Real-Time Triangle-SDF Continuous Collision Detection

JOËL PELLETIER-GUÉNETTE, École de Technologie Supérieure, Canada
ALEXANDRE MERCIER-AUBIN, École de Technologie Supérieure, Université de Sherbrooke, Canada
SHELDON ANDREWS, École de Technologie Supérieure, Canada

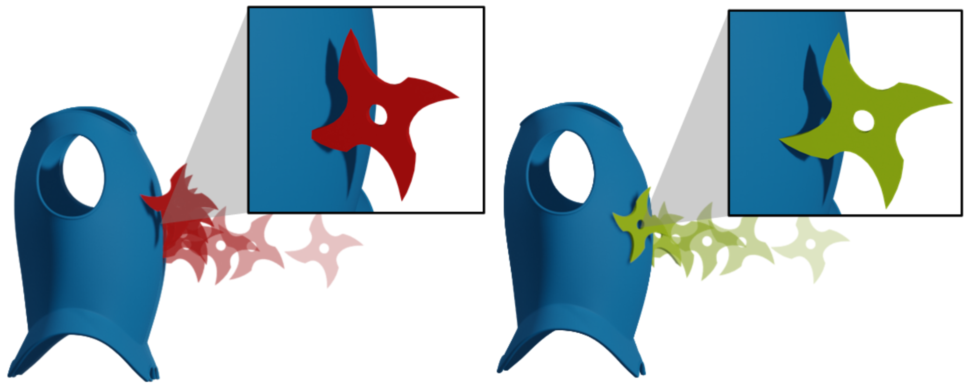

Fig. 1. A fast-flying shuriken moving at 500 m/s hits an armor breastplate. The shuriken is represented by a triangle mesh, and the armor by an implicit function. Left: Discrete collision detection allows penetration into the armor, resulting in a different trajectory after collision response. Right: Our method gives an interpenetration-free simulation.

We introduce an efficient solution to the problem of continuous collision detection (CCD) between triangle geometry and signed distance fields (SDFs). We formulate the triangle-SDF collision problem as a novel spatio-temporal local optimization that solves for the first time of impact between a triangle and an SDF isosurface. Our method offers improved robustness over point sampling methods, and outperforms recent triangle-SDF discrete collision detection (DCD) algorithms. Furthermore, a novel method for adaptively refining the potential collision points on large triangles is proposed for robust triangle-SDF collision detection with coarse meshes. This enables the use of reduced geometry for efficient simulations. We demonstrate the benefits of our approach by comparing to state-of-the-art algorithms for triangle-SDF collision detection, and showcase its effectiveness through simulations involving complex collision scenarios.

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

Additional Key Words and Phrases: Collisions, Real-time, Physics-based, Signed Distance Field,

ACM Reference Format:
Joël Pelletier-Guénette, Alexandre Mercier-Aubin, and Sheldon Andrews. 2025. Real-Time Triangle-SDF Continuous Collision Detection. *Proc. ACM Comput. Graph. Interact. Tech.* 8, 4, Article 49 (August 2025), 22 pages. https://doi.org/10.1145/3747862

Authors' Contact Information: Joël Pelletier-Guénette, joelpg@hotmail.ca, École de Technologie Supérieure, Montréal, Québec, Canada; Alexandre Mercier-Aubin, alexandremercieraubin@gmail.com, École de Technologie Supérieure, Montréal, and Université de Sherbrooke, Sherbrooke, Québec, Canada; Sheldon Andrews, sheldon.andrews@etsmtl.ca, École de Technologie Supérieure, Montréal, Québec, Canada.

1 Introduction

Collision detection is the computational process of determining whether, where, and when objects in a simulated environment are intersecting. It is a key technology for numerous applications in computer graphics, robotics, and virtual reality. Continuous Collision Detection (CCD) is particularly interesting as it permits computing intersection-free trajectories of the simulated objects, whereas discrete methods may miss collisions or require shape separation [Nie et al. 2020].

The accuracy, complexity and performance of collision detection methods are largely dependent on shape representation. For instance, triangle meshes are popular shape representations in computer graphics — due to their ability to model complex geometry — but intersection tests using triangle meshes are expensive, especially as higher resolutions are required to capture fine details. Simulation performance can be drastically improved with simpler shape representations. For example, efficient methods exist for distance, overlap and closest-point queries between spheres, oriented or axis-aligned boxes, and convex hulls [Ericson 2004]. While such primitives cannot accurately represent most objects, they are invaluable when real-time performance is required. They can provide efficient approximate bounding volumes for use in broad-phase collision detection, drastically reducing the number of computationally expensive tests required by more accurate narrow-phase collision detection. Likewise, Signed Distance Fields (SDFs) provide efficient distance and inside-outside information, making them powerful tools for shape approximation in collision detection. They can be stored compactly as analytical functions [Andrews et al. 2022], discrete grids [Koschier et al. 2017a], or even deep neural networks [Park et al. 2019]. They excel at representing smooth or curved surfaces, and can efficiently represent complex geometry [Koschier et al. 2017a]. Common drawbacks of SDFs, however, are that sharp features may be lost due to the smooth nature of implicit surfaces, and they are not well suited to represent non-manifold or self-intersecting surfaces where space cannot be properly partitioned into a well-defined interior, exterior, and surface [Jones et al. 2006]. Triangle meshes, on the other hand, are unaffected by these issues. It is thus of particular interest to combine shape representations and leverage their respective advantages.

Discrete collision detection involving meshes and primitives is well explored already [Andrews et al. 2022; Ericson 2004; Teschner et al. 2005]. Methods involving meshes and SDFs have traditionally been limited to point-based contacts over the mesh vertices or require dense point sampling [McNeely et al. 2005; Xu and Barbič 2017]. The recently proposed method by Macklin et al. [2020] specifically addresses the problem of discrete collision detection between SDFs and triangle meshes. They improve the quality of contact generation compared to point-based approaches, but they only support a single contact point per triangle. Larger triangles may intersect multiple features of an SDF, and thus dense and high-resolution meshes are often required for reliable collision detection. However, low-resolution coarse meshes are preferred by many applications for their efficiency. The existing triangle-SDF methods are further limited to intersection tests performed at discrete points in time. This can result in shapes with a high relative velocity passing through each other (tunnelling). Similar problems occur for SDFs with thin features, where it is easy for shapes to pass through regions where the SDF gradient is undefined or changes direction. This can lead to inaccurate or unstable simulations. Triangle-SDF CCD is, to the best of our knowledge, an unexplored problem.

We address the aforementioned problems by proposing a novel triangle-SDF collision detection algorithm that reformulates the collision detection problem as a local spatio-temporal optimization. Solving this optimization problem provides not only the time of impact, but also the closest points between triangles and an SDF isosurface. Our CCD algorithm avoids tunnelling and offers improved robustness over point sampling methods; it also outperforms recent triangle-SDF discrete collision detection (DCD) algorithms. Our technique does not assume a specific contact model (constraint-based, barrier methods, penalty forces, impulse-based) and can be integrated into existing simulators. Our technique can be used alongside other methods to handle different collision shapes; it does not limit the simulation to triangles and SDFs. Additionally, we introduce a technique to efficiently compute multiple contact points per triangle to support coarse triangle meshes without globally refining the whole mesh. Rather, new contact points are generated only when needed. We evaluate our method for speed, accuracy, and robustness, using a series of hand-crafted extreme collision scenarios involving triangles and SDFs. We demonstrate our approach using several complex rigid body and cloth simulations, with meshes up to hundreds of thousands of triangles and SDFs hand-picked to challenge gradient methods with gradient discontinuities, sharp features, and multiple boundaries along triangle trajectories. Our algorithm is suitable for real-time applications, and interactive speeds are easily achieved, even on a single-threaded CPU-based implementation.

2 Related Work

Collision detection is a key technology for many computer graphics applications, including rendering, animation, 3D modeling, and computational design. It is the process of determining if two or more geometric shapes are intersecting or will intersect. In this section, we provide a brief overview of seminal work in computer graphics on collision detection, with a particular emphasis on the CCD regime of methods, as well as collision algorithms involving geometry represented as an SDF.

2.1 Collision detection

Previous works in computer graphics have provided overviews of the fundamentals of collision detection and response for physics-based animation [Andrews et al. 2022; Ericson 2004; Teschner et al. 2005]. However, numerous algorithms have been developed by the graphics community over the years to address collision detection for specific phenomena and geometry pairs. For instance, Fuhrmann et al. [2003] developed a method for rapid collision detection between rigid and highly deformable objects using distance fields, and Zhang et al. [2023a] use Sum-of-Squares programming to handle higher-order geometry. Alternatively, Guendelman et al. [2003] uses interference detection through testing vertices to the inside/outside function of another model and time step limitation.

Computational complexity and performance are often primary concerns where collision detection is concerned, since the process is often viewed as a bottleneck in the simulation pipeline. Acceleration data structures such as spatial hash grids [Quinlan 1994], BSP trees [Naylor 1998], and bounding volume hierarchies (BVH) [Ericson 2004] are common since they effectively prune the search space of narrow-phase interaction tests, which can be costly. Such acceleration techniques are compatible with our proposed methodology, and we show how a BVH of spheres can be used to effectively reduce the number of triangle-SDF collision tests.

Animations involving fast-moving objects or thin geometry can produce tunnelling and severe penetration artefacts, due mainly to the discrete collision detection used by many applications. Continuous collision detection (CCD) avoids these artifacts by identifying intersections along the entire path of motion, resulting in specialized algorithms for simulation of rigid bodies [Redon et al. 2002], deformable solids [Govindaraju et al. 2006], and cloth [Bridson et al. 2002]. CCD is also an essential component of robust contact simulation with non-interpenetration guarantees, such as in incremental potential contact (IPC) [Li et al. 2020]. Xu and Barbič [2017] developed a continuous algorithm for collisions between point clouds and implicit geometry representations, such as distance functions. Such methods can often use properties like Bernstein basis or Bézier curves to reduce the problem [Tang et al. 2014]. This is useful to model the continuous nature of curved regions. Alternatively, some approaches assume linear motions to reduce computational complexity [Chen et al. 2024; Wang et al. 2022, 2021]. This assumption can significantly streamline calculations, making it easier to predict and manage object trajectories. Similarly, it is possible to use interval calculations to do SDF-SDF collision detection [Liu et al. 2024] and geometrically exact CCD [Brochu et al. 2012], though at prohibitive costs for real-time CCD for multiple contact points queries.

Our work focuses on collision detection combining an explicit mesh-based representation of geometry with an implicit SDF. It can be integrated as part of a typical physical simulation pipeline and does not prevent the use of other collision detection methods to handle different collision shapes. Our proposed acceleration structure relies on this flexibility. It uses less accurate shapes for which vastly more efficient SDF CCD methods can provide early approximations and collision pair culling. Haptic simulation has similarly focused on using similar hybrid representations, with collision between a point cloud and a voxelized distance field receiving particular attention [McNeely et al. 1999, 2005]. Robustness is improved by continuous collision detection versions of these algorithms [Xu and Barbič 2017].

To the best of our knowledge, our method is unique in addressing CCD between triangles and arbitrary SDFs. IPC implementations use triangle-triangle representations [Li et al. 2020], and related methods e.g. [Xu and Barbič 2017], require dense point sampling or specific SDF discretizations.

2.2 Geometry representation

The efficiency of SDF-based algorithms is directly impacted by the efficiency of their underlying data structures and shape representations. Numerous adaptive methods have been proposed to reduce the storage overhead of sampled signed distance fields [Frisken et al. 2000]. Notably, Koschier et al. [2017a] proposed a highly accurate grid-based SDF by fitting a polynomial approximation of the local shape to each grid cell, and we use this representation for several of our experiments. Deep neural models have also demonstrated effectiveness for learning implicit shape representations [Park et al. 2019]. SDF-based swept volumes have proven to be effective in computing collision-free trajectories for animation [Wang et al. 2024] and robotics planning [Zhang et al. 2023b], albeit at computational costs that prohibit real-time applications. Our method treats the SDF as a “black box”, so it is not tied to its representation. Thus, future development of efficient shape representations will remain compatible.

3 Preliminaries

We first offer a brief overview of the concepts used in our paper and the technique introduced in [Macklin et al. 2020] to make our paper self-contained. We note that our technique does not require a specific contact model (constraint-based, barrier methods, penalty forces, impulse-based). Contact handling and the resolution of subsequent potential contacts are out of scope for our paper. For this, we refer the reader to related work on the topic [Andrews et al. 2022]. While our work focuses on collision detection between triangles and SDFs, it does not prevent other methods from handling different collision shapes in the same physics pipeline. Our proposed acceleration structure, presented in Section 4.6, relies on this flexibility. It uses less accurate shapes for which vastly more efficient SDF CCD methods can provide early approximations and collision pair culling. Other efficient collision detection algorithms involving common primitives, and a primer on using multiple specialized collision detection algorithms together can be found in [Ericson 2004].

3.1 Signed Distance Fields

A signed distance field is a function $\phi(\vec{\mathbf{x}})\colon \mathbb{R}^3 \longrightarrow \mathbb{R}$ that gives the signed Euclidean distance from a point $\vec{\mathbf{x}} \in \mathbb{R}^3$ to a shape boundary, where $\phi = 0$, and with $\phi(\vec{\mathbf{x}}) < 0$ for all points inside the shape. Its gradient $\boldsymbol{\nabla}\phi \in \mathbb{R}^3$ satisfies $\|\boldsymbol{\nabla}\phi\| = 1$ and gives the direction in which distance increases the most, so the closest point from $\vec{\mathbf{x}}$ on the boundary is $\vec{\mathbf{x}} - \boldsymbol{\nabla}\phi(\vec{\mathbf{x}})\phi(\vec{\mathbf{x}})$. In the case of physics-based animation, the SDF gradient conveniently provides a contact normal $\vec{\mathbf{n}} = \boldsymbol{\nabla}\phi$, which can otherwise be challenging to compute in triangle-triangle collision scenarios [Erleben 2018].

Certain types of SDFs have limitations. When the SDF uses a discrete representation, its domain may be limited, e.g., to the region covered by a grid. Furthermore, an SDF is not differentiable at points along the medial axis of a shape, resulting in discontinuities of the gradient. However, we find that such issues rarely occur in practice or can be easily circumvented. The impact of SDF quality on our method is further discussed in Section 6.

3.2 Triangle-SDF Discrete Collision Detection

A technique for discrete collision detection (DCD) between triangle meshes and SDFs was introduced in Macklin et al. [2020] using the *closest point methodology* [Erleben 2018]. Collision points are generated by a minimization of the SDF over the barycentric coordinates u, v, w of a triangle with vertices $\vec{\mathbf{a}},\ \vec{\mathbf{b}},\ \vec{\mathbf{c}} \in \mathbb{R}^3$:

$$\min_{u,v,w} \quad \phi(\, u\vec{\mathbf{a}} + v\vec{\mathbf{b}} + w\vec{\mathbf{c}}\,) \tag{1}$$

$$\text{s.t.} \quad u, v, w \geq 0 \tag{2}$$

$$u + v + w = 1\,. \tag{3}$$

The resulting barycentric coordinates give the deepest penetrating point inside the triangle. Gradient methods, such as *projected gradient descent* or the *Frank-Wolfe algorithm* [Frank and Wolfe 1956], can solve for local minima by using the derivative of the SDF with respect to the barycentric coordinates:

$$\vec{\mathbf{d}} = \left[\frac{\partial\phi}{\partial u}, \frac{\partial\phi}{\partial v}, \frac{\partial\phi}{\partial w} \right]^{\mathsf{T}} = \begin{bmatrix} \boldsymbol{\nabla}\phi(\vec{\mathbf{x}}) \cdot \vec{\mathbf{a}} \\ \boldsymbol{\nabla}\phi(\vec{\mathbf{x}}) \cdot \vec{\mathbf{b}} \\ \boldsymbol{\nabla}\phi(\vec{\mathbf{x}}) \cdot \vec{\mathbf{c}} \end{bmatrix}. \tag{4}$$

This approach improves the quality of contact generation compared to point-based methods, but only supports a single contact point per triangle [Macklin et al. 2020]. Larger triangles may intersect multiple features of an SDF, and thus dense and high-resolution meshes are often required for reliable collision detection. Mesh subdivision can be applied ahead of time to provide better coverage, but it increases the cost of the optimization accordingly. We later propose an efficient method to solve these issues (see Section 4.5).

Since DCD only provides contact information if a contact is occurring at a given instant in time, it cannot effectively prevent penetration. Larger time steps, fast movements, or thin SDFs can cause discrete collision techniques to miss contacts. For instance, SDFs may fail to provide a realistic separation direction for sufficiently deep points, or if they are inside sharp features. This results in shapes being forced to penetrate or passing through one another instead of colliding as expected. In such cases, continuous methods for collision detection are preferred.

3.3 Continuous Collision Detection

Continuous collision detection (CCD) is generally applied in order to provide accurate collision detection in simulations where discrete time steps may cause fast-moving or thin objects to pass through each other or interpenetrate in a way such that collisions cannot be handled accurately or

Algorithm 1: Golden Section Search algorithm for minimizing function $f(l)$ over the interval $l_{\text{start}}\ l_{\text{end}}$. Returns the minimum value $l_{\min}$.

```
GSSMinimize($l_{\text{start}}, l_{\text{end}}, f$)
    $r \leftarrow \varphi^{-1}, \quad r^{-1} \leftarrow 1 - r$
    $\alpha_0 \leftarrow 0, \quad \alpha_1 \leftarrow r^{-1}, \quad \alpha_2 \leftarrow r, \quad \alpha_3 \leftarrow 1$
    $l_0 \leftarrow l_{\text{start}}, \quad l_1 \leftarrow \text{lerp}(l_{\text{start}}, l_{\text{end}}, \alpha_1), \quad l_2 \leftarrow \text{lerp}(l_{\text{start}}, l_{\text{end}}, \alpha_2), \quad l_3 \leftarrow l_{\text{end}}$
    $f_0 \leftarrow f(l_0), \quad f_1 \leftarrow f(l_1), \quad f_2 \leftarrow f(l_2), \quad f_3 \leftarrow f(l_3)$
    while $(l_3 - l_0) \leq \text{tol}\,(l_1 + l_2)$ do
        if $\min(f_0, f_1) < \min(f_2, f_3)$ then
            $\alpha_3 \leftarrow \alpha_2, \quad l_3 \leftarrow l_2, \quad f_3 \leftarrow f_2$
            $\alpha_2 \leftarrow \alpha_1, \quad l_2 \leftarrow l_1, \quad f_2 \leftarrow f_1$
            $\alpha_1 \leftarrow r\,\alpha_2 + r^{-1}\,\alpha_0$
            $l_1 \leftarrow \text{lerp}(l_{\text{start}}, l_{\text{end}}, \alpha_1)$
            $f_1 \leftarrow f(l_1)$
        else
            $\alpha_0 \leftarrow \alpha_1, \quad l_0 \leftarrow l_1, \quad f_0 \leftarrow f_1$
            $\alpha_1 \leftarrow \alpha_2, \quad l_1 \leftarrow l_2, \quad f_1 \leftarrow f_2$
            $\alpha_2 \leftarrow r\,\alpha_1 + r^{-1}\,\alpha_3$
            $l_2 \leftarrow \text{lerp}(l_{\text{start}}, l_{\text{end}}, \alpha_2)$
            $f_2 \leftarrow f(l_2)$
        end if
    end while
    $l_{\text{mid}} \leftarrow \text{lerp}\big(l_{\text{start}}, l_{\text{end}}, \frac{1}{2}(\alpha_0 + \alpha_3)\big)$
    $f_{\text{mid}} \leftarrow f(l_{\text{mid}})$
    if $f_0 < f_{mid}$ and $f_0 < f_3$ then
        $l_{\min} \leftarrow l_0$
    else if $f_{mid} < f_3$ then
        $l_{\min} \leftarrow l_{\text{mid}}$
    else
        $l_{\min} \leftarrow l_3$
    end if
    return $l_{\min}$
```

in a stable fashion. CCD gives the time of the first impact between objects in a given time interval $t_{\text{hit}} \in [t_{\text{start}}, t_{\text{end}}]$, and guarantees that no collision occurs up to t_{hit}.

3.4 Golden Section Search

Golden section search (GSS) is an iterative minimization technique discovered by Kiefer [1953], and is well-suited to local minimization problems. We use it to solve single-dimension sub-problems as part of our method in Section 4.3. Although GSS is a well-known algorithm and many public domain implementations are available, we provide algorithmic details in Algorithm 1 for completeness. The method seeks the minimizer for a function $f(l)$ over some interval, such that

$$l_{\min} = \underset{l \in [l_{\text{start}}, l_{\text{end}}]}{\arg\min}\ f(l)\,. \tag{5}$$

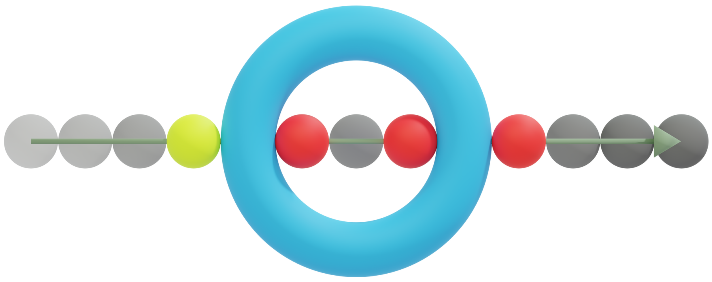

Fig. 2. A sphere moving rightward passes through the boundary of an SDF isosurface (torus) multiple times. Local minima of the spatio-temporal problem are encountered each time the sphere touches the SDF. CCD requires solving for a global minimum with regard to time (green), but gradient methods may converge toward erroneous local minima (red).

The algorithm maintains function values f_0, f_1, f_2, and f_4, for four points l_0, l_1, l_2, and l_4, respectively. The points form three contiguous intervals with a width ratio of $\varphi : 1 : \varphi$, where φ is the golden ratio:

$$\varphi = (1 + \sqrt{5})/2\,, \tag{6}$$

$$\varphi^{-1} = (\sqrt{5} - 1)/2\,. \tag{7}$$

The algorithm operates by successively narrowing the range over which f is evaluated. Note that the points l_i can be of any dimensions. In the context of CCD, we use them both as points in time and as points in space, depending on the specific line search problem. Points are obtained by linear interpolation over the interval $[l_{\text{start}}, l_{\text{end}}]$. The complement φ^{-1} to the golden ratio is used as part of the interval refinement process. Each iteration, we compare the minimum of the left values, i.e., $\min(f_0, f_1)$, to the minimum of the right values, i.e., $\min(f_2, f_3)$. If the left values are smaller, the boundaries of the two leftmost intervals are used to form a new set of three intervals with the same width ratio; the boundaries of the two rightmost intervals are used otherwise. Only a single new point and function value need to be computed when the interval is reduced. One point stays the same, and the other two simply reuse previously computed points. Once the algorithm terminates, the point with the smallest function evaluation from l_0, l_3, and midpoint l_{mid} is returned.

4 Triangle-SDF CCD

In this section, we present an approach that extends triangle-SDF collision to the continuous regime. Previous work on triangle-SDF collision by Macklin et al. [2020] uses iterative gradient-based methods to solve the triangle-SDF discrete collision detection problem. Likewise, we are interested in Frank–Wolfe methods, which are iterative first-order optimization algorithms for constrained convex optimization [Braun et al. 2022; Frank and Wolfe 1956; Jaggi 2013]. However, the landscape of CCD optimization is potentially non-convex, and this can result in tunnelling, as shown in Figure 2. Hence, one of our central contributions is a novel Frank-Wolfe-based algorithm that identifies and converges toward a spatio-temporal global minimum.

We focus on FW algorithms since preliminary experiments revealed that Projected Gradient Descent (PGD) often fails to converge due to a constant step size. Furthermore, backtracking variations were costly due to the increased number of projections back to the problem space required by PGD, whereas FW algorithms require none (as noted by Macklin et al. [2020]).

4.1 Local Optimization

The goal of the CCD optimization is to provide the spatial coordinates and time of the earliest contact over some time interval $t_{\text{hit}} \in [t_{\text{start}}, t_{\text{end}}]$, such that no collision occurs before t_{hit}. We seek, for each triangle, a point of contact $\vec{\mathbf{x}}_c$ at some future time t where $\phi(\vec{\mathbf{x}}_c) = 0$. Since multiple instances of the the time interval can yield $\phi(\vec{\mathbf{x}}_t) = 0$, solving for the unsigned distance $|\phi(\vec{\mathbf{x}}_t)|$ could converge on the wrong side of the SDF (as shown in Figure 2). Therefore, we cannot directly solve for t_{hit} using the unsigned distance $|\phi(\vec{\mathbf{x}}_t)|$ if we intend to find a temporal global minimum. Hence, we minimize the signed distance over the time variable t and the barycentric coordinates u, v, w of a triangle with vertices $\vec{\mathbf{a}}_t,\ \vec{\mathbf{b}}_t,\ \vec{\mathbf{c}}_t \in \mathbb{R}^3$:

$$\min_{u,v,w,t} \quad \phi(\, u\vec{\mathbf{a}}_t + v\vec{\mathbf{b}}_t + w\vec{\mathbf{c}}_t \,)\,, \tag{8}$$

$$\text{s.t.} \quad \phi(\, u\vec{\mathbf{a}}_t + v\vec{\mathbf{b}}_t + w\vec{\mathbf{c}}_t \,) \geq 0\,, \tag{9}$$

$$u, v, w \geq 0\,, \tag{10}$$

$$u + v + w = 1\,, \tag{11}$$

$$t \in [t_{\text{start}}, t_{\text{end}}]\ . \tag{12}$$

Here, $\vec{\mathbf{x}}_t = u\vec{\mathbf{a}}_t + v\vec{\mathbf{b}}_t + w\vec{\mathbf{c}}_t$ and the subscript $\cdot_t$ indicates that the vertex positions change over the time interval, and that we seek the global minimum for t. If no t is found where the unsigned distance is zero, then a collision does not occur over the time interval. Otherwise, the simulation cannot advance past the time of collision and remain intersection-free. Violations of the constraint in Equation 9 provide new upper bounds to the time interval; see Section 4.2 for more details.

We revise the derivative from Equation 4 to account for the added time dimension:

$$\vec{\mathbf{d}} = \begin{bmatrix} \nabla\phi(\vec{\mathbf{x}}_t) \cdot \vec{\mathbf{a}}_t \\ \nabla\phi(\vec{\mathbf{x}}_t) \cdot \vec{\mathbf{b}}_t \\ \nabla\phi(\vec{\mathbf{x}}_t) \cdot \vec{\mathbf{c}}_t \\ \nabla\phi(\vec{\mathbf{x}}_t) \cdot \vec{\mathbf{v}}_t \end{bmatrix}. \tag{13}$$

Here, $\vec{\mathbf{v}}_t$ is the linear velocity of the potential solution $\vec{\mathbf{x}}_t$ at time t. The velocity $\vec{\mathbf{v}}_t$ is obtained differently depending on the type of simulation or required accuracy. For instance, velocities can be extracted directly from the values stored at the nodal coordinates in cloth and elastic simulation. For standard rigid-body scenarios, we project the angular and linear velocities ($\vec{\omega}_g$ and $\vec{\mathbf{v}}_g$) from the rigid-body's centre of rotation $\vec{\mathbf{g}}$ to the surface point $\vec{\mathbf{x}}_t$, such that

$$\vec{\mathbf{v}}_t = \vec{\mathbf{v}}_g + \vec{\omega}_g \times (\vec{\mathbf{x}}_t - \vec{\mathbf{g}})\,. \tag{14}$$

We note that SDFs are queried in their own local space, as it is more efficient and much easier to transform points than it is to update SDFs at runtime [Andrews et al. 2022]. Thus, if the SDF is used as a collision shape for an object which also undergoes rigid motions, the position and velocities should be further projected into the SDF space.

4.2 Frank-Wolfe for CCD

Although the standard Frank-Wolfe algorithm cannot guarantee convergence toward a global minimum, it has excellent convergence properties when seeking a local minimum [Braun et al. 2022; Frank and Wolfe 1956; Jaggi 2013]. The algorithm thus provides an efficient approach to detect if a point violates the lower bound on the SDF value (see Equation 9) for the CCD problem. At any point, $\phi(\vec{\mathbf{x}}_t) \leq 0$ automatically guarantees that $t \geq t_{\text{hit}}$, which gives a new upper bound to the time minimization problem. The time interval is reduced to $t_{\text{hit}} \in [t_{\text{start}}, t]$, thus culling later temporal minima from the problem space. Unfortunately, the gradient at the end of the newly truncated time interval may cause the algorithm to get stuck trying to converge toward the same local minimum as before. We address this issue by splitting the direction-finding sub-problem of the Frank-Wolfe algorithm into two steps.

Our algorithm first proceeds much like Macklin et al. [2020], and at each iterate i seeks a spatial direction by selecting a support vertex from the triangle in its configuration at time t_i:

$$\min_{\vec{\mathbf{s}}_i} \quad \vec{\mathbf{s}}_i^{\top} \nabla\phi(\vec{\mathbf{x}}_{t_i}), \tag{15}$$

$$\text{s.t.} \quad \vec{\mathbf{s}}_i \in \left\{ \vec{\mathbf{a}}_{t_i}, \vec{\mathbf{b}}_{t_i}, \vec{\mathbf{c}}_{t_i} \right\}. \tag{16}$$

Then, we introduce a new temporal direction

$$d_i = \begin{cases} -1, & \text{if } \phi(\vec{\mathbf{x}}_{t_i}) \le 0 \\ -\operatorname{sign}(\nabla\phi(\vec{\mathbf{x}}_{t_i}) \cdot \vec{\mathbf{v}}_{t_i}), & \text{otherwise} \end{cases} \tag{17}$$

that moves $\vec{\mathbf{x}}_{t_i}$ toward the zero isosurface of the SDF when outside , and otherwise moves the solution backward in time. The solution is updated using the step $\alpha = 2/(i+2)$, standard to Frank-Wolfe methods, such that

$$\Delta t = \alpha(t_{\text{end}} - t_{\text{start}})\, d_i, \tag{18}$$

$$t_{i+1} \leftarrow \max(t_{\text{start}}, \min(t_{\text{end}}, t_i + \Delta t)), \tag{19}$$

$$\vec{\mathbf{x}}_{t_{i+1}} \leftarrow \operatorname{proj}\left(\vec{\mathbf{x}}_{t_i} + \alpha(\vec{\mathbf{s}}_i - \vec{\mathbf{x}}_{t_i})\right). \tag{20}$$

Note that in the last line, the position of the updated solution is projected into the triangle at t_{i+1} to obtain $\vec{\mathbf{x}}_{t_{i+1}}$. This can be achieved by barycentric interpolation or by applying the same transformation used to produce the triangle at t_{i+1}.

Splitting the direction-finding sub-problem this way forces the algorithm to ignore the temporal part of the gradient when $\phi(\vec{\mathbf{x}}_t) \le 0$. The spatial direction still follows the gradient, so it may find deeper points, but the temporal direction will be able to push them out of the shape by moving back in time to refine the current solution and avoid penetration at the local minimum. It also allows the algorithm to find earlier local minima, in which case the boundary can be adjusted again until the only remaining minimum is the one at the end of the adjusted time interval. The algorithm terminates by detecting when $t_{i+1} \approx t_i$ and $\vec{\mathbf{x}}_{t_{i+1}} \approx \vec{\mathbf{x}}_{t_i}$, up to a desired precision, or when a hard limit on the number of iterations is reached.

This temporally modified version of the Franke-Wolfe algorithm usually reduces the problem down to a simple convex problem with only a single global temporal minimum, yet offers no guarantees. We found multiple scenarios where this method fails, such as configurations involving repeated tunnelling over a single time step, as shown in Figure 2. We address these issues in the following section.

4.3 Frank-Wolfe with Line Search

The domain of the spatial minimization sub-problem changes as the modified Frank-Wolfe method described in Section 4.2 steps through time. The triangle at the new time point is effectively a different 3D slice of the four-dimensional spatio-temporal domain, which introduces discontinuities. In addition, sudden modifications of the time domain occur when an evaluated point violates the lower bound constraint $\phi(\vec{\mathbf{x}}_{t_i}) \ge 0$. This may cause multiple local minima to be pushed out of bound, and force the spatial minimization to keep seeking new ones. This is made difficult by the ever-decreasing step size $\alpha = 2/(i+2)$, as the step size can become too small to find other minima.

Algorithm 2: Frank-Wolfe with GSS

```
1  FWGSS($t_{\text{start}}, t_{\text{end}}$)
2      $t_1 \leftarrow t_{\text{start}}$
3      Compute the barycentric coordinates $u, v, w$ of the starting iterate.          ▷ Section 4.4
4      for $i \in 1 \ldots$ max iterations do
           // Solve temporal sub-problem
5          $\vec{\mathbf{x}}_{t_i} \leftarrow$ BarycentricInterpolate($u, v, w, t_i$)
6          Compute $\vec{\mathbf{v}}_{t_i}$                                              ▷ Equation 14
7          Query $\phi(\vec{\mathbf{x}}_{t_i})$ and $\nabla\phi(\vec{\mathbf{x}}_{t_i})$
8          if $\phi(\vec{\mathbf{x}}_{t_i}) \leq 0$ then
9              $t_{\text{end}} \leftarrow \min(t_i, t_{\text{end}})$                    // Update interval
10             def UnsignedDistanceAtTime($t$):
11                 $\vec{\mathbf{x}} \leftarrow$ BarycentricInterpolate($u, v, w, t$)
12                 return $|\phi(\vec{\mathbf{x}})|$
13             $t_{i+1} \leftarrow$ GSSMinimize($t_{\text{start}}, t_i$, UnsignedDistanceAtTime)   ▷ Algorithm 1
14         else
15             Compute time direction $d_i$                                             ▷ Equation 17
16             def SignedDistanceAtTime($t$):
17                 $\vec{\mathbf{x}} \leftarrow$ BarycentricInterpolate($u, v, w, t$)
18                 return $\phi(\vec{\mathbf{x}})$
19             if $d_i < 0$ then $t_{i+1} \leftarrow$ GSSMinimize($t_{\text{start}}, t_i$, SignedDistanceAtTime)
20             else $t_{i+1} \leftarrow$ GSSMinimize($t_i, t_{\text{end}}$, SignedDistanceAtTime)
21         end if
           // Solve spatial sub-problem
22         $\vec{\mathbf{x}}_{t_{i+1}} \leftarrow$ BarycentricInterpolate($u, v, w, t_{i+1}$)
23         Compute $\vec{\mathbf{v}}_{t_{i+1}}$                                          ▷ Equation 14
24         Query $\phi(\vec{\mathbf{x}}_{t_{i+1}})$ and $\nabla\phi(\vec{\mathbf{x}}_{t_{i+1}})$
25         if $\phi(\vec{\mathbf{x}}_{t_{i+1}}) \leq 0$ then $t_{\text{end}} \leftarrow \min(t_{i+1}, t_{\text{end}})$   // Update interval
26
27         Compute support vertex $\vec{\mathbf{s}}_1$                                   ▷ Equation 15
28         def SignedDistanceAtPoint($\vec{\mathbf{x}}$):
29             return $\phi(\vec{\mathbf{x}})$
30         $\vec{\mathbf{x}}_{t_{i+1}} \leftarrow$ GSSMinimize($\vec{\mathbf{x}}_{t_{i+1}}, \vec{\mathbf{s}}_i$, SignedDistanceAtPoint)
31         Update barycentric coordinates $u, v, w$ using $\vec{\mathbf{x}}_{t_{i+1}}$
32         if $t_{i+1} \approx t_i$ and $\vec{\mathbf{x}}_{t_{i+1}} \approx \vec{\mathbf{x}}_{t_i}$ then break
33     end for
```

It is possible to avoid this issue by restarting the minimization whenever we reach sufficient precision to ensure no penetration at a local minimum, but this quickly becomes costly. The candidate contact point $\vec{\mathbf{x}}_{t_i}$ will often “wiggle” around a solution, getting continually closer, but may never quite reach its target. We show an example of this behaviour in the supplemental video. This behaviour is expected from Frank-Wolfe methods, however repeating this behaviour for multiple local minima slows down the algorithm drastically. Furthermore, because we ignore the direction of one of the gradient components while in contact with the SDF, the new modified direction does not have to be the descent direction, so theoretically the method can oscillate. We address the aforementioned issues with two more modifications over the temporally modified Frank-Wolfe method presented in Section 4.2.

First, α from Equations 18 - 20 is replaced by two line searches: one to solve for the optimal spatial step toward $\vec{\mathbf{s}}_i$, and another for the optimal temporal step in the direction obtained from Equation 17. Each line search is its own minimization of the SDF over a one-dimensional slice of the problem space – one in time, and the other in space. Solving for the step size with a line search maintains the convergence guarantees of the Frank-Wolfe algorithm [Braun et al. 2022; Jaggi 2013], but allows the spatial minimization to better adjust to the discontinuities and reductions of the domain. For this, we use the GSS method from Section 3.4. Our choice is based on the robustness and efficiency of the method, demonstrated by Macklin et al. [2020], when minimizing SDFs over single-dimension problems. The second modification is to use the result of the temporal sub-problem minimization to improve the spatial minimization. Each iteration, we find the time direction, solve for the temporal step size, and compute t_{i+1} as before. However, we additionally project $\vec{\mathbf{x}}_t$ to t_{i+1} and reevaluate the SDF and gradient before we proceed with Equation 15, solve for the spatial step size, and update $\vec{\mathbf{x}}_{i+1}$. We found this to increase the efficiency of our technique in solving the spatiotemporal problem.

To summarize, our Frank-Wolfe with line search (FWGSS) converges toward the first time of impact by solving different problems depending on the situation. First, it minimizes the current solution by finding barycentric coordinates and time that maximize penetration. When the current solution is inside the SDF, i.e., $\phi(\vec{\mathbf{x}}_{t_i}) < 0$, the temporal line search seeks to minimize the unsigned distance, while the spatial line search continues minimizing the SDF. Additionally, $\phi(\vec{\mathbf{x}}_{t_i}) \leq 0$ provides a new temporal upper bound, and forces the temporal line search to ignore the gradient and seek solutions backward in time. This effectively causes triangles to exit the SDF, culls local minima past the temporal upper bound, and avoids searching for solutions near the local minimum at the end of the time interval. By repeating until no further local minima are found, we obtain a simpler problem with a single temporal (global) minimum at the first time of impact.

Visualizations of the convergence in our supplemental video compare versions of the modified Frank-Wolfe method alongside projected gradient descent and shows that FWGSS (ours) is a clear winner. Our proposed algorithm is further summarized as pseudocode in Algorithm 2.

4.4 Starting Iterate

Picking a good starting iterate can also improve the convergence of gradient-based algorithms. In [Macklin et al. 2020], the triangle vertex $\vec{\mathbf{s}}_i \in \{\vec{\mathbf{a}}, \vec{\mathbf{b}}, \vec{\mathbf{c}}\}$ closest to the SDF is selected. We found this heuristic to hinder convergence in our method, as the closest point to the SDF at the start of the time interval might not necessarily be moving toward the SDF. Instead, the vertex that is most likely to collide with the SDF is selected as the one that minimizes $\vec{\mathbf{v}}_i \cdot \nabla\phi(\vec{\mathbf{s}}_i)$, where $\vec{\mathbf{v}}_i$ is the linear velocity of $\vec{\mathbf{s}}_i$.

4.5 Adaptive Triangle Subdivision

Algorithm 2 and the algorithm described by Macklin et al. [2020] are triangle-local. That is, they provide a single contact point per triangle. As shown in Figure 11 and Figure 10, this is not sufficient when large triangles or edges collide with the SDF at many simultaneous contact points. Subdividing the mesh reduces the risk of encountering this situation by splitting contact planes into smaller coplanar triangles. Unfortunately, this drastically increases the number of triangles on the mesh, all of which must be tested for contacts at every step of the simulation. We address

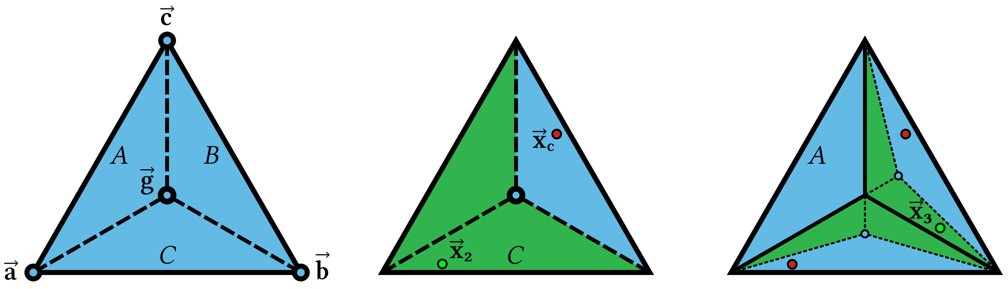

Fig. 3. A typical example of the subdivision process and recursive search for supplemental contacts. Left: a triangle can be subdivided in three sub-triangles at its centroid $\vec{g}$. Centre: based on the position of a first contact point $\vec{x}_c$, two (green) sub-triangles are searched for new contact points. A new contact $\vec{x}_2$ is found in sub-triangle C. Right: The sub-triangles are further subdivided to search for a third contact point. Sub-triangle A is ignored, because a previous search found no contact in it. Based on the existing contacts, four new (green) sub-triangles are searched, and a third point of contact $\vec{x}_3$ is found. The three points satisfy the conditions of being coplanar and not collinear, the algorithm is done.

this issue by providing an adaptive domain subdivision scheme to generate more contact points, when needed, without the cost of subdividing the whole mesh or modifying the geometry.

Collision detection provides a contact point $\vec{x}_c$ with barycentric coordinates $u,\ v,\ w$ on a triangle with vertices $\vec{a},\ \vec{b},\ \vec{c}$. The triangle's centroid $\vec{g} = (\vec{a} + \vec{b} + \vec{c})/3$ is cheap to compute and allows us to define the three barycentric regions $A,\ B,\ C$ shown in Figure 3. These regions each form a sub-triangle with the following vertices:

$$A \leftarrow \vec{b},\ \vec{c},\ \vec{g}, \qquad B \leftarrow \vec{a},\ \vec{g},\ \vec{c}, \qquad C \leftarrow \vec{a},\ \vec{b},\ \vec{g}. \tag{21}$$

The barycentric coordinates can be used to find in which region $\vec{x}_c$ resides. It is within A if $u \leq v$ and $u \leq w$, within B if $v \leq u$ and $v \leq w$, or within C if $w \leq u$ and $w \leq v$. The discrete collision detection algorithm described in Section 3.2 can then search for supplemental simultaneous contact points in sub-triangles which do not include $\vec{x}_c$. DCD is sufficient here, even in a CCD pipeline, because $\vec{x}_c$ is already obtained at the earliest time of impact, and we seek simultaneous contacts.

However, it is not guaranteed that contacts exist in sub-regions, or that only a single contact should be generated in the sub-triangle where $\vec{x}_c$ resides. The subdivision process is therefore applied iteratively to each of the sub-triangles. We show a typical example of one such case in Figure 3. It is possible to balance accuracy and performance by using a threshold, based on a minimum sub-triangle area, to limit iterative subdivisions to bigger triangles or sub-triangles. Only two additional non-collinear points are required for rigid face-face contact stability, so iterative subdivision may be interrupted early upon finding two valid points. In practice, we found that testing the two original sub-regions without iterative subdivision is generally sufficient even for extremely coarse meshes, as shown in the supplemental video and Figure 10.

4.6 Acceleration Structure

Narrow-phase intersection tests can effectively be reduced by using bounding volumes around each triangle, or a hierarchy thereof, and evaluating their distance to the SDF zero isosurface.

Broad-phase collision tests between a bounding sphere and an SDF are particularly efficient, because a single distance query is enough to determine if collision is possible. If the distance from a sphere's centre is bigger than its radius, nothing within the sphere can collide with the SDF.

However, using bound spheres is more difficult with CCD, especially when simulating complex motions. For instance, a swept-sphere test can help cull entire rigid bodies in some scenarios, but cannot capture rotational motions of individual triangles, leading to missed collisions [Ericson 2004].

We found that a sphere tracing method covers complex motions, including rotations, over time well enough to efficiently cull triangles, as long as a sufficient collision margin is added around each sphere, and that the shapes do not rotate more than 180 degrees around their centre of rotation. This is easily ensured by splitting the sphere tests into multiple intervals based on the rotation threshold. Note that we seek the earliest potential contact. There is no need to test later parts of the full interval if a potential contact is found in an earlier segment. Then, our iterative optimization algorithm moves the "padded" spheres linearly, between their position at the start and at the end of the time interval. A visualization of the bounding sphere and padding is shown in Figure 4. The movement at each iteration is defined by the difference between the padded sphere's radius and the distance to the SDF. If the padded sphere intersects the SDF anywhere along this trajectory, the corresponding triangle is considered for further collision testing. For time integration methods assuming constant velocities over a time step, we found that a good heuristic for the proper padding is to compute the distance between the position of the bounding sphere, on the real trajectory in the middle of the time interval, and the midpoint of the line between the start and end position, as shown in Figure 4.

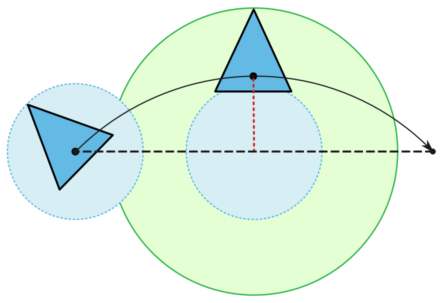
Fig. 4. A triangle undergoes non-linear motion. Linear sphere tracing, using the triangle's bounding sphere (blue), cannot capture the difficult motion. The maximum distance (red) between the trajectory and its linear approximation is added to the radius of the bounding sphere. The augmented bounding sphere (green) covers non-linear motions, even along the linear approximation of the trajectory.

5 Results

We evaluate our proposed method for speed, accuracy and robustness using a series of challenging scenarios involving triangles-SDF collision that requires CCD for success. The scenarios use meshes with up to hundreds of thousands of triangles and SDFs that challenge gradient methods with gradient discontinuities, sharp features, and multiple boundaries along triangles trajectories. All scenarios use the meter and second as base units, and no normalization is applied. A measure of 1 cm is 0.01 m in our simulators. We compare our method against the discrete collision detection method proposed by Macklin et al. [2020], which gives a baseline for comparing performance and robustness.

A custom rigid body and cloth simulator, written in C++, are used for all experiments. We use Discregrid by Koschier et al. [2017b], modified to support both single and double floating point precision, to generate and discretize SDFs from meshes. All examples were run on a consumer-grade laptop computer with an Intel i9 processor and an RTX 4070 laptop GPU, on a single CPU thread. Unless specified otherwise, we run our simulations at 60 time steps per simulated second.

We refer to the Frank-Wolfe with Golden Section Search (FWGSS) as our approach, but we also developed the Frank-Wolfe (FW) method for triangle-SDF CCD, which is included in relevant comparisons.

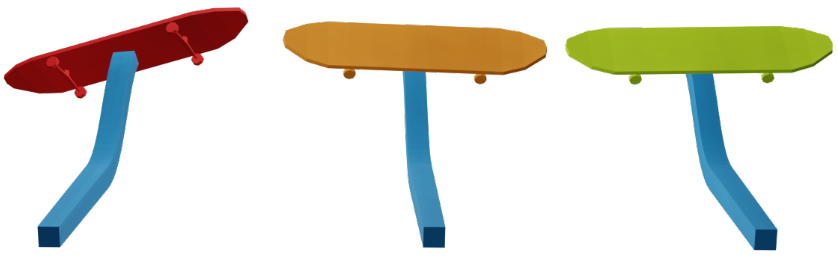

Fig. 5. A skateboard slides down a ramp using the Macklin et al. [2020], FW, and FWGSS methods. The FWGSS method allows the skateboard to settle at the tip of the rail, while the others are unstable.

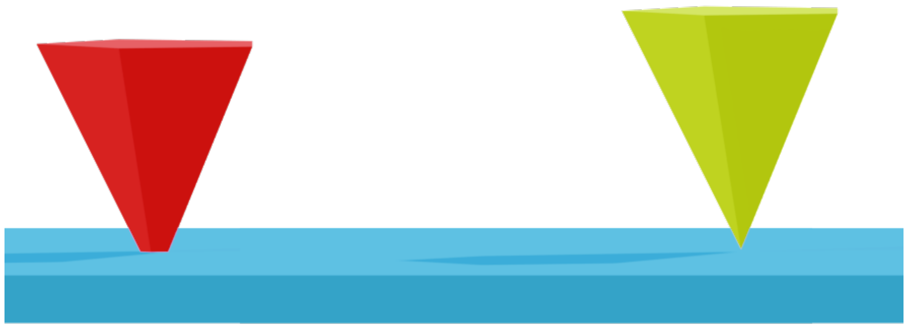

Fig. 6. Two coarse pyramids fall onto an analytical SDF box. The simulation with discrete collision detection (left) shows clear interpenetration, while our method on the right detects sharp contacts in time.

5.1 Contact detection

We first compare the quality of the collision detection to other approaches. In all the examples, our method cheaply offers robust collision detection.

As seen in Figure 5, a poorly chosen first time of impact can have drastic influences on the scene, making the discrete collision skateboard in red fall from the ramp.

In Figure 6 and Figure 7, we compare Macklin et al. [2020] and our method. On first impact, discrete collision detection fails to detect sharp contacts like the ears of the bunny, while our approach accurately captures the fine details of the collision. In Figure 8, we present a hoop made with Boolean operators to have exactly a one by one hole matching the radius of one for the analytical SDF sphere. Our algorithms successfully capture this challenging contact, while the previous state-of-the-art completely misses the collision. This is because the opportunity for contacts is small and can easily be traversed in a single time step.

In Figure 9, we compare the effects with and without adaptive triangle subdivision using a cloth simulation, where any interpenetration is instantly apparent due to the 2D nature of the mesh. We simulate two versions of the scene, one with a time step of 1 ms and one with a large step of 10 ms. The subdivision method already significantly reduces the interpenetration using discrete collision detection.

Likewise, our method without subdivision features some small level of interpenetration at a triangle level because only a single contact point is generated, while the subdivision adds key

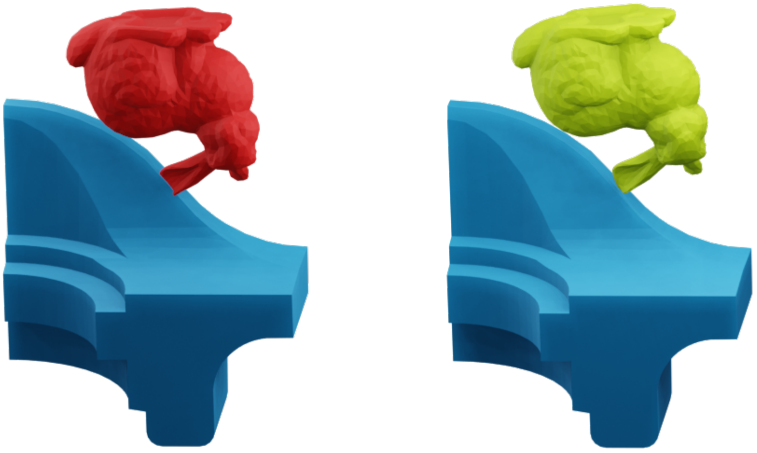

Fig. 7. A bunny simulated with Macklin et al. [2020] and one with FWGSS fall onto an SDF shape. The discrete collision detection bunny clips through the SDF while our method shows no interpenetration.

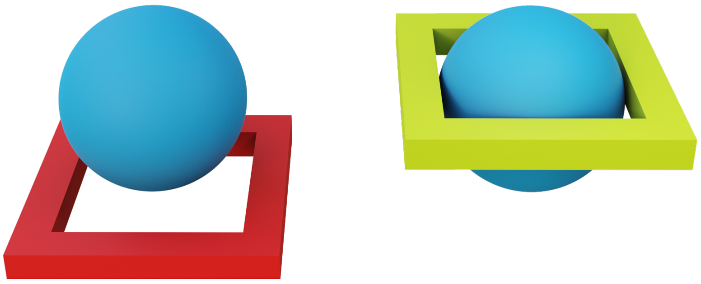

Fig. 8. A triangle mesh hoop falls over an analytical SDF sphere. The hole width exactly matches the sphere diameter. Discrete collision detection [Macklin et al. 2020] (left) misses the collision event, while our method (right) detects it.

contact points. In Figure 10, we show the impact of subdivision over time, where both discrete and continuous collision detection sink into the SDF without it. Subdivision allows the low-poly box to remain stable on top of the spikes, leading to more realistic simulations when used in more complex scenes. Figure 11 shows the first frame after a triangle falls flat against a box SDF. Without subdivision, a single contact constraint is generated, which is not sufficient to avoid penetration. With adaptive subdivision, three contact constraints are generated, allowing the triangle to remain stable.

As for large time steps, like the one frame 360° spin of Figure 12, our method catches them, while discrete collision detection has no way of detecting the contact, staying endlessly in the same position. The collision for a full 360° rotation in a single step is accurately captured with our method at the right time of impact, while discrete collision detection fails to detect the collision entirely, doing a full degree rotation in a time step, and for every time step.

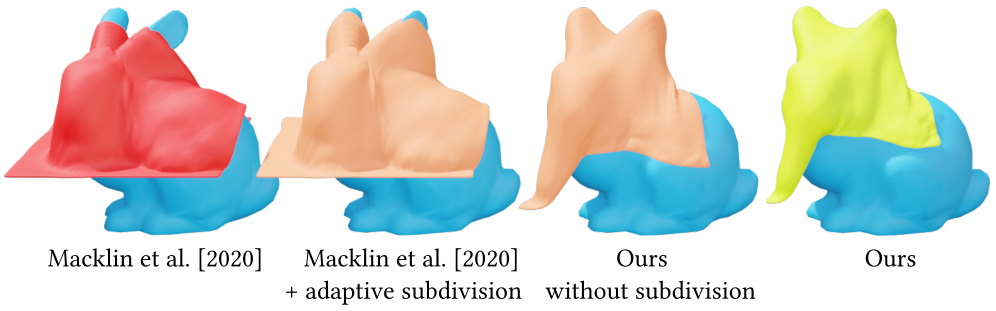

Fig. 9. A mass-spring cloth with triangle mesh geometry falls on a bunny with SDF geometry. Only our method using adaptive triangle subdivision produces a penetration-free simulation.

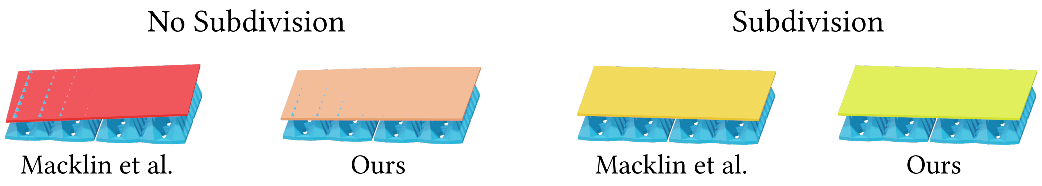

Fig. 10. A box modeled using a coarse triangle mesh falls on a spiky SDF. Without adaptive triangle subdivision, simulations using discrete and continuous collision both exhibit jitter and severe penetration.

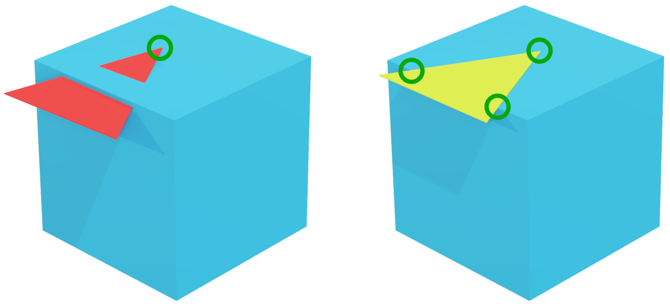

Fig. 11. Left: a single contact point cannot prevent the triangle to pass through the box. Right: we use adaptive subdivision to generate additional contact points, which provide stable face-face contact.

5.2 Ground truth comparisons

To further validate the robustness of our method, we generated a data set containing 2048 randomized starting configurations (position and velocity) along with ground truth (GT) collision solutions. The data set comprises the torus SDF from Figure 2, and two meshes, each with 1024 random configurations. The first mesh consists of a single triangle, and the second is the shuriken with 888 triangles from Figure 1. The torus SDF is 30 cm wide, and the triangle and shuriken are roughly 10 cm wide. We note that no normalization is applied. The meter and second are always

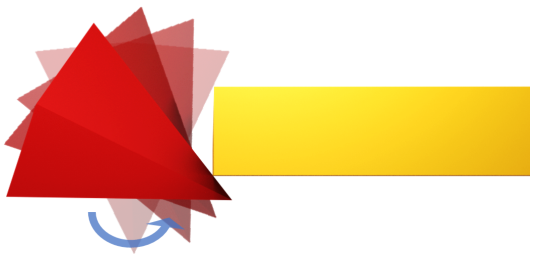

Fig. 12. A triangle mesh pyramid rotates 360° in a single time step. Our CCD method detects the collision on the analytical SDF box, preventing intersection.

Table 1. The time of impact (TOI) for our FWGSS method versus the ground truth. Statistics are shown with BSH acceleration. The last row shows the combined statistics over all 2048 tests. Negative and positive values indicate FWGSS found earlier or later TOI, respectively.

	ΔTOI (s)				
	Min	Max	Avg	Median	STD
Triangle	-1.88E-05	+1.18E-04	+6.87E-07	-1.00E-12	+7.25E-06
Shuriken	-6.47E-06	+7.78E-04	+7.29E-07	-3.00E-12	+2.43E-05
All	-1.88E-05	+7.78E-04	+6.09E-07	-1.00E-12	+1.50E-05

Table 2. The distance travelled by the mesh before impact for our FWGSS method vs ground-truth (GT). Negative and positive values indicate the mesh is closer or further, respectively.

	$\Delta\vec{\mathbf{x}}$ (m)				
	Min	Max	Avg	Median	STD
Triangle	-3.64E-04	+1.76E-03	+1.26E-05	-1.79E-11	+1.23E-04
Shuriken	-2.08E-04	+1.90E-03	+1.22E-06	-5.04E-11	+6.23E-05
All	-3.64E-04	+1.90E-03	+6.98E-06	-3.27E-11	+9.25E-05

the base units. The GT solutions are obtained from a brute force approach by advancing through the collision time interval in steps of 1E-06 s. Each step tests each triangle for collision using a thousand iterations of the DCD method by Macklin et al. [2020]. The method backtracks using a thousand iterations of GSS every time a collision is detected, and repeats the brute force DCD over each triangle until the first time of impact is found.

Our FWGSS method uses the BSH acceleration structure, but the GT method doesn't. The tests are generated from a combination of random mesh positions, speed, and angular velocity, but we ensure that the mesh is on a collision course with the SDF.

Statistical results of the tests are presented in Table 1 and Table 2. They show that our FWGSS method finds the correct time of impact with an average precision of roughly 0.5 μs. As multiple points on the same triangle may be valid contact points, we cannot directly compare the distance between those found by the two methods. Instead, we compare the distance between the stopping point of the mesh between the FWGSS and GT results. We see that FWGSS gives a distance of $<$ 0.5 mm compared to GT, with an average in the order of μm. Of the 2048 tests, FWGSS gave approximately 0.98% false positives for at least one triangle. The false positives are likely due to a slight penetration when FWGSS finds a time of impact a bit later than GT. FWGSS missed at least one triangle in contact for approximately 0.94% of tests. However, all tests for which FWGSS gave

Table 3. We evaluate scalability with regard to triangle count of an armadillo falling on an SDF spike grid by repeating the same scenario with different mesh resolutions. The first column shows the total physics computation time up to the time of impact, and the others show statistics per triangle tested. The results are averaged over 10 repeated executions.

Triangles	Method	Total (ms)	Tri. mean (μs)	Tri. median (μs)	Tri. σ (μs)
1K	DCD	10.41	25.99	25.37	3.69
10K	DCD	63.94	28.57	28.59	2.52
100K	DCD	306.66	27.34	27.10	3.52
1K	FWGSS	6.47	6.29	0.89	25.25
10K	FWGSS	24.18	3.31	0.86	5.91
100K	FWGSS	86.85	0.96	0.65	3.39

Table 4. Performance timings. All examples are simulated until the first contact, and the total time is reported as the sum of ms to compute the triangle-SDF collision detection over all frames. The percentage of runtime compares the collision detection to the full simulation time including time integration, constraint solve, etc. The other columns give the time statistic in μs of the algorithm (based on each triangle tested). Our FWGSS method outperforms the DCD algorithm ([Macklin et al. 2020]) in both accuracy and performance. Timing information for the less robust FW algorithm is also provided on examples where it successfully converged.

Scene	Method	Total (ms)	Runtime %	Tri. Tested	Mean (μs)	Min (μs)	Max (μs)	SD (μs)
Skate	DCD	2.33	94.72	76	30.69	24.99	80.17	13.94
Skate	FW	1.35	32.22	153	8.85	1.71	42.35	3.7
Skate	FWGSS	3.91	57.42	153	25.57	7.14	94.7	24.72
Bunny	DCD	5.09	98.64	89	57.18	26.29	113.5	29.06
Bunny	FW	0.52	27.81	134	3.9	0.39	25.01	4.7
Bunny	FWGSS	1.06	52.74	134	7.93	0.40	36.29	4.94
Shuriken	DCD	1.43	97.28	21	67.97	28.05	86.74	16.79
Shuriken	FWGSS	0.4	74.07	36	11.13	0.57	39.95	7.71

false negatives still found a time of impact very close to the GT method. False positive and false negative collisions occurred only for the shuriken, not the single triangle tests.

5.3 Performance

We evaluate the wall clock time of our method against other approaches by monitoring the computation time of the collision detection and adaptive subdivision. We only compare each technique up to the first time of impact and ignore subsequent simulation frames, as they are influenced by previous results and the different techniques quickly lose a common comparison basis. Other factors unrelated to the compared techniques, like constraint solve or collision response, would also otherwise begin to influence the results.

In Table 3, we evaluate the scalability of the algorithms as the number of triangle increases. This scenario uses the SDF spike grid from Figure 10, and the armadillo mesh at different resolutions. We note that, normally, it would make a lot more sense for the spike grid to be a triangle mesh (coarse and sharp), and for the armadillo to be an SDF (dense and relatively smooth). We deliberately chose this configuration because the spike grid SDF is extremely challenging for gradient methods. Both methods are accelerated using a bounding sphere hierarchy to cull narrow-phase tests. Our method of choice (FWGSS) scales better than the DCD method by Macklin et al. [2020]. The total run time is drastically improved with CCD preventing penetration, whereas DCD runs for a full extra simulation frame before having to handle a lot more intersections. Furthermore, the time interval for CCD shrinks whenever an earlier contact is found by our method. This makes subsequent triangle tests more efficient, and explains the larger standard deviation of our method. Per-triangle statistics demonstrate that our method becomes more efficient as the number of triangle increases. This is also due to the shrinking time interval, which provides a simpler problem space for subsequent narrow-phase triangle tests. We note that the average time per triangle is measured in microseconds.

In Table 4, we compare the time in milliseconds taken by the triangle-SDF collision detection algorithms and triangle subdivision, aggregated over the simulation frames for each scene. We include per-triangle timing statistics in microseconds to provide better insight about the efficiency of each algorithm. In all scenarios, our method of choice (FWGSS) is more accurate, and it outperforms the Macklin et al. [2020] DCD method. This holds true, even if the number of triangles tested over multiple time steps is higher with our methods, because false positives provided by the broad-phase tests are quickly resolved by the narrow-phase tests. The increased number of tests is due to the relative inaccuracy of our acceleration structure (see Section 4.6) compared to structures suited for DCD.

6 Discussion and Limitations

Knowing the lower bound of the minimized function may allow our method to outperform Macklin et al. [2020] while enabling CCD by providing the time of impact, but contact handling can be (very) expensive. Our collision detection method is fast, and compatible with efficient coarse meshes thanks to the adaptive triangle subdivision. It is also time-integration agnostic, and can be used with any type of contact handling. In real-time interactive simulations, where performance is prioritized over accuracy, our method may only be called once per frame for each potential collision pair requiring CCD. In this case, performance and robustness will be improved by our technique, and real time performance is easily maintained. If higher accuracy is required, and interpenetration must absolutely not happen, then our method may need to be called a lot more due to the collision response itself causing more contacts to happen within the time step. Our technique may still improve performance and robustness in these simulations but cannot guarantee real-time performance.

While our method can accurately find the first time and point of impact, it cannot guarantee an absolutely interpenetration-free simulation and maintain its real-time performance capacity without at least a small collision margin. We found that in practice, a margin of about one millimetre is sufficient, but accuracy is otherwise limited by floating point precision in the numerical solver and by limiting the number of iterations to maintain real-time performance.

In the discrete case, we get a cheap, relatively tight bound on the solution space, a static bounding sphere around the triangle. In the CCD case, we cannot accurately and cheaply estimate the full volume a triangle would pass through. Our ray-marching approach is cheap and avoids missing contacts, but requires a wide padding. Therefore, more triangles can pass the filter of the broad phase, even if they do not end up colliding.

Our method depends on the quality of the SDF, but inexact or approximate SDFs are common. We even use approximate polynomial forms in multiple of our scenarios. Imperfect SDFs affect the efficiency and guarantees our method can offer. For example, the more discontinuities there are, the harder it is for gradient methods to converge. Floating point error is also a concern for numerical methods, and may be exacerbated by SDFs with low precision or gradients with non-unit norms. High relative velocities and longer time steps also contribute to this issue. In practice, though, we found that our method still fares well on difficult scenarios with discontinuities, high velocities and rotations, and even difficult SDFs for shapes which would normally be represented by triangle meshes (like the spike grid in Figure 10).

Existing benchmarks for collision detection, such as [Wang et al. 2020], compare continuous triangle-triangle collision detection methods against ground truth solution. Although it is technically possible to generate a triangle mesh from an SDF using methods such as marching cubes [Lorensen and Cline 1987], the resulting geometry only provides a rough discrete approximation of the SDF. Standard triangle-triangle CCD would thus only be able to provide results relative to this approximation, so we exclude them as a baseline for ground-truth comparison. Not to mention that triangle-triangle collision detection brings additional challenges [Erleben 2018]. We therefore opt to compare our results to the ground-truth obtained using the brute force approach described in Section 5.2.

7 Conclusion

Our method addresses challenges in collision detection by reformulating the problem as a spatio-temporal local optimization. Through the introduction of adaptive triangle subdivision and the development of an enhanced problem-specific Frank-Wolfe method with line search, our method leads to robust and accurate collision detection for both coarse and complex geometries. This reduces issues such as tunnelling and missed collisions. Our method offers improved robustness over point sampling methods and outperforms recent triangle-SDF discrete collision detection (DCD) algorithms.

The golden section search is used for the various line searches required by our method. This approach is relatively efficient, and can often find minima more quickly than the version of our algorithm without it. In the future, we would like to study how different line search methods affect the efficiency and accuracy of our method. For instance, it is possible that a simpler bisection method could provide better efficiency.

The current acceleration structure has large bounds for detection. In the future, we would like to improve its efficiency by reusing spatial acceleration structures often used for the discretization of SDFs. Likewise, we could potentially improve the convergence of Frank-Wolfe methods using momentum-based approaches similar to Montaut et al. [2024]. No benchmark exists for triangle-SDF collisions. We evaluated our method with a series of extreme hand-crafted scenarios and random variations. However, it would be beneficial for the research community to develop standard benchmarks to provide rigorous and varied tests for triangle-SDF collisions and related edge cases. We note that our method could potentially be compatible with the incremental potential contacts (IPC) framework [Li et al. 2020], which would completely rid the simulations of the possibility of interpenetration past the first contact. We are excited to see the adoption of fast continuous collision detection in industry applications, and hope that efficient methods using SDFs will enable penetration-free simulations and further advance the field of real-time physics-based animations.

Acknowledgments

This project was partly funded by, and we acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), under Application Id: 585438-2023. The authors would like to thank Thomas Friedlaender, Paul Kry, and Kenny Erleben for providing valuable feedback and discussions. The authors would also like to thank Nicholas Sharp for Polyscope [Sharp et al. 2019], Inigo Quilez for the invaluable resources on SDFs, and the Stanford University Computer Graphics Laboratory for the Bunny, Armadillo, and XYZRGB dragon models, which helped in developing tools and simulations for our experiments.

References

Sheldon Andrews, Kenny Erleben, and Zachary Ferguson. 2022. Contact and friction simulation for computer graphics. In *ACM SIGGRAPH 2022 Courses* (Vancouver, British Columbia, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 3, 172 pages. https://doi.org/10.1145/3532720.3535640

Gábor Braun, Alejandro Carderera, Cyrille W Combettes, Hamed Hassani, Amin Karbasi, Aryan Mokhtari, and Sebastian Pokutta. 2022. Conditional Gradient Methods. arXiv:2211.14103 [math.OC]

Robert Bridson, Ronald Fedkiw, and John Anderson. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.* 21, 3 (July 2002), 594–603. https://doi.org/10.1145/566654.566623

Tyson Brochu, Essex Edwards, and Robert Bridson. 2012. Efficient geometrically exact continuous collision detection. *ACM Trans. Graph.* 31, 4, Article 96 (July 2012), 7 pages. https://doi.org/10.1145/2185520.2185592

Xuwen Chen, Cheng Yu, Xingyu Ni, Mengyu Chu, Bin Wang, and Baoquan Chen. 2024. A Time-Dependent Inclusion-Based Method for Continuous Collision Detection between Parametric Surfaces. *ACM Trans. Graph.* 43, 6, Article 223 (Nov. 2024), 11 pages. https://doi.org/10.1145/3687960

Christer Ericson. 2004. *Real-time collision detection.* CRC Press, USA.

Kenny Erleben. 2018. Methodology for Assessing Mesh-Based Contact Point Methods. *ACM Trans. Graph.* 37, 3 (07 2018), 39:1–39:30. https://doi.org/10.1145/3096239

Marguerite Frank and Philip Wolfe. 1956. An algorithm for quadratic programming. *Naval Research Logistics Quarterly* 3, 1-2 (1956), 95–110. https://doi.org/10.1002/nav.3800030109

Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. 2000. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 249–254. https://doi.org/10.1145/344779.344899

Arnulph Fuhrmann, Gerrit Sobotka, and Clemens Groß. 2003. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon*, Vol. 2003. Graphicon Scientific Society, Keldysh Institute of Applied Mathematics of the Russian Academy of Sciences, Moscow, Russia, 58–65.

Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin, and Dinesh Manocha. 2006. Fast continuous collision detection among deformable models using graphics processors. In *Proceedings of the 12th Eurographics Conference on Virtual Environments* (Lisbon, Portugal) *(EGVE'06)*. Eurographics Association, Goslar, DEU, 19–26.

Eran Guendelman, Robert Bridson, and Ronald Fedkiw. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph.* 22, 3 (July 2003), 871–878. https://doi.org/10.1145/882262.882358

Martin Jaggi. 2013. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *Proceedings of the 30th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 28)*, Sanjoy Dasgupta and David McAllester (Eds.). PMLR, Atlanta, Georgia, USA, 427–435.

M.W. Jones, J.A. Baerentzen, and M. Sramek. 2006. 3D Distance Fields: A Survey of Techniques and Applications. *IEEE Transactions on Visualization and Computer Graphics* 12, 4 (July 2006), 581–599. https://doi.org/10.1109/TVCG.2006.56

J. Kiefer. 1953. Sequential Minimax Search for a Maximum. , 502–506 pages. https://doi.org/10.1090/S0002-9939-1953-0055639-3

Dan Koschier et al. 2017b. *Discregrid Library.* Interactive Computer Graphics. Retrieved August 17, 2023 from https://github.com/InteractiveComputerGraphics/Discregrid

Dan Koschier, Crispin Deul, Magnus Brand, and Jan Bender. 2017a. An hp-adaptive discretization algorithm for signed distance field generation. *IEEE Transactions on Visualization and Computer Graphics* 23, 10 (2017), 2208–2221. https://doi.org/10.1109/TVCG.2017.2730202

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.* 39, 4, Article 49 (Aug. 2020), 20 pages. https://doi.org/10.1145/3386569.3392425

Pengfei Liu, Yuqing Zhang, He Wang, Milo K. Yip, Elvis S. Liu, and Xiaogang Jin. 2024. Real-time collision detection between general SDFs. *Comput. Aided Geom. Des.* 111, C (July 2024), 13 pages. https://doi.org/10.1016/j.cagd.2024.102305

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*. Association for Computing Machinery, New York, NY, USA, 163–169. https://doi.org/10.1145/37401.37422

Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. 2020. Local Optimization for Robust Signed Distance Field Collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 1 (18 04 2020), 1–17. https://doi.org/10.1145/3384538

William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. 1999. Six degree-of-freedom haptic rendering using voxel sampling. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 401–408. https://doi.org/10.1145/311535.311600

William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. 2005. Advances in voxel-based 6-DOF haptic rendering. In *ACM SIGGRAPH 2005 Courses* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 50–es. https://doi.org/10.1145/1198555.1198606

Louis Montaut, Quentin Le Lidec, Vladimir Petrik, Josef Sivic, and Justin Carpentier. 2024. GJK++: Leveraging Acceleration Methods for Faster Collision Detection. *IEEE Transactions on Robotics* 40 (2024), 2564–2581. https://doi.org/10.1109/TRO.2024.3386370

Bruce Naylor. 1998. A Tutorial On Binary Space Partitioning Trees. In *Computer Games Developer Conference Proceedings*. Eurographics Association, Long Beach, California, 433–457.

Quan Nie, Yingfeng Zhao, Li Xu, and Bin Li. 2020. A Survey of Continuous Collision Detection. In *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*. IEEE, Guangzhou, China, 252–257. https://doi.org/10.1109/ITCA52113.2020.00061

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 165–174. https://doi.org/10.1109/CVPR.2019.00025

S. Quinlan. 1994. Efficient distance computation between non-convex objects. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, San Diego, CA, USA, 3324–3329 vol.4. https://doi.org/10.1109/ROBOT.1994.351059

Stéphane Redon, Abderrahmane Kheddar, and Sabine Coquillart. 2002. Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum* 21, 3 (2002), 279–287. https://doi.org/10.1111/1467-8659.t01-1-00587

Nicholas Sharp et al. 2019. Polyscope. www.polyscope.run.

Min Tang, Ruofeng Tong, Zhendong Wang, and Dinesh Manocha. 2014. Fast and exact continuous collision detection with Bernstein sign classification. *ACM Trans. Graph.* 33, 6, Article 186 (Nov. 2014), 8 pages. https://doi.org/10.1145/2661229.2661237

M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. 2005. Collision Detection for Deformable Objects. *Computer Graphics Forum* 24, 1 (2005), 61–81. https://doi.org/10.1111/j.1467-8659.2005.00829.x

Bolun Wang, Zachary Ferguson, Xin Jiang, Marco Attene, Daniele Panozzo, and Teseo Schneider. 2022. Fast and Exact Root Parity for Continuous Collision Detection. *Computer Graphics Forum* 41, 2 (2022), 355–363. https://doi.org/10.1111/cgf.14479

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2020. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5 (24 09 2020), 188:1–188:16. https://doi.org/10.1145/3460775

Bolun Wang, Zachary Ferguson, Teseo Schneider, Xin Jiang, Marco Attene, and Daniele Panozzo. 2021. A Large-scale Benchmark and an Inclusion-based Algorithm for Continuous Collision Detection. *ACM Trans. Graph.* 40, 5, Article 188 (Sept. 2021), 16 pages. https://doi.org/10.1145/3460775

Jingping Wang, Tingrui Zhang, Qixuan Zhang, Chuxiao Zeng, Jingyi Yu, Chao Xu, Lan Xu, and Fei Gao. 2024. Implicit Swept Volume SDF: Enabling Continuous Collision-Free Trajectory Generation for Arbitrary Shapes. *ACM Trans. Graph.* 43, 4, Article 110 (July 2024), 14 pages. https://doi.org/10.1145/3658181

Hongyi Xu and Jernej Barbič. 2017. 6-DoF Haptic Rendering Using Continuous Collision Detection between Points and Signed Distance Fields. *IEEE Transactions on Haptics* 10, 2 (2017), 151–161. https://doi.org/10.1109/TOH.2016.2613872

Paul Zhang, Zoë Marschner, Justin Solomon, and Rasmus Tamstorf. 2023a. Sum-of-Squares Collision Detection for Curved Shapes and Paths. In *ACM SIGGRAPH 2023 Conference Proceedings* (Los Angeles, CA, USA) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 76, 11 pages. https://doi.org/10.1145/3588432.3591507

Tingrui Zhang, Jingping Wang, Chao Xu, Alan Gao, and Fei Gao. 2023b. Continuous Implicit SDF Based Any-Shape Robot Trajectory Optimization. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Detroit, MI, USA, 282–289. https://doi.org/10.1109/IROS55552.2023.10342104