2012). DOI: 10.1145/2185520.2185592. 2.

[BFA02] Bridson, Robert, Fedkiw, Ronald, and Anderson, John. "Robust Treatment of Collisions, Contact and Friction for Cloth Animation". *ACM Trans. Graph.* 21.3 (July 2002), 594–603. DOI: 10.1145/566654.566623. 2.

[BRB*19] Buffet, Thomas, Rohmer, Damien, Barthe, Loïc, et al. "Implicit Untangling: A Robust Solution for Modeling Layered Clothing". *ACM Trans. Graph.* 38.4 (July 2019). DOI: 10.1145/3306346.3323010. 2.

[CDY23] Chen, He, Diaz, Elie, and Yuksel, Cem. "Shortest Path to Boundary for Self-Intersecting Meshes". *ACM Trans. Graph.* 42.4 (July 2023). DOI: 10.1145/3592136. 2, 4, 5.

[Erl18] Erleben, Kenny. "Methodology for Assessing Mesh-Based Contact Point Methods". *ACM Trans. Graph.* 37.3 (July 2018). DOI: 10.1145/3096239. 2, 5.

[FL01] Fisher, Susan and Lin, Ming C. "Deformed Distance Fields for Simulation of Non-Penetrating Flexible Bodies". *Proc. 12th Eurographic Workshop on Computer Animation and Simulation*. Springer-Verlag, 2001, 99–111. DOI: 10.1007/978-3-7091-6240-8_10 2.

[FSG03] Fuhrmann, Arnulph, Sobotka, Gerrit, and Gross, Clemens. "Distance Fields for Rapid Collision Detection in Physically Based Modeling". *International Conference on Computer Graphics and Vision*. 2003 2.

[GBF03] Guendelman, Eran, Bridson, Robert, and Fedkiw, Ronald. "Nonconvex Rigid Bodies with Stacking". *ACM Trans. Graph.* 22.3 (July 2003), 871–878. DOI: 10.1145/882262.882358. 2.

[HFS*01] Hirota, Gentaro, Fisher, Susan, State, Andrei, et al. "An Implicit Finite Element Method for Elastic Solids in Contact". *Proc. 14th Conference on Computer Animation*. 2001, 136–254. DOI: 10.1109/CA.2001.982387 2.

[HTK*04] Heidelberger, Bruno, Teschner, Matthias, Keiser, Richard, et al. "Consistent Penetration Depth Estimation for Deformable Collision Response". *Proc. 9th International Workshop on Vision, Modeling, and Visualization*. Vol. 4. 2004, 339–346 2.

[LFS*20] Li, Minchen, Ferguson, Zachary, Schneider, Teseo, et al. "Incremental Potential Contact: Intersection- and Inversion-free, Large-Deformation Dynamics". *ACM Trans. Graph.* 39.4 (Aug. 2020). DOI: 10.1145/3386569.3392425. 1.

[MASS15] Mitchell, Nathan, Aanjaneya, Mridul, Setaluri, Rajsekhar, and Sifakis, Eftychios. "Non-manifold Level Sets: A Multivalued Implicit Surface Representation with Applications to Self-collision Processing". *ACM Trans. Graph.* 34.6 (Nov. 2015). DOI: 10.1145/2816795.2818100. 2.

[MEM*20] Macklin, Miles, Erleben, Kenny, Müller, Matthias, et al. "Local Optimization for Robust Signed Distance Field Collision". *Proc. ACM Comput. Graph. Interact. Tech.* 3.1 (May 2020). DOI: 10.1145/3384538. 2.

[MHHR07] Müller, Matthias, Heidelberger, Bruno, Hennix, Marcus, and Ratcliff, John. "Position Based Dynamics". *Journal of Visual Communication and Image Representation* 18.2 (2007), 109–118. DOI: 10.1016/j.jvcir.2007.01.005 4.

[MMC16] Macklin, Miles, Müller, Matthias, and Chentanez, Nuttapong. "XPBD: Position-Based Simulation of Compliant Constrained Dynamics". *Proc. 9th International Conference on Motion in Games*. 2016, 49–54. DOI: 10.1145/2994258.2994272 4.

[Möl97] Möller, Tomas. "A Fast Triangle-Triangle Intersection Test". *Journal of Graphics Tools* 2.2 (1997), 25–30. DOI: 10.1080/10867651.1997.10487472. 3, 4.

[MSL*19] Macklin, Miles, Storey, Kier, Lu, Michelle, et al. "Small Steps in Physics Simulation". *Proc. 18th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, New York, 2019. DOI: 10.1145/3309486.3340247. 6.

[MZS*11] McAdams, Aleka, Zhu, Yongning, Selle, Andrew, et al. "Efficient Elasticity for Character Skinning with Contact and Collisions". *ACM Trans. Graph.* 30.4 (July 2011). DOI: 10.1145/2010324.1964932. 2.

[NZXL20] Nie, Quan, Zhao, Yingfeng, Xu, Li, and Li, Bin. "A Survey of Continuous Collision Detection". *Proc. 2nd International Conference on Information Technology and Computer Application*. 2020, 252–257. DOI: 10.1109/ITCA52113.2020.00061 2.

[SOTC22] Santesteban, Igor, Otaduy, Miguel, Thuerey, Nils, and Casas, Dan. "ULNeF: Untangled Layered Neural Fields for Mix-and-Match Virtual Try-On". *Advances in Neural Information Processing Systems*. Vol. 35. Curran Associates, Inc., 2022, 12110–12125. 2.

[SSIF09] Selle, Andrew, Su, Jonathan, Irving, Geoffrey, and Fedkiw, Ronald. "Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction". *IEEE Transactions on Visualization and Computer Graphics* 15.2 (Mar. 2009), 339–350. DOI: 10.1109/TVCG.2008.79. 2.

[ST05] Spillmann, Jonas and Teschner, Matthias. "Contact Surface Computation for Coarsely Sampled Deformable Objects". *Proc. 10th International Workshop on Vision, Modeling, Visualization*. 2005, 289–296 2.

[Wan14] Wang, Huamin. "Defending Continuous Collision Detection against Errors". *ACM Trans. Graph.* 33.4 (July 2014). DOI: 10.1145/2601097.2601114. 2.

[WC21] Wang, Monan and Cao, Jiaqi. "A Review of Collision Detection for Deformable Objects". *Computer Animation and Virtual Worlds* 32.5 (2021). e1987 CAVW-19-0082.R2, e1987. DOI: https://doi.org/10.1002/cav.1987. 2.

[WCL*23] Wang, Tianyu, Chen, Jiong, Li, Dongping, et al. "Fast GPU-based Two-way Continuous Collision Handling". *ACM Trans. Graph.* 42.5 (July 2023). DOI: 10.1145/3604551. 2.

[WFP12] Wang, Bin, Faure, François, and Pai, Dinesh K. "Adaptive Image-based Intersection Volume". *ACM Trans. Graph.* 31.4 (July 2012). DOI: 10.1145/2185520.2185593. 2.

[XB17] Xu, Hongyi and Barbič, Jernej. "6-DoF Haptic Rendering Using Continuous Collision Detection between Points and Signed Distance Fields". *IEEE Trans. Haptics* 10.2 (Apr. 2017), 151–161. DOI: 10.1109/TOH.2016.2613872. 2.

[XMCX22] Xiao, Lei, Mei, Gang, Cuomo, Salvatore, and Xu, Nengxiong. "Comparative Investigation of GPU-Accelerated Triangle-Triangle Intersection Algorithms for Collision Detection". *Multimedia Tools and Applications* 81.3 (2022), 3165–3180. DOI: 10.1007/s11042-020-09066-3. 3.

[YMJ*17] Ye, Juntao, Ma, Guanghui, Jiang, Liguo, et al. "A Unified Cloth Untangling Framework Through Discrete Collision Detection". *Computer Graphics Forum* 36.7 (2017), 217–228. DOI: 10.1111/cgf.13287 2.

[ZMSL23] Zesch, Ryan S., Modi, Vismay, Sueda, Shinjiro, and Levin, David I.W. "Neural Collision Fields for Triangle Primitives". *SIGGRAPH Asia 2023 Conference Papers*. ACM, New York, 2023. DOI: 10.1145/3610548.3618225. 2, 4, 5.

[ZYL13] Zhao, Jing, Ye, Juntao, and Li, Jituo. "Resolving Cloth Penetrations with Discrete Collision Detection". *Proc. International Conference on Computer-Aided Design and Computer Graphics*. 2013, 443–444. DOI: 10.1109/CADGraphics.2013.88 7.