Fast Evaluation of Smooth Distance Constraints on Co-Dimensional Geometry

ABHISHEK MADAN, University of Toronto, Canada
DAVID I.W. LEVIN, University of Toronto, Canada

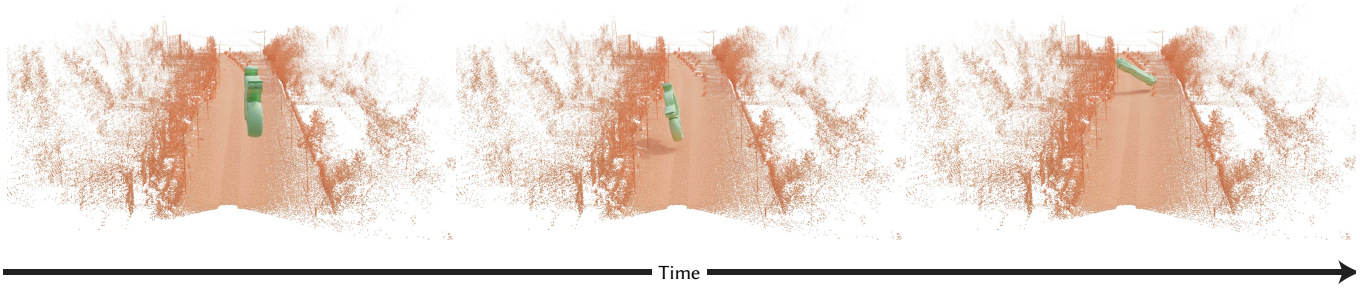


Fig. 1. Using a smooth distance function as an intersection-free constraint, we can simulate rigid body contact between a variety of meshes. Here, a motorcycle jumps onto a point cloud street, where it slides and crashes through the street, hitting electrical poles along the way. Collisions are resolved using a single inequality constraint in a primal-dual interior-point solver.

We present a new method for computing a smooth minimum distance function based on the LogSumExp function for point clouds, edge meshes, triangle meshes, and combinations of all three. We derive blending weights and a modified Barnes-Hut acceleration approach that ensure our method approximates the true distance, and is conservative (points outside the zero isosurface are guaranteed to be outside the surface) and efficient to evaluate for all the above data types. This, in combination with its ability to smooth sparsely sampled and noisy data, like point clouds, shortens the gap between data acquisition and simulation, and thereby enables new applications such as direct, co-dimensional rigid body simulation using unprocessed lidar data.

CCS Concepts: • **Computing methodologies** → **Collision detection**.

Additional Key Words and Phrases: smooth distances, co-dimensional geometry

ACM Reference Format:
Abhishek Madan and David I.W. Levin. 2022. Fast Evaluation of Smooth Distance Constraints on Co-Dimensional Geometry. *ACM Trans. Graph.* 41, 4, Article 68 (July 2022), 17 pages. https://doi.org/10.1145/3528223.3530093

1 INTRODUCTION

Distance fields are integral to many applications in computer graphics and scientific computing. In rendering, distance fields provide an implicit shape representation that enables both flexible editing and fast display, while in physics simulation they provide a convenient way to represent distance-mediated interactions between simulated objects — such as collisions. Any geometric representation can be converted into a distance field, whether it be a point cloud, edge mesh, or triangle mesh. Thus, algorithms that rely on distance field representations are theoretically invariant to input geometry type. This is important because many applications of geometry processing and physics simulation act on mixed geometric input (e.g., self-driving car simulations represent the cars as polygonal models but the environment is acquired as a point cloud via lidar scan).

Unfortunately, current distance field representations fall short of living up to these theoretical advantages. Storing distance fields on grids is memory intensive and can require costly preprocessing, while fitting neural networks alleviates the memory pressure but requires a much higher upfront cost in training time and data consumption, while also being difficult to generalize. Complicating proceedings is the fact that, often, representing the exact distance field is not ideal for practical applications since input geometric models are usually an approximation of the underlying true object. Representing any curved, smooth surface using piecewise linear triangles is an obvious example, but noisy or incomplete data, like lidar point clouds, is another. While methods exist for accurately simulating the latter (Fig. 2 reproduced from Ferguson et al. [2021]), the result is not generally applicable to cases where discrete samples are meant to coalesce into a smooth surface.

Further complicating matters is the fact that exact distance fields are typically not smooth, which limits the choice of algorithms that can be applied. Finally, existing geometry processing pipelines are often set up to receive closed triangle mesh input only [Li et al. 2020a], with co-dimensional inputs (inputs that feature a mixture of triangles, edges, and points) considered special cases [Li et al. 2021].

Authors' addresses: Abhishek Madan, University of Toronto, Toronto, Canada, amadan@cs.toronto.edu; David I.W. Levin, University of Toronto, Toronto, Canada, diwlevin@cs.toronto.edu.

0730-0301/2022/7-ART68 $15.00
https://doi.org/10.1145/3528223.3530093

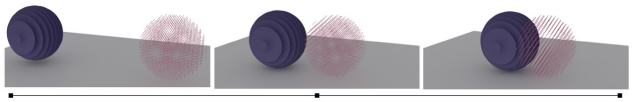


Fig. 2. Using Rigid IPC [Ferguson et al. 2021], a collision between a sphere of points and a sphere of disjoint planes causes them to lock together on impact. While an impressive demonstration of robustness, this result is counter intuitive if the point cloud were meant to represent a solid sphere.

In this paper we present a smooth distance formulation that addresses these issues while simultaneously guaranteeing theoretical properties that are crucial for distance fields to function robustly in rendering and simulation applications. Specifically, our method

- is an implicit function, which naturally fits in an optimization context;
- is an approximation of the exact unsigned minimum distance between two geometric quantities;
- can represent different types of geometry (in this paper we focus on points, edges, and triangles), and can represent both closed and open or co-dimensional geometry;
- is smooth (i.e., able to take derivatives), which is beneficial for optimization;
- is efficient to evaluate;
- conservatively estimates (i.e., underestimates) exact distance.

We achieve these goals by using a smooth minimum distance function based on LogSumExp (sometimes called Kreisselmeier-Steinhauser distance), augmented by weight functions to remove bulge artifacts from edge and triangle mesh isosurfaces, and a conservative Barnes-Hut approximation to speed up function evaluations while inducing slight discontinuities at the near field-far field boundary (see Section 4.3). We demonstrate the efficacy and ease-of-use of our smooth distance field representation on a number of colliding rigid body simulations which directly act on co-dimensional geometry, including difficult cases such as collision-mediated interaction with lidar data featuring millions of points. Our approach could be a drop in improvement to many existing computer graphics applications as well as a major step forward for cutting-edge pursuits such as the direct simulation of self-driving cars in lidar environments.

2 RELATED WORK

The LogSumExp function is commonly used in deep learning (see, e.g., [Zhang et al. 2020]) as a smooth estimate of the maximum of a set of data. Its gradient is the softmax function (which is not a maximum as the name implies, but a smooth estimate of the one-hot argmax of a set of data). LogSumExps can be easily modified to return a smooth minimum distance rather than a maximum (and its gradient is the softmin). Aside from deep learning, LogSumExps also appear in other contexts where smooth approximations to min/max functions are needed: for example, they are known in the engineering literature as the Kreisselmeier-Steinhauser distance [Kreisselmeier and Steinhauser 1979]. LogSumExps have also been used recently in computer graphics by Panetta et al. [2017] to smoothly blend between microstructure joints. LogSumExp is just one of a number of smooth distance functions. For instance, the L_p norm function has been used as a smooth distance as well, for smooth blending between implicit surfaces [Wyvill et al. 1999] and computing smooth shells around surfaces [Peng et al. 2004]. A similar function can be computed directly from boundary integrals [Belyaev et al. 2013]. These functions could act as a drop in replacement for much of our proposed algorithm, but the former function tends to return numerically 0 results for far-away points which makes the zero isosurface ambiguous, and it is unclear if the latter even exhibits the important underestimate property. Not only do LogSumExps satisfy the desired property, but they numerically return `inf` for far-away points which leaves the zero isosurface unambiguous, and so we choose to construct our method around the LogSumExp function.

Gurumoorthy and Rangarajan [2009] demonstrated a relationship between the Eikonal equation and the time-independent Schrödinger equation (which is a screened Poisson equation), and used this to derive the LogSumExp function as an approximate solution to the Eikonal equation. They evaluate the LogSumExp using a convolution in the frequency domain, which requires a background grid to compute the FFT and its inverse. Further, their method requires all data points to be snapped to grid vertices. While more efficient than a full evaluation, our method achieves comparable asymptotic performance without a background grid and therefore respects the input geometry. Sethi et al. [2012] extended this line of work by adding support for edges, but they integrate the exponentiated distance over each edge, which, as we show in Section 3.1, can lead to overestimated distances. Computing a distance approximation by taking a logarithm is also conceptually similar to Varadhan's formula geodesic distance, which was the inspiration for the geodesics in heat method [Crane et al. 2017].

Smooth signed distance functions (SDFs) have recently become popular in machine learning as well. A full accounting is beyond the scope of this paper but see for instance Chen and Zhang [2019]; Mescheder et al. [2019]; Park et al. [2019]. These approaches encode geometric information in latent vectors to be used at evaluation time, along with an input position in space to evaluate a learned signed distance function. Aside from being unsigned rather than signed, our distance approximation diverges in two important ways from work on Neural SDFs. First, our representation is an augmentation to the exact geometry as a smooth approximation rather than an outright replacement. This means that algorithms that still require the original discrete geometry [Li et al. 2020b] for operations such as continuous collision detection can make use of our method. Second, the only preprocessing in our method is building a BVH over the data (which typically takes less than a second) rather than training a neural network, which is significantly more expensive.

One particular feature of smooth distance functions is that they can use point data to construct implicit functions whose zero isosurface represents the surface. There are many other methods that accomplish this, such as radial basis functions [Carr et al. 2001] and Poisson surface reconstruction [Kazhdan et al. 2006; Kazhdan and Hoppe 2013]. Although these methods are capable of producing very accurate surface reconstructions, they require solving large linear systems, while in our approach the implicit function is readily available. Another surface reconstruction method is the point set surface [Alexa et al. 2003], though instead of obtaining an implicit

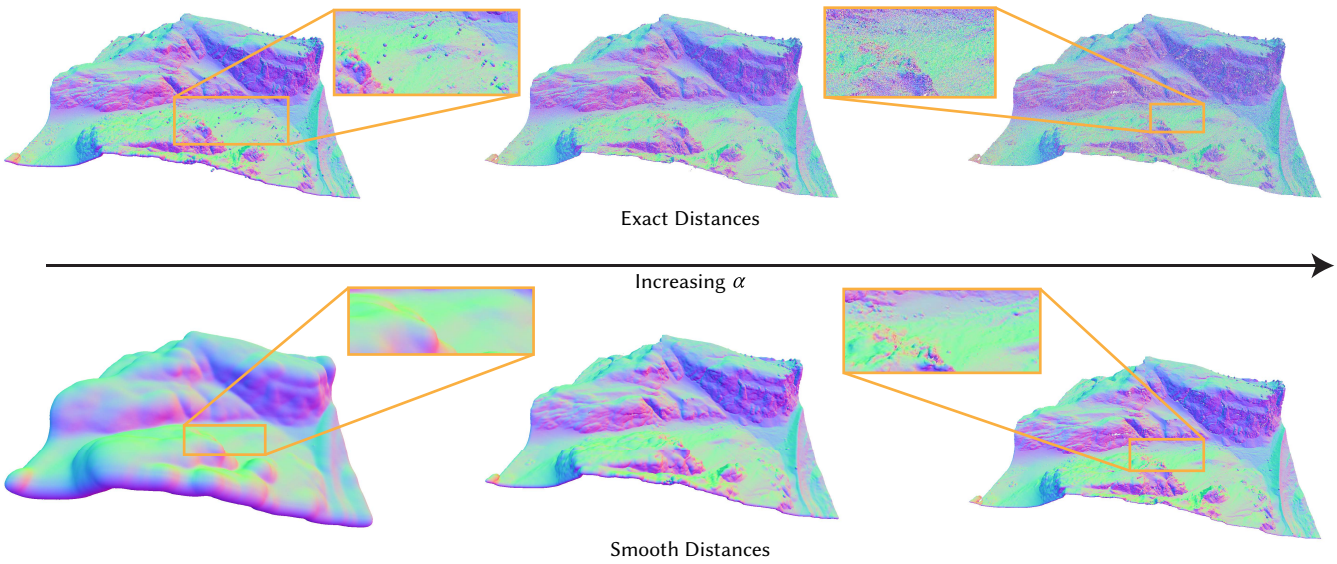


Fig. 3. A comparison between smooth and exact (offset) distance isosurfaces, with gradients represented by colour, at varying values of α, using $1/\alpha$ for exact distance offets. Exact distances trade off between inflating noisy data in the center and far side of the point cloud at low α, and noisy gradients at high α. Meanwhile, smooth distances (bottom) transition from smooth and inflated to more exact as α increases, without exacerbating noisy data or yielding poor gradients.

function, this method finds points on the described surface. Level set methods [Zhao et al. 2001] have also been used to reconstruct surfaces, though this requires a grid and may require prohibitively dense voxels to capture fine detail in the underlying surface.

Collision resolution has long been a difficult problem in physics-based animation and engineering. While no efficient method for deformable SDFs exists, a useful approximation is to use a signed distance function of the undeformed space [McAdams et al. 2011]. Mitchell et al. [2015] extended this work by using a non-manifold grid to accurately represent high-frequency and even zero-width features. Recently, Macklin et al. [2020] have used SDFs to represent rigid objects that robustly collide with deformable objects in an extended position-based dynamics framework [Macklin et al. 2016]. Further, barrier energies for constrained optimization can be seen as a smooth analog for distance-based constraints. These have seen much success in geometry processing [Smith and Schaefer 2015] and physics simulation [Ferguson et al. 2021; Li et al. 2020b], and follow-up work has proposed separate extensions to co-dimensional [Li et al. 2021] and medial geometry [Lan et al. 2021]. While effective for this particular application, the methods of Macklin et al. [2020] and Li et al. [2020b] lack the ability to smooth input data [Ferguson et al. 2021] and so cannot, for instance, approximate point clouds as closed surfaces (Fig. 1) for smooth collision resolution. Another shared technical limitation of these methods is the large number of constraints they generate (one per primitive pair), which must be mitigated through techniques like spatial hashing. For these reasons, we view our method as complementary to the aforementioned approaches: our method smooths the input data while also combining every pairwise primitive constraint into a single constraint.

Simulation frameworks such as Bullet [Coumans and Bai 2021] and PhysX [NVIDIA 2021] often accelerate collisions through bounding proxies that cover sections of geometry such as convex hulls, spheres, and cylinders. These approaches are effective for real-time simulation where speed is preferred over accuracy, but not comparable with our method since we aim for accurate off-line simulation.

The key to our method is a carefully designed weighted smooth distance function, combined with a specialized Barnes-Hut approximation [Barnes and Hut 1986]. The Barnes-Hut algorithm was first developed for N-body simulations to reduce computational effort on far-away bodies with negligible contributions to force. Barnes-Hut approximations have seen use in many graphics applications which use rapidly decaying kernels (e.g., [Alexa et al. 2003; Barill et al. 2018; Yu et al. 2021]), but as we will show, careful modification is needed to ensure that fast evaluation does not break the conservative bounds of the LogSumExp function.

Contributions. In this paper we present a new method for computing smooth distance fields on co-dimensional input geometry. We derive blending weights and a modified Barnes-Hut acceleration approach that ensures our method is conservative (points outside the zero isosurface are guaranteed to be outside the surface), accurate, and efficient to evaluate for all the above data types. This, in combination with its ability to smooth sparsely sampled data like point clouds, enables new applications such as direct, co-dimensional rigid body simulation using unprocessed lidar data.

3 METHOD

Given an input geometry Ω embedded in $\mathbb{R}^3$, the unsigned distance field is defined as

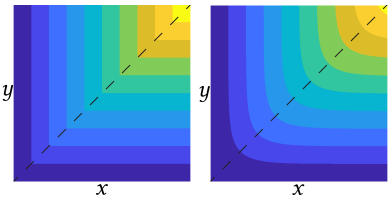
$$d(\mathbf{q}) = \min_{\Omega} d(\mathbf{x}, \mathbf{q}), \tag{1}$$

where $\mathbf{x} \in \Omega$ is a point on the input geometry and $\mathbf{q} \in \mathbb{R}^3$ is an arbitrary query point.

In our case, the input geometry is represented as a co-dimensional data mesh, $\mathcal{M} = (V, F)$. Here V is a set of vertices $\mathbf{v}_i$ in $\mathbb{R}^3$, and F is a set of primitives. In this paper, we deal exclusively with points, edges, and triangles, so we use the terms "simplex" and "primitive" interchangeably. A simplex f_i consists of a tuple of indices into V. For example, a triangle would be represented as $f_i = (i_0, i_1, i_2)$ where each i_k indexes V. We also need to reference the *faces* of a simplex, which we define as any non-empty subset of a simplex; for example, edges and points are faces of triangles. Based on this definition, edges that are faces of triangles can be constructed from triangle indices using, e.g., $i_{01} = (i_0, i_1)$. Lastly, we denote the dimension of a simplex as $n(f_i)$ and the volume of a simplex as $|f_i|$ (which is 1 for points). F can be omitted for point clouds, but we will use it throughout to keep the notation consistent.

With respect to this discretized input, the distance field computation can be reframed as finding the minimum distance between the query point and all constituent simplicies of $\mathcal{M}$:

$$d(\mathcal{M}, \mathbf{q}) = \min_{i} d(f_i, \mathbf{q}). \tag{2}$$



However, min is only C^0, and has discontinuities along the medial axes of the data mesh (see inset, left, for a plot of $\min(x, y)$, with the gradient discontinuity $x = y$ shown as a dashed line). Furthermore, exact distances are a poor representation of co-dimensional geometric representations like point clouds that only loosely approximate the true underlying (volumetric) geometry. The distance isosurfaces to a point cloud have poor gradients at small offsets and amplify sample noise at large offsets (Fig. 3). Our goal is to tackle both problems by designing a distance function that is at least C^1 differentiable and produces a smooth isosurface approximating the underlying geometry.

We begin by converting the true distance to a smooth distance via an application of the LogSumExp smooth minimum function which yields

$$\hat{d}(\mathcal{M}, \mathbf{q}) = -\frac{1}{\alpha} \log\left(\sum_{f_i \in F} w_i(\mathbf{q}) \exp(-\alpha d_i)\right), \tag{3}$$

where $d_i = d(f_i, \mathbf{q})$, α controls the accuracy and smoothness of the approximation and w_i are influence weights for each simplex, which will be discussed in more detail in Section 3.3. (See inset, right, for a plot of $-0.1 \log(\exp(-10x) + \exp(-10y))$.)

Importantly, this function is differentiable for all finite values of α with gradient (with respect to $\mathbf{q}$)

$$\nabla \hat{d}(\mathcal{M}, \mathbf{q}) = \frac{\sum_{f_i \in F} w_i(\mathbf{q}) \exp(-\alpha d_i) \nabla d_i - \frac{1}{\alpha} \exp(-\alpha d_i) \nabla w_i(\mathbf{q})}{\sum_{f_i \in F} w_i(\mathbf{q}) \exp(-\alpha d_i)}, \tag{4}$$

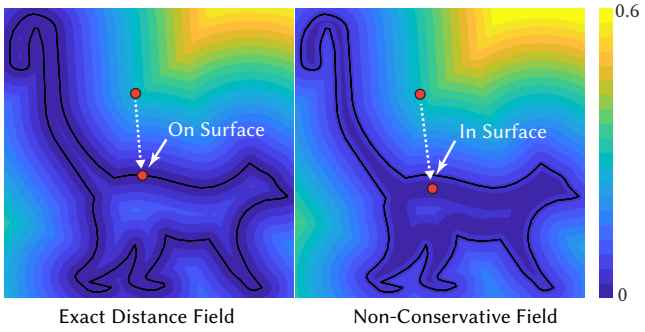


Fig. 4. Points (red) starting outside a shape can be prevented from crossing into it by ensuring their trajectories (dashed line) never cross the 0 isosurface of an unsigned distance field. Non-conservative estimates break this property, allowing interpenetration of the underlying geometry. Such estimates are unusable if downstream algorithmic stages require the geometries to be intersection-free.

and is guaranteed to underestimate the true distance to the input mesh when every $w_i \geq 1$, with error bounded by $\frac{\log(A|F|)}{\alpha}$ where A is the maximum value of all w_i (Appendix A). The differentiability of LogSumExp allows us to preserve the underlying smoothness of the distance functions, which are at least C^1 almost everywhere (Appendix B), and the underestimate (or conservative) property means that our smooth distance will alert us to $\mathbf{q}$ crossing the input surface before it happens. This conservative property is crucial for applications such as collision detection since it ensures that maintaining separation with regards to the smooth distance is sufficient to maintain separation between input shapes (Fig. 4). Ergo, the output of our method will remain usable if downstream applications require the underlying geometric representations to be separated.

An additional nice property of LogSumExp is its accuracy: not only is its error merely logarithmic in mesh size, but as α increases, distances become more accurate.

3.1 Smooth Distance to a Single Query Point

As a didactic example let us apply Eq. 3 to a point cloud which we do by setting $d_i = \|\mathbf{q} - \mathbf{v}_i\|$ and $w_i = 1$. We can directly observe the smoothing effect of α (Fig. 5), which can be used to close point clouds. Decreasing α produces progressively smoother surface approximations, and surfaces produced with smaller α values nest those produced with higher α's. This nesting is a consequence of the conservative behaviour of the LogSumExp formula.

All of this taken together means that the naive LogSumExp works well for point cloud geometry.

3.2 Smooth Distances to Co-Dimensional Geometry

A natural extension of Eq. 3 to edge and triangle meshes is to replace the discrete sum over points with a continuous integral (see, e.g., [Sethi et al. 2012]) over the surface:

$$\hat{d}(f_i, \mathbf{q}) = -\frac{1}{\alpha} \log\left(\int_{f_i} \exp(-\alpha \|\mathbf{x} - \mathbf{q}\|) d\mathbf{x}\right).$$

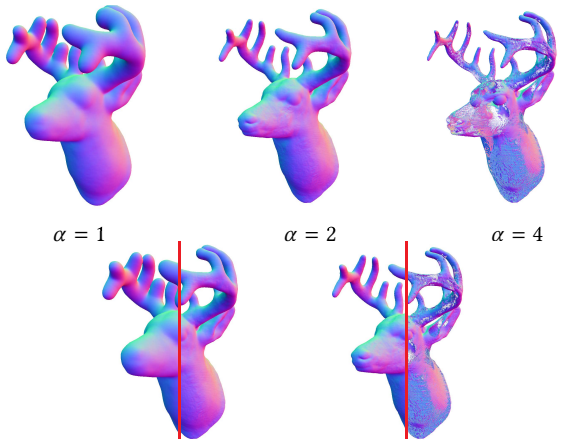


Fig. 5. At low values of α, the sharp antlers and face of this deer point cloud become puffy and smooth (top left). Increasing α resolves features in the face more clearly (top middle), and the individual points become visible at high α (top right). The bottom row shows the transition between different α values — higher-α surfaces are contained in lower-α surfaces.

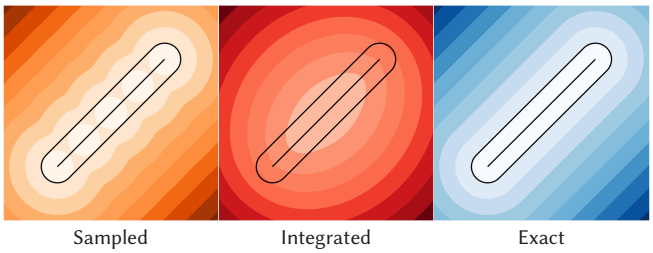


Fig. 6. Computing edge distances with LogSumExp and sampled point quadrature (left) creates holes in the isosurface at high α. Integrating over the edge (middle) using 5th order Gaussian quadrature can remove the holes but will overestimate distance at low α. Computing exact distances (right) produces the correct isosurface. The edge and a small offset surface are shown in each, where the underestimate property requires that only the first color interval should be contained in the offset region.

When discretized, this becomes equivalent to applying Eq. 3 to quadrature points on the mesh, while the w_i become the quadrature weights.

While simple, this formulation will unfortunately break the important conservative property of the LogSumExp function because the quadrature weights will, in general, not be greater than or equal to 1. Fig. 6 shows examples of the overestimation errors introduced by this approach.

Rather we must return to Eq. 3 and compute the respective d_i's to the constituent mesh triangles and edges exactly. This can be accomplished via efficient quadratic programming which we detail in Appendix B. However, using unit valued weights, as we did for points, produces bulging artifacts where primitives connect (Fig. 7). What remains is to compute weight functions that mitigate these effects while simultaneously satisfying our ≥ 1 constraint.

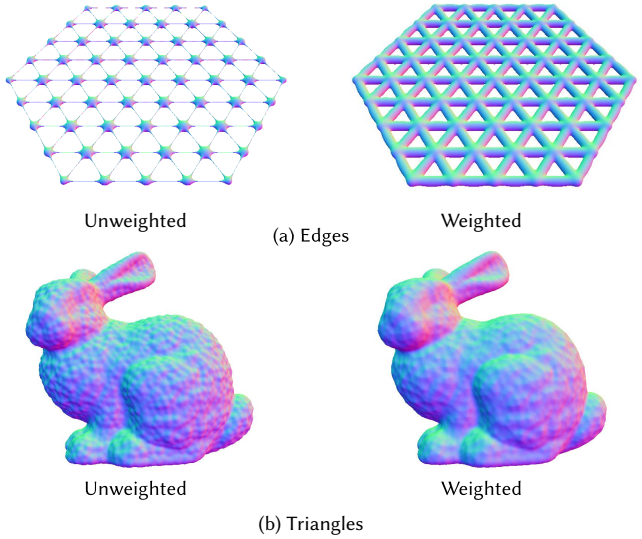


Fig. 7. Distances tend to concentrate where primitives overlap, creating thin edges (top left) and bumps in the surface (bottom left). Weight functions help mitigate these effects (right), for both edge meshes (a) and triangle meshes (b).

3.3 Weight Functions

First, we must understand why these bulging artifacts occur. If the closest point on $\mathcal{M}$ to $\mathbf{q}$ is on a simplex face, then every simplex containing that face will have the same closest point, resulting in a more severe underestimate of the true distance than usual. While this phenomenon occurs any time $\mathbf{q}$ is on the medial axis of $\mathcal{M}$ (the cause of the logarithmic error term), separately computing the distance to each primitive effectively creates an artifical medial axis where there is only one true closest point that is contained in multiple primitives. Panetta et al. [2017] observe the same concentration artifacts when using LogSumExp on edge meshes, but they propose fixing it using weighted blending, where the weight calculation relies on knowing the convex hull of the edge mesh neighborhood and blending between smooth and exact distance fields. Unfortunately, it is unclear how to extend this to triangles or apply acceleration schemes like Barnes-Hut. Instead, we propose a different weighting scheme which is fast, local and can be easily adapted to triangles.

We center the design for our per-primitive weight functions w_i around high-accuracy use cases, which correspond to high values of α. The weights must be spatially varying to counteract the local concentration artifacts, and for simplicity, the function will be defined in terms of the barycentric coordinates of the closest point projection of $\mathbf{q}$ onto f_i. The projection function is denoted by $\boldsymbol{\pi}_i(\mathbf{q})$ and the barycentric coordinates of a point $\mathbf{p}$ within f_i are denoted $\boldsymbol{\phi}_i(\mathbf{p})$; using these definitions, our weight function is $w_i(\mathbf{q}) = w_i(\boldsymbol{\phi}_i(\boldsymbol{\pi}_i(\mathbf{q})))$. We will first design a weight function that mitigates the bulge artifacts without regard for maintaining the conservative property, which we will denote as $\tilde{w}_i$, and then derive an appropriate global scale factor for every $\tilde{w}_i$ in $\mathcal{M}$ to achieve the conservative property.

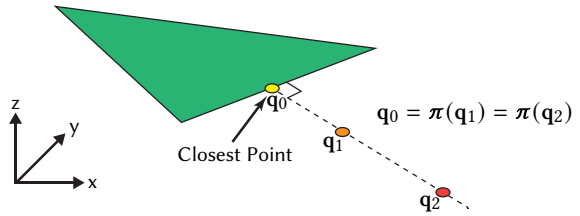


Fig. 8. All points (such as $\mathbf{q}_1$ and $\mathbf{q}_2$) along a line extending perpendicular to a simplex boundary share a closest point $\mathbf{q}_0$, and thus the normal derivative of the closest point projection $\boldsymbol{\pi}$ is the zero vector.

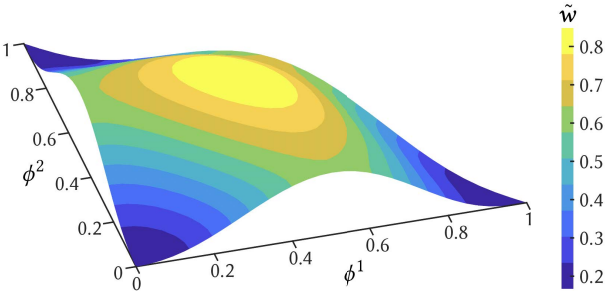


Fig. 9. An example of the unscaled triangle weights, $\tilde{w}$, computed by our method for a single triangle.

The weight functions will be defined as polynomials in terms of $\boldsymbol{\phi}_i$, so constructing the weight functions is a matter of determining the polynomial coefficients by solving a system of linear equations, determined by both point constraints and derivative constraints. Qualitatively, the point constraints aim to both assign a low weight to the simplex boundary and assign a high weight to the simplex interior (see Appendix C for more details). The derivative constraints constrain the normal derivative to be 0 along the boundary, which is necessary in order to ensure w_i is smooth everywhere. To see this, we note that the gradient of $\tilde{w}_i$ with respect to $\mathbf{q}$ is

$$\nabla \tilde{w}_i = \frac{\partial \boldsymbol{\pi}_i}{\partial \mathbf{q}} \frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{\pi}_i} \frac{\partial \tilde{w}_i}{\partial \boldsymbol{\phi}_i}, \tag{5}$$

where we use the indexing convention $\left[\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right]_{pq} = \frac{\partial \mathbf{y}_q}{\partial \mathbf{x}_p}$ for a generic vector function $\mathbf{y}(\mathbf{x})$ (i.e., gradients with respect to a scalar function are column vectors). $\boldsymbol{\pi}_i$ is in fact a C^0 function, with a derivative discontinuity on the boundary. When $\boldsymbol{\pi}_i(\mathbf{q})$ is in the interior of f_i, the gradient $\frac{\partial \boldsymbol{\pi}_i}{\partial \mathbf{q}} \in \mathbb{R}^{3\times 3}$ is the identity matrix, but when it is on the boundary, the gradient has a null space in the direction perpendicular to the boundary, making the normal derivative the zero vector (Fig. 8) and creating a derivative discontinuity at the simplex boundary. Left unchecked, this discontinuity will propagate to w_i as well. We opt to hide this discontinuity by coercing $\frac{\partial w_i}{\partial \boldsymbol{\phi}_i}$ to have zero normal derivative along the boundary while also being smooth. See Appendix C for details on how the derivative constraints are enforced, and Fig. 9 for an example weight function.

Since the point constraints require $\tilde{w}_i$ to be less than 1 along the simplex boundary, they break the conservative property. We rectify

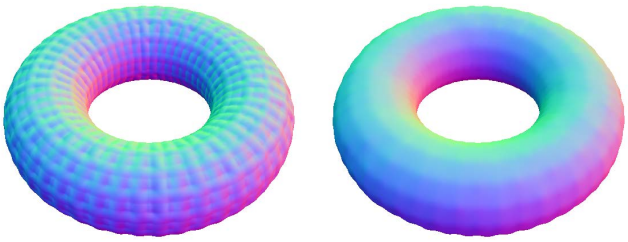


Fig. 10. Small meshes produce large weight gradients which produce noticeable ridges on this torus (left); uniformly scaling the ambient space allows us to control the length of these gradients and smooth out the ridges (right).

this by scaling $\tilde{w}_i$ by a factor A associated with $\mathcal{M}$ and not individual simplices, so that $A\tilde{w}_i \geq 1$ for all $f_i \in \mathcal{M}$. Although this inflates the zero isosurface, it does so uniformly, so we do not reintroduce the artifacts we wanted to remove. This error is logarithmic in A and becomes negligible at higher values of α.

We can further refine our weight functions to improve their behaviour. One such improvement targets the $\frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{\pi}_i}$ term in the gradient. Since $\boldsymbol{\pi}_i$ is restricted to producing points within f_i, this term is in fact equivalent to linear shape function gradients from finite element analysis. For triangles, our barycentric coordinate vector is $\boldsymbol{\phi}_i = [\phi_i^1, \phi_i^2]^\top$, and our gradients (with respect to points in f_i) are $\nabla\phi_i^1 = \frac{(\mathbf{v}_{i_0} - \mathbf{v}_{i_2})^\perp}{2|f_i|}$ and $\nabla\phi_i^2 = \frac{(\mathbf{v}_{i_1} - \mathbf{v}_{i_0})^\perp}{2|f_i|}$ where $\mathbf{x}^\perp$ represents a 90-degree counterclockwise rotation of $\mathbf{x}$. We see that gradients are inversely proportional to triangle area, and thus create gradient artifacts in smaller triangles (Fig. 10). Our goal is to control the magnitude of these gradients.

Since larger triangles have smaller gradients, a simple way to fix the gradients is to isotropically scale the space by a factor ρ, compute $\hat{d}$, and then scale back to the original space at the end. This way, weight gradients are computed in the scaled space and we can directly control the magnitude of the barycentric gradients. However, if we expand Eq. 3 using distances scaled by ρ, we notice that ρ behaves the exact same way as α, and so we do not actually need a new parameter. Instead, α itself can be interpreted as a uniform scale factor in space, and controlling accuracy with α is equivalent to measuring lengths with the metric $\alpha^2 I$ where I is the identity matrix. Going back to weight gradients, we now measure lengths in α-scaled space, and we get barycentric gradients such as $\nabla\phi_i^1 = \frac{(\mathbf{v}_{i_0} - \mathbf{v}_{i_2})^\perp}{2\alpha|f_i|}$, which allows us to reduce the norm of the problematic gradient term (Fig. 10). Other quantities stay the same since they are either independent of the metric (e.g., $\nabla\boldsymbol{\pi}_i$), or their dependence on α and thus the metric space is explicit and already accounted for.

Another improvement is the behaviour of the w_i's at low α. A key assumption in our design of w_i is that α is sufficiently large, but since α is a controllable user parameter, this assumption can sometimes be violated, and the weights can overcompensate for concentration. In these cases w_i needs to be further modified to avoid artifacts (Fig. 11). If α surpasses some threshold α_U (which at a high level represents a

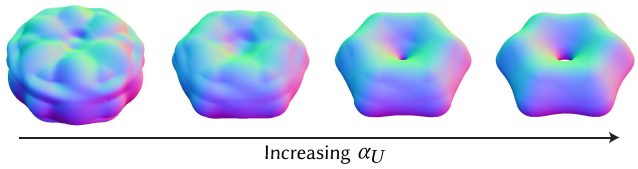


Fig. 11. At a low value of α, the weight function overcompensates and produces creases and other spurious artifacts in this very blobby hexagonal torus; adjusting α_U makes it possible to blend between weighted and unweighted distances to mitigate these artifacts.

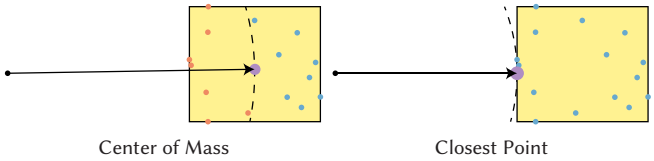


Fig. 12. The center of the Barnes-Hut approximation (purple) has a significant effect on the sign of the error. The center of mass can potentially be farther away than some data points (orange), which can lead to an overestimate of the true distance. On the other hand, the closest point on the bounding box is guaranteed to be closer than every data point (blue), so the final result will always be an underestimate.

upper bound on α — see Section 4.3 for more information on how it is selected), then we can use w_i as-is; if $\alpha < \alpha_U$, then we must flatten or attenuate w_i. Experimentally we observe that using a scale factor $S = \frac{\alpha}{\max(\alpha,\alpha_U)}$ to define an *attenuated* weight function $w_i(\mathbf{q}) = (A\tilde{w}_i(\mathbf{q}))^S$, with gradient $\nabla w_i(\mathbf{q}) = S(A\tilde{w}_i(\mathbf{q}))^{S-1}\nabla\tilde{w}_i(\mathbf{q})$, helps minimize these issues. Note that this heuristic does not eliminate the issue for all α but does mitigate it sufficiently for all ranges of α used in this paper.

3.4 Generalized Query Primitives

We will briefly discuss how to generalize our query point into a general query primitive g, which can also be an edge or a triangle. Just like with points, we can compute the distance to another simplex f_i as $d(f_i, g)$ using a positive semidefinite quadratic program with linear constraints (see Appendix B). We now also obtain barycentric coordinates for g through the argmin, which we will denote as $\boldsymbol{\lambda}$. All formulas in the preceding subsections can be adapted for this generalization by simply replacing $\mathbf{q}$ with g, and using the closest point to $\mathcal{M}$ on g, $g(\boldsymbol{\lambda})$, when a single point is needed (e.g., in weight functions). These changes imply that $\nabla\hat{d}$ is now with respect to an entire query primitive, but this simply describes a rigid translation of g. Taking gradients with respect to entire primitives also ensures that, even in configurations with multiple closest point pairs such as parallel edges, the gradient is the same for each closest point pair and thus well-defined.

3.5 Barnes-Hut Approximation

Although our distance function is smooth and easy to differentiate, it requires evaluation of a distance between every pair of primitives in the most general case. However, its use of exponentially decaying functions enables the application of the well-known Barnes-Hut approximation [Barnes and Hut 1986] to accelerate evaluation. Barnes-Hut uses a *far field approximation* to cluster groups of far-away primitives together (typically using a spatial subdivision data structure like an octree or bounding volume hierarchy) and treat them as a single point. The approximation is characterized by the centers of each bounding region, and a user parameter β which controls where the far field approximation is employed — see Appendix D for details. At a high level, lower β is more accurate, and $\beta = 0$ results in an exact evaluation.

Placing the far-field expansion at the center of mass reduces overall error [Barnes and Hut 1986] but it can possibly overestimate the true distance, breaking the conservative nature of the LogSumExp smooth distance. A minor modification can reinstate this property – placing the expansion center on the closest point of the cluster region to the query primitive g, rather than at the center of mass. When using an octree or BVH, we pick the closest point on the bounding box to the query point. This may increase the error relative to using the center of mass, but it guarantees that the Barnes-Hut estimate only underestimates the exact smooth distance (Fig. 12). Note that this choice does not affect the gradient if the bounding region is convex (which is the case for bounding box hierarchies and octrees), since rigidly translating g farther away along the gradient direction does not change the closest point.

Finding the closest point on a box to a query point is simple, but finding the closest point on a box to a query edge or triangle is more complex, and requires solving a quadratic program. However, this is relatively expensive for what is meant to be a fast check to help reduce computation time, so we instead use an approximation of the problem. Viewing a box as the intersection of 6 halfspaces, we identify which halfspaces the primitive lies completely outside of. If there are 3 such halfspaces, they identify a corner of the box that is closest to the primitive; if there are 2 halfspaces, they identify an edge of the box; and if there is 1 halfspace, it identifies a face of the box. If there are no halfspaces that satisfy this criteria, then we simply return the distance to the box as 0 and force the traversal to visit its children. Although this test somewhat inhibits the Barnes-Hut approximation's ability to group primitives together, the computational savings per visited node more than make up for it — we noticed over a 5× improvement in performance in our experiments.

Psueodocode summarizing our method so far is given in Algorithm 1 and Algorithm 2.

3.6 Smooth Distance to a Query Mesh

Now that we are equipped with an efficient method for computing $\hat{d}(\mathcal{M}, g)$, we can combine these distances using LogSumExp to obtain a smooth distance between $\mathcal{M}$ and a *query mesh* $\bar{\mathcal{M}} = (\bar{V}, \bar{F})$:

$$\hat{d}(\mathcal{M}, \bar{\mathcal{M}}) = -\frac{1}{\alpha_q}\log\left(\sum_{g_j \in \bar{F}} \exp\left(-\alpha_q \hat{d}(\mathcal{M}, g_j)\right)\right), \tag{6}$$

where we have introduced a new accuracy-controlling parameter α_q that is independent of the inner α. In an exact evaluation ($\beta = 0$),

ALGORITHM 1: Collecting contributions for $\hat{d}(\mathcal{M}, g)$ collectContributions

Inputs : Data mesh $\mathcal{M}$, BVH node B, query primitive g, parameters α, α_U, β
Outputs : Sum of exponentials c and sum of weighted distance gradients ∇c

```
// B has children B.l and B.r
if BarnesHutCondition(M, B, g, β) ;        // App. D, Sec. 3.5
 then
    Return far field expansion ;                          // App. D
else if B is a leaf containing f_i then
    (d_i, ∇d_i, φ_i, λ) ← d(f_i, g) ;                     // Sec. 3.4, App. B
    (w_i, ∇w_i) ← weightFn(f_i, φ_i, g, λ, α, α_U) ;      // Sec. 3.3
    c ← w_i exp(−α d_i) ;                                 // Eq. 3
    ∇c ← c∇d_i − (1/α) exp(−α d_i) ∇w_i ;                 // Eq. 4
    Return (c, ∇c);
else
    (c_l, ∇c_l) ← collectContributions(M, B.l, g, α, α_U, β);
    (c_r, ∇c_r) ← collectContributions(M, B.r, g, α, α_U, β);
    Return (c_l + c_r, ∇c_l + ∇c_r);
end
```

ALGORITHM 2: Computing $\hat{d}(\mathcal{M}, g)$ smoothMinDist

Inputs : Data mesh $\mathcal{M}$, BVH node B, query primitive g, parameters α, α_U, β
Outputs : Smooth min distance $\hat{d}$ and gradient $\nabla\hat{d}$

```
(c, ∇c) ← collectContributions(M, B, g, α, α_U, β);
d̂ ← −(1/α) log c;
∇d̂ ← ∇c / (c + ε) ;                          // Avoid divide-by-zero
```

this strongly resembles the LogSumExp of all the pairwise distances between each f_i and g_j.

A subtle issue with the current formulation is that the distance gradients $\nabla\hat{d}(\mathcal{M}, g_j)$ are with respect to g_j as a whole, but we need per-vertex gradients as those are the true degrees of freedom. One simple way to do this is to split $\exp(-\alpha_q\hat{d}(\mathcal{M}, g_j))$ equally between the vertices of g_j, and rewriting the summation over vertices and one-ring neighbourhoods gives us:

$$\hat{d}(\mathcal{M}, \bar{\mathcal{M}}) = -\frac{1}{\alpha_q}\log\left(\sum_{\mathbf{q}_k\in\bar{V}}\sum_{g_j\in\mathcal{N}_k}\frac{1}{n(g_j)+1}\exp\left(-\alpha_q\hat{d}(\mathcal{M}, g_j)\right)\right), \tag{7}$$

where $\mathcal{N}_k$ is the set of one-ring neighbours of $\mathbf{q}_k$. Then, the gradient with respect to query vertex $\mathbf{q}_k$ is

$$\nabla_k\hat{d}(\mathcal{M}, \bar{\mathcal{M}}) = \frac{\sum_{g_j\in\mathcal{N}_k}\frac{1}{n(g_j)+1}\exp\left(-\alpha_q\hat{d}(\mathcal{M}, g_j)\right)\nabla\hat{d}(\mathcal{M}, g_j)}{\sum_{g_j\in\bar{F}}\exp\left(-\alpha_q\hat{d}(\mathcal{M}, g_j)\right)}. \tag{8}$$

Once again, the convexity of the query and data primitives means that we can interpret the $\nabla\hat{d}(\mathcal{M}, g_j)$ as a rigid translation of g_j that affects all its vertices equally.

The summation in Eq. 6 can be easily parallelized, and the gradient computation requires only a small amount of serialization at the end to redistribute gradients onto vertices.

4 RESULTS

4.1 Implementation

We implemented our method using C++ with Eigen [Guennebaud et al. 2010] and libigl [Jacobson et al. 2018]. We designed the implementation so that it could be ported onto the GPU, and to this end, we implemented the BVH traversal algorithm outlined in Algorithm 1 using the stackless method of Hapala et al. [2011]. Since GPUs exhibit poor performance for double-precision floating point, most of our computations (particularly vector arithmetic) are conducted in single-precision, while exponential sums are tracked using double-precision to increase the range of usable α values. Primitive distances were hand coded if feasible, and were otherwise implemented using a null-space quadratic program solver written using Eigen (edge-triangle and triangle-triangle distances). An important aspect of these distance computations is robustness, which becomes particularly important because distances are computed using single-precision floats, and errors in the distance can result in breaking the conservative property. The hand-coded distance functions made extensive use of an algorithm by Kahan which employs the fused-multiply-add instruction to reduce cancellation error [Kahan 2004], while the quadratic program solver leveraged Eigen's numerically stable algorithms.

4.2 Sphere Tracing

Sphere tracing is a method to render signed (and unsigned) distance functions [Hart 1996]. We can use sphere tracing to visualize the zero isosurface of $\hat{d}$ with $\bar{\mathcal{M}}$ as a single point, which we have done throughout the paper for demonstrative purposes.

4.3 Parameter Analysis

To demonstrate the effectiveness of Barnes-Hut, we conduct an ablation study on β. In order for Barnes-Hut to be useful in approximating a constraint function, it primarily needs to be accurate near the zero isosurface, as that is where it is evaluated in constrained optimization problems (e.g., rigid body contact constraints are only evaluated when there is a potential collision). The approximation becomes increasingly inaccurate farther away from the surface, but since we are only concerned with the zero isosurface, we use sphere tracing as a sampling technique. Using the Stanford bunny mesh's vertices, edges, and triangles, with $\alpha = 200$, $\alpha_U = 1200$, and β between 0 and 1, we measure the amount of time taken to render a 512×512 image, as well as the distance along each ray between the approximation's estimate of the isosurface and the actual isosurface ($\beta = 0$). The results are shown in Fig. 13, using 4 threads on a 2015 MacBook Pro. We see that running time decreases by an order of magnitude even for $\beta = 0.2$ in all three cases, and all renders take less than 10s at $\beta = 0.5$. The error relative to the bounding box diagonal also remains below 4% for these values of β. As a result, we use $\beta = 0.5$ in all of our examples in the paper unless otherwise stated. We find that β can be increased with low error for higher

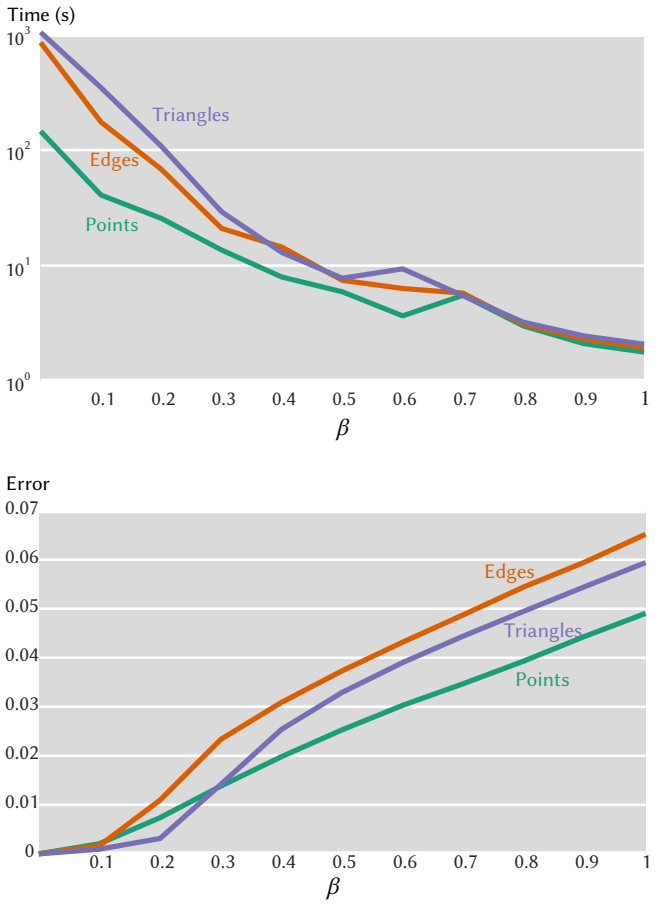


Fig. 13. Sphere tracing the Stanford bunny's smooth distance function over its vertices (green), edges (orange), and triangles (purple) obtains order-of-magnitude speedups (top) even for small β, while achieving low isosurface approximation error relative to the bounding box diagonal (bottom).

values of α (or equivalently, meshes with lower sampling density), but this is not necessary to obtain significant speedups.

Due to Barnes-Hut relying on a switch between a near field and far field approximation, the smooth distance field can exhibit discontinuities. In order to quantify the effect of these discontinuities, we perform a sensitivity analysis on β at points where the evaluation switches between a near field and far field expansion, since we expect a change in β to also change the leaves visited in the evaluation. Using the city lidar point cloud from Fig. 1 and $\beta = 0.5$, we generate sample points by uniformly sampling points on the top and bottom of the bounding box, and sphere tracing rays along the vertical coordinate axis (z-axis in this case) until the ray either misses the surface or has a smooth distance of 0.1 or less. Then, with these sample points on the 0.1 isosurface, we trace small rays in the same vertical direction, and whenever the evaluation switches from a far field to near field expansion in a bounding box (i.e., traverses inside a new bounding box), we also measure the smooth distance using $\beta = 0.5001$ and compute the discrepancy in smooth distance and the cosine between gradient vectors. We collect 259 field switch points where $\beta = 0.5001$ traverses less of the BVH than $\beta = 0.5$, and observe that the mean smooth distance discrepancy is 7.6×10^{-5} and the maximum discrepancy is 6.6×10^{-3}; for gradients, the mean cosine is 1 and the minimum cosine is 0.9974. These results are sufficient for our rigid body simulations with geometry orders of magnitude larger than these discrepancies, though for applications with more demanding smoothness requirements, smooth Barnes-Hut approximations are a potential area for future work.

The remaining parameters α and α_U are associated with the geometry of $\mathcal{M}$. In cases where we want a function resembling our underlying geometry, we want to select an α that is high enough to produce a good approximation of the surface, but low enough to prevent numerical problems. Since we can interpret α as a metric, we can select α based on the density of our geometry. One heuristic we found to be a useful starting point for edge and triangle meshes is to set α to be the reciprocal of the minimum edge length. For points, we set α to be 100 times the length of the bounding box diagonal. These results can be refined by sphere tracing the zero isosurface and tweaking the results until the surface looks satisfactory. α_U is essentially an upper bound on α and can be determined using this same procedure, which allows it to be used in low-α scenarios as well. α_q is very similar to α but is related to the geometry of $\bar{\mathcal{M}}$, and can be selected using the same process. A fully automated solution for selecting these parameters is left as future work.

4.4 Performance Benchmark

We performed a large-scale performance evaluation of our method on the Thingi10K dataset [Zhou and Jacobson 2016]. For each model, we created 3 data meshes: all of the model's vertices V, all of its edges E, and all of its triangles F. We then scaled and translated each of these meshes so that their bounding boxes were centered at $[0.5, 0.5, 0.5]^\top$, with a bounding box diagonal of 0.5. Then, we evaluated $\hat{d}$ from each mesh to each of the voxel centers of a $100 \times 100 \times 100$ grid between $[0, 0, 0]^\top$ and $[1, 1, 1]^\top$, in parallel using 16 threads. The results are reported in Fig. 14. We can see that performance is proportional to the number of visited leaves (i.e., the number of primitive distance calculations and far field expansions employed), and the percentage of visited leaves drops significantly as meshes increase in size. These results show that our method is quite scalable, and is very good at handling large meshes. Just like in Section 4.2, points tend to perform much better than edges and triangles, but now we can see why this is the case. Point clouds have lower variances than edge meshes and triangle meshes in their average leaves visited percentage — for example, even at at 1000 edges/triangles, several meshes require the traversal to visit well over 10% of the leaves in their BVH on average. Also, points generally visit far fewer leaves — point cloud tests visit at most 70 leaves, while edge meshes and triangle meshes can require over 2000 visited leaves. As we can see from the first graph, less leaves visited corresponds to faster query times, and so points are empirically faster than the other two types of meshes.

We also ran this benchmark on a GPU — see Appendix E for the results.

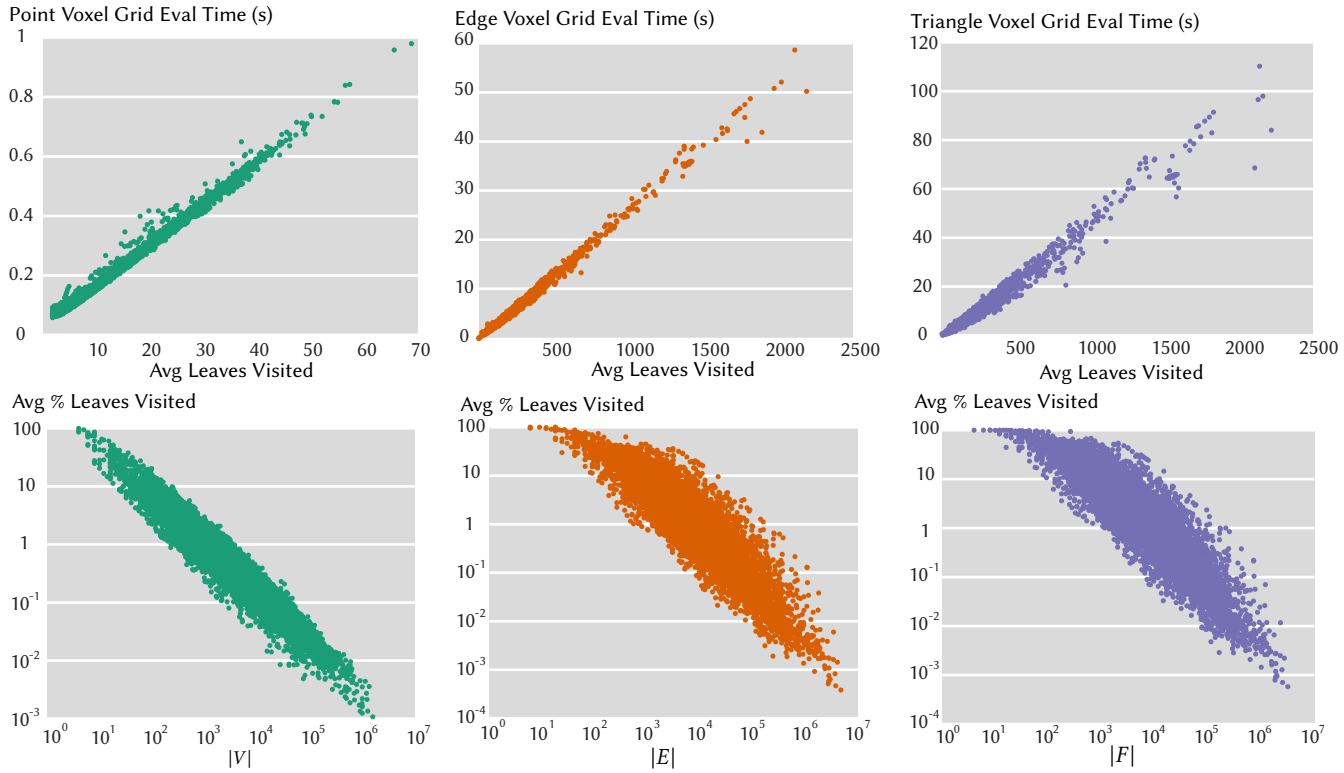


Fig. 14. Results of the Thingi10K benchmark, using the vertices V, edges E, and triangles F of each mesh in the dataset to evaluate queries on a $100 \times 100 \times 100$ voxel grid. The top row shows that running time is linearly proportional to the number of visited leaves for all mesh types, and the percentage of visited leaves drops sharply as mesh size increases. Furthermore, points tend to perform much better than edges and triangles.

4.5 Rigid Body Collisions

One popular application of distance functions is in collision resolution, where they can be used as an intersection-free optimization constraint. To focus on the distance function constraint rather than the mechanics, we simulate frictionless rigid body contact, using a variety of configurations and a variety of geometry. Each time step of our simulation is driven by an incremental potential energy based on Ferguson et al. [2021] with a single smooth distance constraint. For example, the optimization problem corresponding to a single object $\bar{\mathcal{M}}$ colliding with one other static object $\mathcal{M}$ in the scene is

$$\begin{aligned} \min_{\mathbf{p},\boldsymbol{\theta}} \quad & E(\mathbf{p},\boldsymbol{\theta}) \\ s.t. \quad & \hat{d}(\mathcal{M}, \bar{\mathcal{M}}(\mathbf{p},\boldsymbol{\theta})) \geq 0, \end{aligned} \tag{9}$$

where $\mathbf{p}$ and $\boldsymbol{\theta}$ are vectors in $\mathbb{R}^3$ describing the world space position and orientation of $\bar{\mathcal{M}}$, respectively, and E is an objective function whose unconstrained minimizer is equivalent to an implicit Euler time step in position and an exponential Euler time step in orientation. We solved this optimization problem using a primal-dual interior-point solver [Nocedal and Wright 2006] written in C++, where every iteration produces a feasible point, and we replace the Hessian block of the primal-dual system with an identity matrix (similar to gradient descent on the optimization problem's Lagrangian). For multi-object simulations, each object in the scene has an associated α and α_U to use in smooth distance evaluations, and every pairwise smooth distance constraint is combined into a single constraint using LogSumExp with the maximum α among all objects in the scene. Inertial quantities were computed using the underlying geometry. We did not implement continuous collision detection, so we use relatively small time steps in our examples.

Since we wrap every pairwise primitive distance constraint into a single constraint, a much simpler alternative to smooth distances is the exact minimum of all pairwise distances. However, as we discussed in Section 3, the exact minimum is C^0, which is undesirable (other methods that use exact distance use multiple constraints, which is better behaved than a single exact minimum constraint [Nocedal and Wright 2006]). We demonstrate its poor behaviour in two 2D examples where a point is dropped into a V-shaped bowl, and compare the results between exact minimum distance and smooth distance constraints (Fig. 15). In both simulations, exact distances perform qualitatively worse than smooth distances, because they do not smooth the geometry and must deal with the sharp gradient change after passing the medial axis passing vertically through the base of the bowl.

We now show a variety of simulations in three dimensions using smooth distances. First, we show two rigid bunnies colliding together and falling onto a bumpy ground plane as a simple test

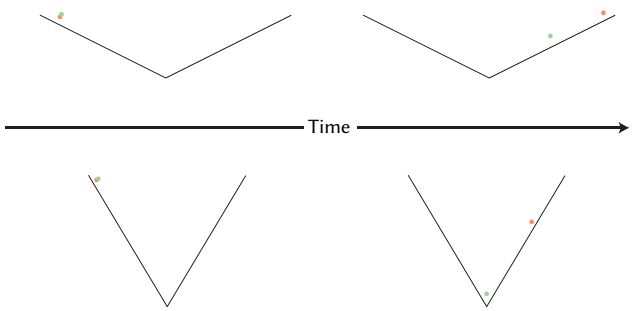


Fig. 15. A comparison of exact and smooth distance constraints in rigid body simulations of a point mass dropping onto two different V-shaped bowls. In the shallow bowl (top), exact distances (green) lose a lot of kinetic energy after the sharp base, while smooth distances (orange) allow the point to roll out of the bowl. In the deep bowl (bottom), exact distances simply get stuck, while smooth distances are able to continue past the sharp base for a time.

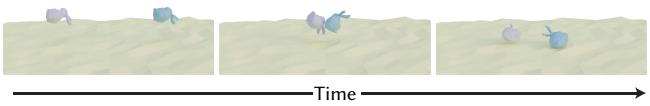


Fig. 16. Two bunnies colliding and falling onto a bumpy plane, where they bounce along the surface.

(Fig. 16). The two bunnies bounce off each other and fall down, without intersections. Now we depart from traditional examples and simulate co-dimensional geometry. We can simulate the edges of a faceted icosphere falling into a net-like bowl (Fig. 17), where it rolls to the other side and falls back into the bowl. Both of these meshes are represented as edge meshes in the simulation, and we see again that there is no interpenetration. Similarly, we can show a sphere roll down a slide represented as an edge mesh (Fig. 18). In a more complex scenario, we can mix primitive types in a mesh and still compute distances. We can take a sphere mesh, attach some spikes represented as edges to it, and then simulate this shape falling into the net bowl (Fig. 19). We see one of the spikes hit the lip of the bowl and the spiky ball rolling up the other side of the bowl.

We can go even further and use our function to simulate cases that have not been well-defined in previous methods, like point cloud collisions. Although recent work has simulated contact with highly co-dimensional geometry [Ferguson et al. 2021; Li et al. 2021], the inflated isosurfaces provided by $\hat{d}$ allow us to close the surface defined by the point cloud without a surface reconstruction preprocess. We demonstrate this by tossing a trefoil knot, represented as a piecewise linear curve, into a ring toss game represented as a point cloud (Fig. 20). In larger examples, we simulate an octopus triangle mesh (Fig. 21) and a point cloud bunny (Fig. 22) sliding and rolling down point cloud terrain acquired from a lidar scanner. We also simulated a motorcycle jumping onto a lidar-acquired point cloud street (Fig. 1).

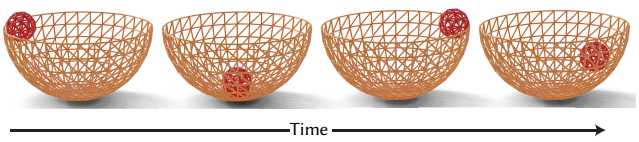


Fig. 17. An icosphere edge mesh is dropped into a hemispherical bowl which is also represented as an edge mesh, where it rolls around in the bowl. Note that both meshes are only rendered as triangle meshes, and are represented in the simulation as edge meshes.

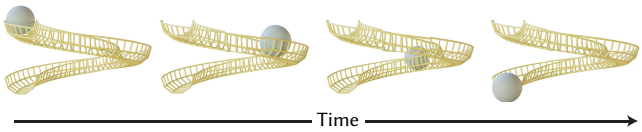


Fig. 18. A triangle mesh sphere rolls down a twisting slide made of edges.

We summarize the performance results of the 3D simulations in Table 1, in particular the average time taken to evaluate $\hat{d}(\mathcal{M}, \bar{\mathcal{M}})$. Each evaluation was run with 28 CPU threads parallelizing over per-primitive distance computations.

5 LIMITATIONS AND FUTURE WORK

We have demonstrated a variety of applications of our method, but it is not a panacea. For example, our method tends to be overly conservative in contact resolution, and may need to be augmented with a more exact method to accurately handle tight contacts like a nut screwing into a bolt. Furthermore, it can be difficult to select parameters optimally. Although we can select β satisfactorily, α and related parameters require some extra care to select in a way that does not produce `inf` values and yields highly accurate solutions. One possibility would be to analyze the distribution of mesh edge lengths to pick a reasonable α that is robust to noise in the distribution. Furthermore, our weight functions, while effective, are designed heuristically, and it would be interesting to see if there is a more exact way to define them based on the underlying geometry as well as the connectivity. They are also only applied to the data mesh in the current formulation because our weights require a global minimizer for computing barycentric coordinates of the closest point. Generalizing weights to work for both data and query meshes, perhaps through a more theoretically grounded weight function, is interesting future work. Another related direction of future work is analyzing the distribution of a point cloud to eliminate concentration artifacts caused by non-uniform point distributions. Although this is not a problem for point clouds that come from most lidar scanners, it is a useful property to ensure the generalizability of our method. Working with noisy point clouds also makes it desirable to eliminate such noise from the dataset entirely. Smooth distances do not amplify noisy data like exact distances, but the noise still exists in the underlying dataset, which can be problematic particularly at high values of α where the noise begins to separate from the main body into small satellite regions.

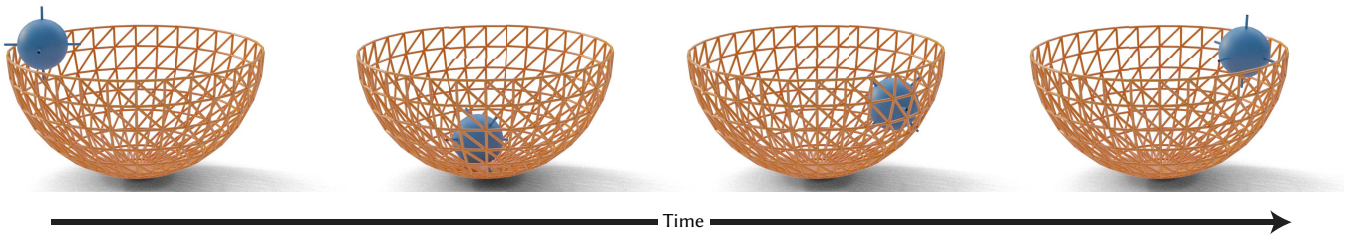


Fig. 19. A spiky sphere made from a triangle mesh and edges for the spikes, falls into a bowl edge mesh and rolls up the side of the bowl, with the spikes hitting the bowl's wires. Due to low α values, the spikes rarely poke through the bowl.

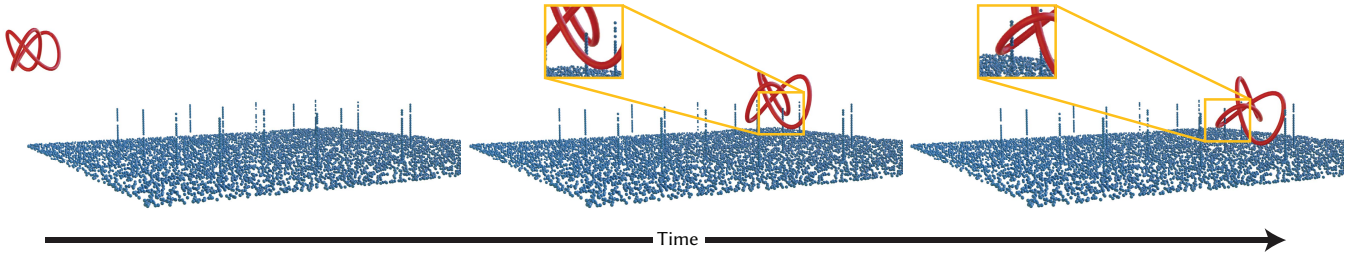


Fig. 20. A trefoil knot is thrown into a ring toss game, represented by point samples, where it hits a ring spike and slides around it before resting on the base.

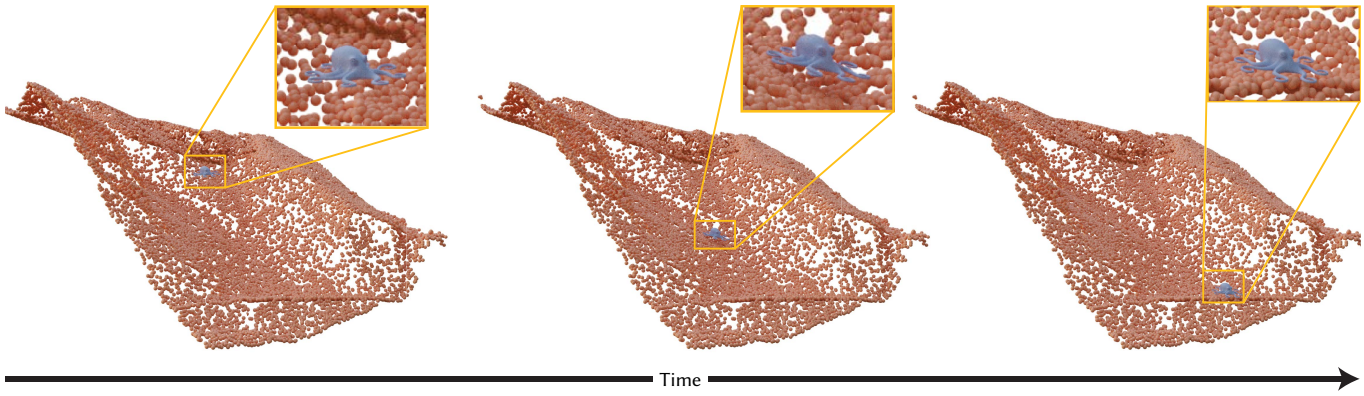


Fig. 21. An octopus triangle mesh slides through terrain defined by a lidar point cloud.

Table 1. Timing results from the simulations presented in the paper. We give information about the data mesh and query mesh, as well as the average distance evaluation time and the simulation time step size. Each distance evaluation was parallelized over 28 CPU threads.

$\mathcal{M}$	$\mathcal{M}$ *Type*	$\lvert F\rvert$	α	$\bar{\mathcal{M}}$	$\bar{\mathcal{M}}$ *Type*	$\lvert\bar{F}\rvert$	α_q	*Avg. Dist Time (ms)*	*dt (s)*
Bowl	Edges	736	50	Icosphere	Edges	120	50	2.72934	1/200
Bowl	Edges	736	50	Spiky Sphere	Tris+Edges	966	20	34.7789	1/1000
Terrain	Points	6591087	100	Bunny	Points	3485	20	250.0031	1/100
Terrain	Points	6591087	100	Octopus	Tris	4432	1000	189.5989	1/100
Bunny	Tris	6966	200	Bunny	Tris	6966	200	92.5717	1/400
Bumpy Plane	Tris	800	200	Bunny	Tris	6966	200	470.3857	1/400
Slide	Edges	615	100	Sphere	Tris	960	100	33.5601	1/100
Ring Toss	Points	12400	20	Trefoil	Edges	100	20	1.4727	1/200
City Street	Points	6747648	50	Motorcycle	Tris	8800	100	581.2362	1/100

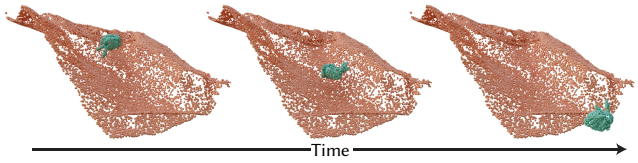


Fig. 22. A bunny point cloud rolls and bounces along a lidar point cloud hill.

Another avenue for future work is applying our method to deforming meshes in both elastodynamics simulations and friction simulation. Due to α's dependence on the underlying mesh, it would also need to change over time, so it would be interesting to treat α as part of the simulation's evolving state. Friction is particularly interesting because it involves computing a tangent plane at each contact point, and since contact points are implicitly wrapped in the constraint, such a formulation would require a smooth friction computation over every primitive in the data and query mesh. We also believe that there are interesting applications in purely particle-based simulation methods like spherical particle hydrodynamics that have no background grid to store implicit functions, where our method could be dropped in to render the fluid surface or induce collision forces onto the particles from other objects.

Another useful application is to integrate smooth distances into existing simulation frameworks such as IPC [Li et al. 2020b] and application extensions [Fang et al. 2021], replacing multiple exact distance constraints with a single smooth distance constraint. Although these methods can be implemented using first-order derivative information via gradient descent, their efficient implementations require Hessians to use in Newton iterations. Smooth distance Hessians inherit discontinuities at projected primitive boundaries from their constituent pairwise primitive distances (though C^1 functions are not nearly as problematic as C^0 functions — for example, see the Supplemental of Li et al. [2020b]), but the more pressing issue is computing and storing them efficiently, since they are large dense matrices. It would be worthwhile to investigate how these Hessians can be approximated using techniques like hierarchical matrices.

The explosion of popularity in implicit functions through reconstruction work such as neural SDFs means that improved computational tools for implicit functions are on the horizon, which can also be used to improve the versatility of smooth distances. For example, computing inertial quantities by directly integrating over the volume enclosed by the zero isosurface would allow simulation frameworks to forget about the underlying geometry almost entirely.

The parallelizability of our method also makes it interesting to consider its integration in purely GPU-based applications. In particular, there are many possible locations for parallelization; for example, while our current implementation parallelizes the query primitive distance computations, an alternative approach could parallelize the traversals in each per-query primitive evaluation. We believe analyzing the tradeoffs of these sort of approaches in the style of Halide [Ragan-Kelley et al. 2013] is promising future work.

6 CONCLUSION

We have presented a smooth distance function that can be efficiently evaluated in such a way that it conservatively estimates the distance to the underlying geometry. Our function works on various types of geometry that can be encountered: points, line segments, and triangles, and utilizes weight functions to eliminate isosurface artifacts. We have benchmarked our method on the Thingi10K dataset and shown that it scales quite well for large geometry. It enables applications such as rigid body contact with lidar point cloud data. We believe this geometric abstraction is very powerful, and due to its basic preprocessing requirements (simply building a BVH), it can provide a lightweight yet versatile augmentation to the underlying geometry.

ACKNOWLEDGMENTS

This research has been funded by in part by NSERC Discovery (RGPIN-2017-05524), Connaught Fund (503114), Ontario Early Researchers Award (ER19-15-034), Gifts from Adobe Research and Autodesk, and the Canada Research Chairs Program.

The authors would like to thank Hsueh-Ti Derek Liu, Silvia Sellán, Ty Trusty, Yixin Chen, and Honglin Chen for proofreading, and Silvia Sellán and Ty Trusty for help in figure rendering. The motorcycle, deer, and octopus models are from the Thingi10K dataset; the bunny is from the Stanford 3D Scanning Repository; the city lidar point cloud is from the Toronto-3D dataset; and the terrain point cloud is from OpenTopography.

REFERENCES

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. 2003. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics* 9, 1 (2003), 3–15.

Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–12.

Josh Barnes and Piet Hut. 1986. A hierarchical O (N log N) force-calculation algorithm. *nature* 324, 6096 (1986), 446–449.

Alexander Belyaev, Pierre-Alain Fayolle, and Alexander Pasko. 2013. Signed Lp-distance fields. *Computer-Aided Design* 45, 2 (2013), 523–528. https://doi.org/10.1016/j.cad.2012.10.035 Solid and Physical Modeling 2012.

Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. 2001. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 67–76.

Zhiqin Chen and Hao Zhang. 2019. Learning Implicit Fields for Generative Shape Modeling. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5932–5941. https://doi.org/10.1109/CVPR.2019.00609

Erwin Coumans and Yunfei Bai. 2016–2021. PyBullet, a Python module for physics simulation for games, robotics and machine learning. http://pybullet.org.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2017. The Heat Method for Distance Computation. *Commun. ACM* 60, 11 (Oct. 2017), 90–99. https://doi.org/10.1145/3131280

Yu Fang, Minchen Li, Chenfanfu Jiang, and Danny M. Kaufman. 2021. Guaranteed Globally Injective 3D Deformation Processing. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 75 (2021).

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 Trans. Graph.* 40, 4, Article 183 (2021).

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

Karthik S. Gurumoorthy and Anand Rangarajan. 2009. A SchröDinger Equation for the Fast Computation of Approximate Euclidean Distance Functions. In *Proceedings of the Second International Conference on Scale Space and Variational Methods in Computer Vision* (Voss, Norway) *(SSVM '09)*. Springer-Verlag, Berlin, Heidelberg, 100–111. https://doi.org/10.1007/978-3-642-02256-2_9

Alireza Ghaffari Hadigheh, Oleksandr Romanko, and Tamás Terlaky. 2007. Sensitivity analysis in convex quadratic optimization: simultaneous perturbation of the objective and right-hand-side vectors. *Algorithmic Operations Research* 2, 2 (2007), 94–94.
Michal Hapala, Tomáš Davidovič, Ingo Wald, Vlastimil Havran, and Philipp Slusallek. 2011. Efficient stack-less BVH traversal for ray tracing. In *Proceedings of the 27th Spring Conference on Computer Graphics*. 7–12.
John C Hart. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer* 12, 10 (1996), 527–545.
Alec Jacobson, Daniele Panozzo, et al. 2018. libigl: A simple C++ geometry processing library. https://libigl.github.io/.
William Kahan. 2004. On the cost of floating-point computation without extra-precise arithmetic. *World-Wide Web document* (2004), 21.
Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7.
Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)* 32, 3 (2013), 1–13.
G. Kreisselmeier and R. Steinhauser. 1979. Systematic Control Design by Optimizing a Vector Performance Index. *IFAC Proceedings Volumes* 12, 7 (1979), 113–117. https://doi.org/10.1016/S1474-6670(17)65584-8 IFAC Symposium on computer Aided Design of Control Systems, Zurich, Switzerland, 29-31 August.
Lei Lan, Yin Yang, Danny M. Kaufman, Junfeng Yao, Minchen Li, and Chenfanfu Jiang. 2021. Medial IPC: Accelerated Incremental Potential Contact With Medial Elastics. *ACM Trans. Graph.* (2021).
Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020a. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Trans. Graph. (SIGGRAPH)* 39, 4, Article 49 (2020).
Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. 2020b. Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics. *ACM Transactions on Graphics* 39, 4 (2020).
Minchen Li, Danny M. Kaufman, and Chenfanfu Jiang. 2021. Codimensional Incremental Potential Contact. *ACM Trans. Graph.* 40, 4, Article 170 (2021).
Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. 2020. Local Optimization for Robust Signed Distance Field Collision. *Proc. ACM Comput. Graph. Interact. Tech.* 3, 1, Article 8 (April 2020), 17 pages. https://doi.org/10.1145/3384538
Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*. 49–54.
Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.* 30, 4, Article 37 (July 2011), 12 pages. https://doi.org/10.1145/2010324.1964932
Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.
Nathan Mitchell, Mridul Aanjaneya, Rajsekhar Setaluri, and Eftychios Sifakis. 2015. Non-Manifold Level Sets: A Multivalued Implicit Surface Representation with Applications to Self-Collision Processing. *ACM Trans. Graph.* 34, 6, Article 247 (Oct. 2015), 9 pages. https://doi.org/10.1145/2816795.2818100
Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2e ed.). Springer, New York, NY, USA.
NVIDIA. 2021. NVIDIA PhysX SDK. https://developer.nvidia.com/physx-sdk.
Julian Panetta, Abtin Rahimian, and Denis Zorin. 2017. Worst-Case Stress Relief for Microstructures. *ACM Trans. Graph.* 36, 4, Article 122 (July 2017), 16 pages. https://doi.org/10.1145/3072959.3073649
Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
Jianbo Peng, Daniel Kristjansson, and Denis Zorin. 2004. Interactive Modeling of Topologically Complex Geometric Detail. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 635–643. https://doi.org/10.1145/1015706.1015773
Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.
Manu Sethi, Anand Rangarajan, and Karthik Gurumoorthy. 2012. The Schrödinger distance transform (SDT) for point-sets and curves. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 198–205. https://doi.org/10.1109/CVPR.2012.6247676
Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4 (2015).
Brian Wyvill, Andrew Guy, and Eric Galin. 1999. Extending the CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System. *Computer Graphics Forum* 18, 2 (1999), 149–158. https://doi.org/10.1111/1467-8659.00365 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00365
Chris Yu, Henrik Schumacher, and Keenan Crane. 2021. Repulsive Curves. *ACM Trans. Graph.* 40, 2 (2021).
Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. 2020. *Dive into Deep Learning*. https://d2l.ai.
Hong-Kai Zhao, Stanley Osher, and Ronald Fedkiw. 2001. Fast surface reconstruction using the level set method. In *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision*. IEEE, 194–201.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

A PROOF OF UNDERESTIMATE PROPERTY

Here we prove that smooth distances underestimate the true distance, when each distance contribution has an associated weight function.

Theorem A.1. *Let $\mathcal{M} = (V, F)$ be the data mesh, and let g be a query primitive, and each $f_i \in F$ has an associated weight function $w_i(g)$, where $1 \leq w_i(g) \leq A$ for some constant A. Suppose $d_{min} = \min_{f_i \in F} d(f_i, g)$. Then, defining $\hat{d}(\mathcal{M}, g)$ as in Eq. 3 (but using a general query primitive g instead of a point), we have $d_{min} \geq \hat{d}(\mathcal{M}, g) \geq d_{min} - \frac{\log A|F|}{\alpha}$ for all $\alpha > 0$.*

Proof. Let $k = \operatorname{argmin}_{f_i \in F} d(f_i, g_j)$. Using this notation, $d_{min} = d_k$. Then,

$$
\begin{aligned}
d_k &= -\frac{1}{\alpha} \log\left(\exp\left(-\alpha d_k\right)\right) \\
&\geq -\frac{1}{\alpha} \log\left(\exp\left(-\alpha d_k\right)\right) - \frac{1}{\alpha} \log w_k(g) && (w_k(g) \geq 1) \\
&= -\frac{1}{\alpha} \log\left(w_k(g) \exp\left(-\alpha d_k\right)\right) \\
&\geq -\frac{1}{\alpha} \log\left(\sum_{f_i \in F} w_i(g) \exp(-\alpha d(f_i, g))\right) && (\hat{d}(\mathcal{M}, g)) \\
&\geq -\frac{1}{\alpha} \log\left(\sum_{f_i \in F} A \exp(-\alpha d(f_i, g))\right) && (w_i(g) \leq A) \\
&\geq -\frac{1}{\alpha} \log\left(\sum_{f_i \in F} A \exp\left(-\alpha d_k\right)\right) \\
&= d_k - \frac{\log A|F|}{\alpha}
\end{aligned}
$$

Therefore, we have $d_{min} \geq \hat{d}(\mathcal{M}, g) \geq d_{min} - \frac{\log A|F|}{\alpha}$. □

B EXACT DISTANCE FORMULATION

Here we discuss our distance formulation between simplices f and g. Denoting barycentric coordinates of f as $\boldsymbol{\phi}$ and g as $\boldsymbol{\lambda}$, the points on each simplex referenced by the barycentric coordinates are denoted as $f(\boldsymbol{\phi})$ and $g(\boldsymbol{\lambda})$ respectively. (For points, the barycentric coordinate vector is 0-dimensional, so the barycentric coordinate vector can be omitted.) For example, if f is a triangle consisting of 3 vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$, and $\boldsymbol{\phi} = [\phi^1, \phi^2]^\top$, $f(\boldsymbol{\phi}) = (1 - \phi^1 - \phi^2)\mathbf{v}_0 + \phi^1 \mathbf{v}_1 + \phi^2 \mathbf{v}_2$. Furthermore, we are restricted to convex combinations of $\mathbf{v}_i$ (i.e., $0 \leq \phi^i \leq 1$, and $\sum_i \phi^i \leq 1$). Since any point on a simplex can be referenced using barycentric coordinates, we can determine the

closest point pair on f and g by minimizing the squared distance between all points on each simplex:

$$d^2(f,g) = \min_{\boldsymbol{\phi}^*,\boldsymbol{\lambda}^*} \|f(\boldsymbol{\phi}^*) - g(\boldsymbol{\lambda}^*)\|^2, \tag{10}$$

where $\boldsymbol{\phi}$ and $\boldsymbol{\lambda}$ are the argmin of the problem. Combined with the aforementioned constraints on $\boldsymbol{\lambda}$ and $\boldsymbol{\phi}$, we have a quadratic program with linear constraints (and in the case of points, the problem simplifies to the L^2 norm).

We are also interested in taking derivatives of this distance. It is well-known that quadratic programs are differentiable [Hadigheh et al. 2007], and in this case, the distance gradient (with respect to g) is equivalent to the distance gradient of the closest point pair $f(\boldsymbol{\phi})$ and $g(\boldsymbol{\lambda})$ with respect to $g(\boldsymbol{\lambda})$, which is

$$\nabla d(f,g) = \begin{cases} \frac{g(\boldsymbol{\lambda}) - f(\boldsymbol{\phi})}{\|g(\boldsymbol{\lambda}) - f(\boldsymbol{\phi})\|} & d(f,g) \neq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Although there is a gradient discontinuity when f and g intersect, this is not an issue in our simulation applications, because smooth distance constraints ensure that the underlying geometries will never touch.

This formulation is also tied to the closest point projection function used to define weight functions in Section 3.3. Given a closest point projection $\boldsymbol{\pi}$ onto simplex f, $\boldsymbol{\pi}(g) = \boldsymbol{\pi}(g(\boldsymbol{\lambda})) = f(\boldsymbol{\phi})$.

C WEIGHT FUNCTIONS

Here we provide more details on the construction and storage of weight functions $w_i(\mathbf{q})$ for edges and triangles.

C.1 Pointwise Case Analysis

To identify appropriate point constraints for the weight polynomials, we derive the *exact* weights in various cases depending on the location of the closest point on a data mesh $\mathcal{M} = (V, F)$ to point $\mathbf{q}$. First, suppose the closest point is on a vertex $\mathbf{v}_k \in V$. Looking at $\hat{d}$, and assuming α is large, we have

$$\begin{aligned} \hat{d}(\mathcal{M},\mathbf{q}) &= -\frac{1}{\alpha}\log\left(\sum_{f_i\in F} w_i(\mathbf{q})\exp(-\alpha d(f_i,\mathbf{q}))\right) \\ &\approx -\frac{1}{\alpha}\log\left(\left(\sum_{f_i\in \mathcal{N}_k} w_i(\mathbf{q})\right)\exp(-\alpha d(\mathbf{v}_k,\mathbf{q}))\right) \\ &= d(\mathbf{v}_k,\mathbf{q}) - \underbrace{\frac{\log\left(\sum_{f_i\in\mathcal{N}_k} w_i(\mathbf{q})\right)}{\alpha}}_{r_k} \end{aligned}$$

In the above derivation, $\mathcal{N}_k$ denotes the set of one-ring neighbours of $\mathbf{v}_k$, and the second line is a consequence of high α making the other terms in the summation negligible. Our goal is to ensure that $r_k = 0$, which is equivalent to the condition $\sum_{f_i\in\mathcal{N}_k} w_i(\mathbf{q}) = 1$. We can accomplish this by setting $w_i(\mathbf{q}) = \frac{1}{|\mathcal{N}_k|}$ if $\mathbf{q}$'s closest point to f_i is its vertex $\mathbf{v}_k$ (which will be the case for all $f_i \in \mathcal{N}_k$ when $\mathbf{q}$'s closest point on $\mathcal{M}$ is $\mathbf{v}_k$). When $\mathcal{M}$ is a triangle mesh and the closest point is along an edge (k,ℓ), we can use an identical argument to derive $w_i(\mathbf{q}) = \frac{1}{|\mathcal{N}_{k\ell}|}$ where $\mathcal{N}_{k\ell}$ denotes the set of triangles incident on (k,ℓ).

Now suppose the closest point is on a simplex $f_i \in F$ (but not on one of its faces, in which case the earlier discussion applies). Again assuming α is large, we have

$$\begin{aligned} \hat{d}(\mathcal{M},\mathbf{q}) &= -\frac{1}{\alpha}\log\left(\sum_{f_i\in F} w_i(\mathbf{q})\exp(-\alpha d(f_i,\mathbf{q}))\right) \\ &\approx -\frac{1}{\alpha}\log\left(w_i(\mathbf{q})\exp(-\alpha d(f_i,\mathbf{q}))\right) \\ &= d(f_i,\mathbf{q}) - \frac{\log w_i(\mathbf{q})}{\alpha} \end{aligned}$$

In this case, we can simply set $\tilde{w}_i(\mathbf{q}) = 1$.

Of course, we cannot directly use these exact weights, since they are not smooth and do not satisfy the conservative property $w_i(\mathbf{q}) \geq 1$. Instead, we use these cases as guidelines for point constraints in building non-conservative polynomial weights $\tilde{w}_i$ and computing an appropriate scale factor A to ensure the conservative property holds.

To simplify the notation in the remainder of this section, we will work with barycentric coordinates of the closest point projection of $\mathbf{q}$ onto f_i (i.e., $\tilde{w}_i(\boldsymbol{\phi}_i)$).

C.2 Edge Weights

Edges have a single barycentric coordinate, so the weight function of an edge f_i in terms of barycentric coordinates is $\tilde{w}_i(\phi_i)$. The weight function is a 4th order polynomial, with point constraints $\tilde{w}_i(0) = \frac{1}{|\mathcal{N}_{i_0}|}$, $\tilde{w}_i(1) = \frac{1}{|\mathcal{N}_{i_1}|}$, and $\tilde{w}_i(0.5) = 1$, and derivative constraints $\tilde{w}_i'(0) = \tilde{w}_i'(1) = 0$. Using the polynomial coefficients as unknowns, the five above equations become a linear system that can be solved for unique coefficients that satisfy the constraints. We have set the three extrema of this quartic function by construction, which allows us to cheaply compute $A = \max_k |\mathcal{N}_k|$.

C.3 Triangle Weights

Triangles have two barycentric coordinates, so the weight function of a triangle f_i in terms of barycentric coordinates is $\tilde{w}_i(\phi_i^1, \phi_i^2)$. In order to satisfy the constraints, $\tilde{w}_i$ is a 7th order polynomial, with 36 coefficients. Since we must store a set of coefficients for each triangle in a mesh, we additionally require that the function is symmetric, $\tilde{w}_i(\phi_i^1, \phi_i^2) = \tilde{w}_i(\phi_i^2, \phi_i^1)$, so that we can roughly halve the number of coefficients that must be stored.

We use 10 point constraints: one per vertex, two per edge (spaced equally along their lengths), and one for the triangle barycenter. In order to ensure that $\tilde{w}_i$ is symmetric, we assign the same value at each vertex constraint and at each edge constraint, based on the maximum vertex and edge valence, respectively. More concretely, we first find $v_i = \max\{|\mathcal{N}_{i_0}|, |\mathcal{N}_{i_1}|, |\mathcal{N}_{i_2}|\}$ and $e_i = \max\{|\mathcal{N}_{i_{01}}|, |\mathcal{N}_{i_{12}}|, |\mathcal{N}_{i_{20}}|\}$ (noting that $v_i \geq e_i$), and assign $\tilde{w}_i(0,0) = \tilde{w}_i(1,0) = \tilde{w}_i(0,1) = \frac{1}{v_i}$, and $\tilde{w}_i(1/3,0) = \tilde{w}_i(2/3,0) = \tilde{w}_i(1/3,2/3) = \tilde{w}_i(2/3,1/3) = \tilde{w}_i(0,1/3) = \tilde{w}_i(0,2/3) = \frac{1}{e_i}$. Without carefully setting the final barycenter constraint, it is possible to produce spurious local extrema in regions other than the vertices, edges, and barycenter, which is problematic for determining an appropriate A, and can

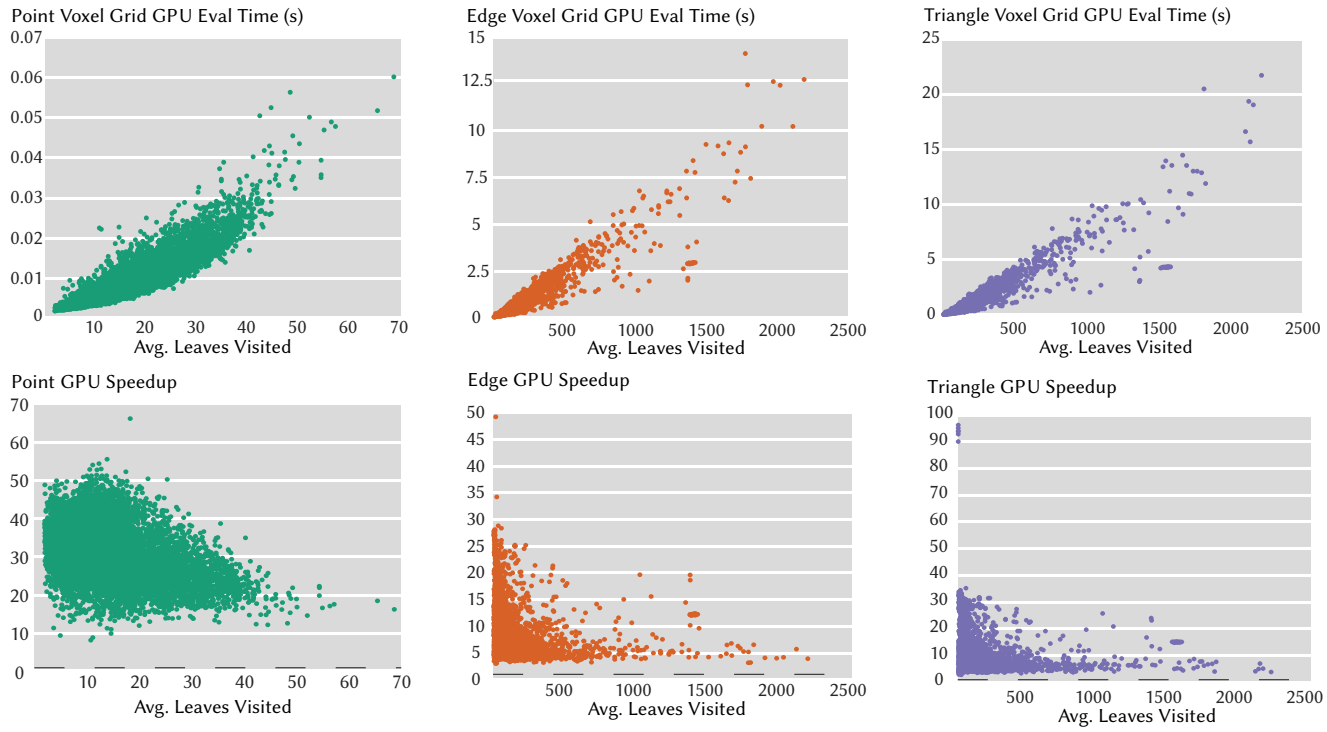


Fig. 23. Results of the Thingi10K benchmark run on the GPU, with the same test as in Fig. 14. The top row shows times and average leaves visited, while the bottom row shows speedups relative to the CPU benchmark. The dotted lines represent a speedup of 1 (i.e., identical performance on the CPU and GPU).

even lead to phenomena like negative weights. We have found that a barycenter constraint of $\tilde{w}_i(1/3, 1/3) = \frac{v_i - 1}{v_i}$ prevents this spurious extrema issue; although the barycenter weight is no longer 1, the weight at the barycenter is still significantly higher than at the vertices, so the deviation from exact weights is acceptable. Then, like with edges, we have $A = \max_i v_i$.

The normal derivative constraints, are more complex than for edges: we must enforce constraints on the lines $\phi_i^1 = 0$, $\phi_i^2 = 0$, and $\phi_i^1 + \phi_i^2 = 1$ within the barycentric triangle. Taking care to ensure symmetry, these constraints are $\frac{\partial}{\partial \phi_i^1}\tilde{w}_i(0,t) = \frac{\partial}{\partial \phi_i^2}\tilde{w}_i(t,0) = 0$ for all $t \in [0,1]$, and $\left(\frac{\partial}{\partial \phi_i^1} + \frac{\partial}{\partial \phi_i^2}\right)\tilde{w}_i(t,1-t) = \left(\frac{\partial}{\partial \phi_i^1} + \frac{\partial}{\partial \phi_i^2}\right)\tilde{w}_i(1-t,t) = 0$ for all $t \in [0,1]$. Essentially, each constraint equation is assigning a 6th order univariate polynomial to be zero over a line segment, which can only happen with the zero polynomial. Thus, each coefficient of these polynomials, which is a linear combination of coefficients of $\tilde{w}_i$, must be 0 as well, creating 7 new linear equations per constraint. Noting that the 1st order coefficients of $\frac{\partial}{\partial \phi_i^1}\tilde{w}_i(0,t)$ and $\frac{\partial}{\partial \phi_i^2}\tilde{w}_i(t,0)$ are equal, as well as the 6th order coefficients of $\left(\frac{\partial}{\partial \phi_i^1} + \frac{\partial}{\partial \phi_i^2}\right)\tilde{w}_i(t,1-t)$ and $\left(\frac{\partial}{\partial \phi_i^1} + \frac{\partial}{\partial \phi_i^2}\right)\tilde{w}_i(1-t,t)$, we have 26 unique equations, leading to a total of 36 equations. This system of equations still has a non-trivial null space, but we can solve for reasonable weight coefficients using the pseudoinverse.

To further reduce the memory footprint of these weights, we observe that the constraint equations corresponding to $\phi_i^1 = 0$ and $\phi_i^2 = 0$ are each in terms of a single $\tilde{w}_i$ coefficient, so we do not need to store those 13 coefficients at all. Combined with symmetry, we only need to store 13 unique coefficients per weight function.

D FAR FIELD APPROXIMATION

Here we describe the Barnes-Hut far field approximation in more detail. Let B be a bounding region of points, n_B be the number of points in B, $|B|$ be the diameter of B (e.g., the bounding box diagonal), and $\mathbf{y}_B$ be the center of the far field approximation of B. If $\frac{|B|}{d(\mathbf{y}_B, g)} < \beta$ for some user-defined β, then approximate the exponential summation over points B as $\sum_{f_i \in B} \exp(-\alpha d(f_i, g)) \approx n_B \exp(-\alpha d(\mathbf{y}_B, g))$. Similarly, the gradient contribution of B (in the numerator summation of Eq. 4) is now $n_B \exp(-\alpha d(\mathbf{y}_B, g)) \frac{g(\boldsymbol{\lambda}) - \mathbf{y}_B}{\|g(\boldsymbol{\lambda}) - \mathbf{y}_B\|}$. When our data primitives are edges or triangles, we must take some care with the weight function, which must be incorporated to ensure the approximation is reasonably smooth. Since the approximation is just a single point, there is no undesirable concentration in some regions and we can simply use a weight of $w = A^S$, which corresponds to a constant weight function scaled by A and attenuated by S.

Viewing the far field approximation as a Taylor series, we have only included the constant term. However, we found that higher-order Taylor series terms tended to produce worse results due to the off-center expansion point amplifying the error of those terms.

E GPU BENCHMARK

Here we present the results of our GPU benchmark. Structurally, it is identical to the benchmark in Section 4.4, but it is run on a GPU instead of several CPU threads. We ran these tests on a Titan RTX, and implemented our method in CUDA by ensuring our CPU code was also GPU-compatible. The results are presented in Fig. 23. We see that, while performance is still linearly proportional to leaves visited, and the GPU code consistently outperforms CPU code by staying over the speedup=1 line and achieving an average 30× speedup on points and 5× speedup on edges and triangles, the relative speedups vary greatly, and are significantly reduced as more leaves are visited. This is likely because global memory accesses become a significant issue in these cases, on top of the additional computation. This is an especially pronounced issue on GPUs since memory is much slower to access on a GPU than a CPU.