

Real-time Collision Detection between General SDFs

Pengfei Liu^{a,1}, Yuqing Zhang^{a,1}, He Wang^b, Milo K. Yip^c, Elvis S. Liu^c and Xiaogang Jin^{a,*}

^aState Key Lab of CAD&CG, Zhejiang University, China

^bUniversity of Leeds, United Kingdom

^cMoreFun Studios, Tencent, China

ARTICLE INFO

Keywords:

Collision Detection

Sign Distance Function

Optimization

Physically Based Animation

ABSTRACT

Signed Distance Fields (SDFs) have found widespread utility in collision detection applications due to their superior query efficiency and ability to represent continuous geometries. However, little attention has been paid to calculating the intersection of two arbitrary SDFs. In this paper, we propose a novel, accurate, and real-time approach for SDF-based collision detection between two solids, both represented as SDFs. Our primary strategy entails using interval calculations and the SDF gradient to guide the search for intersection points within the geometry. For arbitrary objects, we take inspiration from existing collision detection pipelines and segment the two SDFs into multiple parts with bounding volumes. Once potential collisions between two parts are identified, our method quickly computes comprehensive intersection information such as penetration depth, contact points, and contact normals. Our method is general in that it accepts both continuous and discrete SDF representations. Experiment results show that our method can detect collisions in high-precision models in real time, highlighting its potential for a wide range of applications in computer graphics and virtual reality.


1. Introduction


SDFs (Frisken et al., 2000) have advantages in applications such as 3D reconstruction, surface fitting (Flöry and Hofer, 2010), sculpting, rendering, path planning (Chen et al., 2018) and collision detection due to their desirable characteristics, such as the analytic forms of implicit surface representation (Quilez, 2013) or neural networks (Park et al., 2019; Davies et al., 2020; Sitzmann et al., 2020; Tan et al., 2022), as well as their efficient $O(1)$ complexity for distance queries. As a result, they have emerged as one of the most prominent unifying geometry representations in a wide range of contexts, particularly in collision detection (CD), which is a fundamental problem in many fields. Analytic SDFs are even used by some developers to model entire scenes (Quilez, 2019).

SDFs allow us to query the minimum Euclidean distance between a point and the surface. As a result, collision detection algorithms for SDFs have previously focused on detecting collisions between SDFs and explicit representations, as characterized by the two approaches described below. The first is point sampling of the geometry prior to detection, followed by checking each point SDF value to determine the status of point-SDF overlap (Fuhrmann et al., 2003; Guendelman et al., 2003). Another method is to find the point on a triangle with the smallest SDF value in order to determine the status of triangle-SDF overlap (Macklin et al., 2020). While methods for detecting point-SDF and triangle-SDF collisions have been introduced and widely used in particle systems and physical simulations, little research has been conducted on collision detection algorithms between two arbitrary SDFs.

We discuss SDF-SDF collision detection for various SDF representations in this paper. SDF can be represented using a variety of techniques, such as analytic distance functions, voxels, neural networks, and others. The analytic distance function stands out among these methods due to its accurate representation and easy derivation of SDF values through fast computation. For certain shapes represented by distance functions, such as spheres and ellipsoids, fast and accurate collision detection methods already exist (Dube et al., 2011; Jia et al., 2011; Brozos-Vázquez et al., 2019, 2022). These methods, however, are limited to specific shapes or classes of shapes and cannot easily be extended to accommodate general analytic distance functions. To address this issue, we propose using interval arithmetic (Moore and Yang, 1996) to determine the range of SDF values within a region. This method enables fast intersection testing of SDF represented by the analytic distance function.

*Corresponding author

 jin@cad.zju.edu.cn (X. Jin)

 <http://www.cad.zju.edu.cn/home/jin/> (X. Jin)

¹Indicates equal contribution

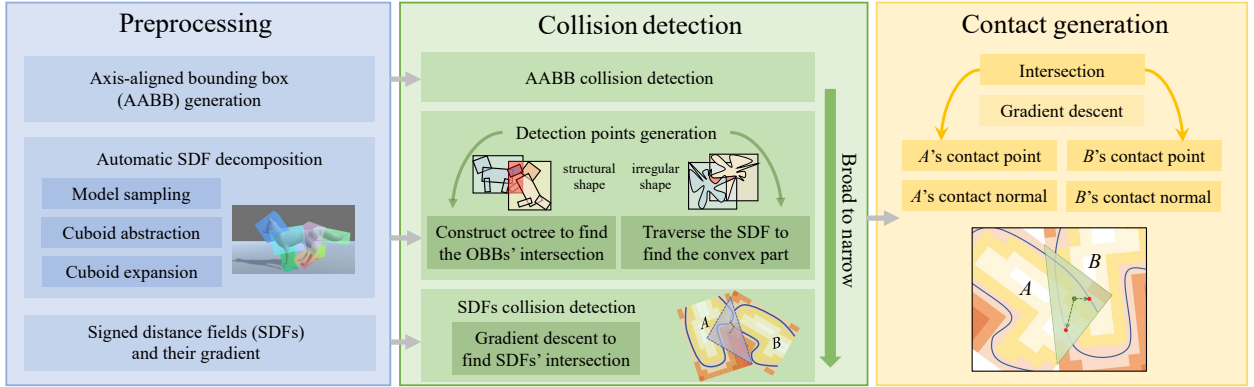


Figure 1: The pipeline of general SDFs collision detection in our method. We decompose the object in the preprocessing stage, keep the SDF within the specified bounds. In the collision detection stage, an octree subdivision algorithm is used to determine the intersection region of OBBs for simple and structurally regular objects, whereas irregularly shaped objects require a traversal of the SDF to identify convex protrusions on the object surface. Then, using gradient descent, we generate detection points and find intersection regions of SDFs. Finally, contact information such as penetration depth, contact points, and contact normals are generated.

Despite its advantages, the analytic distance function cannot represent arbitrary shapes, for which voxel or neural network representations are frequently preferred. As a result, a general collision detection method between general SDF representations is also extremely important. The following issues arise when detecting collisions with general SDFs. First, determining the range of SDF values in a given region efficiently for SDFs represented by voxels and neural networks is a significant challenge. A method for querying the intersection region of general neural implicit surfaces was proposed by Nicholas et al (Sharp and Jacobson, 2022). However, the computational speed of this method (80ms per query) is still insufficient for real-time applications. Second, because the intersection shape is arbitrary, it is necessary to compute the exact intersection. Gradient-based methods are fast numerically, but they are prone to getting stuck in local minima due to the concave nature of object geometries. Global methods, on the other hand, such as simulated annealing, can avoid local minima but are significantly slower. Third, while SDFs can provide detailed contact information, calculating specific indicators such as penetration depth and contact point locations remains difficult. Finally, there has been little comprehensive research into the optimal trade-off between accuracy and speed for SDFs. This emphasizes the need for further investigation of the problem.

To address the aforementioned challenges, we present a new real-time CD method between SDFs of analytic distance functions, as well as a dual-gradient-based approach for rapid collision detection and computation of contact/intersection information between two arbitrary objects represented by SDFs. For analytic distance functions, we use octrees for rapid identification of potential collisions to improve the efficiency of the collision detection algorithm. For general SDFs, we create oriented bounding boxes in order to segment the given objects into multiple parts. This object segmentation operation significantly reduces the issue of local minima in our gradient-based method. In addition, we use the analytical distance functions of bounding boxes to quickly identify potential collision parts and our dual-gradient approach to determine key contact points. Fig. 1 depicts the framework of general SDFs collision detection. We show that our method is both accurate in detecting collisions and fast enough for real-time applications through extensive evaluation and comparison. Our SDF-SDF collision detection method provides an efficient and accurate solution for a wide range of applications. In summary, our paper makes the following contributions:

- The first real-time and accurate general SDF-SDF collision detection method.
- A novel method for testing the intersection of analytic distance functions.
- An accurate method for estimating contact information for SDF-SDF collision response stages.

We compare our method to mesh-mesh and SDF-mesh methods. When it comes to collision detection for general SDFs, our method is comparable to and complements these methods. Furthermore, when SDFs are expressed analytically, our method outperforms the competition. Our method may have interesting potential applications when all of the scenes are represented by closed-form SDFs (Quilez, 2019).

2. Related Work

2.1. Collision Detection

Collision detection is a fundamental problem in computer graphics, of which many algorithms have been proposed for different objects' representations and application scenarios (Gilbert et al., 1988; Montanari et al., 2017; Ferguson et al., 2021; Wang et al., 2021; Macklin et al., 2020; Zesch et al., 2023). For complex objects represented by polygonal meshes, the basic problem of collision detection is to determine whether there exist colliding polygons or not. There are two main optimization ideas: one is to cull polygons that are improbable to collide (Zheng and James, 2012; Wang, 2014) and the other is to speed up the collision detection of polygons (Du et al., 2017). Since polygonal meshes only have surface information, they can't handle penetration between objects. While Continuous Collision Detection (CCD) can help to prevent penetration and increase stability, its high computational cost will impose a strain on real-time applications.

Another common object representation in collision detection is implicit surfaces, which include algebraic surfaces, SDFs, constructive solid geometry (Laidlaw et al., 1986; Sharma et al., 2018), and neural implicit representations (Sharp and Jacobson, 2022). Most implicit surface methods turn the collision detection problem into determining whether there are intersecting regions or intersecting points. Although accurate intersection points can be calculated analytically for primitives (Choi et al., 2006; Ketchel and Larochelle, 2006; Dube et al., 2011), it is difficult to find intersection regions of general SDFs. Sharp et al. (Sharp and Jacobson, 2022) provide an interval query scheme for neural implicit surfaces. It can be used in intersection tests; however, it has a poor time performance and cannot generate contact information. Calculating the minimum distance between objects can also help with collision detection. Fernandez et al. (Fernández-Layos and Merchante, 2024) developed an algorithm to determine the minimum distance and penetration depth between two convex SDF objects. However, this method does not provide a comprehensive analysis of SDF collisions between general objects. In comparison to earlier work, our method can quickly find intersecting regions and provide contact information for collision response.

2.2. SDF

SDFs can be defined as a function $\phi(\mathbf{x}) : R^3 \rightarrow R$ that attributes to each point \mathbf{x} its signed distance $\phi(\mathbf{x})$ to the closest surface point. $\phi(\mathbf{x})$ is positive if \mathbf{x} is outside the shape and negative if it is inside. Thanks to their strong shape representation capability and distance query efficiency, SDFs have been widely used recently in fields such as 3D modeling (Chen and Zhang, 2019; Li et al., 2022), 3D reconstruction (Chibane et al., 2020; Stier et al., 2021; Yao et al., 2021; Ortiz et al., 2022; Driess et al., 2022), and rendering (Takikawa et al., 2021; Jiang et al., 2020). SDF can not only be discretized for storage on a grid or volume texture, but it can also be encoded by a neural network (Park et al., 2019; Sitzmann et al., 2020; Davies et al., 2020) and retain its continuity. In our method, the representation form is irrelevant as long as the gradient information of SDF is provided.

SDFs are also useful for collision detection because they can instantly provide distance from any point in space to the object's surface, regardless of geometric complexity, and they allow the collision reaction to be easily computed alongside detection. SDFs have previously been used to identify collisions not just between rigid bodies (Xu and Barbic, 2014; Koschier et al., 2017; Macklin et al., 2020), but also in rigid-deformable simulations (Fuhrmann et al., 2003; Teschner et al., 2005). Xu et al. (Xu and Barbic, 2016) present a fast CCD algorithm for SDFs. It has been demonstrated that it can resolve collisions with intricate contact at high speeds. The preceding approaches all share the goal of detecting collisions between points and SDFs. However, sampling points directly on the object's surface may miss edge or face intersections with SDFs in particle-based simulations, resulting in penetration at some sharp features, especially when the number of sampling points is insufficient. Macklin et al. (Macklin et al., 2020) extend SDF-based collision detection to polygon meshes; they employ a GPU axis-aligned bounding box (AABB) tree to cull triangles and perform a local optimization loop per triangle to find the closest point to the SDFs. Although SDF has been widely used in collision detection, there is currently no research on real-time collision detection between general SDFs.

3. Intersection of Analytic Distance Functions

We begin with SDFs represented by analytic distance functions. In our approach, we use interval arithmetic to detect collisions between two distance functions. To demonstrate the application of interval calculations for collision detection, we use the distance function of a cuboid as an example. Given a cuboid (an axis-aligned unit cube) centered

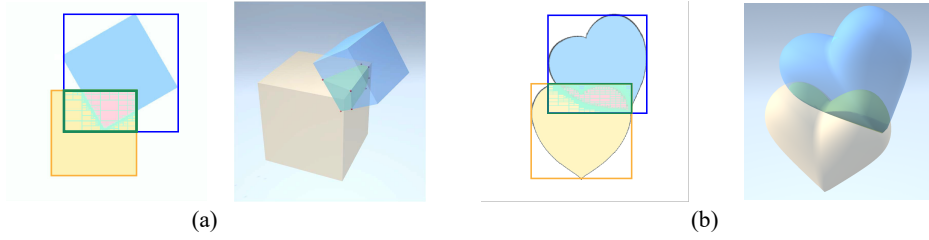


Figure 2: An illustration of intersecting regions in analytic distance functions. The intersection of two cubes (a) and two hearts (b). The intersection of the front view is on the left. The green wireframe represents the AABB intersection region. The octree structure of the intersection region is represented by the lime green wireframes.

at the origin with a scaling factor of $\mathbf{s} \in \mathbb{R}^3$, we define its analytic distance function $\phi_c(\mathbf{p})$ as follows:

$$\phi_c(\mathbf{p}) = \|\max(\mathbf{q}, 0)\|_2 + \min(\max(\mathbf{q}_x, \max(\mathbf{q}_y, \mathbf{q}_z)), 0), \quad (1)$$

$$\mathbf{q} = |\mathbf{p}| - \frac{1}{2}\mathbf{s}, \quad (2)$$

where \mathbf{p} is a point in space and \mathbf{s} is the length, width, and height of the cuboid.

In a coordinate frame, a region can be expressed as interval vectors like:

$$R = \{[x_-, x_+], [y_-, y_+], [z_-, z_+]\}. \quad (3)$$

where the positive and negative subscripts indicate the upper and lower bounds of the interval vector. Recognizing intervals as an extension of the real number system, a real number x can be represented as an interval $[x, x]$. To distinguish the calculation from real numbers, we use Max and Min to represent the operation of taking the maximum and minimum interval. Referring to (Moore and Yang, 1996; Stol and De Figueiredo, 1997), the basic interval operation rules for interval calculations are as follows. Interval arithmetic operations are defined by:

$$\begin{aligned} [a_-, a_+] + [b_-, b_+] &= [a_- + b_-, a_+ + b_+], \\ [a_-, a_+] - [b_-, b_+] &= [a_- - b_+, a_+ - b_-], \\ [a_-, a_+] * [b_-, b_+] &= [\min(a_-b_-, a_-b_+, a_+b_-, a_+b_+), \max(a_-b_-, a_-b_+, a_+b_-, a_+b_+)]. \end{aligned}$$

Max and Min operations are defined by:

$$\begin{aligned} \text{Max}([a_-, a_+], [b_-, b_+]) &= [\max(a_-, b_-), \max(a_+, b_+)], \\ \text{Min}([a_-, a_+], [b_-, b_+]) &= [\min(a_-, b_-), \min(a_+, b_+)]. \end{aligned}$$

The absolute value and square root of $[a_-, a_+]$ are defined by:

$$\begin{aligned} |[a_-, a_+]| &= \begin{cases} [\min(|a_-|, |a_+|), \max(|a_-|, |a_+|)], & a_- * a_+ > 0 \\ [0, \max(|a_-|, |a_+|)], & \text{otherwise} \end{cases} \\ \sqrt{[a_-, a_+]} &= \begin{cases} [], & a_+ < 0 \\ [0, \sqrt{a_+}], & a_- \leq 0 \\ [\sqrt{a_-}, \sqrt{a_+}], & \text{otherwise} \end{cases} \end{aligned}$$

In this sense, given any region R^p in 3D space, we can get the distance range from points in R^p to the cuboid. The distance bounds are defined as d_{\min} and d_{\max} :

$$[d_{\min}, d_{\max}] = \phi_c(R^p), \quad (4)$$

$$\phi_c(R^p) = \|\text{Max}(R^q, 0)\|_2 + \text{Min}(\text{Max}(R_x^q, \text{Max}(R_y^q, R_z^q)), 0), \quad (5)$$

$$R^q = |R^p| - \frac{1}{2}s. \quad (6)$$

Algorithm 1 describes the entire process of computing the intersection region between two cuboids, C_a and C_b , which are represented by their respective analytical distance functions, ϕ_{C_a} and ϕ_{C_b} . The detection region is the intersection region of two cubes' AABBs. The detection region R^p is partitioned using an octree, and its subregions are maintained by a queue. We can use the octree's nodes to fit the intersecting regions of the two cuboids with varying degrees of accuracy by varying the depth of the octree. Partitioning ends when a region is entirely within or entirely outside of both cuboids. When the upper bounds of the distance intervals d_{\max}^a and d_{\max}^b are both negative, a region is considered to be entirely within the two cuboids. When the lower bounds of the distance intervals d_{\min}^a and d_{\min}^b are both positive, a region is considered to be entirely outside the two cuboids.

The same interval arithmetic operations can be applied to any analytical SDF equation, such as a heart, which is represented by the following analytical equation:

$$x^2 + (4 + 1.2y - |x| \sqrt{\frac{20 - |x|}{15}})^2 + (z - \frac{y}{15})^2 - 15^2 = 0. \quad (7)$$

The results of this method are shown in Fig. 2 with the octree depth set to 10. The minimum size of the intersection region we can detect is determined by the depth of the octree. Deeper octrees can detect smaller intersecting regions, but with more cost.

Algorithm 1: The analytic distance function intersection test algorithm

input : ϕ_{C_a} , ϕ_{C_b} , R^p , and the depth of octree *level*
output: The intersection region of analytic distance function R^I

```

1  $Q \leftarrow \text{push}(R^p, \text{level});$ 
2 while  $Q$  is not empty do
3    $R^q \leftarrow Q.\text{front}();$ 
4    $[d_{\min}^a, d_{\max}^a] = \phi_{C_a}(R^q);$ 
5   if  $d_{\min}^a > 0$  then
6      $\text{continue};$ 
7    $[d_{\min}^b, d_{\max}^b] = \phi_{C_b}(R^q);$ 
8   if  $d_{\min}^b > 0$  then
9      $\text{continue};$ 
10  if  $d_{\max}^a \leq 0$  and  $d_{\max}^b \leq 0$  then
11     $R^I \leftarrow \text{push}(R^q)$ 
12  else
13     $l \leftarrow \text{getOctreeLevel}(R^q);$ 
14    if  $l > 0$  then
15       $R^{sub} \leftarrow \text{octree}(R^q, l - 1);$ 
16       $Q \leftarrow \text{push}(R^{sub}, l - 1);$ 

```

4. Intersection of General SDFs

While analytical distance functions can represent some specific geometric shapes, they have difficulty representing more general objects. In practice, voxels and neural networks are typically used to represent SDFs. Our approach, on the other hand, introduces a dual-gradient-based method for fast collision detection between two SDFs. This method effectively addresses the limitations of existing techniques, allowing for the detection of a wide range of objects while maintaining the speed required for real-time applications.

As shown in Fig. 1, our framework is divided into three stages. First, we decompose an arbitrary SDF into several parts by building an Oriented Bounding Box (OBB) for each part. Further, we use an AABB to completely contain

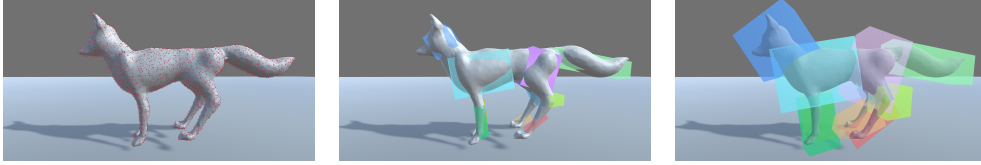


Figure 3: Pipeline for decomposition of SDFs. We first sample points on the model’s surface and calculate their corresponding normals (left). Then, we employ Cuboid abstraction network (Yang and Chen, 2021) to approximate parts of the model to cuboids (middle). After that, we expand cuboids to cover the whole model using the assigned matrix (right).

it. The decomposition method is described in Section 4.1. In the collision detection stage, we first reduce the number of possible pairs of AABBs in a collision, similar to the broad-phase collision detection in traditional methods. In the narrow phase, given the two colliders C_A and C_B , we define their SDFs as ϕ_A and ϕ_B .

When their AABBs intersect, we use the gradient decent to find the intersecting region. In Section 4.2, we present two approaches for determining the initial positions of gradient decent in various scenarios. In Section 4.3, we describe our new gradient descent method for identifying whether there exists a point in the intersecting region. Finally, in the contact extraction stage, we get the deepest penetration points with their separation normals in C_A and C_B . More details are presented in Section 4.4.

4.1. Automatic Decomposition of SDFs

To improve detection efficiency, an object is typically segmented into multiple parts, each of which is detected independently in collision detection. This strategy is also used in our approach. Because of its high computational speed, we use a gradient-based approach. Gradient-based methods, on the other hand, can be vulnerable to local minima when dealing with two arbitrary SDFs. Using oriented bounding boxes to divide an SDF into multiple parts is mathematically equivalent to dividing a non-convex optimization problem into several simpler problems. This method aids in avoiding local minima and reduces complexity.

Binding bounding boxes to objects is a time-consuming and skill-dependent task. Unlike discrete geometries (Mamou et al., 2016), there is currently no algorithm which is designed for dividing SDFs into multiple parts. Therefore, a pipeline for automatically decomposing an SDF is designed as follows. First, we conduct a dense and uniform sampling of the space, recording the value of SDF. Then, we select points on the input object’s surface (with a zero SDF value) and calculate their normals. The number of sampling points depends on the geometry of the object and the input of a neural network used for cuboid shape abstraction. After that, the unsupervised cuboid shape abstraction network (Yang and Chen, 2021) is employed to embed the input 3D points into a latent code and decode it into a set of parametric cuboids $\{C_i\}_{i=1,\dots,M}$. Each cuboid C can approximate a part of the original model and it is parameterized by a translation vector $\mathbf{t} \in R^3$, a scale vector $\mathbf{s} \in R^3$, and a quaternion $\mathbf{q} \in R^4$ representing the 3D orientation. For those points that are not inside any cuboid, we finally select the cube where the point is most likely to be located and enlarge the cube to exactly include that point. As shown in Fig. 3, the whole pipeline can ensure that the cuboids can totally contain their corresponding parts.

Although the automatically constructed oriented bounding boxes are sometimes not tight, they give a reference solution and are proven to be effective in our experiments. Artifacts caused by sampling can easily be mitigated by increasing the number of sampled points.

4.2. Detection Points Selection

We can determine whether two objects intersect by examining whether a detection point can be moved along the SDF gradient to a position inside both objects. If the intersection region is singular and fully convex, successful detection can be accomplished with only one detection point and a carefully chosen initial position. However, complex geometries of colliding objects can result in multiple and irregular intersection regions, as shown in Fig. 4(a), necessitating the use of multiple detection points to avoid local optima. Furthermore, the initial detection point positions must be carefully chosen, as shown in Fig. 4(b).

To address this issue, we choose multiple initial detection point positions in the intersection region of OBBs. The method outlined in Section 3 provides information about the intersection region of OBBs using an octree. In our approach, the center points of the region, defined by octree nodes, serve as the candidate detection points.

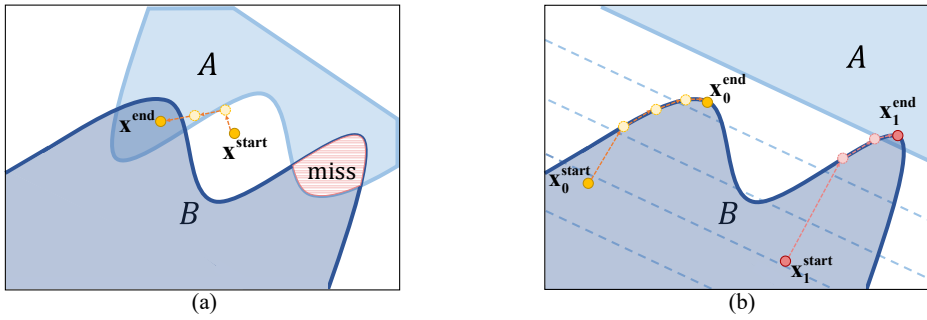


Figure 4: This figure shows two cases where detection fails. The detection point is first projected onto the surface of an object, then moved along the gradient of another object until it reaches the area where the objects intersect. (a) When multiple intersection regions are present, using only one detection point may result in missed collisions. (b) The importance of selecting the initial position of the detection point with care. Detection point x_0^{start} , in particular, becomes stuck in a local minimum and fails to reach the true intersection region, whereas x_1^{start} successfully identifies the collision.

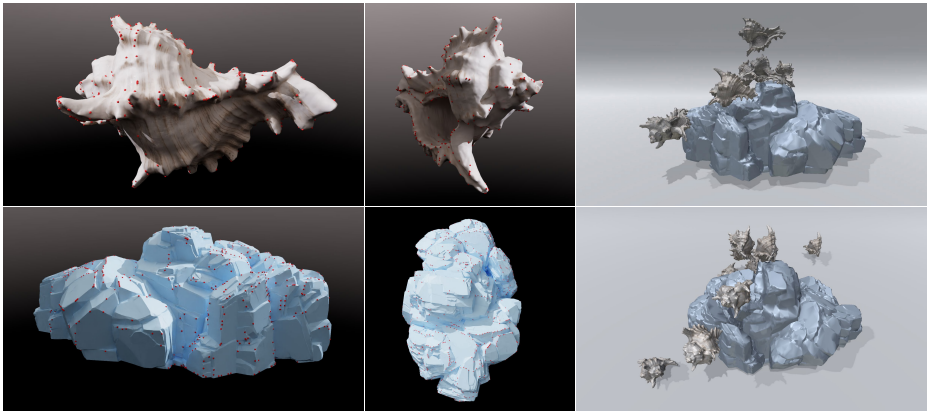


Figure 5: The detection points generated at convex protrusions on the surface of complex geometries with irregular shapes are shown in the left and middle columns. The collision detection results using these detection points are displayed in the right column.

While increasing the depth of an octree reduces the likelihood of missed detections, constructing large octrees solely for the purpose of locating small intersecting regions is ineffective. This is especially true for objects with uneven surfaces, which are difficult to decompose into multiple parts using OBBs. In such cases, We propose that the initial positions of detection points on the object's surface be pre-selected prior to implementing SDF collision detection. This strategy proves more effective in detecting objects with intricate geometries and irregular surfaces, such as shells.

Certain constraint conditions are imposed when selecting detection points on an object's surface to ensure that at least one detection point is initialized in each surface protrusion. To begin, the selected position must have an SDF value less than a pre-defined boundary value, ensuring that the detection points are within the object. Second, when querying the SDF values of the six points that are n units away from the selected point in each of the six adjacent directions as shown in Fig. 6 (a), at least four of these points must have SDF values greater than the boundary value. While a point that meets this condition does not have to be in a convex part of the object, there must be points that meet this condition on the convex portion. This condition ensures that points in the convex part of the object boundary are chosen. We traverse the SDF to determine the initial positions of all detection points that satisfy the conditions stated above. Because the SDF is defined in the local coordinate system of the object itself, these detection point positions should be transformed into the global coordinate system for fast collision detection. Fig. 5 shows the experimental results of this approach.

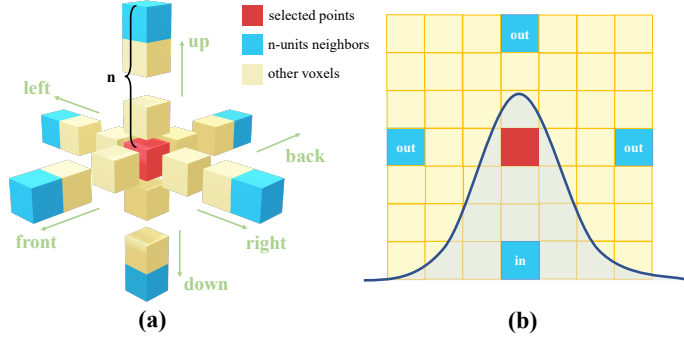


Figure 6: This is a representation of how detection points are selected on the surface of an object. In (a), we can see the selected point and its neighboring points in six adjacent directions. And if the selected point is on a convex part of the object's boundary, the SDF values of most of its neighbors will be greater than the boundary values, as shown in (b). For different objects, we need to manually adjust the parameter n to ensure that the detection points can cover the protrusions on the object surface.

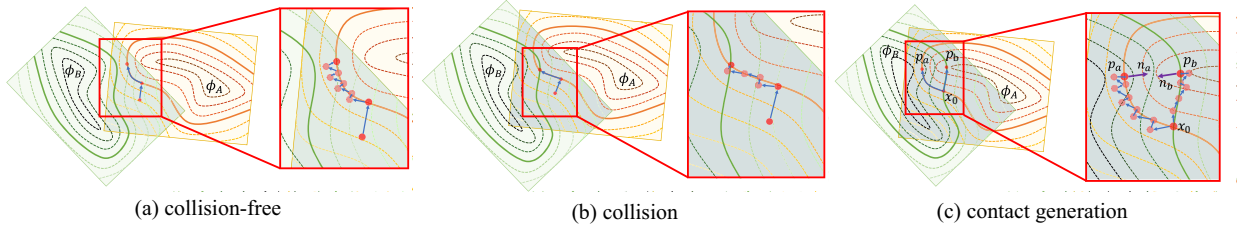


Figure 7: SDFs collision detection and contact generation using gradient descent. ϕ_A and ϕ_B are SDFs. The red point serves as a detection point and its blue trajectory is visualized as an optimized path to achieve a desired position under constraints. (a) shows the collision-free scenario while (b) demonstrates the scenario where objects A and B collide. The process of contact generation is illustrated in (c).

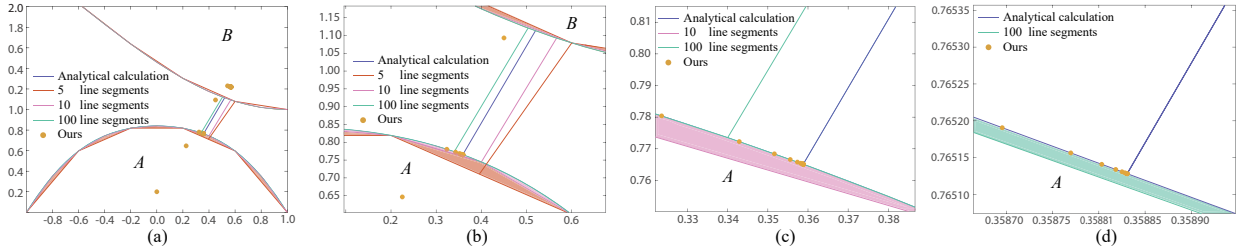


Figure 8: (a) We use analytic computation, line approximation, and our method to compute the line of closest points of two curves, respectively. (b) Results with approximation error greater than 0.1 using 5 line segments. (c) Results with approximation error greater than 0.01 using 100 line segments. (d) After a small number of iterations, the error of our method is less than 10^{-17} . The colored area is the error of line segment approximation. The orange points represent our method's detection point positions at each iteration.

4.3. Intersection of SDFs

The problem for the intersection test of SDFs is to determine whether there exists a point \mathbf{x} in the space that satisfies both $\phi_A(\mathbf{x}) \leq 0$ and $\phi_B(\mathbf{x}) \leq 0$.

Assuming that a detection point \mathbf{x} is located outside of object A, we project it onto the nearest point within A, including the point on the surface. We can easily calculate the closest point on the isosurface where $\phi_A = 0$ for any point in 3D space due to the advantageous properties of SDFs. As a result, the projection operator P is as follows:

$$P(\mathbf{x}) = \arg \min_{\mathbf{x}' \in A} \frac{1}{2} \|\mathbf{x}' - \mathbf{x}\|^2 = \mathbf{x} - \nabla \phi_A(\mathbf{x}) \phi_A(\mathbf{x}). \quad (8)$$

We update the detection point \mathbf{x} iteratively along the the gradient of ϕ_B with a step size s , subject to the constraint that \mathbf{x} stays inside the object A . The gradient for voxel and neural network representations of SDF can be computed numerically by querying the SDF values. Similar to other methods using gradient-based iteration (Macklin et al., 2020; Bridson, 2015), the iterative process is governed by the following formula:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - s \nabla \phi_B(\mathbf{x}_i), \text{ s.t. } \phi_A(\mathbf{x}_{i+1}) \leq 0. \quad (9)$$

The constraint is satisfied by performing another projection operation on the predicted position \mathbf{x}_{i+1} , which is outside of object A .

Finally, if the detection point \mathbf{x} moves to the internal region of B , where it simultaneously satisfies $\phi_A(\mathbf{x}) \leq 0$ and $\phi_B(\mathbf{x}) \leq 0$, then object A and object B are considered as in a collision (see Fig. 7). If we are unable to find a point within both SDFs, as shown in Fig. 7(b), we will restart our search for the next detection point. When all of the detection points indicate that there is no collision, we conclude that objects A and B do not intersect. More details can be found in Algorithm 2.

Algorithm 2: The SDF intersection test algorithm

input : ϕ_A, ϕ_B , the detection point \mathbf{x}
output: An intersection point or the closest point \mathbf{x}_i

```

1  $\mathbf{x}_i \leftarrow \mathbf{x}$ ;
2 while ( $\phi_A(\mathbf{x}_i) > 0$  or  $\phi_B(\mathbf{x}_i) > 0$ ) and  $|\mathbf{x}_{i+1} - \mathbf{x}_i| > \epsilon$  and  $Num_{iter} < MaxSteps$  do
3   if  $\phi_A(\mathbf{x}_i) > 0$  then
4      $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i - \nabla \phi_A(\mathbf{x}_i) \phi_A(\mathbf{x}_i)$ ;
5   else
6      $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i - s \nabla \phi_B(\mathbf{x}_i)$ ;
7     if  $\phi_A(\mathbf{x}_{i+1}) > 0$  then
8        $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1} - \nabla \phi_A(\mathbf{x}_{i+1}) \phi_A(\mathbf{x}_{i+1})$ ;
9    $\mathbf{x}_i \leftarrow \mathbf{x}_{i+1}$ ;

```

4.4. Contact Generation

When the collision between A and B is detected, we generate the contact information including contact points, contact normals, and contact depth. We define the contact points as \mathbf{p}_a and \mathbf{p}_b . \mathbf{p}_a is the deepest point on the surface of A that penetrates into B . \mathbf{p}_b is the deepest point on the surface of B that penetrates into A . Then the contact depth can be easily obtained by querying $\phi_A(\mathbf{p}_b)$ and $\phi_B(\mathbf{p}_a)$. Contact normals are defined as \mathbf{n}_a and \mathbf{n}_b which are the best directions to separate colliders:

$$\begin{aligned} \mathbf{n}_a &= -\nabla \phi_A(\mathbf{p}_a), \\ \mathbf{n}_b &= -\nabla \phi_B(\mathbf{p}_b). \end{aligned} \quad (10)$$

The projected gradient descent method is used to find contact points using the intersection point found in Section 4.3 as the starting point \mathbf{x}_0 , as shown in Fig. 7 (c). In contrast to Algorithm 2, \mathbf{p}_a and \mathbf{p}_b can be uniquely determined with precise positions during the contact generation process. Specifically, we compute the next point position along the gradient of A 's SDF at each iteration, with the constraint that the point lies inside B . The iterative process ends when the step size of the solver falls below a user-defined tolerance ϵ , yielding the convergence point \mathbf{p}_a . Similarly, the position of \mathbf{p}_b can be determined. Algorithm 3 contains more information. The step size must be manually adjusted for various scenarios. A step size that is too large can result in frequent detection failures. A too-small step size, on the other hand, can result in excessively long detection times. We begin the adjustment process in experiments with a step size of one unit length.

5. Results

We implemented our method on a PC with an AMD Ryzen 5800X CPU and Nvidia RTX 2060 GPU. SDFs are represented by 3D voxel grids and neural networks (Park et al., 2019). The analytic distance functions are based on (Quilez, 2019). All experiments are shown in the video.

Algorithm 3: Contact generation

input : ϕ_A, ϕ_B , the starting point \mathbf{x}_0
output: $\mathbf{p}_a, \mathbf{p}_b, \mathbf{n}_a, \mathbf{n}_b$

```

1  $\mathbf{x}_i \leftarrow \mathbf{x}_0$ ;
2 while  $\phi_A(\mathbf{x}_i) \leq 0$  and  $\phi_B(\mathbf{x}_i) \leq 0$  and  $|\mathbf{x}_{i+1} - \mathbf{x}_i| > \epsilon$  and  $Num_{iter} < MaxSteps$  do
3    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i - s\nabla\phi_B(\mathbf{x}_i)$ ;
4   if  $\phi_A(\mathbf{x}_{i+1}) > 0$  then
5      $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1} - \nabla\phi_A(\mathbf{x}_{i+1})\phi_A(\mathbf{x}_{i+1})$ ;
6    $\mathbf{x}_i \leftarrow \mathbf{x}_{i+1}$ ;
7  $\mathbf{p}_a \leftarrow \mathbf{x}_i$ ;
8  $\mathbf{n}_a \leftarrow \nabla\phi_B(\mathbf{p}_a)$ ;
9  $\mathbf{x}_i \leftarrow \mathbf{x}_0$ ;
10 while  $\phi_A(\mathbf{x}_i) \leq 0$  and  $\phi_B(\mathbf{x}_i) \leq 0$  and  $|\mathbf{x}_{i+1} - \mathbf{x}_i| > \epsilon$  and  $Num_{iter} < MaxSteps$  do
11    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i - s\nabla\phi_A(\mathbf{x}_i)$ ;
12   if  $\phi_B(\mathbf{x}_{i+1}) > 0$  then
13      $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_{i+1} - \nabla\phi_B(\mathbf{x}_{i+1})\phi_B(\mathbf{x}_{i+1})$ ;
14    $\mathbf{x}_i \leftarrow \mathbf{x}_{i+1}$ ;
15  $\mathbf{p}_b \leftarrow \mathbf{x}_i$ ;
16  $\mathbf{n}_b \leftarrow \nabla\phi_A(\mathbf{p}_b)$ ;
    
```

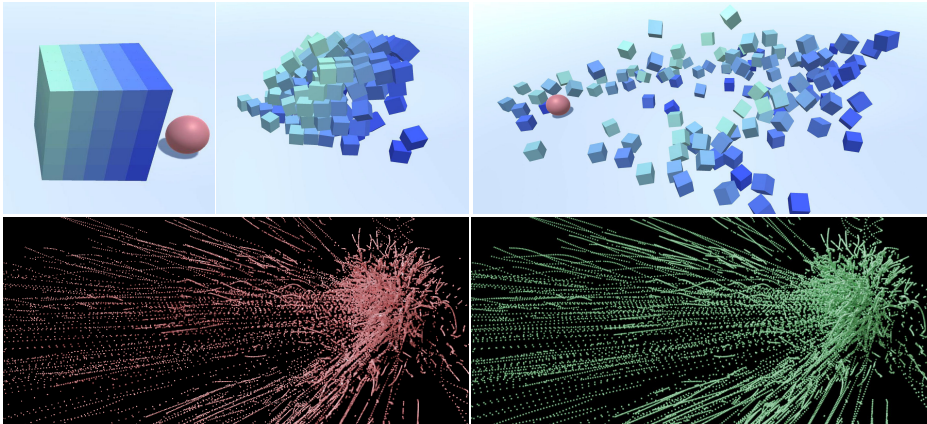


Figure 9: We can obtain precise collision data between spheres and cubes. Each frame's contact points are recorded as an image. The red points on the bottom left are accurate contact points. Our result is represented by the distribution of green points on the bottom right.

5.1. Detection Precision

Using two objects A and B with smooth curved boundaries, we compare our method to the analytic computation method and the line approximation method in Fig. 8. To determine their shortest distance, an analytical computation is performed. The curves are approximated by 5, 10, and 100 line segments, respectively, to mimic existing discretization strategies. We can calculate the approximation error by comparing the shortest distances computed from these approximations to the analytical result. Although increasing the number of line segments produces more accurate results, it incurs a higher computational cost. In our experiments, the computational cost of 5 line segments is 0.0050~0.0053 milliseconds, and the computational cost of 100 line segments is 1.2457~1.5063 milliseconds. There is a significant difference between analytical and experimental results. Using analytic distance functions representation, on the other hand, can produce more accurate results in a more efficient manner. We choose a detection point within A and use our method to move it to the position closest to B with a fixed step size $s = 0.5$. Our method significantly improved precision and efficiency, lowering the error rate to 10^{-17} after only 89 iterations.

Real-time Collision Detection between General SDFs

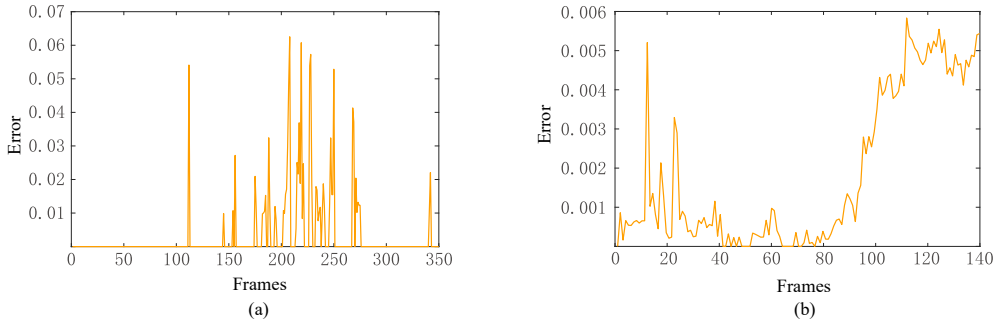


Figure 10: (a) The figure shows the mean distance between the detected contact point positions and their ground truth values for each frame in the cube-sphere scenario. The maximum size of an object's OBB is 2.54 units. (b) The mean distance of the contact point positions between our method and the mesh-SDF method (Macklin et al., 2020) for each frame in the shell-rock scenario. The maximum size of the shell's OBB is 5.56 units.

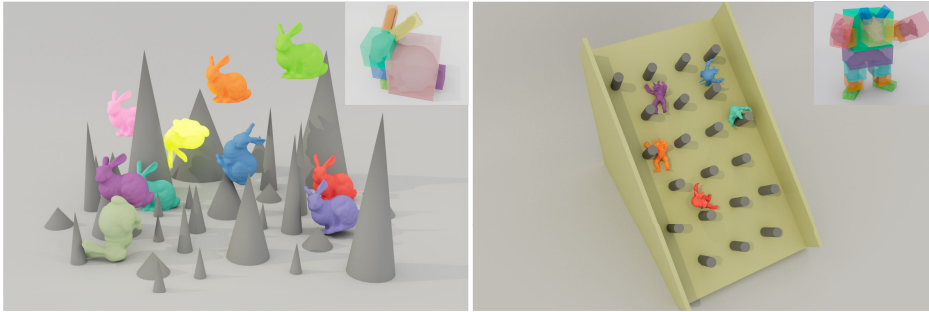


Figure 11: We design two scenarios, bunny-cone and armadillo-cylinder, to compare different methods. In the mesh-based methods, the armadillo has 9,994 triangles and bunny has 5,880 triangles. In our proposed method, the cone and cylinder are represented using analytical distance functions, the discrete SDFs are stored in 253^3 grids, and the continuous SDFs are encoded by multilayer perceptrons (Davies et al., 2020). The results of the SDF decomposition are displayed in the top right subplot.

We compare our results to the exact results in a 3D cube-sphere collision scenario to further validate the accuracy of our proposed method. Because cubes and spheres are regular geometric shapes, all collision information is exact. To calculate the positions of all collision points and record the trajectories of the objects, we use the Separating Axis Theorem (SAT) (Huynh, 2009) and CCD algorithms (Redon et al., 2002). We compute the contact point positions using our proposed SDF-based gradient descent method, which uses analytic distance functions to represent the sphere and cubes in each frame. The entire process is visualized in Fig. 9. In this experiment, we set ϵ to 10^{-5} and calculated the average difference between the contact point positions and the exact results in each frame, as shown in Fig. 10 (a). In theory, as long as ϵ is small enough, our results can approach the exact solution infinitely closely.

5.2. Time Performance

We compare the time efficiency of collision detection algorithms in two scenarios: bunny-cone and armadillo-cylinder in this experiment. The fully mesh-based algorithm (Coumans and Bai, 2016–2021), the algorithm proposed in (Macklin et al., 2020) for collision detection between triangle meshes and SDF, and our own algorithm were all evaluated. The contact point trajectory during the process is visualized in Fig. 12, with all collisions in sharp areas accurately detected. Triangle meshes are used in collision detection between triangle meshes and SDF to represent bunnies and armadillos, whereas analytical distance functions are used to represent cylindrical and conical objects. Bunnies and armadillos are represented in our proposed method by either a discrete SDF stored in voxel grids or a continuous SDF encoded by a neural network, while other objects in the scene are represented by analytical distance functions.

The comparison of detection times per frame for various collision detection algorithms (Coumans and Bai, 2016–2021; Macklin et al., 2020), all of which are run on a single thread, is shown in Fig. 13. Despite the use of culling,

Real-time Collision Detection between General SDFs

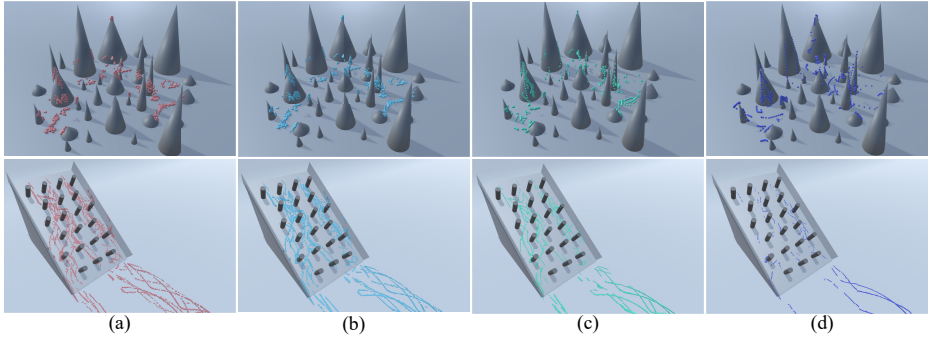


Figure 12: Comparison of the contact point distributions of three different collision detection methods: (a) the fully mesh-based method, (b) the mesh-SDF method, and (c) our proposed method. Images (d) are the results of our method in the absence of SDF decomposition.

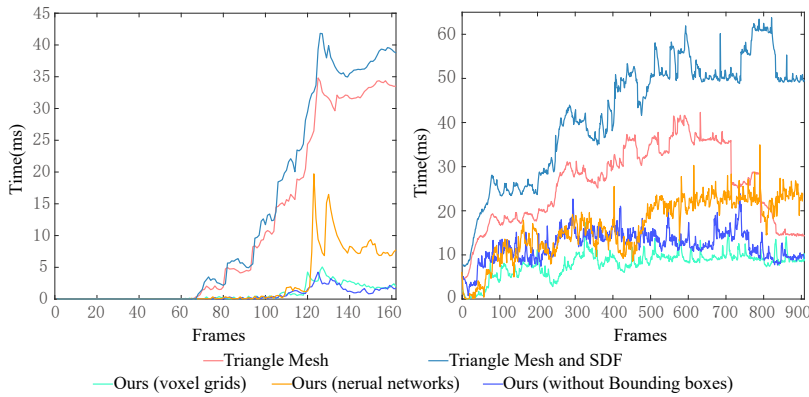


Figure 13: Time performance for different collision detection methods between bunnies and cones (left) and armadillos and cylinders (right).

Table 1

Objects information and memory usage comparison between meshes and SDFs.

Scene	Triangle Num	Voxel Num	Step Size	Memory Usage			
				Triangle Mesh	Triangle Mesh and SDF(voxel)	Ours (voxel)	Ours (neural networks)
Bunny	5880	256^3	0.005	468MB	528MB	513MB	463MB
Armadillo	9994	256^3	0.01	465MB	774MB	755MB	468MB

the completely mesh-based method is computationally expensive for dynamic scenarios with high-precision models. This is due to the fact that each frame during sustained contact between objects requires the processing of intersections of numerous triangles. Due to the time-consuming nature of performing the local optimization for each triangle, the mesh-SDF collision detection method is even less efficient than the completely meshed-based method without the use of the GPU AABB tree. Furthermore, neural network-encoded SDFs necessitate more calculations for distance queries, resulting in lower detection efficiency than SDFs discretely stored within voxel grids with $O(1)$ query time complexity. Their advantage, however, is that they ensure SDF continuity while significantly reducing memory usage.

5.3. Ablation Study

Preprocessing begins with decomposing the SDF into multiple parts and generating corresponding OBBs. Calculating the intersection region of OBBs not only reduces the gradient descent search space but also guides the detection point selection. We conduct an ablation study to show how the OBB generated for each object component improves collision detection accuracy and speed.

Fig. 11 shows the decomposition of bunny and armadillo models. In contrast, we generate only one OBB for the

entire model and then select detection points from the intersection region of the OBBs. The final positions of the contact points are shown in Fig. 12. In the absence of SDF decomposition, the generated contact point error increases significantly, resulting in a large number of false and missed contacts.

The missed detections can be attributed to the two factors listed below. To begin, detection points in the complex search space are prone to becoming trapped in local optima, resulting in failure to converge towards the true intersection regions of SDFs. Second, generating the intersection area with only two OBBs can reduce the number of detection points.

Although increasing the depth of the octree can avoid these problems, it comes with a significant increase in collision detection time. With a fixed octree depth, not using SDF decomposition can also reduce algorithm efficiency, as shown by the blue curve in Fig. 13. It becomes more obvious as model complexity increases, because larger intersection regions of OBBs necessitate more iterations for optimizing detection points via gradient descent.

5.4. The Optimization of GPU Parallelization

Since the gradient descent process of each detection point is independent, our method can be parallelized for which we use Nvidia wrap (Macklin, 2022). We compare the parallelization performance of our method to that of (Macklin et al., 2020) in a shell-rock collision scenario.

To ensure collision detection accuracy, we use a high-poly model with 1,300,000 triangles to represent the shell and 620,992 triangles to represent the rock in collision detection. In addition, the SDF resolution is 1000^3 .

A shell, unlike the bunny and armadillo, lacks distinct structural features. Decomposing it into multiple convex bodies using OBBs is extremely difficult due to the presence of various sized protrusions on its surface. As a result, we chose 484 detection points on the shell and 384 detection points on the rock using the algorithm described in Section 4.2. During the narrow phase, all detection points begin simultaneously searching for the collision region by moving along the gradient of SDFs. In our collision detection method, the average time of gradient descent per frame is 0.22ms.

For the same scene, we used the SDF-mesh collision detection method and collected collision detection data. The time of gradient descent on object surfaces is 0.26ms per frame with parallel computing. As shown in Fig. 10 (b), the error in contact information generated by both methods is calculated for each frame. According to the statistical results, the difference in contact information generated by the two methods is usually less than 0.05 unit, with a maximum of 0.064 unit. However, both methods missed some collisions that the other method captured. We discovered that these points are precisely on the boundary of the collision judgment condition after further investigation. As a result, we believe that this discrepancy is caused by an error between SDF and mesh.

6. Conclusion and Limitation

In this paper, we discussed the SDF-SDF collision method and proposed collision detection method between SDF-SDF in different representation. For analytic Distance Function, we employed interval arithmetic for intersection detection. For general SDFs, We incorporate an unsupervised cuboid shape abstract neural network into the SDF decomposition task. SDF decomposition can aid in improving detection time performance and reducing SDF memory usage. SDF decomposition, on the other hand, is useful in avoiding local minima. In the detection of collisions, we begin by using an octree to locate the intersection region of two OBBs. The detection points are then moved along the gradient of the SDF to find SDF intersection regions. Finally, we find contacts by looking for the deepest points and gradient directions.

Our method has limitations: (1) The local optimum is still not completely avoidable. We need a large number of detection points for multiple detections, which causes a lot of resource consumption. (2) The query efficiency of SDFs varies depending on the representation. The query efficiency of voxels and analytic distance functions is high. For neural implicit surfaces, we need develop more efficient methods. (3) The method's parameters, such as the depth of the octree, the step size of the detection point movement, and so on, must be manually adjusted according to the scene. There is currently no good automated method for calculating it.

It is worth noting that our method complements the mesh-mesh and SDF-mesh methods. When the entire scene is represented by SDFs (Quilez, 2019), our method has advantages. In the future, we will investigate how SDFs can be used in both rendering and physically based animation in order to realize the application of SDFs in a broader range of scenarios.

Acknowledgements

Xiaogang Jin was supported by Key R&D Program of Zhejiang (No. 2024C01069), the National Natural Science Foundation of China (Grant No. 62036010), and the FDCT under Grant 0002/2023/AKP.

References

- Bridson, R., 2015. Fluid simulation for computer graphics. AK Peters/CRC Press.
- Brozos-Vázquez, M., Pereira-Sáez, M.J., Rodríguez-Raposo, A.B., Souto-Salorio, M.J., Tarrío-Tobar, A.D., 2022. Contact detection between a small ellipsoid and another quadric. *Computer Aided Geometric Design* 98, 102136.
- Brozos-Vázquez, M., Pereira-Sáez, M.J., Souto-Salorio, M., Tarrío-Tobar, A.D., 2019. Classification of the relative positions between a small ellipsoid and an elliptic paraboloid. *Computer Aided Geometric Design* 72, 34–48.
- Chen, R., Gotsman, C., Hormann, K., 2018. Path planning with divergence-based distance functions. *Computer Aided Geometric Design* 66, 52–74.
- Chen, Z., Zhang, H., 2019. Learning implicit fields for generative shape modeling, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5939–5948.
- Chibane, J., Alldieck, T., Pons-Moll, G., 2020. Implicit functions in feature space for 3d shape reconstruction and completion, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6970–6981.
- Choi, Y.K., Wang, W., Liu, Y., Kim, M.S., 2006. Continuous collision detection for two moving elliptic disks. *IEEE Transactions on Robotics* 22, 213–224.
- Coumans, E., Bai, Y., 2016–2021. Bullet 3 physics sdk (bullet physics library). <http://pybullet.org>.
- Davies, T., Nowrouzezahrai, D., Jacobson, A., 2020. On the effectiveness of weight-encoded neural implicit 3d shapes. arXiv preprint arXiv:2009.09808 .
- Driess, D., Ha, J.S., Toussaint, M., Tedrake, R., 2022. Learning models as functionals of signed-distance fields for manipulation planning, in: *Conference on Robot Learning*, PMLR. pp. 245–255.
- Du, P., Liu, E.S., Suzumura, T., 2017. Parallel continuous collision detection for high-performance gpu cluster, in: *Proceedings of the 21st ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 1–7.
- Dube, C., Tsoeu, M., Tapon, J., 2011. A model of the humanoid body for self collision detection based on elliptical capsules, in: *2011 IEEE International Conference on Robotics and Biomimetics*, IEEE. pp. 2397–2402.
- Ferguson, Z., Li, M., Schneider, T., Gil-Ureta, F., Langlois, T., Jiang, C., Zorin, D., Kaufman, D.M., Panozzo, D., 2021. Intersection-free rigid body dynamics. *ACM Transactions on Graphics (SIGGRAPH)* 40, Article No.: 183.
- Fernández-Layos, P.L.A., Merchante, L.F., 2024. Convex body collision detection using the signed distance function. *Computer-Aided Design* , 103685.
- Flöry, S., Hofer, M., 2010. Surface fitting and registration of point clouds using approximations of the unsigned distance function. *Computer Aided Geometric Design* 27, 60–77.
- Friskén, S.F., Perry, R.N., Rockwood, A.P., Jones, T.R., 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*, pp. 249–254.
- Fuhrmann, A., Sobotka, G., Groß, C., 2003. Distance fields for rapid collision detection in physically based modeling, in: *Proceedings of GraphiCon*, pp. 58–65.
- Gilbert, E.G., Johnson, D.W., Keerthi, S.S., 1988. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation* 4, 193–203.
- Guendelman, E., Bridson, R., Fedkiw, R., 2003. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics (TOG)* 22, 871–878.
- Huynh, J., 2009. Separating axis theorem for oriented bounding boxes. <https://jkh.me/files/tutorials/Separating%20Axis%20Theorem%20for%20Oriented%20Bounding%20Boxes.pdf>.
- Jia, X., Choi, Y.K., Mourrain, B., Wang, W., 2011. An algebraic approach to continuous collision detection for ellipsoids. *Computer Aided Geometric Design* 28, 164–176.
- Jiang, Y., Ji, D., Han, Z., Zwicker, M., 2020. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1251–1261.
- Ketchel, J., Larochelle, P., 2006. Collision detection of cylindrical rigid bodies for motion planning, in: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, IEEE. pp. 1530–1535.
- Koschier, D., Deul, C., Brand, M., Bender, J., 2017. An hp-adaptive discretization algorithm for signed distance field generation. *IEEE Transactions on Visualization and Computer Graphics* 23, 2208–2221.
- Laidlaw, D.H., Trumbore, W.B., Hughes, J.F., 1986. Constructive solid geometry for polyhedral objects, in: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 161–170.
- Li, T., Wen, X., Liu, Y.S., Su, H., Han, Z., 2022. Learning deep implicit functions for 3d shapes with dynamic code clouds, in: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12840–12850.
- Macklin, M., 2022. Warp: A high-performance python framework for gpu simulation and graphics. <https://github.com/nvidia/warp>. NVIDIA GPU Technology Conference (GTC).
- Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S., Corse, Z., 2020. Local optimization for robust signed distance field collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 1–17.
- Mamou, K., Lengyel, E., Peters, A., 2016. Volumetric hierarchical approximate convex decomposition, in: *Game Engine Gems 3*. AK Peters, pp. 141–158.
- Montanari, M., Petrinic, N., Barbieri, E., 2017. Improving the gjk algorithm for faster and more reliable distance queries between convex objects. *ACM Transactions on Graphics (TOG)* 36, 1–17.

- Moore, R.E., Yang, C., 1996. Interval Analysis. volume 2. Prentice-Hall Englewood Cliffs, NJ.
- Ortiz, J., Clegg, A., Dong, J., Sucar, E., Novotny, D., Zollhoefer, M., Mukadam, M., 2022. isdf: Real-time neural signed distance fields for robot perception. arXiv preprint arXiv:2204.02296 .
- Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S., 2019. Deepsdf: Learning continuous signed distance functions for shape representation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 165–174.
- Quilez, I., 2013. Shadertoy. <https://www.shadertoy.com/user/iq/>.
- Quilez, I., 2019. Happy jumping. <https://www.shadertoy.com/view/3lsSzf>.
- Redon, S., Kheddar, A., Coquillart, S., 2002. Fast continuous collision detection between rigid bodies, in: Computer graphics forum, Wiley Online Library. pp. 279–287.
- Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., Maji, S., 2018. Csgnet: Neural shape parser for constructive solid geometry, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5515–5523.
- Sharp, N., Jacobson, A., 2022. Spelunking the deep: Guaranteed queries on general neural implicit surfaces via range analysis. ACM Transactions on Graphics (TOG) 41, 1–16.
- Sitzmann, V., Chan, E., Tucker, R., Snavely, N., Wetzstein, G., 2020. Metasdf: Meta-learning signed distance functions. Advances in Neural Information Processing Systems 33, 10136–10147.
- Stier, N., Rich, A., Sen, P., Höllerer, T., 2021. Vortex: Volumetric 3d reconstruction with transformers for voxelwise view selection and fusion, in: 2021 International Conference on 3D Vision (3DV), IEEE Computer Society. pp. 320–330.
- Stol, J., De Figueiredo, L.H., 1997. Self-validated numerical methods and applications, in: Monograph for 21st Brazilian Mathematics Colloquium, IMPA, Rio de Janeiro. Citeseer.
- Takikawa, T., Litalien, J., Yin, K., Kreis, K., Loop, C., Nowrouzezahrai, D., Jacobson, A., McGuire, M., Fidler, S., 2021. Neural geometric level of detail: Real-time rendering with implicit 3d shapes, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11358–11367.
- Tan, Q., Zhou, Y., Wang, T., Ceylan, D., Sun, X., Manocha, D., 2022. A repulsive force unit for garment collision handling in neural networks, in: European Conference on Computer Vision, Springer. pp. 451–467.
- Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.P., Faure, F., Magnenat-Thalmann, N., Strasser, W., et al., 2005. Collision detection for deformable objects. Computer Graphics Forum 24, 61–81.
- Wang, B., Ferguson, Z., Schneider, T., Jiang, X., Attene, M., Panozzo, D., 2021. A large-scale benchmark and an inclusion-based algorithm for continuous collision detection. ACM Transactions on Graphics (TOG) 40, 1–16.
- Wang, H., 2014. Defending continuous collision detection against errors. ACM Transactions on Graphics (TOG) 33, 1–10.
- Xu, H., Barbic, J., 2014. Continuous Collision Detection between Points and Signed Distance Fields, in: Bender, J., Duriez, C., Jaillet, F., Zachmann, G. (Eds.), Workshop on Virtual Reality Interaction and Physical Simulation, The Eurographics Association.
- Xu, H., Barbič, J., 2016. 6-dof haptic rendering using continuous collision detection between points and signed distance fields. IEEE Transactions on Haptics 10, 151–161.
- Yang, K., Chen, X., 2021. Unsupervised learning for cuboid shape abstraction via joint segmentation from point clouds. ACM Transactions on Graphics (TOG) 40, 1–11.
- Yao, G., Wu, H., Yuan, Y., Zhou, K., 2021. Dd-nerf: Double-diffusion neural radiance field as a generalizable implicit body representation. arXiv preprint arXiv:2112.12390 .
- Zesch, R.S., Modi, V., Sueda, S., Levin, D.I., 2023. Neural collision fields for triangle primitives, in: SIGGRAPH Asia 2023 Conference Papers, pp. 1–10.
- Zheng, C., James, D.L., 2012. Energy-based self-collision culling for arbitrary mesh deformations. ACM Transactions on Graphics (TOG) 31, 1–12.