

Neural Collision Fields for Triangle Primitives

Ryan S. Zesch Texas A&M University USA rzesch@tamu.edu

Shinjiro Sueda Texas A&M University USA sueda@tamu.edu Vismay Modi University of Toronto Canada vismay@cs.toronto.edu

David I.W. Levin University of Toronto Canada NVIDIA Canada diwlevin@cs.toronto.edu

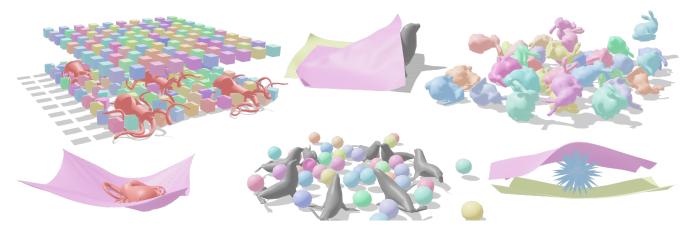


Figure 1: We propose a new smoothed surface integral formulation for collision detection and resolution between triangle meshes, along with a neural integrated triangle-triangle collision primitive for use during dynamic simulations of elastica. This figure shows a collection of large-scale simulations carried out using our method.

ABSTRACT

We present neural collision fields as an alternative to contact point sampling in physics simulations. Our approach is built on top of a novel smoothed integral formulation for the contact surface patches between two triangle meshes. By reformulating collisions as an integral, we avoid issues of sampling common to many collisionhandling algorithms. Because the resulting integral is difficult to evaluate numerically, we store its solution in an integrated neural collision field — a 6D neural field in the space of triangle pair vertex coordinates. Our network generalizes well to new triangle meshes without retraining. We demonstrate the effectiveness of our method by implementing it as a constraint in a position-based dynamics

This work is licensed under a Creative Commons Attribution International 4.0 License.

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia © 2023 Copyright held by the owner/author(s). ACM ISBN 979-8-4007-0315-7/23/12. https://doi.org/10.1145/3610548.3618225 framework and show that our neural formulation successfully handles collisions in practical simulations involving both volumetric and thin-shell geometries.

CCS CONCEPTS

 \bullet Computing methodologies \rightarrow Collision detection; Neural networks.

KEYWORDS

Collision Detection; Collision Resolution; Neural Network

ACM Reference Format:

Ryan S. Zesch, Vismay Modi, Shinjiro Sueda, and David I.W. Levin. 2023. Neural Collision Fields for Triangle Primitives. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/ 3610548.3618225

1 INTRODUCTION

Collision handling of triangle meshes is critical to physics simulation, movement planning, and robotics, among many other fields. Collision handling algorithms for polygonal meshes rely on the availability of two low-level operations. The first is an intersection query, which reports whether two polygonal primitives are in contact, and the second is a method for generating appropriate constraints or penalty energies if contact has occurred. Principally, this amounts to being able to evaluate a scalar function that represents some measure of distance between mesh primitives, and its gradient. The piecewise nature of triangle meshes along with the presence of features such as sharp points at vertices and creases at edges makes designing these functions difficult. The need for a (potentially approximate) gradient direction means suitable collision functions do more work than geometrically intersecting two triangles. And while contact point sampling is a long-studied problem in graphics, a recent work by Erleben [2018] shows that even relatively modern solutions suffer from artifacts in many cases.

Collision handling methods for triangle meshes fall into three common paradigms: point sampling, signed distance fields, and barrier functions. Point sampling is perhaps the oldest and most used technique. In the point sampling paradigm, once a triangletriangle collision has been confirmed, contact point samples are generated at the intersection of the two primitives. These contact samples are used as either attachment points for penalty springs, or as pairwise contact constraints. Choosing sampling points correctly is difficult, and there is no accepted perfect algorithm for this task. Contact point sampling requires handling special cases such as vertex-face, vertex-edge, and edge-edge collisions carefully, so as to ensure that these contacts are resolved properly by a downstream simulation algorithm. Simulation results can vary significantly with sampling quality and density. In recognition of this challenge, evaluation methodologies for contact point sampling schemes have been proposed to vet various approaches [Erleben 2018].

Sampling based on signed distance fields (SDFs) has been proposed to mitigate the above difficulties with per-primitive pointbased sampling [Macklin et al. 2020]. An SDF stores the minimum distance to a mesh at any point in nearby space and contact samples can be generated by sampling at the minimum distance on a colliding triangle. The SDF also provides a means of checking for contact, and the gradient of an SDF can be used to calculate the contact normal. The limitation of SDF sampling is the SDF itself, which requires recomputation if an object deforms significantly, though some methods sidestep this issue by transforming to a material space SDF [Macklin et al. 2020]. Even on a GPU, SDF recomputation for deformable objects is slow as it requires propagating information from the boundary of the object into the ambient space. Further, by definition an SDF requires an object to have a well-defined inside and outside, making them unsuitable for thin objects such as cloth. Minimum distance fields, like SDFs, are also non-smooth and this non-smoothness can have a deleterious effect on optimization-based solver convergence.

The most recent additions to the pantheon of contact handling methods are barrier methods, which prevent collisions by defining barrier potentials between all objects in a scene. The Incremental Potential Contact (IPC) algorithm [Li et al. 2020] defines log barrier potentials between minimum distances on mesh primitives (faces and edges). IPC is highly robust and, using these potentials, coupled with judicious use of continuous collision detection, is able to guarantee non-interpenetration during simulation. However, the IPC barriers, defined using minimum distances, suffer from similar engineering difficulties as the methods above—one must have barriers that handle collisions between different mesh components (vertices, faces, and edges) and non-smoothness of minimum distances requires extra mollification to avoid convergence issues.

In this paper, we propose Neural Collision Fields for handling triangle-triangle collisions. We base our method around an integrated formulation of the contact patch between two potentially colliding triangles and couple it with a neural integration scheme for efficient evaluation that stores the results of contact-surface integration in a Neural Collision Field – a 6D neural field in the space of triangle pair vertex positions. Our method acts as a blurred contact area between primitives - it is not a binary collision indicator of if primitives are intersecting. We show that neural fields provide both a suitable collision distance-like approximation and gradient for physics simulation. This field implicitly handles point-face, point-edge, edge-edge, edge-face, and face-face triangle collisions without special casing. We train the neural field once, on a large database of triangle-triangle pairs, and it can then be applied to novel triangle meshes without further modification. Incorporating our primitive into a simulation code base is simple, as one only requires the weights for the network and code to evaluate the network value and its gradient on whatever hardware is being used. As a proof-of-concept, we integrate our method into an open-source Position-Based Dynamics framework [Bender 2022] and perform a number of simulations ranging from small contact unit tests to large-scale scenes. We demonstrate that our method handles both the Erleben benchmarks [2018] and more complicated tests, and that it is suitable for simulations featuring both volumetric and thin-shell objects.

Since our method works in the spatial domain, it may miss collisions occurring between time steps. However, our method is compatible with standard continuous collision detection techniques.

Our main contributions can be summarized as:

- A sampling-free, surface integral based collision formulation
- A fast, smooth, neural representation of the integral
- A large dataset of intersecting triangles.

2 RELATED WORK

Mechanisms for detecting collisions and deriving per-primitive constraints for collision resolution are many and varied. In this section, we limit discussion to those that are used in penalty- or constraintbased dynamic simulations of rigid or deformable objects. Because our contributions center around primitive-primitive collision resolution (narrow phase collision detection) we omit work on fast broad phase for collision detection which is, in principle, compatible with our neural collision primitive. For more details, we refer the reader to the survey papers in graphics [Lin and Gottschalk 1998; Teschner et al. 2005; Wang and Cao 2021], robotics [Haddadin et al. 2017], and HCI [Lazaridis et al. 2021].

2.1 Mesh Sampling Collisions

Point sampling mesh primitives is the most commonly encountered approach in simulation software used today. Erleben [2018] provides an excellent overview of point sampling for contact and highlights the difficulty of the problem—handling special cases Neural Collision Fields for Triangle Primitives

for point and edge collision queries, choosing the correct number of points to generate, and computing appropriate contact normals from those samples. This is compounded by the fact that, if one is not careful, sampling-based contact resolution can be mesh dependent—producing different simulation results depending on the resolution of the simulation meshes (Fig. 9). Our work proposes a neural blackbox replacement to collision sampling schemes that avoids or significantly reduces these issues.

2.2 Signed Distance Field Collisions

Signed Distance Fields (SDFs) are often employed to improve over contact handling schemes that operate only on triangle meshes. Macklin et al. [2020] perform optimization over an SDF in order to determine contact points. They aim to solve issues of contact point sampling in collision detection, such as failing to sample small geometric details. Their focus is on point-SDF contact, and their method is limited to a single collision point per primitive, though subdivision may be used to handle multiple contacts. Strecke and Stueckler [2021] use a differentiable SDF for contacts in order to run a differentiable rigid body simulation. Rigid body simulations are an ideal application domain for SDF collisions as computing or updating an SDF can be time-consuming. As such, the above methods are difficult to apply to deformable object simulation, a limitation our neural collision primitive avoids.

2.3 Barrier Collisions

Barrier methods based on Incremental Potential Contact (IPC) formulation by Li et al. [2020] have quickly proliferated throughout graphics due to their non-interpenetration guarantees. IPC methods use a smoothed barrier method, coupled with continuous collision detection to ensure that models remain intersection-free. This methodology has been successfully applied to co-dimensional [Li et al. 2021], rigid [Ferguson et al. 2021], and quasi-rigid objects [Lan et al. 2022a] as well. IPC-like barriers have also been incorporated in fast, Projective Dynamics solvers on the GPU by Lan et al. [2022b]. In terms of quality of output, IPC methods are currently peerless, barring difficulties in friction modeling. Their primary limitations lie on the theoretical and engineering sides. One must have barriers that handle collisions between different mesh components (vertices, faces, and edges) and non-smoothness of barriers based on minimum distances requires extra mollification to avoid convergence issues. Our neural contact primitives avoid these issues as implementation only requires evaluating the network and its gradient, with no special casing for contact type required.

2.4 Neural Collisions

We are (obviously) not the first to apply neural networks to collision processing. Park et al. [2019] learn neural representations of classes of objects as SDFs, using a latent space to move between different objects within a class. This method has been applied to deformable physics simulation collisions by Zesch et al. [2022] by using linear modes of deformation or joint angles as a network input. Similar work by Cai et al. [2022] based on linear modes of deformation focused on self-collisions. Zesch et al. [2023] likewise use linear modes of deformation for neural broad phase collision detection. Since these works learn reduced space deformations through linear modes, they are not applicable to larger non-linear deformations, such as for cloth. These networks must be fine-tuned per shape, which is computationally expensive.

In recent neural simulators, general physical neural constraints are learned, including collision constraints [Yang et al. 2020; Rubanova et al. 2021; Alldieck et al. 2021]. These approaches similarly must be fined-tuned per scene, failing to be a broadly generalizable collision solution. Romero et al. [2022] use a collider space deformation network to apply full space deformations on top of subspace-simulated deformable objects. However, this approach focuses more on producing visually pleasing deformations due to contact rather than tackling difficulties in collision constraint generation.

While our method does rely on a neural network, it is significantly different from standard neural approaches to collision dominated simulation. Rather than learning whole object response, we exploit the fact that we can generate arbitrarily large datasets of triangle-triangle interactions and have the network memorize the resulting collision integrals. This allows our single network to be reused for new triangle-triangle collision operations inside of a simulation algorithm, without retraining no matter what meshes are present.

2.5 Continuous Collision Detection

Many simulation algorithms augment their collision resolution with continuous collision detection (CCD). CCD prevents collisions resulting from the movement of vertices between time steps. Brochu et al. [2012] perform CCD, yielding a guaranteed correct boolean result by considering the parity of roots of a raycast against a parameter polyhedra. This approach is used to address floating point errors of other methods. This method is not robust to degenerate configurations. Wang et al. [2022] solve this degeneracy issue. This method is comparable in runtime to other CCD methods, but is held back by not predicting collision time itself. Ferguson et al. [2021] include curved CCD for rigid body trajectories, and Lan et al. [2022a] improve contact culling in CCD. Li et al. [2021] propose additive CCD, based on conservative advancement [Mirtich 1996], in order to handle codimensional objects with thickness inside of IPC. Our method is not a replacement for CCD as it is a spatial, rather than spatial-temporal, collision primitive. However, like other approaches discussed above, it can be complemented by CCD to improve robustness.

3 METHODS

We present a novel collision formulation based on surface integrals as an alternative to traditional methods. By taking a primitive-first approach, we are able to compute these surface integrals over arbitrary deformable objects. In order to make integration possible, we utilize neural networks to learn our integrated potential function.

3.1 Integral Formulation

The total contact area between two arbitrary surfaces in world space can be described as an integral

$$V = \iint_{\Omega_A} \iint_{\Omega_B} \delta(||\mathbf{x}_A(\alpha) - \mathbf{x}_B(\beta)||^2) \, d\Omega_B d\Omega_A, \tag{1}$$

where $\delta(\cdot)$ is an indicator function, and $\mathbf{x}_A(\alpha), \mathbf{x}_B(\beta)$ are points on surfaces Ω_A and Ω_B , parameterized by α, β . We define our indicator as

$$\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0. \end{cases}$$
(2)

If our surfaces are triangle meshes this integral becomes

$$V = \sum_{i}^{N} \sum_{j}^{M} \iint_{A} \iint_{B} \delta(||\mathbf{x}_{A}(\alpha) - \mathbf{x}_{B}(\beta)||^{2}) \, dB \, dA, \qquad (3)$$

where we are summing over the two meshes with N and M triangles, and the integral is now over individual triangles A and B.

Let $\alpha \in \mathbb{R}^2$ be a set of barycentric coordinates for A, and $\beta \in \mathbb{R}^2$ a set of barycentric coordinates for B. Given two points represented by α and β , we utilize a negative exponential function $f(\cdot)$ to smooth the discontinuous function $\delta(\cdot)$, allowing us to determine if these points are within a tolerance of each other:

$$f(\mathbf{x}_A, \mathbf{x}_B) = \exp(-k \cdot ||\mathbf{x}_A(\alpha) - \mathbf{x}_B(\beta)||^2), \tag{4}$$

where $\mathbf{x}_A(\alpha) \in \mathbb{R}^3$ and $\mathbf{x}_B(\beta) \in \mathbb{R}^3$ are the points on the two triangles at the respective barycentric coordinates, and the parameter k > 0 can be used to tune how sharp this exponential is. In order to extend this function to triangle primitives, we integrate it over all barycentric coordinates of *A* and *B* using standard barycentric integration. This yields a final integral expression for a triangle pair, $V : \mathbb{R}^{18} \mapsto \mathbb{R}$,

$$V(p_A, p_B) = \iint_A \iint_B \exp(-k \cdot ||\mathbf{x}_A(\alpha) - \mathbf{x}_B(\beta)||^2) \, dB \, dA, \quad (5)$$

where p_A and p_B are 9D vectors representing the vertices of A and B. While this formulation implicitly handles inter-object collisions, it is easily extended to self-collisions by culling adjacent triangles. As an illustration, we show in Fig. 2 the analogous integrand over two line segment primitives.

This surface integral accomplishes two things. First, the strength of the response is determined by how in-contact the triangles A and *B* are, giving a representation of how large the collision area is. This is visualized in Fig. 3 and Fig. 20, where we color intersecting triangles based on their contribution to the integrated value. In essence, the integral can be thought of as a blurred contact area between triangles. Secondly, the gradient of this function with respect to the vertices of triangles A and B can be used to handle the resolution of collisions, as this gradient direction minimizes the blurred contact area. Unfortunately, however, this integral does not have a closed-form solution. This means that numerical integration is required for its solution. The non-polynomial nature of the integrand means that standard quadrature schemes yield poor accuracy. Gauss-Hermite methods are tailor-made for these types of integrals but require complex and time-consuming adaptive schemes to be effective. Brute force sampling is effective but slow, especially as the sharpness k increases. Therefore, we turn to neural networks as an alternative integration technique.

3.2 Neural Integration

3.2.1 Foundation. Rather than use quadrature to evaluate our collision integral, we will instead train a neural network to evaluate any triangle-triangle collision integral. The network takes as input

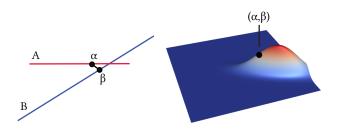


Figure 2: In a 2D analog working on line segment primitives, our integral integrates over barycentric coordinates α and β of A and B respectively. The function inside the integrand is plotted on the right, with the spike in the value corresponding to the crossing point of the line segments.

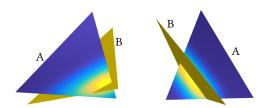


Figure 3: In this image, we color each point in triangle A based on the inner integral in our formulation, $\iint_{B} \exp(-k \cdot ||\mathbf{x}_{A}(\alpha) - \mathbf{x}_{B}(\beta)||^{2}) dB.$

the vertices of any triangle pair and produces the scalar integrated value as output. Because this network acts on triangle-triangle pairs, it can be trivially applied to any triangle mesh. Neural networks are especially well suited for our triangle pair integration. First, training data is easy to generate in large quantities, as generating triangles and computing dense approximations of their integrals is straightforward. Moreover, neural networks are known for their ability to smooth over and generalize within a dataset, which helps ensure the consistency of integral estimates. Second, we find the plug-and-play nature of this approach appealing. A network only needs to be trained once, and can be integrated into diverse applications without the need for further fine-tuning. Importantly, by focusing on primitive elements, we sidestep the issues faced by SDF neural networks, which must be fine-tuned per model before they can be deployed [Zesch et al. 2022; Alldieck et al. 2021]. Third, the gradient of the neural network can easily be computed, which can be used for collision resolution. The Hessian can be computed as well if required.

3.2.2 Data Generation. In order to generate training data, we first generate two random triangles *A* and *B* in space, utilizing a multivariate normal distribution to generate each coordinate. We then select a random barycentric coordinate on each of *A* and *B*, and align those points. In order to include nearly-intersecting triangles in our dataset, we then apply a normally distributed translation to one of the triangles. We then center the pair of triangles by moving their collective centroid to the origin. We finally normalize the neural network inputs by scaling the triangle pairs such that the furthest point has a distance of one from the origin.

Zesch, R. et al.

Neural Collision Fields for Triangle Primitives

Once triangle pairs have been generated, we compute a dense approximation of the surface integral over each triangle pair in normalized space for training. We use uniform quadrature points and weights. The number of samples depends on the stiffness k. For our simulations, which use standard meshes in graphics, we empirically determined that the values of k = 1000 and k = 1500give good results, for softer and harder responses, respectively. We find that when the stiffness is set to k = 1000, we need s = 35quadrature points along each triangle edge, yielding $35 \times 36/2$ sample points per triangle. To then compute the integral, we must compute the integrated function on each pair of points between the triangles, giving a total of $(35 \times 36/2)^2 = 396,900$ exponential function evaluations per integral. For our steeper value of k =1500, we find we need s = 40 quadrature points, giving (40 × $(41/2)^2 = 672,400$ exponential function evaluations per integral. Integral values are normalized to [0, 1] by a simple uniform scaling such that the highest integrated value in our dataset is 1.

3.2.3 Network Architecture. We find a multilayer perception (MLP) with 4 hidden layers of 512 neurons to be sufficient for learning this integral. We use ReLU intermediate activation functions and a sigmoid final activation. Triangle pairs are normalized as described above, and passed to the network as an 18D vector. In order to augment our training data, we randomly permute the triangles *A* and *B* in our input encoding. We also permute the order of the points within each triangle. Finally, we apply a random rotation to each batch. In training, we use standard L_1 loss, as we found this to work best empirically. In addition, we use an L_1 loss on the predicted values under different augmented states, in order for our network to be invariant to input orientation and permutation. We use the Adam optimizer [Kingma and Ba 2014] with a learning rate of 1E–4, and use a batch size of 4096.

3.2.4 *Training*. We train the network on an increasing stiffness k, for only a few epochs per k, ramping up to the desired stiffness, to avoid predicting 0 uniformly. In particular, we use k = 5, 50, 250, 500, 1000, 1500. If desired, this warm-starting strategy can be used to train networks with higher stiffness values. Our network is trained for only a specific k value. While training, we use early stopping with a patience of 10 epochs. We train the final network for 60 epochs, using a dataset of 500M intersecting triangle pairs for the desired stiffness value. Other similar settings may be sufficient.

3.3 Simulation

Our neural integration technique is generalizable to any type of mesh-based simulator for both rigid and deformable objects. In order to demonstrate our network, we use the open-source CPUbased Position Based Dynamics [Müller et al. 2007] framework provided by Bender [2022] as a foundation. We add our neural collisions as a PBD constraint, and apply it to highly deformable objects and cloths in order to highlight its applicability where SDFbased collisions perform poorly.

During each simulation step, we first check for potential intersections using Axis-Aligned-Bounding-Box trees for the objects. This broad-phase step provides us with pairs of nearby triangles. Next, we iterate through these pairs of triangles and create a neural collision constraint for any face pairs within a predetermined SA Conference Papers '23, December 12-15, 2023, Sydney, NSW, Australia

Table 1: Simulation settings for our examples. All simulationsuse 5 substeps per timestep.

Example	k	h	α	Iters.
Erleben	1500	5E-3s	Various	1
BoxResolution	1500	5E-3s	1E-6	1
Bars	1500	5E-3s	1E-6	1
Seals	1500	5E-3s	1E-6	1
Bunnies	1500	1E-3s	1E-6	3
OctopiCubes	1500	5E-3s	1E-6	3
OctopiSub	1500	5E-3s	1E-6	1
OctopiCloth	1000	5E-3s	1E-7	3
SealCloths	1000	3E-3s	1E-7	5
ClothArma	1500	5E-3s	1E-6	1
ClothsTorus	1500	5E-3s	1E-6	1
ClothsSpiky	1500	5E-3s	1E-9	20
ClothTris	1000	5E-3s	1E-6	3
Jumble	1500	1E-3s	1E-6	1
BigJumble	1500	1E-3s	1E-6	5
ClothTwist	1000	5E-4s	1E-6	3
PegBoard	1500	5E-4s	1E-6	5

collision threshold. Each neural collision PBD constraint

 $C(p_A, p_B) = \text{Normalize}^{-1}(\text{MLP}(\text{Normalize}(p_A, p_B)))$ (6)

takes in a pair of faces, *A*, *B*, each denoted by three points $p_A \in \mathbb{R}^9$ and $p_B \in \mathbb{R}^9$, totalling 18 DOFs per constraint. A PBD constraint compliance α is set low across all examples. We use a minimum constraint threshold, but find its effects negligible except when setting $\alpha = 0$. Before evaluating the MLP, we normalize the input as described in §3.2.2, and denormalize with this scale factor in computing the constraint *C* and its gradient ∇C . The network gradient allows for easy computation of ∇C with respect to each of the input triangle vertices. The constraint can be solved in a standard PBD fashion, using a Gauss-Seidel or Jacobi update to the positions of the vertices p_A and p_B in each of the two faces A and B. In practice, however, we batch collision queries together and solve them in parallel on the GPU, in order to increase performance. This has the additional benefit of blending collision response gradients at a single state, increasing simulation stability. While we do not use CCD in our implementation, our method is entirely compatible with traditional methods of performing CCD.

4 RESULTS

4.1 Robustness

We test our neural surface integral constraint over many test cases with various settings, as described below. Note that in our simulator, we do not model frictional contact. We demonstrate our ability to handle self-collisions in our simulations with the octopus model. Our test settings are summarized in Table 1. We use k = 1000 or 1500 for all examples. As usual, lower stiffness provides a looser collision tolerance, but allows for larger simulation steps.

4.1.1 Erleben Fundamental Test Cases. In order to verify that our method handles common problem cases in collision handling, we verify our method successfully resolves cases presented by Erleben

[2018]. We find that in all cases, we successfully resolve the collisions without issue, as seen in Fig. 8. We perform these tests on both nearly-rigid and highly deformable models, and find success in both cases.

4.1.2 Erleben Comparisons. While these cases have already been demonstrated to be hard, we verify that off-the-shelf solutions such as Blender [Community 2018] and PyBullet [Coumans and Bai 2021] face problems with some of these tests when running the same configurations as our method, with reasonable settings. In Blender, we find that while initial contacts may be handled accept-ably, subsequent edge-edge collisions can cause extremely large impulses. In PyBullet, we use a triangle mesh for concave objects rather than a convex decomposition, as this is a more reasonable comparison to our method for deformable objects. We find that in the cases with sharp objects falling on the wedge base, tunneling is common and becomes unrecoverable using standard, exact mesh-based collision approaches. Please see 1:47 in our video for these failure cases.

4.1.3 Box Resolution. In Fig. 9, we test the collision of two large nearly-rigid boxes at multiple resolutions. Traditional methods rely on the mesh vertices as sample points, so the resulting simulation is dependent on the mesh resolution. Our surface integral, however, is independent of mesh resolution, resulting in our method producing similar simulation results across different resolutions. When performing a similar simulation in Blender, we find that simulation results diverge with different mesh resolutions, as seen in our video at 1:54.

4.1.4 Various Shapes. With these examples, we verify that the same network and simulation settings can be used across different shapes. We drop 36 nearly-rigid spheres onto 8 deformable seals. We fix the back flippers of the seals to the ground in order to produce more deformations. We then drop 36 deformable bunnies on top of each other, and demonstrate that collisions are still handled correctly.

4.1.5 *Large Deformations.* We drop a deformable octopus on a submarine. We find that even at large deformations, our method is easily able to handle the collisions between objects, unlike an SDF-based approach. We also drop five octopi between spaced cubes to further demonstrate that our method handles complex deformations.

4.1.6 Cloth Examples. In Fig. 4 we drop a deformable octopus on a cloth, as well as a cloth onto a deformable octopus. While an SDF-based method is unable to handle cloth (not defined inside), our method handles these cases with no modifications. See Table 1 for simulation settings. For these simulations, we demonstrate our neural constraint's ability to prevent self-collisions of the octopus. We similarly drape a cape on an armadillo in Fig. 13. We also demonstrate that our method is able to handle cloth-cloth collisions in Fig. 14 and Fig. 15, and cloth self-collisions in Fig. 17, allowing for many simulation types to rely on a single unified collision solution. Finally, in Fig. 5 we drop many soft 2D FEM triangles onto a cloth, with the triangle normals facing horizontally. Our method is easily able to catch them all. In comparison, running a similar simulation in Blender often lets the triangles pass through the cloth unless

Figure 4: An octopus lands and bounces off of a cloth, and a cloth drapes an octopus.

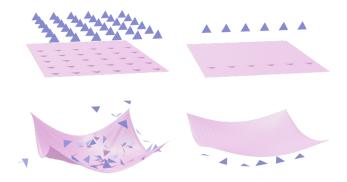


Figure 5: We drop many FEM 2D triangles onto a cloth and successfully catch them all. Our method (Left) catches them with ease, while Blender (Right) requires much fine tuning in order to avoid penetration.



Figure 6: A spiky object is sandwiched between two cloths.

collision parameters are tuned carefully, as seen in our video at 0:08. Note that in all cases, we are able to handle cloth with no modification.

4.1.7 Sharp Contacts. In Fig. 6 we demonstrate the capability of our method to handle sharp collisions. We drop a very spiky object onto a cloth, and drape a second cloth on top. We find that our method is able to prevent penetrations entirely.

Zesch, R. et al.

Table 2: A timing comparison of our neural integration against an optimized sampling method to compute $V(p_A, p_B)$. Comparisons are run in PyTorch. A batch of 256 triangle pair queries are integrated using each method, averaged over many runs. For k = 1000 we use s = 35 quadrature points, resulting in 396,900 function evaluations per query, and for k = 1500 we use s = 40 quadrature points, resulting in 672,400 function evaluations per query. Neural net timings include time to normalize input data and GPU methods include transfer times. We also include timings with an 8k batch size (*) and with constraint gradient computation timing (∇) .

	Time Per Query $(s = 35)$	Time Per Query $(s = 40)$
PyTorch CPU Sampling	1.14E-02 s	1.92E-02 s
PyTorch CPU NN	1.21E-05 s	1.15E-05 s
PyTorch GPU Sampling	1.13E-04 s	1.94E-04 s
PyTorch GPU NN	5.60E-06 s	5.55E-06 s
CUDA GPU NN	2.18E-06 s	2.36E-06 s
CUDA GPU NN*	1.33E-06 s	1.32E-06 s
CUDA GPU NN⊽	4.16E-06 s	4.31E-06 s
CUDA GPU NN ∇^*	2.48E-06 s	2.45E-06 s

4.2 Performance

We implement our method in CUDA as a custom MLP implementation. We test against PyTorch [Paszke et al. 2019] based implementations. Our timings are performed on a 4-core Intel i7-4790K CPU and an NVIDIA GeForce RTX 3070 Ti GPU.

We compare our neural network performance against an optimized sampling-based method, which was used to generate training data. These comparisons are done in PyTorch. We find that for stiffness k = 1000 or k = 1500, we require 35 or 40 quadrature points respectively per integral in order to have stable approximations. At this sample density, our network method outperforms a sampling-based approach both on the CPU and the GPU in Py-Torch. Performance data is shown in Table 2. Our CPU based PBD simulation is capable of evaluating constraints either through a Gauss-Seidel approach on the CPU, or a parallelized Jacobi solver on the GPU. This Jacobi solver for constraints allows us to batch the MLP queries, thus significantly speeding up each call. Parallelized PBD simulation on the GPU would benefit even further from this approach.

In order to have comparable run times, a sampling-based method can only use 15 quadrature points for exponential constants of both k = 1000 and k = 1500. At this number of samples, however, we find that our neural method has better accuracy than the samplebased approximation, as seen in Table 3. In practice, the effect of too sparse of sampling is that the constraint gradient is inconsistent in direction, as seen in Fig. 7, which results in poor collision resolution. Notice that in the numerical method, error accumulates in a periodic fashion when triangles are overlapping, which is detrimental to simulation, as the constraint gradient is used in collision resolution. In Fig. 21 we compare accuracies of networks trained on reduced datasets.

We now compute the number of queries that must be performed so that it is quicker to train our network and query *n* times than to simply perform a quadrature *n* times. Our k = 1500 neural network SA Conference Papers '23, December 12-15, 2023, Sydney, NSW, Australia

Table 3: We compare the accuracy of our neural integration against our sampling method, using a number of samples to match performance between the methods when querying a batch of 256 triangle pair on the GPU in PyTorch. We find that both for k = 1000 and k = 1500, our neural method outperforms the sampling method. Additionally, we compare with nets trained at k = 1500 with half the layer size and with one fewer layers.

Method	k	Samples	Runtime	Relative % Error
Sampling	1000	<i>s</i> = 15	1.81E-03 s	14.7304
NN	1000	-	1.43E-03 s	6.3518
Sampling	1500	s = 15	1.81E-03 s	19.8588
NN	1500	-	1.42E-03 s	6.8509
NN half	1500	-	1.35E-03 s	13.9281
NN -1	1500	-	1.27E-03 s	10.6444



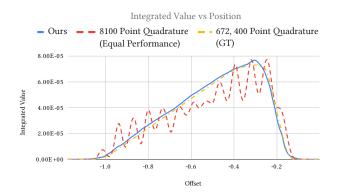


Figure 7: Two triangles rigidly slide through each other. We plot the integrated value with k = 1500 as our constant. If the numerical integral is sampled too sparsely, the gradient with respect to the triangle position is erratic, which performs poorly in simulation. Here, we use s = 15 for our numerical integral, as visualized above, in order to have the same performance between neural and sample-based integration.

took 20 hours to train on an HPC cluster. With this network, at a batch size of 256, we can query at a rate of 2.36E-06 seconds per query. Our sampling based solution takes 1.94E-04 seconds per query. Therefore after n = 20h/(1.94E-04s - 2.36E-06s) = 400M queries, we have already amortized the cost of training the network. We find this number of queries has been performed many times over in the process of generating images and videos for this paper.

5 CONCLUSION & FUTURE WORK

5.1 Conclusion

We have presented a novel contact formulation based on Neural Collision Fields and have demonstrated its use as a *unified* collision handler for a variety of simulation scenarios.

5.2 Future Work and Limitations

There are some limitations to our neural approach. First, although we showed that our approach works well with cloth, if the penetrations are large, the constraints can start to push models into each other. Higher stiffness and/or CCD are needed to mitigate these issues. Also, since our constraints become active slightly before the primitives come into contact, the broad phase must be padded with looser bounds, as in IPC.

There are many directions in which our work could be extended. One could consider formulating similar surface integrals over other geometric primitives (*e.g.*, quadrilaterals, NURBS patches). These could include integrals that are sensitive to inside/outside information, or which are sided. Additionally, one could consider using our methods to develop a neurally integrated continuous collision detection scheme over primitive elements. One might extend our model to handle friction between primitives, either in the integral formulation itself or as an additional parameter in the neural approach. Excitingly, these future additions would be easy to import into any simulation engine running neural collision primitives as at worst, the architecture and (at best) the weights would need to be updated. This opens the door to simple over-the-air updates of collision handling in simulations running in games and other applications.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This work was sponsored in part by the National Science Foundation (#1846368 & #2313076), NSERC Discovery, the Ontario Early Researchers Award program, The Canada Research Chairs Program, and gifts from Adobe Research and Autodesk. R. Zesch was supported by an NVIDIA internship. Portions of this research were conducted with the advanced computing resources provided by Texas A&M High Performance Research Computing.

REFERENCES

- Thiemo Alldieck, Hongyi Xu, and Cristian Sminchisescu. 2021. imGHUM: Implicit generative models of 3d human shape and articulated pose. In Proceedings of the IEEE/CVF International Conference on Computer Vision. 5461–5470.
- Jan Bender. 2022. Position Based Dynamics. https://github.com/ InteractiveComputerGraphics/PositionBasedDynamics
- Tyson Brochu, Essex Edwards, and Robert Bridson. 2012. Efficient geometrically exact continuous collision detection. ACM Transactions on Graphics (TOG) 31, 4 (2012), 1–7.
- Xinhao Cai, Eulalie Coevoet, Alec Jacobson, and Paul Kry. 2022. Active Learning Neural C-space Signed Distance Fields for Reduced Deformable Self-Collision. In Graphics Interface 2022.
- Blender Online Community. 2018. Blender a 3D modelling and rendering package. Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www. blender.org
- Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.
- Kenny Erleben. 2018. Methodology for assessing mesh-based contact point methods. ACM Transactions on Graphics (TOG) 37, 3 (2018), 1–30.
- Zachary Ferguson, Minchen Li, Teseo Schneider, Francisca Gil-Ureta, Timothy Langlois, Chenfanfu Jiang, Denis Zorin, Danny M. Kaufman, and Daniele Panozzo. 2021.

Intersection-free Rigid Body Dynamics. ACM Transactions on Graphics (SIGGRAPH) 40, 4, Article 183 (2021).

- Sami Haddadin, Alessandro De Luca, and Alin Albu-Schäffer. 2017. Robot collisions: A survey on detection, isolation, and identification. *IEEE Transactions on Robotics* 33, 6 (2017), 1292–1312.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014).
- Lei Lan, Danny M Kaufman, Minchen Li, Chenfanfu Jiang, and Yin Yang. 2022a. Affine body dynamics: Fast, stable & intersection-free simulation of stiff materials. arXiv preprint arXiv:2201.10022 (2022).
- Lei Lan, Guanqun Ma, Yin Yang, Changxi Zheng, Minchen Li, and Chenfanfu Jiang. 2022b. Penetration-free projective dynamics on the GPU. ACM Transactions on Graphics (TOG) 41, 4 (2022), 1–16.
- Lazaros Lazaridis, Maria Papatsimouli, Konstantinos-Filippos Kollias, Panagiotis Sarigiannidis, and George F Fragulis. 2021. Hitboxes: A Survey About Collision Detection in Video Games. In International Conference on Human-Computer Interaction. Springer, 314–326.
- Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R 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 (2020), 49.
- Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. ACM Trans. Graph. (SIGGRAPH) 40, 4, Article 170 (2021).
- Ming Lin and Stefan Gottschalk. 1998. Collision detection between geometric models: A survey. In Proc. of IMA conference on mathematics of surfaces, Vol. 1. 602–608.
- 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 (2020), 1–17.
- Brian Vincent Mirtich. 1996. Impulse-based dynamic simulation of rigid body systems. University of California, Berkeley.
- Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. Journal of Visual Communication and Image Representation 18, 2 (2007), 109–118.
- Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning continuous signed distance functions for shape representation. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 165–174.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-stylehigh-performance-deep-learning-library.pdf
- Cristian Romero, Dan Casas, Maurizio M Chiaramonte, and Miguel A Otaduy. 2022. Contact-centric deformation learning. ACM Transactions on Graphics (TOG) 41, 4 (2022), 1–11.
- Yulia Rubanova, Alvaro Sanchez-Gonzalez, Tobias Pfaff, and Peter Battaglia. 2021. Constraint-based graph network simulator. arXiv preprint arXiv:2112.09161 (2021).
- Michael Strecke and Joerg Stueckler. 2021. DiffSDFSim: Differentiable rigid-body dynamics with implicit shapes. In 2021 International Conference on 3D Vision (3DV). IEEE, 96–105.
- Matthias Teschner, Stefan Kimmerle, Bruno Heidelberger, Gabriel Zachmann, Laks Raghupathi, Arnulph Fuhrmann, M-P Cani, François Faure, Nadia Magnenat-Thalmann, Wolfgang Strasser, et al. 2005. Collision detection for deformable objects. In *Computer graphics forum*, Vol. 24. Wiley Online Library, 61–81.
- Bolun Wang, Zachary Ferguson, Xin Jiang, Marco Attene, Daniele Panozzo, and Teseo Schneider. 2022. Fast and Exact Root Parity for Continuous Collision Detection. In Computer Graphics Forum, Vol. 41. Wiley Online Library, 355–363.
- Monan Wang and Jiaqi Cao. 2021. A review of collision detection for deformable objects. Computer Animation and Virtual Worlds 32, 5 (2021), e1987.
- Shuqi Yang, Xingzhe He, and Bo Zhu. 2020. Learning physical constraints with neural projections. Advances in Neural Information Processing Systems 33 (2020), 5178– 5189.
- Ryan S. Zesch, Bethany R. Witemeyer, Ziyan Xiong, David I.W. Levin, and Shinjiro Sueda. 2022. Neural Collision Detection for Deformable Objects. arXiv preprint arXiv:2202.02309 (2022).
- Ryan S. Zesch, Bethany R. Witemeyer, Ziyan Xiong, David I.W. Levin, and Shinjiro Sueda. 2023. NBD-Tree: Neural Bounded Deformation Tree for Collision Culling of Deformable Objects. In Southwest Data Science Conference.

Neural Collision Fields for Triangle Primitives

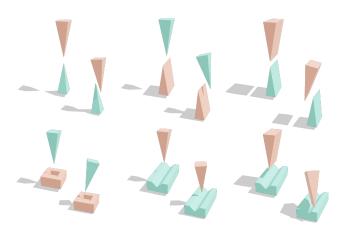


Figure 8: We successfully handle the fundamental cases presented by Erleben [2018].

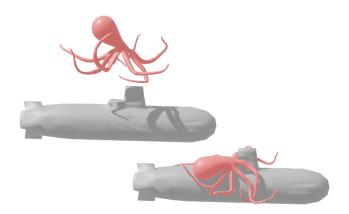


Figure 11: A deformable octopus is dropped onto a nearly rigid sub. Tentacles deform and wrap around the sub.

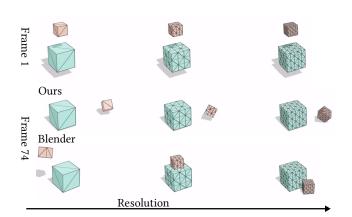


Figure 9: We remesh a box to have 1, 3, or 5 triangles per edge. Our method produces visually similar results during simulation regardless of mesh resolution, while traditional methods often have larger variance in outcome.

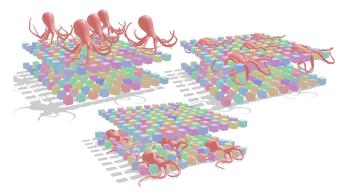


Figure 12: Five deformable octopi slide through a grid of fixed cubes onto a grid of deformable cubes.



Figure 13: An armadillo has a cloth draped over it like a cloak.

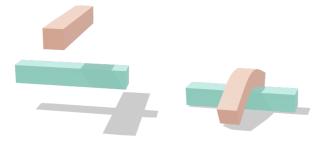


Figure 10: Two soft clay bars are dropped onto each other and deform after the collision.



Figure 14: A deformable seal has two cloths draped over it.

Zesch, R. et al.



Figure 15: We drape a cloth over a torus onto a second cloth, and handle this cloth-cloth contact with ease.



Figure 16: Eight deformable seals have 36 spheres dropped onto them.



Figure 17: We twist a cloth and handle all self collisions. We compare our simulation against an equivalent one in Blender in our supplemental material.



Figure 18: We drop nearly-rigid bunnies through a peg board. We compare our simulation against equivalent ones in Blender and Bullet in our supplemental material.



Figure 19: We drop over 1000 deformable objects into a pile, and successfully resolve all collisions.

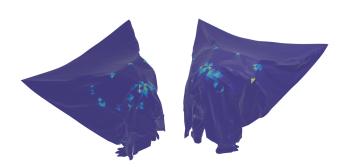


Figure 20: A heat map of integrated values in the armadillo cloth scene, Fig. 13. Triangles are colored by the total sum of all integrals to which they contribute, as predicted by our trained model.

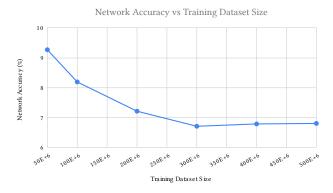


Figure 21: We train networks on subsets of our data set of various sizes, and compare the learned accuracy of networks trained on these subsets.

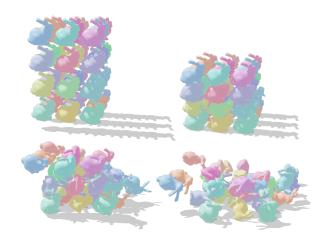


Figure 22: Thirty six deformable bunnies collide with each other.