# *Robust Biharmonic Skinning Using Geometric Fields*

# Robust Biharmonic Skinning Using Geometric Fields

ANA DODIK, CSAIL, Massachusetts Institute of Technology, Cambridge, United States
VINCENT SITZMANN, CSAIL, Massachusetts Institute of Technology, Cambridge, United States
JUSTIN SOLOMON, CSAIL, Massachusetts Institute of Technology, Cambridge, United States
ODED STEIN, University of Southern California, Los Angeles, United States


Fig. 1. Bounded biharmonic weights produced by our method can be used to extrapolate color (top row) or deformations (bottom row) from control handles. Our method succeeds on meshes that are typically difficult to handle with standard tools. As an extreme example, the Scorpion (Randomized) triangle soup is missing 30% of its faces and has had the remaining ones randomly perturbed. We can robustly compute weights for meshes created using off-the-shelf 3D scanning software even if they are non-watertight and contain thin and self-intersecting triangles as does the Beaver mesh. Similarly, our method can be used to deform virtual reality ribbon drawings [Rosales et al. 2019], such as the Piggybank mesh.



Bounded bihramonic weights are a popular tool used to rig and deform characters for animation, to compute reduced-order simulations, and to define feature descriptors for geometry processing. They necessitate tetrahedralizing the volume bounded by the surface, introducing the possibility of meshing artifacts or tetrahedralization failure. We introduce a *mesh-free* and

Authors' Contact Information: Ana Dodik, CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States; e-mail: anadodik@mit.edu; Vincent Sitzmann, CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States; e-mail: sitzmann@mit.edu; Justin Solomon, CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States; e-mail: jsolomon@mit.edu; Oded Stein, University of Southern California, Los Angeles, California, United States; e-mail: oded.stein.graphics@gmail.com.


ACM 1557-7368/2025/10-ART
https://doi.org/10.1145/3771928

*robust* automatic skinning technique that generates weights comparable to the current state of the art, but works reliably even on open surfaces, triangle soups, and point clouds where current methods fail. We achieve this through the use of a specialized Lagrangian representation enabled by the advent of hardware ray-tracing, which circumvents the need for finite elements while optimizing the biharmonic energy and enforcing boundary conditions. The flexibility of our formulation allows us to integrate artistic control through weight painting during the optimization. We offer a thorough qualitative and quantitative evaluation of our method.

CCS Concepts: • **Computing methodologies** → **Mesh geometry models**; **Mesh models**; **Animation**; **Point-based models**; **Modeling and simulation**; *Ray tracing*; *Visibility*.

Additional Key Words and Phrases: skinning weights, bounded biharmonic weights, geometry processing, deformation, partial differential equations, variational problems

## 1 Introduction

The dominant pipeline for computer animation relies on *deformation skeletons* composed of control handles (typically points and bones), each of which has an associated region of influence on the shape. These regions of influence, referred to as *skinning weights*, specify how deformations of the control handles are to be blended and transferred to the shape. The skeleton interface requires users to specify the skinning weights, traditionally through the slow and tedious process of manualweight painting .

To ease this burden, various automatic skinning weight computation methods have been developed. Despite growing interest, data driven approaches remain class-specific, require an abundance of data, and struggle with out-of-distribution examples. Instead, a popular alternative is to formulate skinning weights as minimizers of some smoothness objective, side-stepping generalization issues of data-driven methods (§2.2). Such optimization-based approaches have seen adoption in commercial software, notably, Pinnochio [Baran and Popović 2007] in Blender and bounded biharmonic weights (BBW) [Jacobson et al. 2011] in Adobe Character Animator.

Despite this adoption, manual painting of skinning weights remains predominant. This is, in part, due to **robustness** issues with automatic methods, as they rely on mesh-based discretizations using the finite-element method (FEM). Famously, FEM-based approaches struggle with data common in the real world [Gillespie et al. 2021; Hu et al. 2019; Miller et al. 2023; Sawhney and Crane 2020; Sawhney et al. 2023, 2022; Sharp and Crane 2020a,b; Sharp et al. 2021]. For example, Blender's implementation raises the error `Bone Heat Weighting: Failed to find solution...` due to issues with FEM, resulting in a plethora of online discussions and tutorials dedicated to cleaning up meshes to avoid this error.

Robustness issues are especially pronounced with methods that rely on tetrahedral meshes, as fast and robust tetrahedralization remains a challenging open problem [Diazzi et al. 2023; Hu et al. 2020, 2018]. This is particularly unfortunate, as the state-of-the-art smoothness-based formulation of skinning weights—bounded biharmonic weights [Jacobson et al. 2011]—requires a tetrahedral mesh. Figure 3 demonstrates a common failure case of state-of-the-art fast tetrahedralization software [Hu et al. 2020]. If we want to guarantee that our software can compute a tetrahedral mesh, we either get a fast approximate solution that can lead to visible artifacts, or the tetrahedralization software can take multiple hours only for the downstream FEM solver to fail due to the tetrahedral mesh being too high-resolution.

In addition, manual weight painting remains prevalent because it offers complete **artistic control** over the final look of the animation. Therefore, instead of trying to fully automate the entire skinning pipeline, automatic skinning methods need to offer ways to integrate artistic edits.

Recent function representations popularized in machine learning and vision promise to replace FEM in optimization pipelines for geometry processing tasks while introducing additional artistic control [Dodik et al. 2023]. However, representations like neural fields [Xie et al. 2022] or Gaussian splatting [Kerbl et al. 2023] are not directly applicable to geometry processing due to hard constraints on the set of permissible solutions as dictated by the geometry of the shape (Fig. 5, §4.1), boundary conditions (§4.3) or the problem formulation itself (Eq. 1).

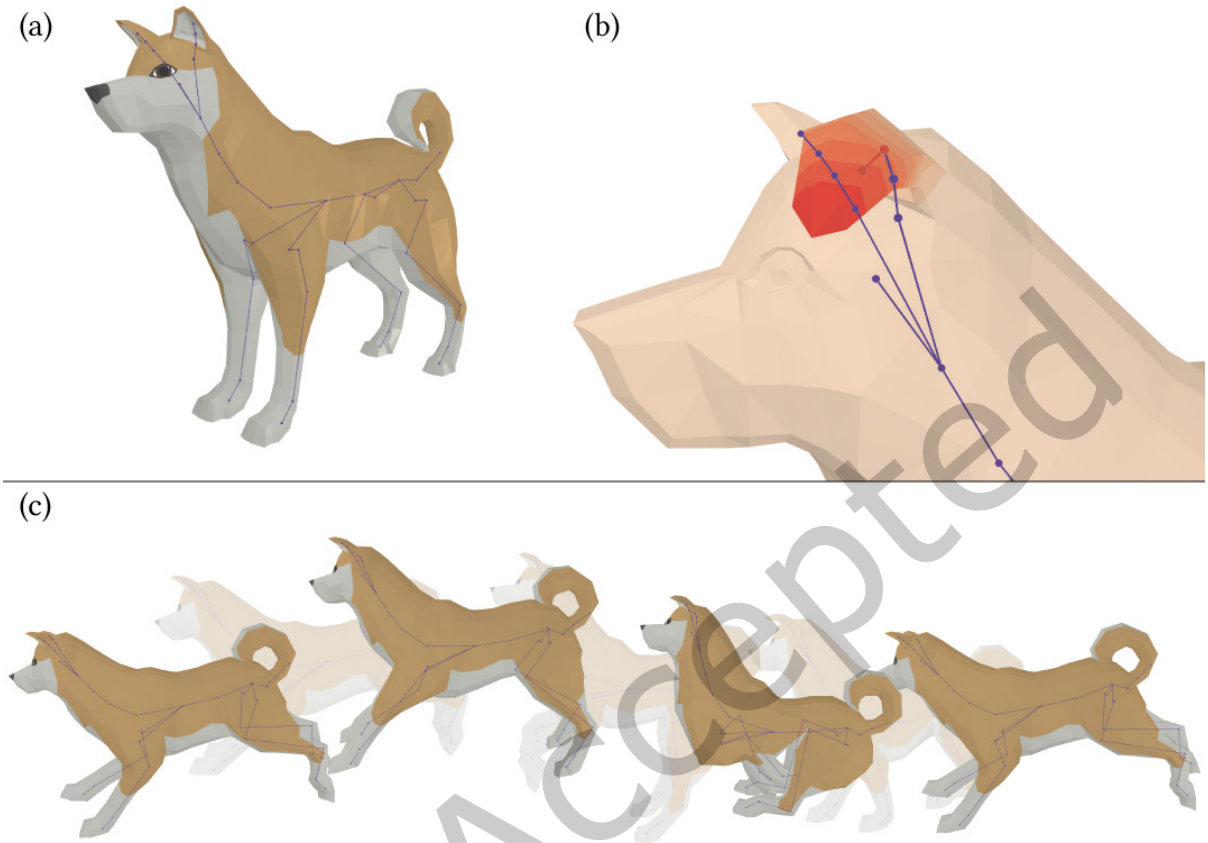


Fig. 2. An illustrative example on the low-poly Shiba Inu mesh, with the skeleton depicted in purple (a). The skinning weight function of the ear bone determines the region of influence of that bone (b). Our method for skinning weight computation produces smooth-looking deformations (c).



We take inspiration from Gaussian splatting but incorporate hard constraints directly into the architecture of the differentiable field, dubbing our construction *geometric fields*.

In this article, we introduce a mesh-free, tailor-made representation for skinning weights that satisfies the constraints of the problem by construction. Our representation not only accelerates weights computation, but also respects the constraints imposed by the geometry of the shape, avoiding artifacts like bleeding (Fig. 5). With our architecture, we can optimize for bounded biharmonic weights via stochastic gradient descent (SGD).

Our approach inherits other benefits of differentiable programming and gradient-based optimization. For example, we can incorporate user-painted weights, offering additional control over the output. Instead of requiring the user to paint the entire shape, our method enables weight painting on a subset of the boundary, which are then included as Dirichlet boundary conditions in the optimization problem. Our system quickly produces an initial solution that a user can paint over according to preference—and then resume the optimization to incorporate the prescribed weights into the solution.

Our weights are universally faster to compute than the original BBW implementation [Jacobson et al. 2011], while fast quasi-harmonic weights (QHW) [Wang and Solomon 2021] are faster only on well-behaved meshes. As soon as robust tetrahedralization is needed, our method is more robust and faster end-to-end than existing alternatives. In many cases, tetrahedralization can entirely fail or produce unusable results (Figure 3); in cases where it succeeds, QHW can crash or be slower than our method (Table 1, Figure 4).

In summary, our main contributions are:

- A robust automatic method for computing skinning weights.
- A Lagrangian representation and approach to solving the bounded biharmonic weights problem without finite elements.
- The use of hardware-accelerated ray tracing for a geometry-aware function parameterization.
- Modifications that allow the method to work on non-watertight domains like triangle soups, ribbon drawings, or point clouds.
- A way of incorporating weight painting into the optimization by introducing Dirichlet boundary conditions.
- A thorough qualitative and quantitative investigation of our method.



## 2 Related Work

In this section, we mention connections of our work to other methods in the geometry processing literature.

### 2.1 Robust Geometry Processing

A significant body of prior work has been dedicated to making FEM-based geometry processing robust. FEM-based algorithms put the burden on the end-user to ensure that meshes are "well-behaved," requiring the user to e.g. remove self-intersections or to ensure mathematical properties such as manifoldness, watertightness, or boundedness of interior angles. While FEM can be a natural fit for meshes, a mesh that is sufficient for representing a shape is not necessarily also a good FEM mesh; moreover, FEM-based methods often require meshing the interior of a volume bounded by a boundary representation, while the latter is sufficient for 3D modeling.

Robust triangle meshing algorithms rely on complex data structures and often parallelize poorly [Gillespie et al. 2021; Hu et al. 2019; Sharp and Crane 2020a,b; Sharp et al. 2021]. Existing tetrahedral meshing algorithms either have strong requirements on the quality of the input (e.g., crashing in the presence of self-intersections) [Diazzi et al. 2023; Tournois et al. 2024] or drastically change the appearance of the mesh [Hu et al. 2020, 2018] as in Figure 3.

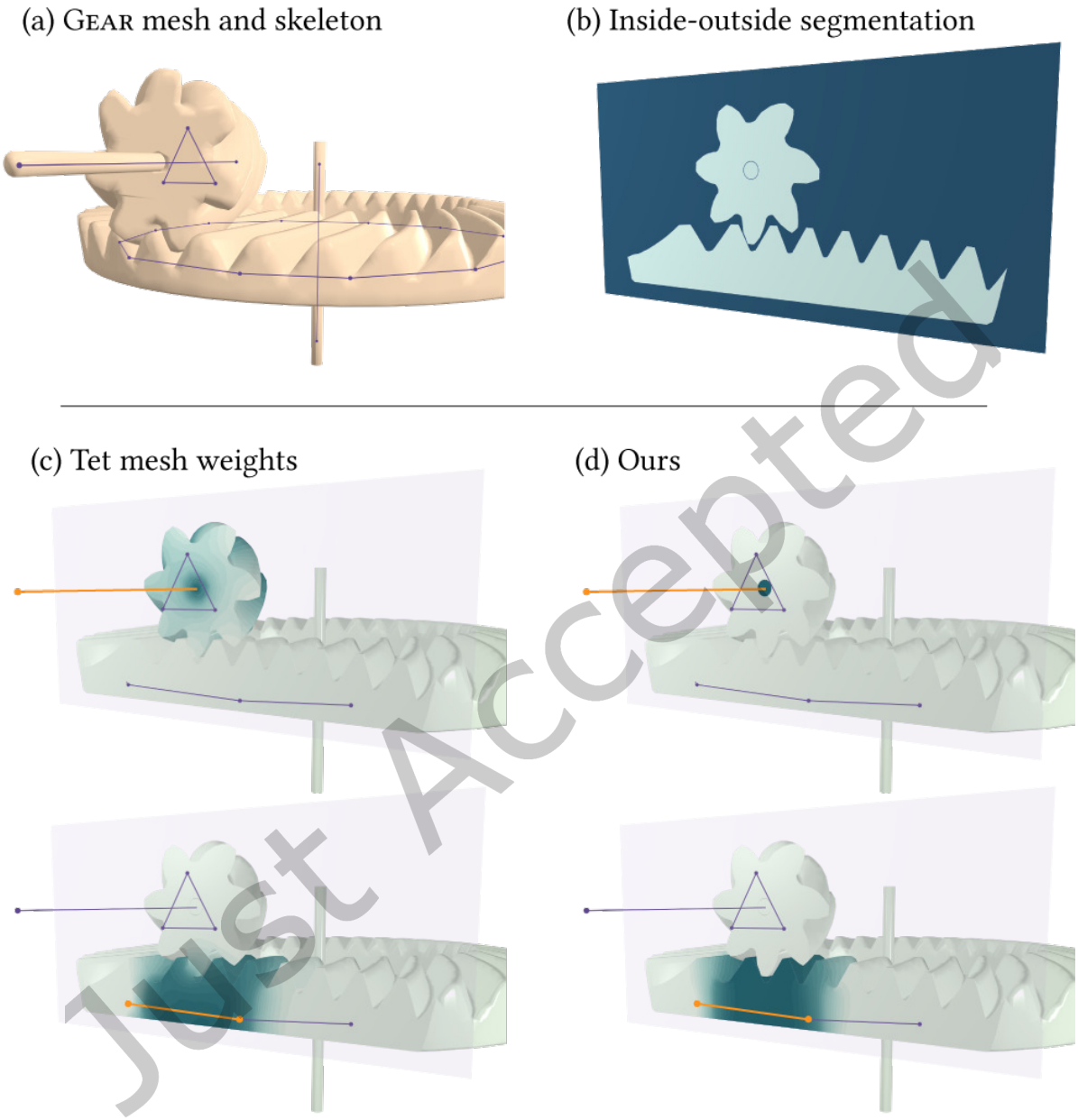




Fig. 3. Our method for bounded biharmonic weights [Jacobson et al. 2011] works on poorly behaved geometry such as the challenging Gear mesh (a). We visualize quantities as colors on a planar slice through the volume. The mesh does not self-intersect, as evidenced by the generalized winding number [Jacobson et al. 2013] (b). Solving for BBW using on a tetrahedral mesh from FastTetWild [Hu et al. 2020] with default parameters results in weights bleeding over boundaries and discretization artifacts (c). Modifying FastTetWild's parameters to respect boundaries increases computation time to **1.78 hours**. In comparison, our weights are smooth, respect boundaries, and can be computed without tetrahedralization in **32.2 seconds** (d).

To sidestep the issues above, there is emerging interest in mesh-free approaches to geometry processing using Monte Carlo methods [Miller et al. 2023; Sawhney and Crane 2020; Sawhney et al. 2023, 2022]. As of now, this approach is not applicable to our setting, as it applies to linear PDEs rather than the inequality-constrained variational problem needed for skinning. Similarly, the boundary element method (BEM) holds some promise for alleviating dependence on tetrahedralization of a boundary representation, but typical BEM algorithms are restricted to linear PDE in the interior of the domain.

A seminal construction in robust geometry processing, the generalized winding number gives a robust means of deciding whether a point is inside or outside a given shape [Jacobson et al. 2013]. It degrades gracefully and produces meaningful results even in the presence of heavily degenerate boundary geometry. Its main application has been to approximate tetrahedral meshing methods, such as the one used to create Figure 3. We use the generalized winding number to alleviate the need for tetrahedralization while evaluating our optimization objective (see Section 5.2).

## 2.2 Automatic Skinning Weights

In an effort to alleviate the largely manual process of painting skinning weights, automatic skinning weights algorithms use input geometry and/or example poses to infer the weight functions. This paper focuses on the former setting—by far the most common, since obtaining an exemplary set of poses for a shape that suggest the underlying skinning weights is itself a hard problem. See e.g. [James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012, 2014; Wampler 2016] for data-driven methods in the latter category; we also mention some neural network architectures that predict skinning weights below.

*PDE-based methods.* Many classical methods for automatic skinning weights are built around the solution of a partial differential equation (PDE) such as the Laplace equation $\Delta w \equiv 0$ [Joshi et al. 2007], which can be efficiently solved with Dirichlet boundary conditions and whose solutions automatically fulfill the constraints needed for skinning. The related *Pinocchio* method [Baran and Popović 2007] is implemented in common modeling software like Blender and Maya. Low-order PDEs like the Laplace equation can lead to artifacts near control handles. This is alleviated by solving the higher-order biharmonic equation $\Delta^2 \alpha \equiv 0$ [Botsch and Kobbelt 2004] or even the triharmonic equation [Jacobson et al. 2010; Tosun 2008]. These methods, however, lose the maximum principle that holds for the Laplace equation, leading to undesirable oscillatory skinning weights.

*Variational methods.* Variational methods add constraints on top of a PDE-derived energy, making the problem nonlinear. The widest-known and state-of-the-art method of this class is *bounded biharmonic weights* (BBW) [Jacobson et al. 2011], which uses a biharmonic energy with nonnegativity and partition of unity constraints—and an implicit Neumann boundary condition [Stein et al. 2018]—leading to a convex quadratic program that can be discretized on a triangle/tetrahedral mesh. BBW provides high-quality, smooth automatic skinning weights at the cost of efficiency and mesh dependence. It has been extended to avoid local extrema [Jacobson et al. 2012a] or to improve efficiency [Wang and Solomon 2021].

These methods, however, still rely on meshing the entire computational domain (a volume in 3D), even though skinning weights are only needed at the outer surface; it is unclear how to extend them to domains that cannot easily be meshed, e.g., because they are not watertight. Jacobson et al. [2013] briefly illustrate how generalized winding numbers—used as a means of distinguishing inside from outside in non-watertight models—can possibly bridge the gap (see their Figure 20). Our extension to non-watertight domains in Section 5.2 also makes use of Jacobson et al. [2013]'s model.

*Additional approaches.* A zoo of methods propose models adjacent to automatic skinning weights computation. For example, Thiery and Eisemann [2018] jointly optimize bone positions and skinning weights using models

from elasticity. Delta mush methods [Le and Lewis 2019] bypass steps of skinning and weight computation by applying *a posteriori* smoothing to linear-blend skinning deformations using binary binding weights. Bang and Lee [2018] propose a spline-based interface to efficiently create and edit skinning weights.

Dionne and de Lasa [2013, 2014] consider the problem of computing robust skinning weights using a closed-form formula in terms of distances; they use a variant of Dijkstra's algorithm to compute distances on a voxel grid of the interior of the domain. Their method is robust to non-watertight edges but requires a dense voxelixation in the presence of thin features to avoid bleeding between disconnected parts of the shape; they also do not optimize a smoothness energy, which can yield artifacts at points where shortest-path distance is not differentiable. Xian et al. [2018] use a similar approach for 2D image deformation but compute interior distances along a graph of mesh edges.

*Neural representations.* Dodik et al. [2023]'s method to compute generalized barycentric coordinates using a neural representation is closely related to our approach; see their work for for past literature on generalized barycentric coordinates. Although the idea of using machine learning-inspired function representations for geometry processing also underlies our work, their method is not directly applicable to our problem. In particular, building in hard constraints of the function class of barycentric coordinates requires a complicated purpose-built architecture that is unnecessary in our formulation (see Section 4.1).

Other works use neural networks to predict skinning weights learned from data. These methods are *not* comparable to ours, since they are data-driven and typically omit the necessary hard constraints, although their architectures parameterize skinning weights functions in various ways, providing a point of high-level comparison. Several axes can be used to compare methods for learning skinning weights. A key decision is the architecture and its interaction with the shape representation; for example, many learned skinning weights employ graph neural networks [Liu et al. 2019; Mosella-Montoro and Ruiz-Hidalgo 2022; Pan et al. 2021], while others use convolutional neural networks [Ouyang and Feng 2020] or Gaussian splatting [Kocabas et al. 2023] to represent and process shapes. Another axis is the training data. While many works rely on direct supervision, some infer skinning weights from deformations or by jointly optimizing bone positions with skinning weights [Chen et al. 2021; Li et al. 2021; Ma and Zhang 2023; Xu et al. 2020; Yang et al. 2021]; others incorporate modalities like video [Liao et al. 2023]. Jeruzalski et al. [2020]; Kant et al. [2023] use neural representations to solve inverse skinning problems. A few methods also introduced specialized architectures and loss terms, e.g., for handling interactions between cloth and human bodies [Ma et al. 2022; Wu et al. 2020] or for supporting a pipeline to learn articulated 3D animals [Wu et al. 2023]. Beyond using Laplacian regularization (see e.g. [Liao et al. 2023]), these papers primarily infer skinning weights from data rather than using neural representations to optimize for geometric skinning weights.

### 2.3 Physics Informed Neural Networks in Graphics

A recently popular approach for discretization-free PDE solutions on geometric domains employs physics-informed neural networks (PINNs), such as the works of Raissi et al. [2019]. These are used, e.g., to solve PDEs for physical simulation [Chen et al. 2023], find watertight surfaces from point clouds [Gropp et al. 2020; Sitzmann et al. 2020], and to solve general hyperbolic equations [Rodriguez-Torrado et al. 2021]. Similar to the way we enforce boundary conditions, these networks can be enhanced with explicit boundary constraints [Chalapathi et al. 2024; Chen et al. 2024; Lu et al. 2021; Sukumar and Srivastava 2022]. Other approaches for enforcing constraints on PDEs include [Djeumou et al. 2022; Liu et al. 2022; Mohan et al. 2023; Zhong et al. 2024].

## 3 Robust Biharmonic Skinning

Suppose we are given a shape $\mathcal{P} \subset \mathbb{R}^d$ for $d$ either 2 or 3, with boundary $\partial\mathcal{P}$ and a skeleton consisting of $K$ *control handles* $\{\boldsymbol{h}_i \subset \mathcal{P} : 1 \leq i \leq K\}$. Most commonly, control handles are either points ($\boldsymbol{h}_i$ is a singleton set) or bones ($\boldsymbol{h}_i$ is a line-segment). Each handle has an associated *skinning weight function*, $\alpha_i : \mathcal{P} \to [0, 1]$, with $\boldsymbol{\alpha}(\boldsymbol{x}) = [\alpha_1(\boldsymbol{x}), \ldots, \alpha_K(\boldsymbol{x})]^\top$ denoting the vector-valued skinning weights function.

Skinning weights functions satisfy a set of properties at each point $\boldsymbol{x} \in \mathcal{P}$:

- Non-negativity. A handle cannot have negative influence over a point: $\alpha_i(\boldsymbol{x}) \geq 0$.
- Partition of unity. The influence of all handles over any point must sum up to 100%: $\sum_i \alpha_i(\boldsymbol{x}) = 1$.
- Lagrange property. A handle must have 100% influence over itself: $\forall \boldsymbol{x} \in \boldsymbol{h}_j : \alpha_i(\boldsymbol{x}) = \delta_{ij}$, where $\delta_{ij}$ is the Kronecker delta.
- Boundary conditions. Optionally, a model for skinning weights can include boundary conditions that specify the behavior of the weights at $\partial\mathcal{P}$, e.g., prescribing the value of the normal derivative at the boundary $\boldsymbol{n}_{\boldsymbol{p}}^\top \nabla \alpha_i(\boldsymbol{p}) = 0$ via Neumann conditions. We introduce the ability to optionally prescribe the value of the weights on a subset of the boundary, $\alpha_i|_{\Gamma \subseteq \partial\mathcal{P}}$ via Dirichlet conditions.

Many functions $\boldsymbol{\alpha}$ satisfy these constraints, so algorithms for skinning weights computation typically seek weights that extremize a functional like smoothness.

Although state-of-the-art algorithms that optimize (1) are fairly efficient, they necessitate tetrahedral meshing, which forces a compromise between boundary approximation quality (Figure 3), runtime (Figure 4, Table 1), and/or manual intervention to repair bad boundary elements. Once the tetrahedral mesh is generated, success is still not guaranteed thanks to bad mesh elements and/or poorly-scaling optimization algorithms (Figure 4, red crosses).

To address this challenge, the key idea of this work is to define the space of *geometric fields*—smooth parametric functions which incorporate into their definition the geometry of a given shape, as well as necessary hard constraints and boundary conditions. Higher-order differentiability allows us to optimize for smoothness using stochastic gradient descent and to easily incorporate artistic edits into the optimization (Figures 6, 17).

To make geometric fields conform to the interior of a shape, we design a geometry-aware point-based formulation that exploits modern ray-tracing hardware (§4.1), making them robust to boundaries with bad or non-manifold mesh elements and self-intersections. As a consequence, our method inherits the practically sub-linear scaling of ray-tracing with respect to boundary resolution.

We showcase geometric fields by robustly solving the seminal *bounded biharmonic weights* skinning weights problem [Jacobson et al. 2011]. Combining the constraints and objectives above into a single optimization problem, Jacobson et al. [2011] solve:

$$\min_{\alpha_{i\in[1,K]}} \sum_{i=1}^{K} \int_{\mathcal{P}} |\Delta\alpha_i(\boldsymbol{x})|^2 \, \mathrm{d}V(\boldsymbol{x}), \tag{1}$$

$$\text{s.t.} \quad \alpha_i(\boldsymbol{x}) \geq 0\,, \ \forall i, \boldsymbol{x} \in \mathcal{P}, \qquad \text{(non-negativity)} \tag{1.1}$$

$$\textstyle\sum_i \alpha_i(\boldsymbol{x}) = 1\,, \ \forall \boldsymbol{x} \in \mathcal{P}, \qquad \text{(partition of unity)} \tag{1.2}$$

$$\alpha_i(\boldsymbol{x}) = \delta_{ij}\,, \ \forall i, \boldsymbol{x} \in \boldsymbol{h}_j, \qquad \text{(Lagrange property)} \tag{1.3}$$

where $\delta_{ij}$ is the Kronecker delta, $V$ is the volume form of $\mathcal{P}$.

*Boundary Conditions.* The constraints in Equations 1.1, 1.2, 1.3 can further be combined with boundary conditions for the BBW variational problem. In addition to natural boundary conditions, our method can be modified to support zero Neumann conditions or Dirichlet conditions defined on a subset of the boundary as a way of incorporating user-defined brush strokes into the skinning weights computation. Formally, a user can

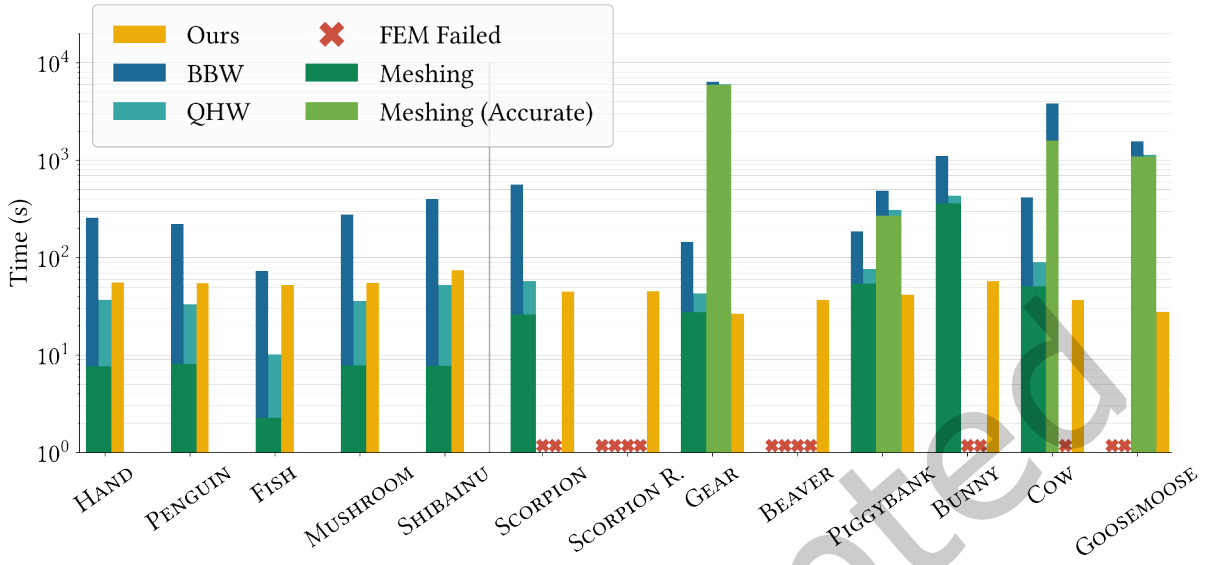


Fig. 4. On well behaved meshes, our method is faster than BBW and somewhat slower than QHW (left). On meshes requiring robust tetrahedralization, our method succeeds in all test cases, whereas existing alternatives either fail to produce a solution or are up to orders of magnitude slower (right).

choose to enable either of the following boundary conditions,

$$\boldsymbol{n}_{\boldsymbol{p}}^{\top} \nabla \alpha_i(\boldsymbol{p}) = 0\,, \;\; \forall i, \boldsymbol{p} \in \Gamma_1 \subseteq \partial \mathcal{P}, \qquad \text{(zero Neumann)} \quad (1.4a)$$

$$\boldsymbol{\alpha}(\boldsymbol{p}) = \boldsymbol{g}(\boldsymbol{p})\,, \;\; \forall \boldsymbol{p} \in \Gamma_2 \subseteq \partial \mathcal{P}, \qquad \text{(Dirichlet)} \quad (1.4b)$$

where $\Gamma_1$ and $\Gamma_2$ are non-overlapping subsets of the boundary, $\boldsymbol{n}_{\boldsymbol{p}}$ is the surface normal at the boundary point $\boldsymbol{p}$, and $\boldsymbol{g}$ represents user-drawn Dirichlet conditions. Note that, while the original BBW article [Jacobson et al. 2011] does not explicitly employ zero Neumann constraints, Stein et al. [2018] show that their discretization induces these conditions (see their §2.2.1). The choice of boundary conditions is subjective, and our system is flexible enough to turn boundary conditions on or off as desired—even during optimization—allowing users to manually prescribe weights at certain locations.

## 4 Geometric Fields

We begin by presenting the design of a flexible differentiable field architecture that respects the geometry of the shape and satisfies the hard constraints and boundary conditions necessary for skinning. At a high level, our model composes a smooth parametric *geometry-aware* function $\boldsymbol{f}_\theta : \mathcal{P} \to \mathbb{R}^K$, which maps every point in the domain to one value per control handle (§4.1), with different *activation functions* that ensure the output satisfies constraints and boundary conditions (§4.2,4.3).

### 4.1 Function Representation

In this section, we formulate a geometry-aware parametric function representation, $\boldsymbol{f}_\theta$, which we will use to represent skinning weights. It is tempting to use a neural field as a general-purpose function representation, yet

these are typically unaware of a shape's geometry. Instead, we propose a more interpretable and geometry-aware representation inspired by classical computer graphics models as well as modern splat-based rendering methods.

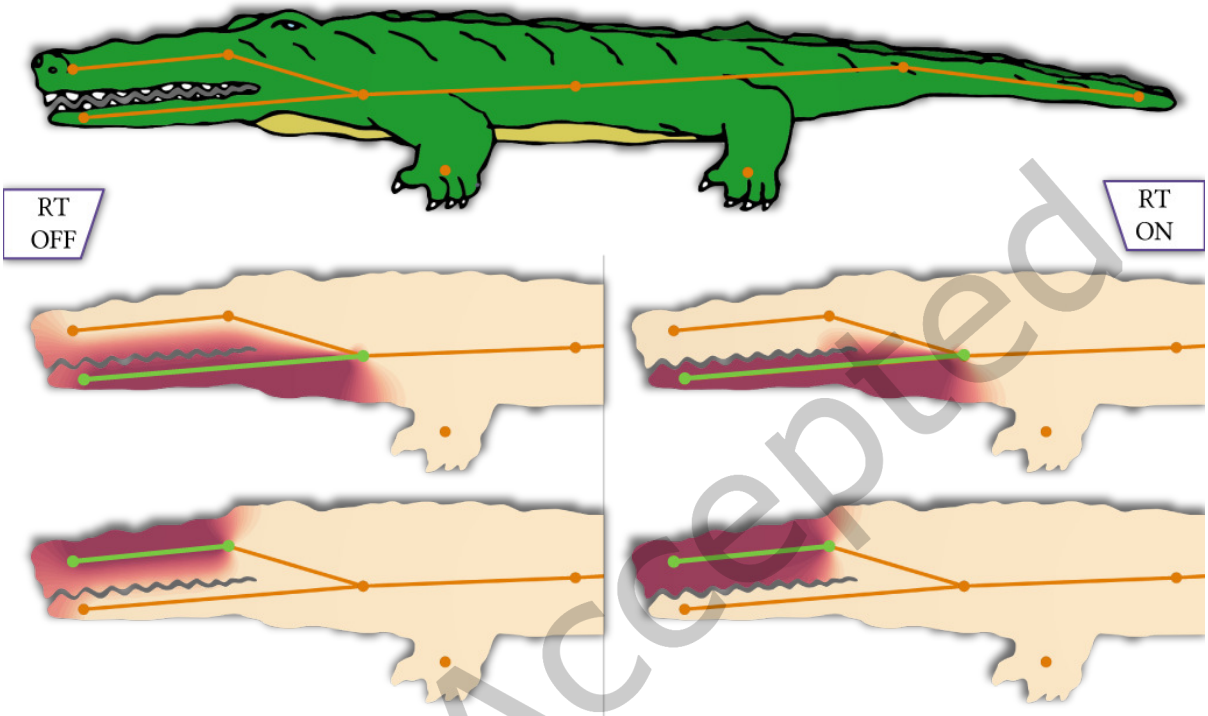


Fig. 5. We demonstrate the effects of our visibility-aware kernel on the Crocodile mesh. On the left, skinning weights optimized using $k_{\mathrm{E}}$ exhibit undesirable *bleeding artifacts*. On the right, we can see that optimizing for skinning weights using $k_{\mathrm{RT}}$ mitigates the issue.



*Kernel-Based Parameterization.* We parameterize $\boldsymbol{f}_\theta$ using a collection of pairs $\mathcal{X} \coloneqq \{(\boldsymbol{x}_i, \boldsymbol{f}_i) : 1 \le i \le N\}$, where $\boldsymbol{x}_i \in \mathcal{P}$ is a randomly sampled point in the domain and $\boldsymbol{f}_i \in \mathbb{R}^K$ is a parameter vector associated to that point. The points $\boldsymbol{x}_i$ are fixed during optimization; the optimization variables are the parameter vectors $\boldsymbol{f}_i$ , which we can think of as rows of $\theta \in \mathbb{R}^{N \times K}$. Section 5.1 and Appendix A detail how the $\boldsymbol{x}_i$'s are sampled.

Evaluated at a point $\boldsymbol{x} \in \mathcal{P}$, we take

$$\boldsymbol{f}_\theta(\boldsymbol{x}) \coloneqq \frac{\sum_{i=1}^N k(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{f}_i}{\sum_{i=1}^N k(\boldsymbol{x}, \boldsymbol{x}_i)}, \tag{2}$$

where $k(\boldsymbol{x}, \boldsymbol{x}_i)$ is an affinity kernel between $\boldsymbol{x}$ and $\boldsymbol{x}_i$.

This point-based form for $\boldsymbol{f}_\theta$ is built on classical methods for kernel regression [Nadaraya 1964; Watson 1964] and is a common tool in computer graphics [Alexa et al. 2004; Coifman and Lafon 2006; Gingold and Monaghan 1977; Lucy 1977; Pauly et al. 2003]. Unlike these formulations, which are oblivious to the boundary of $\mathcal{P}$, relying on modern machine learning techniques and advances in GPU hardware lets us design a kernel $k$ that transforms this into a viable representation for variational problems in geometry.

Introducing a kernel-based formulation rather than a neural network is justified by several factors. Bounded biharmonic weights are smooth, meaning they can be well-approximated by samples on a point cloud. Furthermore, this form for $f_\theta$ localizes the degrees of freedom on the domain, making it possible to visualize and reason about design parameters as shown in §6 and Appendix A. Most importantly, this formulation admits for a simple modification to the kernel $k$ that incorporates structural information and prevents *bleeding* over the boundaries of the shape, shown in Figure 5. Avoiding bleeding artifacts would be non-trivial using a coordinate neural network, since a coordinate network in its usual configuration is extrinsic by nature—information that is far away intrinsically can be close together extrinsically (for example, the raised arm of the robot and its head in Figure-14), and an extrinsic coordinate network will cause bleeding across narrow gaps.

*Visibility-based Kernel.* A standard choice for an affinity kernel is the exponential kernel [Coifman and Lafon 2006; Nadaraya 1964; Watson 1964]:

$$k_{\text{E}}(\boldsymbol{x}, \boldsymbol{x}_i) = \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{x}_i\|^2\right\}, \tag{3}$$

where $\sigma$ governs the *spread* of the kernel. However, as this kernel is based on the extrinsic distance between $\boldsymbol{x}$ and $\boldsymbol{x}_i$, it can lead to bleeding artifacts demonstrated in Figure 5.

Previous work [Jacobson et al. 2011, 2012b; Wang and Solomon 2021] avoids bleeding by meshing the interior of $\mathcal{P}$ and associating degrees of freedom and objective terms with elements of the mesh. Reliance on a conforming mesh with sufficient quality to discretize and solve skinning weights problems hamstrings the efficiency and reliability of automatic skinning weights computation, since design of fast and robust meshing algorithms remains challenging [Diazzi et al. 2023; Hu et al. 2020, 2018]. Moreover, this limitation prevents computation of skinning weights on triangle soups and other disconnected domains, as we consider in Section 5.2.

To address bleeding using GPU-friendly queries without a volume mesh, we modify the kernel $k$ to only allow for pairs of points if they are *visible* to each other:

$$k_{\text{RT}}(\boldsymbol{x}, \boldsymbol{x}_i) = \mathcal{V}(\boldsymbol{x} \leftrightarrow \boldsymbol{x}_i) \exp\left\{-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{x}_i\|^2\right\}. \tag{4}$$

Here, $\mathcal{V}(\boldsymbol{x} \leftrightarrow \boldsymbol{x}_i)$ is a visibility indicator function between $\boldsymbol{x}$ and $\boldsymbol{x}_i$, defined as 0 if the line segment between $\boldsymbol{x}$ and $\boldsymbol{x}_i$ intersects the boundary and 1 otherwise. This construction is illustrated visually in Figure 9.

In practice, checking visibility distills down to a ray-mesh intersection query. Efficient, robust algorithms support this query [Pharr et al. 2016, Chapter 6.8] on modern GPU hardware via the Optix library [Parker et al. 2010]. As a point of reference, a similar approach has been proposed to interpolate irradiance caches in real-time rendering [Halen and Hayward 2021; Majercik et al. 2019]. To enable fast GPU radius queries within the hardware-accelerated ray tracing context, we truncate the kernel after a distance $r = 3\sigma$ and employ a hash-grid data structure. In Section 5, we use a similar strategy to estimate smoothness energies while respecting the interior structure of the domain.

Remark (Barycentric coordinates). *Dodik et al.'s work on barycentric coordinates [2023] provides a point of contrast to our own. The barycentric coordinates problem is closely linked to our Equation 1, with one additional "Reproduction" constraint. This additional constraint, however, motivates their function parameterization, which is built by sampling simplices in the domain and combining their linear barycentric coordinate functions. Thanks to the missing reproduction constraint, our parameterization is much simpler and does not scale cubically in the size of the domain.*

## 4.2 Non-negativity and Partition of Unity

We can incorporate the non-negativity and partition of unity constraints into the final activation layer of $f_\theta$ using standard machine learning tools. We can satisfy Equations 1.1 and 1.2 by first applying an elementwise

`softplus` function [Dugas et al. 2000] to the outputs of $f_\theta$—which maps them to the non-negative reals—and then normalizing such that they sum to one. Appendix B contains the full definitions of less common functions adopted in our work..

*Discussion.* A more common choice in machine learning would be the exponential function instead of a `softplus`. We found `softplus` to be preferable for several reasons. A normalized exponential, also known as a soft maximum, amplifies the relative magnitude of the largest output. However, we desire the opposite behavior as multiple control handles can exert influence over the same region. More practically, an exponential activation has a larger span of possible gradient values, which yields instabilities during optimization.

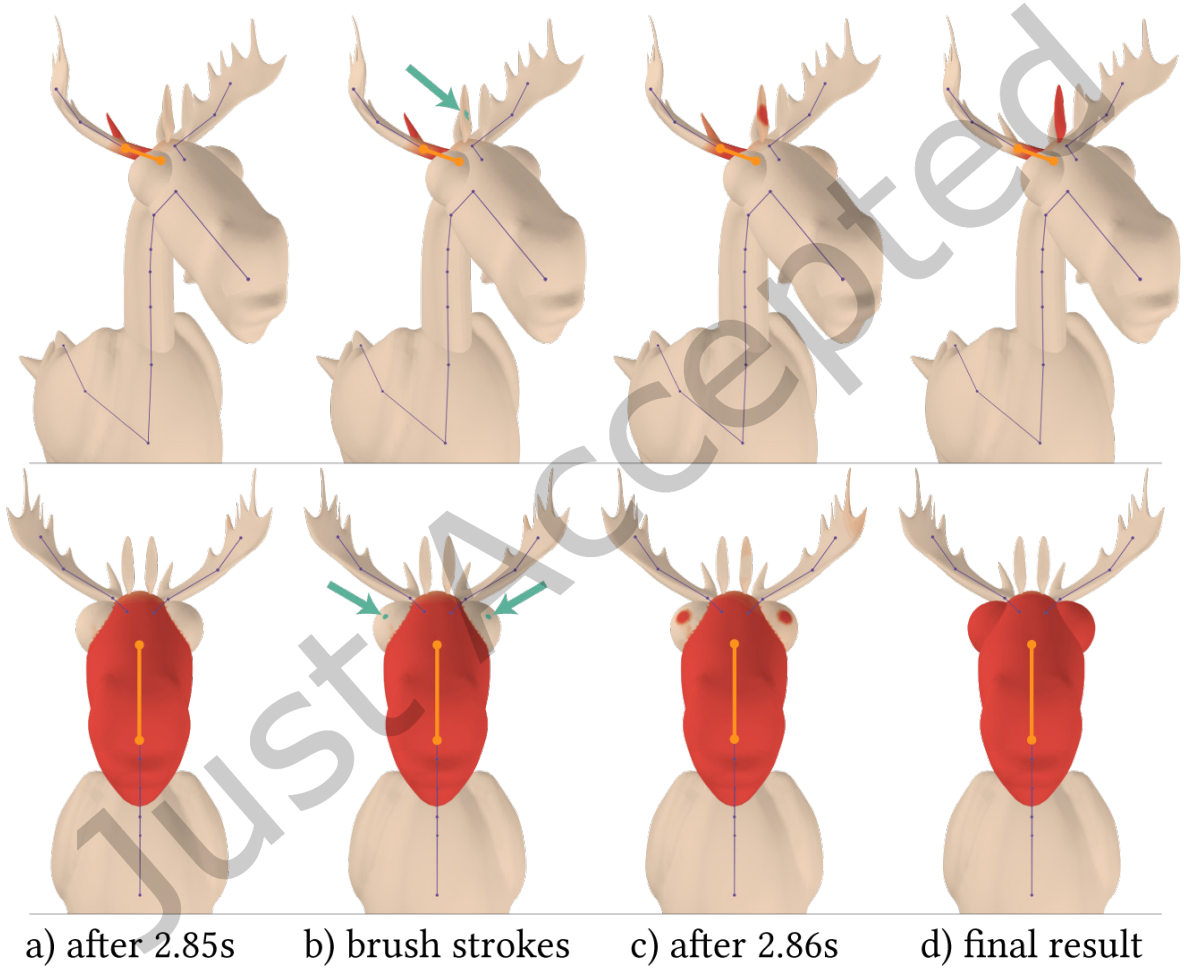




Fig. 6. We combine the popular painting interface with our optimization method via Dirichlet boundary conditions. Our method quickly generates an initial solution (a). The ears and eyes of the Goosemoose mesh are disconnected as there is no path through the shape's interior that reaches them from any of the control handles. After inspection, the user can paint on skinning weights on a small subset of the boundary (b). The optimization then resumes, incorporating user-specified weights as Dirichlet boundary conditions (c, d).

### 4.3 Enforcing Boundary Conditions

It remains to enforce the Lagrange property (1.3), itself a kind of Dirichlet boundary condition (1.4b), and the Neumann boundary condition (1.4a). In physics-informed neural networks (PINNs), boundary conditions are commonly enforced *weakly*, with a loss term [Lu et al. 2021; Raissi et al. 2019]. In this framework, it remains unclear how the interior and the boundary loss terms should be combined, leading to case-by-case parameter tuning and a brittle optimization procedure. As a solution, recent work enforces such hard constraints by reparameterizing the function [Lu et al. 2021; Sukumar and Srivastava 2022]. We opt for this strategy due to its robustness. Our approach:

- does not require a specific shape representation such as a signed distance field or restrict the topology or geometry of the boundary—we require only closest-point queries;
- only reparameterizes the function in an $\varepsilon$-neighborhood of the boundary, meaning that the majority of the computation remains unmodified; and
- only requires access to function values and not gradients, even when enforcing zero Neumann boundary conditions, reducing the necessary computation and avoiding the need for fourth-order derivatives.

*Lagrange property and Dirichlet boundary conditions.* The Lagrange property (Equation 1.3) and Dirichlet boundary conditions (Equation 1.4b) give hard constraints on the values of the skinning weights functions at certain locations. The Lagrange property is automatically enforced at the bones, while the Dirichlet conditions are optionally painted on. We describe how we implement the Lagrange property below; Dirichlet conditions are implemented in a nearly identical fashion.

We begin by restating Equation 1.3 in terms of a control handle indicator function, $\boldsymbol{e} : \mathcal{P} \to \{0,1\}^K$. We define $\boldsymbol{e}$ such that, for all $\boldsymbol{x}$ that lay on handle $\boldsymbol{h}_i$, the $i^{\text{th}}$ element of $\boldsymbol{e}(\boldsymbol{x})$ equals 1 and all other elements equal 0:

$$e_i(\boldsymbol{x}) \coloneqq \begin{cases} 1 & \text{if } \boldsymbol{x} \in \boldsymbol{h}_i, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

It follows that the Lagrange condition can be satisfied via

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \boldsymbol{e}(\boldsymbol{x}) + \left(1 - \sum_{i=1}^{K} e_i(\boldsymbol{x})\right) \boldsymbol{f}_\theta(\boldsymbol{x}). \tag{6}$$

In general, this transformation makes $\boldsymbol{f}_\theta$ discontinuous at the handles. Therefore, to enforce this boundary condition, we construct a *mollified* version of $\boldsymbol{e}$—denoted as $\widetilde{\boldsymbol{e}}$—that equals $\boldsymbol{e}$ on the control handles and quickly falls off to zero as we move away. As such, we are essentially interpolating between $\boldsymbol{f}_\theta(\boldsymbol{x})$ and $\boldsymbol{e}(\boldsymbol{x})$ as $\boldsymbol{x}$ approaches a control handle inside cylindrical $\varepsilon$-neighborhoods of bones and spherical $\varepsilon$-neighborhoods of point handles.

$$\boldsymbol{\alpha}(\boldsymbol{x}) \coloneqq \widetilde{\boldsymbol{e}}(\boldsymbol{x}) + \left(1 - \sum_{i=1}^{K} \widetilde{e_i}(\boldsymbol{x})\right) \boldsymbol{f}_\theta(\boldsymbol{x}). \tag{7}$$

We smooth out (6) by replacing the binary 0 or 1 decision inside of $e_i$ with a so-called smooth *bump function*, $w\colon [0,\varepsilon] \to [0,1]$, ensuring $w(0) = 1$, $w(\varepsilon) = 0$, and $w'(0) = 0$. The exact construction of $\widetilde{e_i}$ and the choice of $w$ are described in Appendix B.

*Zero Neumann boundary conditions.* To match the implementation of Jacobson et al. [2011], we optionally modify $\boldsymbol{\alpha}$ near the boundary to enforce the zero Neumann boundary condition (Equation 1.4a). Its effects can be seen in Figure 7.

To evaluate $\boldsymbol{\alpha}$ with zero Neumann conditions at $\boldsymbol{x} \in \mathcal{P}$, we first find the closest boundary point $\boldsymbol{p} \in \partial\mathcal{P}$. We will refer to the distance between $\boldsymbol{x}$ and $\boldsymbol{p}$ as $t$ and to the surface normal at $\boldsymbol{p}$ as $\boldsymbol{n}_{\boldsymbol{p}}$. Our goal is to edit $\boldsymbol{\alpha}$ to enforce that, as $t \to 0$, we have $(\boldsymbol{\alpha}(\boldsymbol{x}) - \boldsymbol{\alpha}(\boldsymbol{p}))/t \to \boldsymbol{0}$.

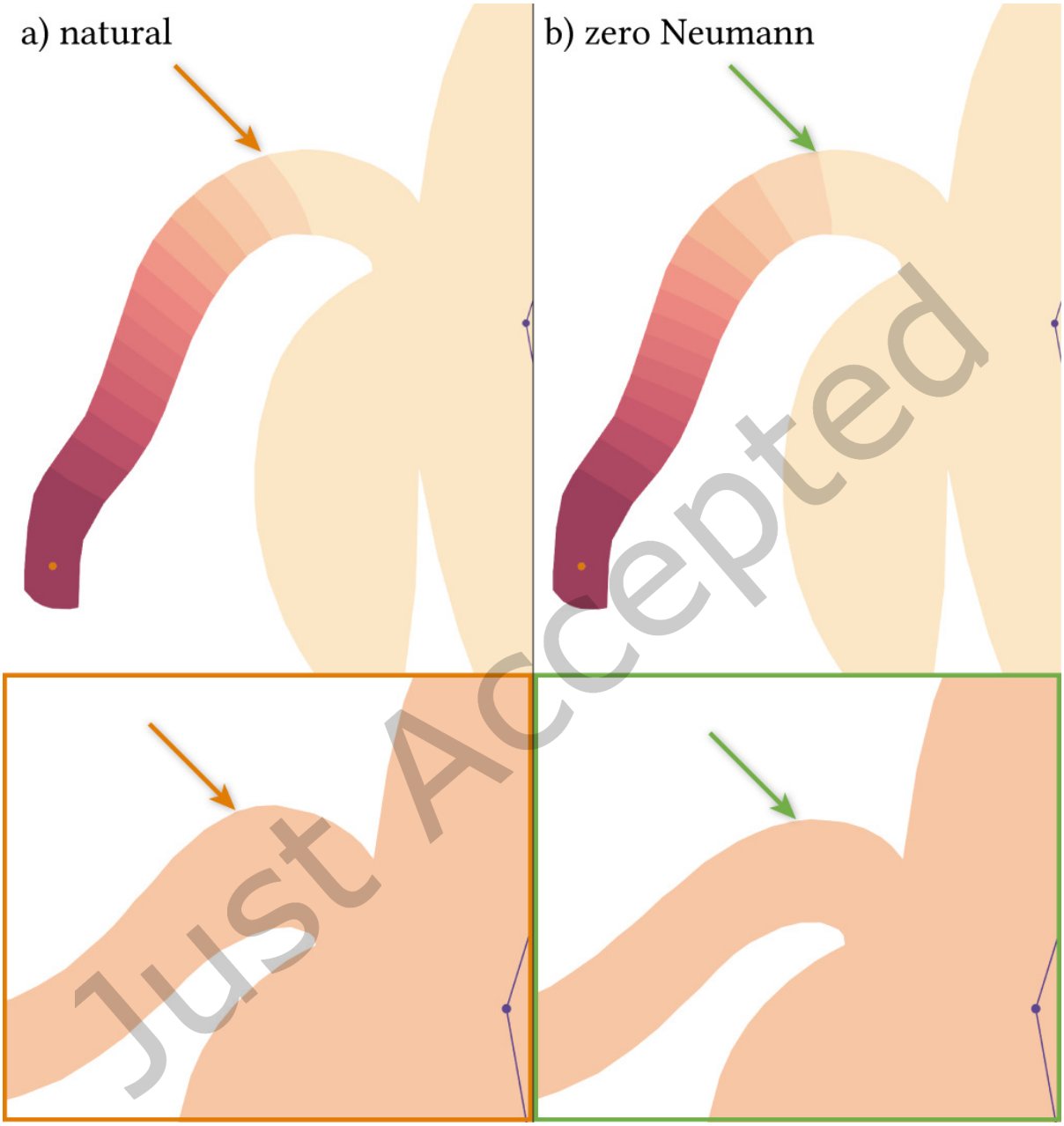




Fig. 7. The user can optionally enable zero Neumann boundary conditions, making the level sets of the weights orthogonal to the boundary. In this example, zero Neumann weights (top) make it so that the control handle symmetrically deforms its region of influence (bottom).

As there are many ways to design this procedure, we opt to preserve as much of the original function as possible, i.e., $\boldsymbol{\alpha}$ should remain unmodified for all $\boldsymbol{x}$ outside an $\varepsilon$ neighborhood of $\partial\mathcal{P}$. We begin by defining a parametric line, $\boldsymbol{\gamma} : [-\varepsilon, \varepsilon] \to \mathcal{P}$, that connects $\boldsymbol{x}$ and $\boldsymbol{p}$, satisfying $\boldsymbol{\gamma}(0) = \boldsymbol{p}$, $\boldsymbol{\gamma}(t) = \boldsymbol{x}$, and $\boldsymbol{\gamma}' \equiv \boldsymbol{n_p}$. This allows us to rewrite the boundary condition as

$$\frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{\alpha}(\boldsymbol{\gamma}(t))\Big|_{t=0} = \boldsymbol{n_p} \cdot \nabla\boldsymbol{\alpha}(\boldsymbol{p}) = 0. \tag{8}$$

Denoting the unmodified function as $\hat{\boldsymbol{\alpha}}$, we can write

$$\boldsymbol{\alpha}(\boldsymbol{x}) = \hat{\boldsymbol{\alpha}}(\boldsymbol{x}) + w(t)(\hat{\boldsymbol{\alpha}}(\boldsymbol{p}) - \hat{\boldsymbol{\alpha}}(\boldsymbol{x})), \tag{9}$$

where we use $w\colon [0, \varepsilon] \to [0, 1]$ with $w(0) = 1$, $w(\varepsilon) = 0$, $w'(0) = 0$, as defined previously and in Appendix B. This formulation is illustrated in Fig. 8. By construction, $\boldsymbol{\alpha}$ satisfies the following property:

Proposition 4.1. *$\boldsymbol{\alpha}$ satisfies zero Neumann conditions for any smooth $\hat{\boldsymbol{\alpha}}$ on the interiors of the boundary facets of $\mathcal{P}$.*

Proof. We need to check for $\boldsymbol{n_p} \cdot \nabla\boldsymbol{\alpha}(\boldsymbol{p}) = 0$. By the chain rule,

$$\begin{aligned}\boldsymbol{n_p} \cdot \nabla\boldsymbol{\alpha}(\boldsymbol{p}) &= \boldsymbol{n_p} \cdot \Big(\nabla\hat{\boldsymbol{\alpha}}(\boldsymbol{p}) + w'(0)\underbrace{(\hat{\boldsymbol{\alpha}}(\boldsymbol{p}) - \hat{\boldsymbol{\alpha}}(\boldsymbol{p}))}_{=\mathbf{0}} - \underbrace{w(0)}_{=1}\nabla\hat{\boldsymbol{\alpha}}(\boldsymbol{p})\Big) \\ &= \boldsymbol{n_p} \cdot \Big(\nabla\hat{\boldsymbol{\alpha}}(\boldsymbol{p}) - \nabla\hat{\boldsymbol{\alpha}}(\boldsymbol{p})\Big) = 0.\end{aligned}$$

In the first equality, note that the term next to $w(0)$ contains only a single derivative of $\hat{\boldsymbol{\alpha}}$; the other term vanishes since we are projecting onto the flat boundary facets of $\mathcal{P}$. □

The proposition above has to be stated carefully because polygonal choices of $\mathcal{P}$ necessarily have sharp corners, where Neumann conditions are ill defined. Empirically, we find that behavior of our model near these corners is still reasonable and converges to the proper global boundary conditions as the boundary is refined.

## 5 Optimization

With our representation of skinning weight functions in place, we proceed to describe our method for computing and optimizing the biharmonic energy. Our approach uses a randomized estimator compatible with stochastic gradient-based optimization techniques.

### 5.1 Estimation and Optimization of Biharmonic Energy

In each iteration of the optimization, we uniformly randomly sample $M$ samples $\{\boldsymbol{y}_j\}_{j=1}^{M}$ in the interior of the domain $\mathcal{P}$, used as quadrature points to compute our approximation of the biharmonic energy objective. If a mesh of the interior is available, we use it for random sampling of the interior; this mesh is used only for sampling and does *not* have to satisfy the quality conditions that are typically necessary for FEM. If a mesh is unavailable, we randomly sample the ambient space and use the generalized winding number [Jacobson et al. 2013] to reject samples below a threshold, as described below in Section 5.2. It is important to sample points in the interior of the domain and not just on the boundary surface, since the integral in (1) is formulated on the *volume*.

Given our set of samples, we use a Monte Carlo estimate of the bilaplacian energy in Equation 1:

$$\sum_{i=1}^{K}\int_{\mathcal{P}} |\Delta\alpha_i(\boldsymbol{y})|^2 \,\mathrm{d}V(\boldsymbol{y}) \approx \frac{V(\mathcal{P})}{M}\sum_{i=1}^{K}\sum_{j=1}^{M} |\hat{\Delta}\alpha_i(\boldsymbol{y}_j)|^2, \tag{10}$$

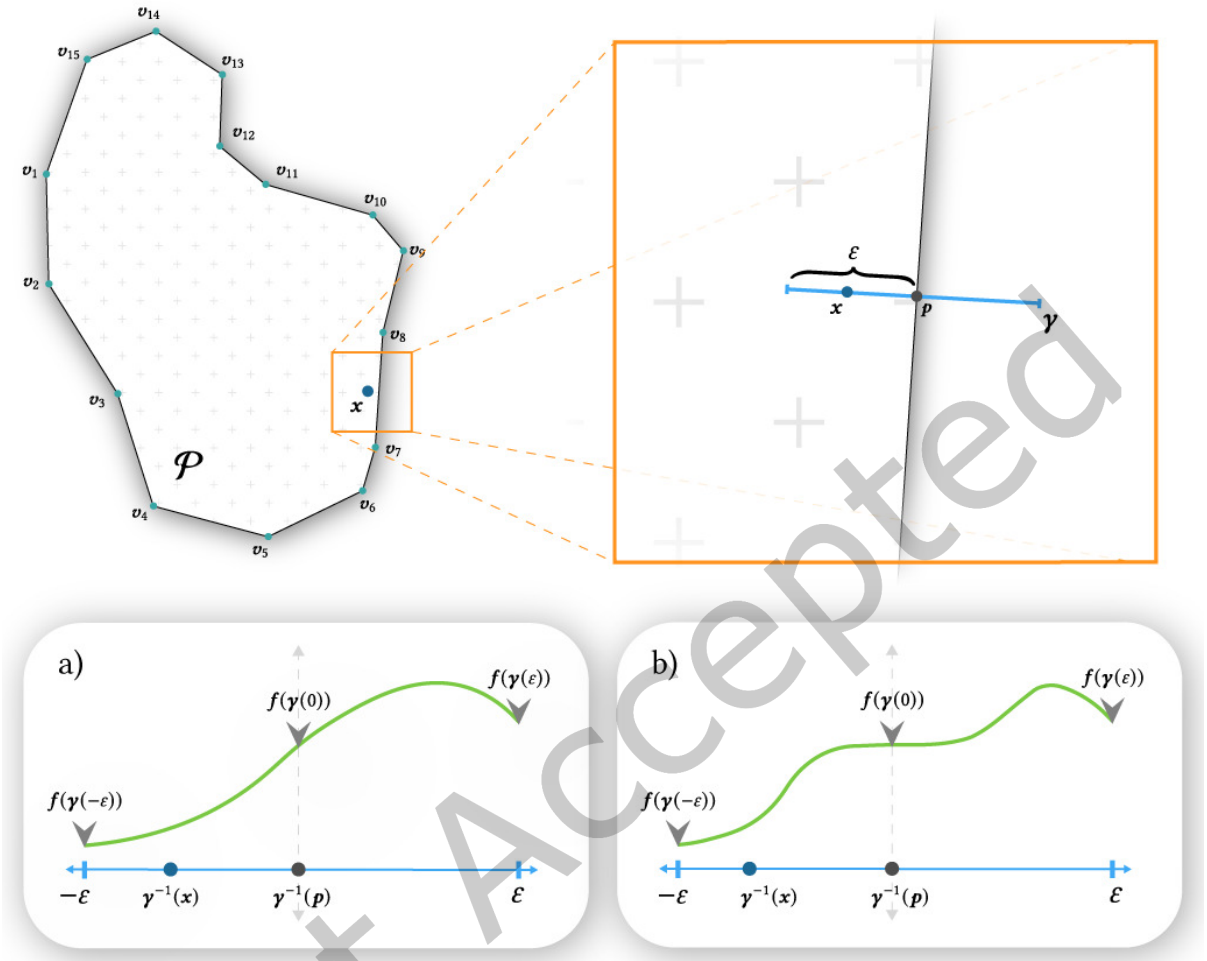


Fig. 8. An illustration of our zero Neumann reparameterization within a $\varepsilon$-neighborhood of the boundary.



where $\hat{\Delta}$ is a visibility-aware finite-difference Laplacian estimator defined in Section 5.3.

We optimize this objective using the Adam optimizer [Kingma and Ba 2014]. A new set of samples $\mathbf{y}_j$ is drawn in each iteration of Adam. Note Wang and Solomon [2021] also find Adam to be an effective optimizer for skinning weights problems. Please see Appendix A for more implementation details.

## 5.2 Generalized-Winding Numbers Bilaplacian Energy

We are able to apply our method to non-watertight geometries through an extension that relies on robust methods popular in other parts of geometry processing.

In particular, we rely on generalized winding numbers [Barill et al. 2018; Jacobson et al. 2013] to perform robust inside-outside segmentation. For a watertight shape, the generalized winding number is a function $\text{wn} : \mathbb{R}^d \to \mathbb{R}$,

which equals 1 for all points inside the shape and −1 for those outside of the shape. For an open shape, the generalized winding number smoothly varies from 1 to −1 as the percentage of the shape's faces surrounding a point decreases.

We define two thresholds, $\mathrm{wn}_l$ and $\mathrm{wn}_h$. Anything below $\mathrm{wn}_l$ is considered completely outside the shape, and anything above $\mathrm{wn}_h$ is completely inside. Our implementation uses $\mathrm{wn}_l = 0.1$ and $\mathrm{wn}_h = 0.25$.

We use $\mathrm{wn}_l$ to reject points both when sampling the point cloud $\{\boldsymbol{x}_i\}$, as well as the optimization samples $\{\boldsymbol{y}_i\}$. For all remaining points, we clip their generalized winding numbers above $\mathrm{wn}_h$ compute a weighted biharmonic energy:

$$\frac{V(\mathcal{P})}{M} \sum_{i=1}^{K} \sum_{j=1}^{M} \frac{\mathrm{wn}(\boldsymbol{y}_j) - \mathrm{wn}_l}{\mathrm{wn}_h - \mathrm{wn}_l} |\hat{\Delta}\alpha_i(\boldsymbol{y}_j)|^2. \tag{11}$$

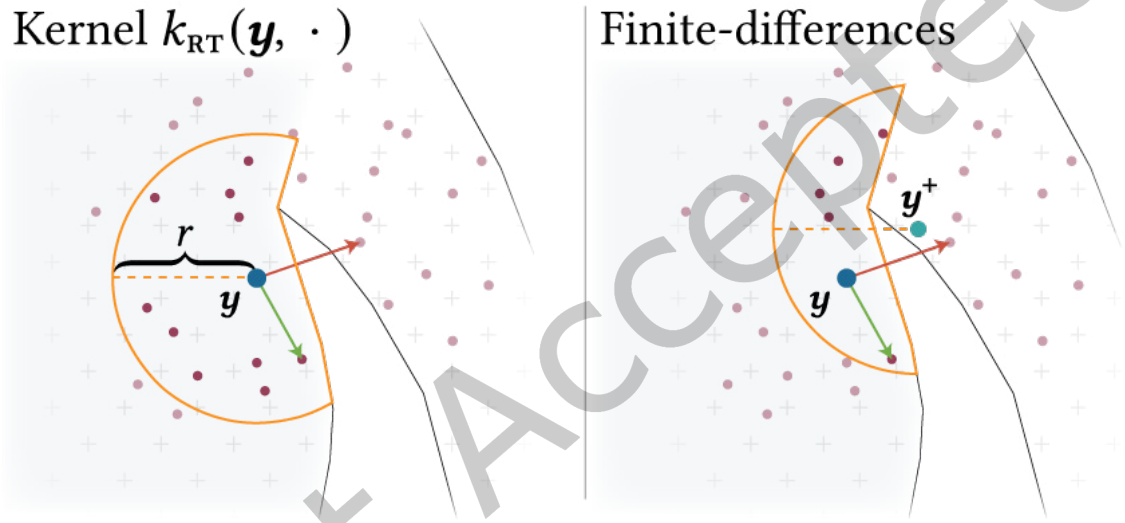


Fig. 9. The figure on the left visualizes our ray-tracing kernel $k_{\mathrm{RT}}$. For an evaluation point $\boldsymbol{y}$, we look up all points within a radius $r$, and include into our estimate points which are visible from $\boldsymbol{y}$. Here we visualize a visible point with a green arrow pointing to it, and an occluded point with a red arrow. The figure on the right illustrates our visibility-aware finite-difference scheme. Our construction ensures that the kernel estimate at $\boldsymbol{y}^+$ can only include points which are in a star-shaped neighborhood of $\boldsymbol{y}$.



## 5.3 Finite Differences

Previous work highlights problems in relying on automatic differentiation to compute higher-order derivatives of function representations like ours [Chetan et al. 2023; Li et al. 2023]; in particular, differential quantities obtained through automatic differentiation tend to be noisy, and using them for optimization results in visible artifacts [Chetan et al. 2023]. On the other hand, a naïve finite-difference estimator can yield bleeding artifacts similar to those in Figure 5. Here, we suggest a stochastic estimator of the Laplacian that reduces computation time and avoids bleeding.

A standard finite-difference estimator of the Laplacian, $\Delta_h$ can be defined as

$$\Delta_h f(\mathbf{y}) = \frac{-2df(\mathbf{y}) + \sum_{i=1}^{d}(f(\mathbf{y} + h\boldsymbol{e}_i) + f(\mathbf{y} - h\boldsymbol{e}_i))}{h^2}, \tag{12}$$

where $h$ is the finite-difference step size, and $\boldsymbol{e}_i$ represents the $i^{\text{th}}$ basis vector in $\mathbb{R}^d$. This well-known finite-difference Laplacian requires a linear number $2d + 1$ of computations in the dimensionality $d$ of the ambient space; this means a single term on the right-hand side of (11) would require 7 evaluations of $\alpha_i$ in 3D ($d = 3$) and 5 evaluations in 2D ($d = 2$).

Instead, since our evaluation of the objective function is stochastic anyway, we use a modified stochastic Laplacian estimator:

$$\hat{\Delta}_h f(\mathbf{y}) = \frac{-2f(\mathbf{y}) + f(\mathbf{y} + h\mathbf{v}) + f(\mathbf{y} - h\mathbf{v})}{h^2}, \tag{13}$$

where $\mathbf{v}$ is uniformly drawn from the unit sphere $S^{d-1}$; we draw a different $\mathbf{v}$ for each term in (11). The expectation of this expression over $\mathbf{v} \in S^{d-1}$ converges to $\Delta f$ as $h \to 0$. Our expression (13) simultaneously reduces the number of function evaluations to a constant independent of dimension, and it removes the axis alignment bias of the standard estimator (12).

Remark (bias). *There are two sources of minor bias in our estimate* (11) *relative to the true biharmonic energy on the left-hand side. First, we use a positive step size $h > 0$ rather than limiting $h \to 0$. Second, by Jensen's inequality, squaring* (13) *leads to a slight overestimate of the true objective on average; this bias is also present in the Dirichlet energy of Dodik et al. [2023]. We do not find either source of bias to be significant and leave derivation of practical fully-debiased estimates of the biharmonic energy to future work.*

We now address bleeding in the evaluation of the smoothness objective. Take $\boldsymbol{y}^{\pm} \coloneqq \boldsymbol{y} \pm h\mathbf{v}$ to be a displaced sample used in evaluation of (13). When evaluating $\boldsymbol{f}_\theta(\boldsymbol{y}^{\pm})$, we simply zero out the kernel $k$ for those points $\boldsymbol{x}_i$ that are not visible from $\boldsymbol{y}$. In other words, we ensure that all $\boldsymbol{x}_i$ used for the evaluation of the Laplacian at $\boldsymbol{y}$ are within a star-shaped neighborhood of $\boldsymbol{y}$ (see Fig. 9).

## 5.4 Multiscale Optimization

Using a gradient descent method for optimizing smoothness energies incurs a trade-off. If we use a finite-difference step that is too small, we will have to wait longer for the boundary conditions to propagate inward. Since our formulation is nonconvex, this issue also can lead to local minima.

In mesh-based pipelines, these difficulties are resolved via preconditioning, either by using Sobolev gradients or through higher-order methods [Wang and Solomon 2021]. In our scenario, however, it is not clear how to precondition the optimization of the biharmonic energy over a smooth geometric field that is nonconvex in its parameters. Therefore, we introduce a simple yet effective alternative to preconditioning that results in high-resolution solutions while reducing the number of iterations for convergence.

Inspired by multigrid methods and prior work on optimizing differential energies in neural fields [Li et al. 2023], we found that a multiscale approach offers an effective solution. Specifically, we initiate our method by randomly sampling $\mathcal{X}$. We let the method optimize for a number of iterations before doubling the number of samples in $\mathcal{X}$. To initialize the new point cloud, we use $\boldsymbol{f}_\theta$ to interpolate existing values $\boldsymbol{f}_i$ onto the new points. Additionally, we also interpolate Adam's gradient moment estimates [Kingma and Ba 2014] in a similar fashion. As the radius of our kernel $k$ and the finite-differences step size are related to the density of the point cloud (see Figure 11), they shrink as we up-sample the point-cloud.

The upsampling generally increases the variance in the evaluation of the biharmonic energy as the interpolation kernel and finite-difference estimator progressively do less and less smoothing (see Figures 12 and 13).

To stabilize the optimization, we progressively decrease the learning rate every time we upsample the underlying representation (see Appendix A for details). We demonstrate the practical effect of this strategy qualitatively in Figure 12 and quantitively in Figure 13.

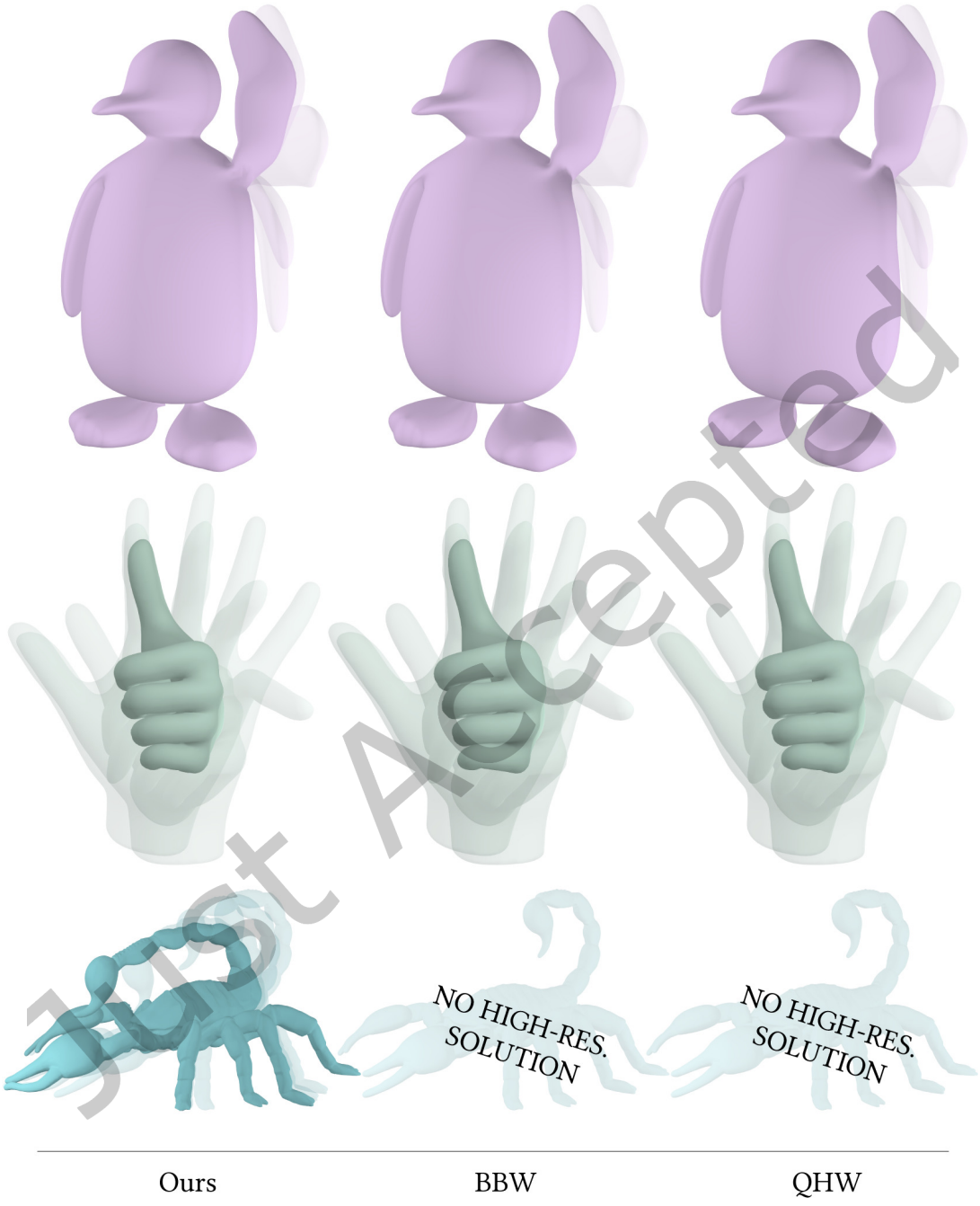




Fig. 10. We compare the deformations produced by our skinning weights with previous volumetric skinning weights methods: BBW [Jacobson et al. 2013] and QHW [Wang and Solomon 2021]. As expected, all methods produce similar results on simple geometries such as the Penguin and Hand meshes. However, robust tetrahedralization software is not able to produce a high-resolution solution for the more complicated Scorpion mesh.

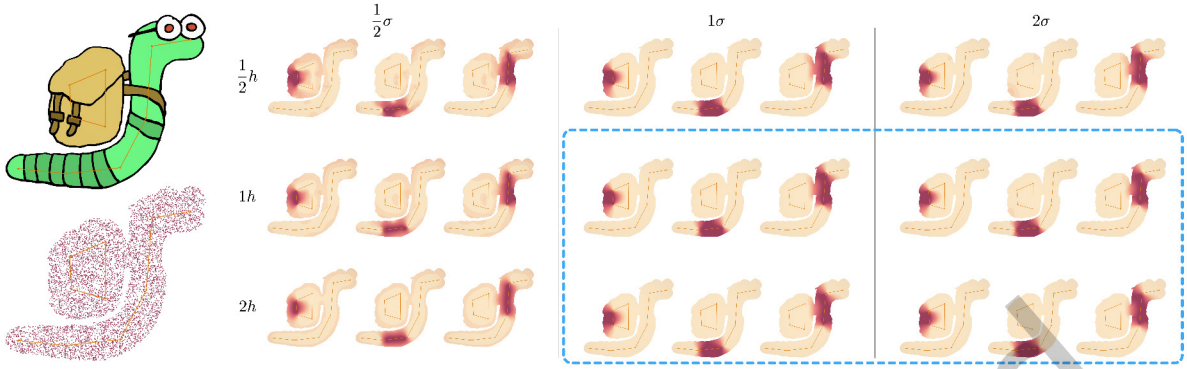


Fig. 11. Results of our ablation study. On the top, we show the effect of varying the kernel parameter $\sigma$, as well as the finite-difference step size, $h$. Making $\sigma$ or $h$ too small compared to the median spacing of points makes optimizing differential quantities difficult and leads to wrong results. Doubling the size of $\sigma$ or $h$ produces a similar result and would be viable choices of parameters, but it does unnecessarily increase the kernel look-up time.

## 6 Results

We ran all of our experiments on a machine with an Intel i9-13900 CPU, 32 GB of memory, and an Nvidia GeForce RTX 4090. See Appendix A for implementation details, including used libraries and hyper-parameter choices. The comparison results for bounded biharmonic weights (BBW) [Jacobson et al. 2011] were generated in Matlab using the `gptoolbox` library [Jacobson et al. 2021], and for quasi-harmonic weights (QHW) [Wang and Solomon 2021] using the authors' implementation. All of our deformations were produced using dual quaternion skinning [Kavan et al. 2008]. We will release the code upon acceptance.

To the best of our knowledge, there are currently no large-scale datasets or benchmarks for robust skinning weight computation. To evaluate our algorithm, we curated a sample of 3D meshes and manually created the skeletons. Most of the meshes are based on diverse and realistic scenarios for animation; 5 are clean and well-behaved, 3 are visually clean yet happen to break previous methods (Scorpion, Gear, Cow), and 4 consist of open and self-intersecting meshes with poor quality elements that a user might nonetheless plausibly wish to animate (Beaver, Piggybank, Bunny, Goosemoose). Lastly, Scorpion Rand., Gear, and the meshes in Figure 18 represent extreme stress-test scenarios that are likewise handled well by our algorithm.

We first validate the properties of our method and ablate parts of our algorithm and our parameter choices in Section 6.1. To provide evidence for robustness, Section 6.2 offers a qualitative and quantitative comparison against previous work that solves for bounded biharmonic weights. We then examine the influence of different boundary conditions in Section 6.3. Finally, we discuss limitations and future work in Section 6.4.

### 6.1 Validation and Ablation

We first validate that our weights look correct and ablate the algorithm's hyperparameters. We attempt to tetrahedralize all 3D meshes before comparing them with BBW [Jacobson et al. 2011] and QHW [Wang and Solomon 2021] in Table 1 and Figures 4, 10, and 20.

Figure 20 compares our skinning weights with weights computed using bounded biharmonic weights (BBW) [Jacobson et al. 2011], as well as those computed using quasi-harmonic weights (QHW) [Wang and Solomon 2021]. As expected, all methods produce qualitatively similar weights functions, but the setting is quite different: ours

can work with a boundary mesh/polygon, while BBW/QHW require filling the domain with elements by tet-/tri-meshing. Figure 10 compares the animations produced by our method to those produced by BBW and QHW. Again, the results are nearly indistinguishable on simple meshes. The major difference is that our method does not require tetrahedralization.




Fig. 12. Final results of our multiscale optimization strategy (top row) on Banana (top left). When optimizing with a coarse point cloud of $2^{11}$ samples, the optimization converges quickly, but the function is overly smooth near the control handles (middle row). Naïvely increasing the resolution to $2^{14}$ points results in slow convergence and possible local minima (bottom row).

*Ablation.* Our method includes some parameters, the most important of which are the number of points in $\mathcal{X}$, the size of $\sigma$, and the size of the finite-difference step $h$. To determine suitable values, we ran a series of ablation studies.

Figure 11 demonstrates the effects of varying $\sigma$ and $h$. There is an inherent tension regarding the size of $\sigma$. On one hand, we want $\sigma$ to be as small as possible to accelerate radius queries. On the other, too small of a $\sigma$ results in what is effectively a 1-nearest-neighbor estimator, which makes estimating differential quantities difficult. As a compromise, we found the following to be a useful heuristic: We set $\sigma$ to the median distance to the $9^{\text{th}}$ nearest neighbor. As shown in our ablation, this choice of $\sigma$ makes it large enough to avoid any issues associated with a degenerate kernel estimator; making it significantly larger produces correct, if slightly blurry, results but offers no benefit while increasing the runtime.

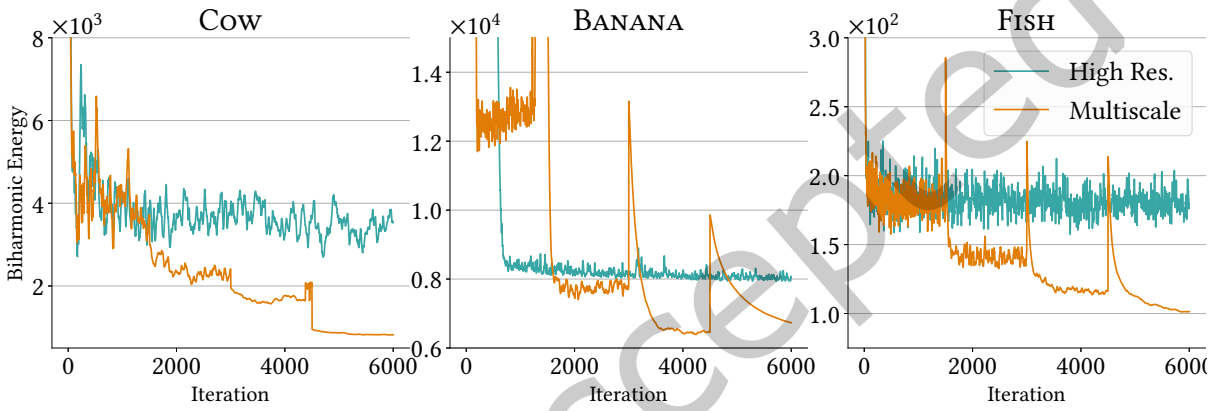


Fig. 13. Our multiscale optimization results in lower biharmonic energy compared to a fixed high-resolution sampling of $\mathcal{X}$.

*Multiscale optimization.* The multiscale optimization strategy is an important component of our method; we demonstrate its benefits by comparing it to fixed-size $\mathcal{X}$.

We qualitatively compare different fixed sizes of of $\mathcal{X}$ to our strategy in Figure 12. Too small of a point cloud results in an overly blurry representation near the handles, whereas too large of a point cloud results in a higher loss, slower convergence, and possible local minima. In comparison, our multiscale optimization combines the benefits of both.

For a quantitative evaluation, Figure 13 shows the energy during optimization for three different meshes, comparing directly optimizing a high-resolution $\mathcal{X}$ to using the multiscale strategy. To make the plots comparable, we make sure that the final stage of the multiscale optimization has the same number of points as the high-resolution setup, and compute the energies using a finite difference estimator with $h$ equal to the one used for the high-resolution scenario. To decrease the visual noise, we use a standard axis-aligned finite-difference estimator for computing the plots only.

As evidenced by this experiment, low-resolution solutions—i.e., the first stage of the the multiscale optimization—are quick to converge, but can result in a higher energy than the high-resolution solutions such as for the Banana mesh. Directly optimizing the high-resolution mesh results in slow convergence and overall higher error in all three scenarios.

These plots are instructive in other ways; for example, it is likely that optimizing for longer would further decrease the energy in all cases. Similarly, it is likely that upsampling sooner in the first two stages would have

| | Meshing type | Meshing (s) | Interior points | BBW Optim. (s) | QHW Optim. (s) | BBW Total (s) | QHW Total (s) | **Ours Natural BC (s)** | **Ours Neumann BC (s)** |
|---|---|---|---|---|---|---|---|---|---|
| Hand | tetgen | 7.64 | 67341 | 250.07 | 29.29 | 257.7 | **36.92** | 59.79 | 82.37 |
| Penguin | tetgen | 8.05 | 67440 | 212.57 | 25.19 | 220.62 | **33.24** | 58.67 | 85.54 |
| Fish | tetgen | 2.26 | 18875 | 71.17 | 7.86 | 73.43 | **10.13** | 56.86 | 80.68 |
| Mushroom | tetgen | 7.8 | 65738 | 270.5 | 28.1 | 278.29 | **35.9** | 59.95 | 82.45 |
| Shibainu | tetgen | 7.79 | 65596 | 391.66 | 44.22 | 399.46 | **52.01** | 76 | 102.45 |
| Scorpion | default | 26.19 | 54963 | 535.47 | 31.03 | 561.66 | 57.22 | **46.31** | 81.91 |
| | accurate | Failed | Failed | Failed | Failed | Failed | Failed | | |
| Scorpion (Rand.) | default | Failed | Failed | Failed | Failed | Failed | Failed | **48.23** | 82.27 |
| | accurate | Failed | Failed | Failed | Failed | Failed | Failed | | |
| Gear | default | 27.53 | 43128 | 117.25 | 15.57 | 144.78 | 43.11 | **30.48** | 50.65 |
| | accurate | 5974.59 | 63072 | 440.33 | 63.52 | 6414.92 | 6038.1 | | |
| Beaver | default | Failed | Failed | Failed | Failed | Failed | Failed | **38.97** | 66.93 |
| | accurate | Failed | Failed | Failed | Failed | Failed | Failed | | |
| Piggybank | default | 54.12 | 65981 | 132.47 | 22.34 | 186.59 | 76.46 | **45.99** | 69.91 |
| | accurate | 269.73 | 66502 | 221.4 | 40.61 | 491.13 | 310.34 | | |
| Bunny | default | 361.74 | 64382 | 737.58 | 73.28 | 1099.32 | 435.01 | **59.58** | 91.16 |
| | accurate | Failed | Failed | Failed | Failed | Failed | Failed | | |
| Cow | default | 50.64 | 61500 | 364.03 | 39.03 | 414.67 | 89.67 | **40.00** | 84.54 |
| | accurate | 1593.12 | 63894 | 2260.49 | Failed | 3853.6 | 1592.12 | | |
| Goosemoose | default | Failed | Failed | Failed | Failed | Failed | Failed | **31.55** | 67.86 |
| | accurate | 1094.18 | 66738 | 468.43 | 48.78 | 1562.62 | 1142.97 | | |

Table 1. Run-time statistics. For previous work, we include the tetrahedral meshing algorithm, either tetgen, tetwild (default), or tetwild (accurate), the number of interior quadrature points, BBW and QHW optimization time, total run-times for BBW and QHW including tetrahedral meshing. Cells highlighted in red signify crashes, whereas cells highlighted in yellow are cases where the FEM pipeline produced a result, but either the result was unusable (Gear mesh) or tetgen was unable to get within 10% of the 64k interior quadrature points. For our work, we include timings with and without Neumann boundary conditions.



been possible with little impact to the quality. We found the current parameters to offer a good trade-off between simplicity of implementation, speed, and quality, and leave further experimentation to future work.

## 6.2 Comparisons with Finite Elements

*Experimental Setup.* We first attempt to tetrahedralize the boundary mesh with tetgen [Si 2015], including samples on the control handles. As QHW requires ellipsoidal handles, we first convert bones and points into ellipsoids with width equal to the width of our Lagrange constraint mollifier $\varepsilon$ from Equation 15. If tetgen fails, we then attempt to tetrahedralize with FastTetWild [Hu et al. 2020]. As FastTetWild cannot incorporate prescribed quadrature points into the mesh, we refine it with tetgen to make it usable for skinning.

Due to the many hyperparameters in tetrahedralization software, direct comparisons to our method are difficult. The first important parameter is how well FastTetWild adheres to the boundary mesh. As demonstrated by Figure 3, the default parameters forego mesh quality for speed, resulting in low-resolution and occasionally

incorrect solutions. FastTetWild can increase boundary adherence, at the expense of longer runtimes. We include timings for both parameter settings, default and accurate, in Table 1 and Figure 4.

Similarly, tetgen offers a parameter to control the quality—and therefore density—of the tetrahedral mesh. The default parameter used by Jacobson et al. [2011] yields meshes of varying density, ranging from only 6244 internal vertices for Fish to 150713 for Cow. Both differ from the total number of points used by our method in 3D, 64000. To facilitate a fair comparison, we ran our experiments twice, once with default tetgen parameters and once attempting to match the number of internal vertices to our method. Since the number of vertices is not exposed as a parameter in tetgen, we ran a binary search on the "quality" parameter and picked the result closest to 64000. We were unable to match the internal mesh density in three cases (highlighted yellow in Table 1). We include the results for the matched vertex count in Table 1 and Figure 4, and include the remaining results and raw data in the supplemental material.

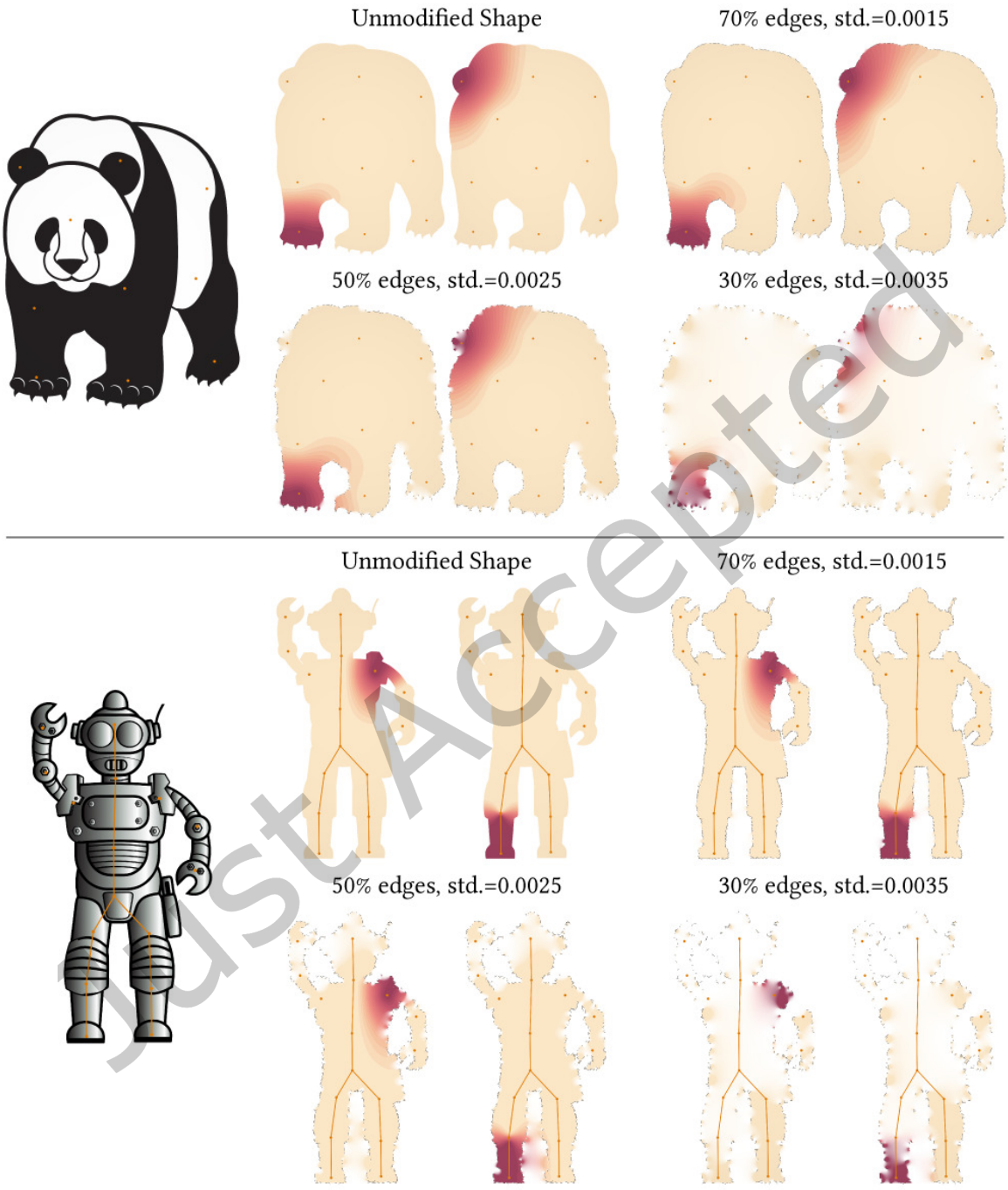




Fig. 14. Our method is effective even on heavily degraded geometric data. This figure shows the results of our method on two shapes with progressively fewer boundary edges whose vertices have been randomly perturbed by adding progressively larger Gaussian noise.

*Findings.* Five meshes, Hand, Penguin, Fish, Mushroom, and Shiba Inu, can be tetrahedralized with tetgen. While our method is faster than BBW on these meshes, QHW is faster in comparison to our method. However, once meshes have to be tetrahedralized with FastTetWild and refined with tetgen, robustness becomes a large issue. Default FastTetWild parameters result in the BBW and QHW pipelines failing to produce results on three meshes (Scorpion (Randomized), Beaver, and Goosemoose) and yield unusable weights on the Gear mesh (Figure 3) Accurate FastTetWild settings lead to tetrahedralization crashes on four meshes: Scorpion, Scorpion (Randomized), Beaver, and Bunny. In addition, QHW (but not BBW) crashes on the Cow mesh on the accurate FastTetWild setting. Piggybank is the *only* FastTetWild mesh that succeeds under all experimental settings. In the remaining cases, our method is either comparable to, or faster than previous work. Figures 1, 3, 10, 14, 15, and 18 demonstrate these key findings qualitatively.

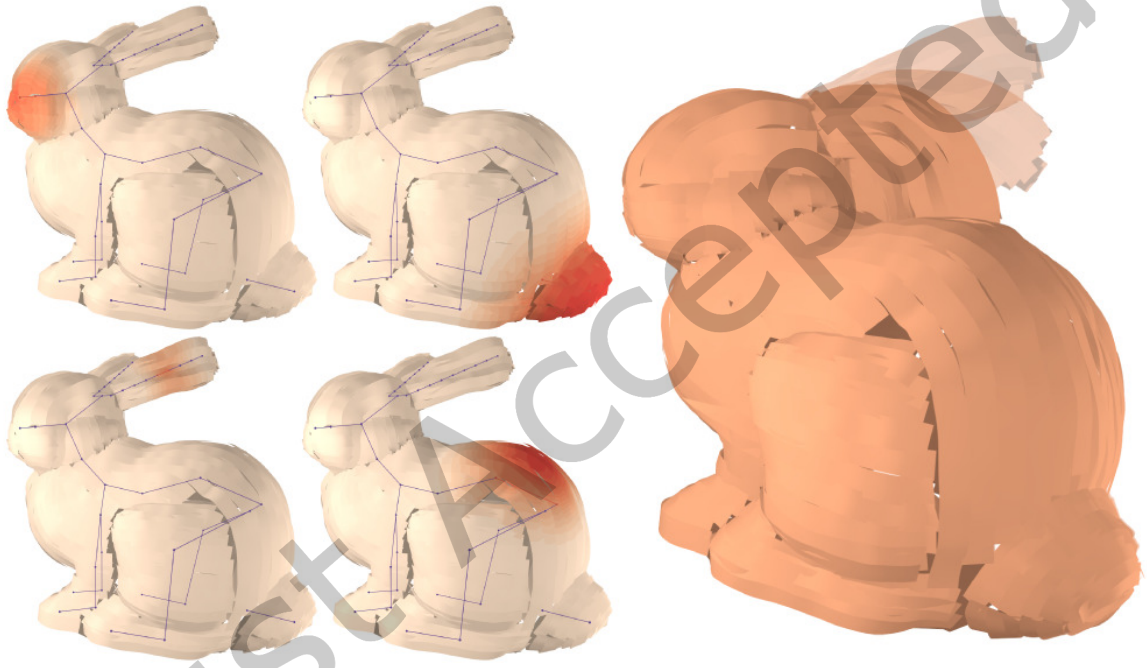

Fig. 15. Our method allows us to animate non-watertight, self-intersecting shapes, such as these virtual reality ribbon drawings created in SurfaceBrush [Rosales et al. 2019]. Despite the quality of the geometry, our method produces smooth-looking weights and animations.

Our main contribution, i.e., effectiveness on imperfect data, is exemplified by the extreme example of triangle soups in Figures 1 and 14, as well as others in Figure 18. Despite severe mesh deterioration, our method still produces meaningful solutions resulting in smooth looking animations. Figure 14 demonstrates the graceful degradation of our method inherited from the generalized winding number; even with 70% of the polygon edges removed and the remaining ones randomly displaced, our method succeeds. In Figure 1, we use our weights to animate Scorpion (Randomized), obtained by deteriorating Scorpion; FEM-based algorithms are unable to produce a solution for such a degraded mesh.

While triangle soups are an extreme example, algorithms relying on tetrahedral meshing fail in more common scenarios. For example, they fail on at least one experimental setting on Scorpion (Figure 10), Goosemoose (Figure 6), and Cow (Figure 17) meshes, despite no obvious problems with the meshes. BBW and QHW results on the Gear mesh either contain artifacts that render the result unusable or require a 1.66 hour meshing time.

Two practically-relevant classes of shapes our method handles out-of-the box are 3D scans from computer vision pipelines (Figure 1) and virtual reality ribbon drawings created by e.g. SurfaceBrush [Rosales et al. 2019] (Figures 1 and 15). Both of these modalities produce non-watertight, self-intersecting meshes, which necessitate slow and brittle tetrahedralization software. The 3D scan of the Beaver mesh (Figure 1) fails to tetrahedralize, regardless of settings, Bunny (Figure 15) either fails to tetrahedralize or results in a 7.25 minute runtime for QHW and a 18.32 minute runtime for BBW (in comparison to our method's 71.74 seconds). While Piggybank (Figure 1) succeeds with all FEM-based methods, our method is nonetheless faster than the best QHW result on that mesh.

Similar to past work and most applications of BBW, we found that solving the full BBW quadratic program was prohibitively slow on realistic examples and consequently dropped the partition-of-unity constraint, which was enforced *a posteriori* by dividing the weights by their sum pointwise. This simplified problem was nonetheless often orders of magnitude slower than our method.

### 6.3 Boundary Conditions

Next, we demonstrate our method's flexibility to boundary conditions. Figures 7 and 8 visualize the difference between natural boundary conditions and the optional Neumann conditions. Figure 7 provides a concrete example where Neumann conditions might be desirable, as they make the weights symmetric through narrow passages. We include all of the 3D results with and without Neumannn boundary conditions as supplemental material. The differences between the boundary conditions are noticeable albeit subtle, but the natural boundary conditions are significantly faster than the Neumann conditions (Table 1). We leave enabling or disabling the zero Neumann conditions as an optional choice for the user.

More interesting are the Dirichlet conditions, which open new connections between bounded biharmonic weights and weight painting, the predominant user interaction for skinning weight design. As demonstrated in Figure 6, painted weights can allow users to connect disconnected components to a given handle. Figure 17 demonstrates how painting can also allow for artistic control. This boundary condition is a crucial missing puzzle piece for optimization-based automatic skinning weight design.

### 6.4 Discussion

Our results show conclusively that geometric fields are useful for the computation of bounded biharmonic weights. While QHW still produces results somewhat quicker than our approach, our method is more robust and faster on meshes requiring robust tetrahedralization. Our formulation also introduces useful new controls to the bounded biharmonic weights in the form of boundary conditions. Dirichlet conditions are especially important, as they allow for manual user intervention to produce a specific result.

*Hyperparameters.* Our method includes some parameters, namely the resolution of the underlying representation, sizes of boundary condition mollifiers, and optimizer parameters. While these parameters are different than the ones used by BBW and QHW (which require parameters to the tet meshing tool and optimization software), we have a similar number of parameters to past work, and our default parameters are effective across all examples.

Controlling the exact resolution of the tetrahedral meshes in our comparisons proved difficult in our experiments and required a binary search over a "quality" parameter. As the different experimental settings demonstrate, we

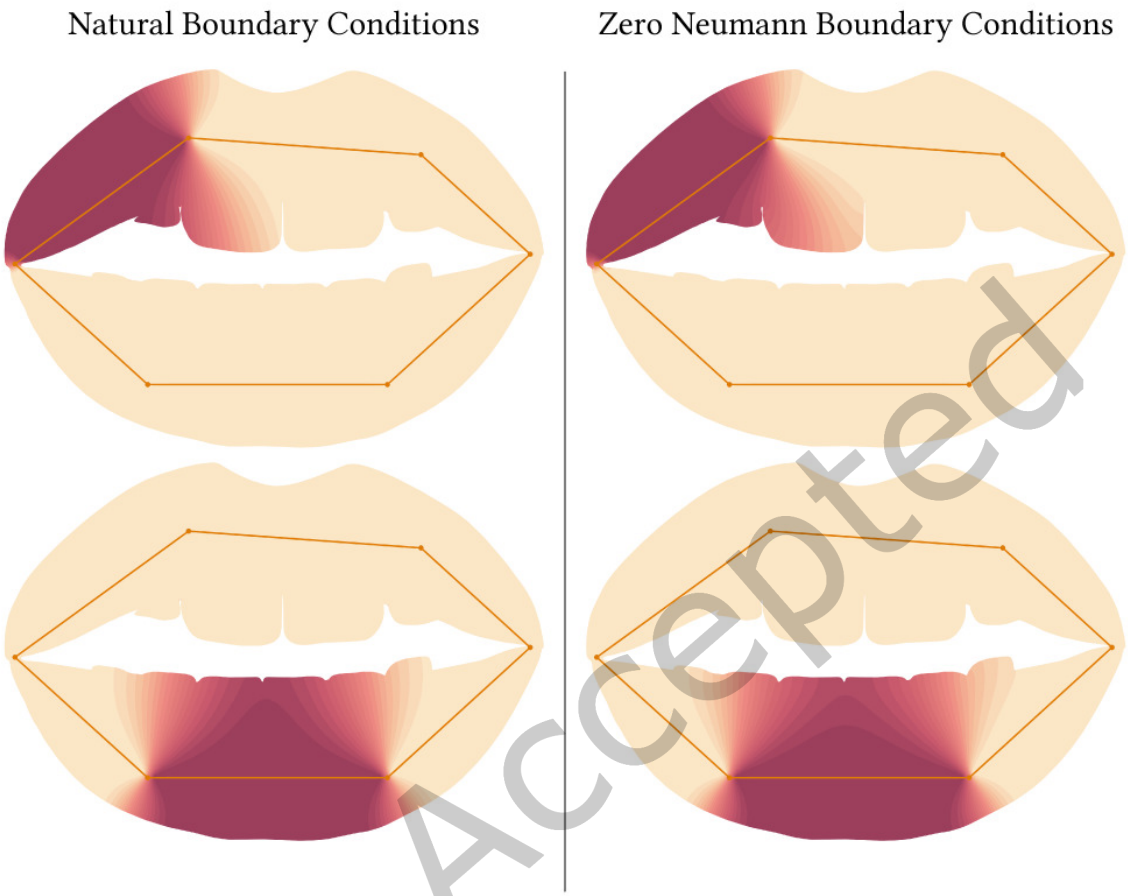




Fig. 16. We demonstrate the effect of the optional zero Neumann boundary condition constraints; on the right, the isolines of the level-sets are indeed orthogonal to the boundary of the shape.

were unable to find a consistent setting of the meshing tools' parameters that make previous methods produce usable results on all meshes.

*Limitations and Future Work.* Our boundary condition mollifiers are small but finite, and they are not geometry aware. This simple construction could result in unintuitive results if, for example, there were a bone placed extremely close to the boundary the Gear mesh, as this situation would cause bleeding. We did not observe this situation in our examples, since the width of the boundary conditions decreases as the underlying resolution increases (see Appendix A). This boundary case could be resolved by tracing rays to the closest point on the bone to check visibility, automatically determining the size of the boundary conditions based on the boundary geometry, or by adding a way to detect this case and informing the user.

Similarly, there can be situations where the distance between bones is so small that the Lagrange property is broken *if* our mollifiers and $\sigma$ are large-enough. Note, however, that even in the continuous formulation (1), it is possible for the Lagrange property to be broken, if bones touch or intersect. Furthermore, when animating

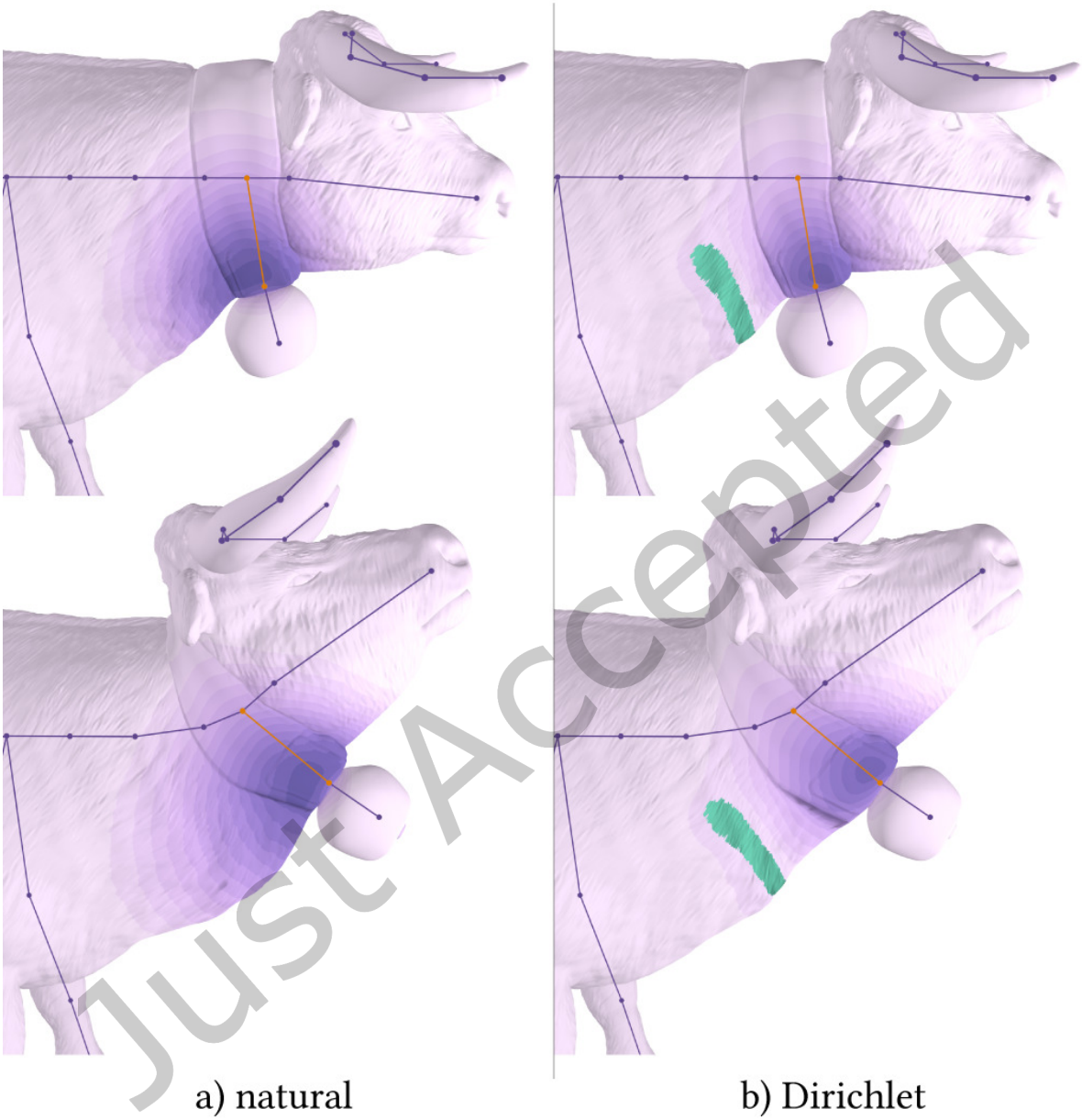




Fig. 17. Optimizing bounded biharmonic weights with natural boundary conditions offers little control in the final look of the animation (a). By pausing the optimization after 500 iterations and painting Dirichlet conditions onto a small region (pictured in green), the user can, for example, attenuate a handle's influence over said region (b).

filigree-like features that are small compared to the rest of the mesh, it can happen $\sigma$ is too large to meaningfully optimize the biharmonic energy in those regions. Compounding this effect, small filigree regions can be poorly sampled at lower resolutions. Such a case is demonstrated in Figure 19; the individual bones of the human's fingers are too small and too close to each other compared to the rest of the human to be meaningfully represented by the point cloud at the lowest resolutions. We anticipate that the solution to this will be importance sampling such that larger percentages of $\mathcal{X}$ and of the optimization samples are focused on narrow regions.‘

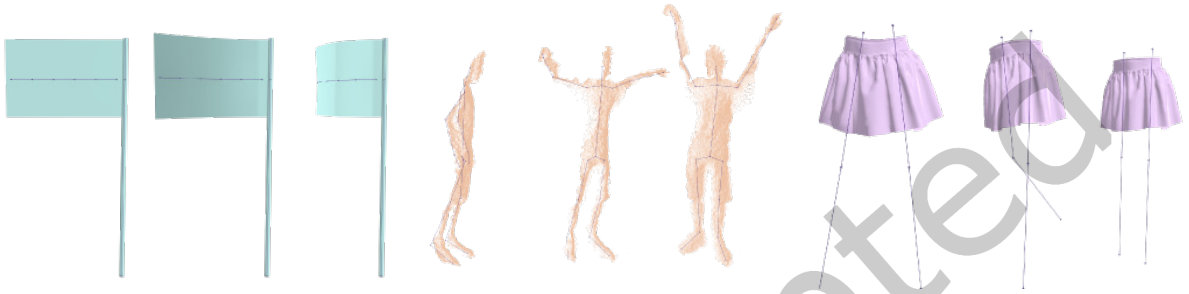

Fig. 18. Our method succeeds in computing weights in extreme stress test scenarios including a flag, a partial point cloud of a human from a smartphone depth sensor, and a skirt mesh with multiple disconnected patches.

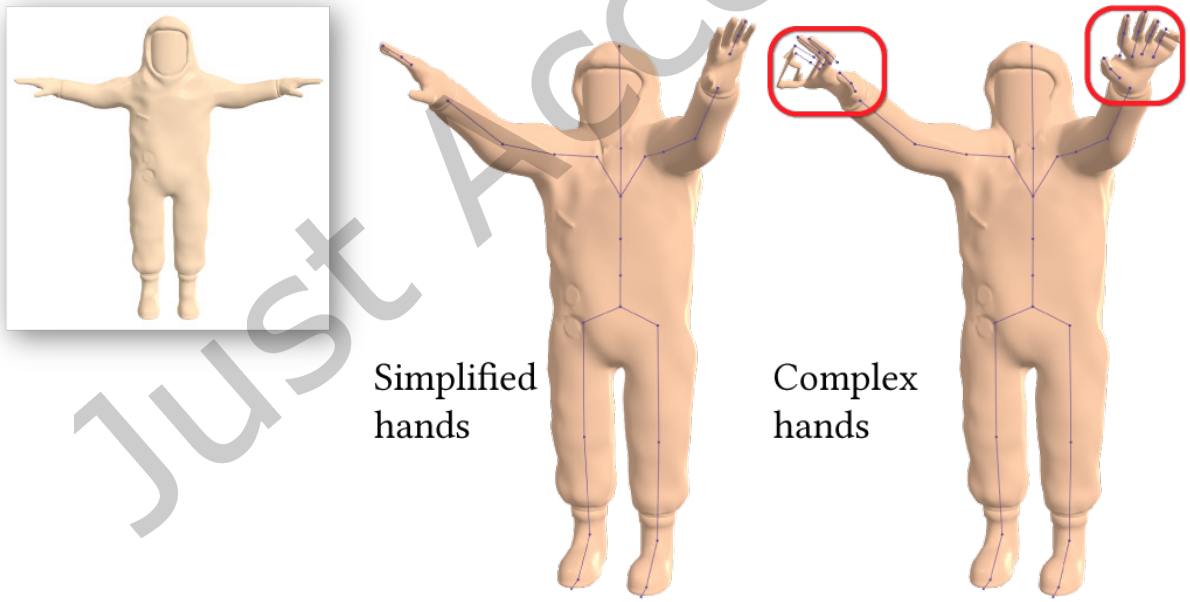


Fig. 19. The body of the person depicted in the Hazmat mesh is large compared to their fingers; the undeformed mesh is shown on the left. Despite disconnected and self-intersecting geometry, our method works as expected if we care to animate the body alone (center). If we need to animate each finger separately using a high-resolution hand rig (right), our method still produces a result in under 30 seconds, but the animations include visible artifacts. See discussion in Section 6.4.

Even though our method is reasonably fast, it involves ray-tracing hardware that is not yet ubiquitous; the majority of the runtime is spent tracing rays. While this limitation will become less of an issue as the availability of ray-tracing hardware increases in 3D workstations, it is worth finding ways of amortizing this cost.

Even though our method is able to handle extreme scenarios, our random sampling relies on the generalized winding number for inside-outside testing and therefore inherits the preconditions of the generalized winding number, including a consistent orientation of normals. However, our method is mostly orthogonal to the specific inside-outside testing algorithm—modifying it to work with a different robust inside-outside algorithm is likely possible, albeit out of the scope of the present works.

While we found our multiscale optimization strategy to be effective, we believe that the runtime could be further improved by finding a way to precondition geometric fields. Methods such as QHW [Wang and Solomon 2021], while using the Adam optimizer, rely on preconditioning for acceleration.

Lastly, our method is specifically tuned for bounded biharmonic weights, but one could imagine a general framework for solving variational problems using geometric fields. Many problems in geometry processing can be formulated in terms of a variational principle, including, e.g., mesh parameterization and deformation, various formulations of developable surface approximation, geometric flows, etc. In the specific context of the Dirichlet energy, future work might want to connect to the work of Sawhney and Crane [2020], where our representation could be used as a direct replacement, or—when convergence guarantees are needed—as a control variate.



## 7 Conclusion

We propose a mesh-free approach to automatic computation of skinning weights. This is enabled by our geometric field representation of skinning weights, through which we may optimize the biharmonic energy without the need for finite elements or constraints. Our method enables skinning weights computation on previously challenging or even impossible shapes, such as those with open surfaces or triangle soups, while achieving quality comparable to prior state-of-the-art approaches requiring volumetric meshes.

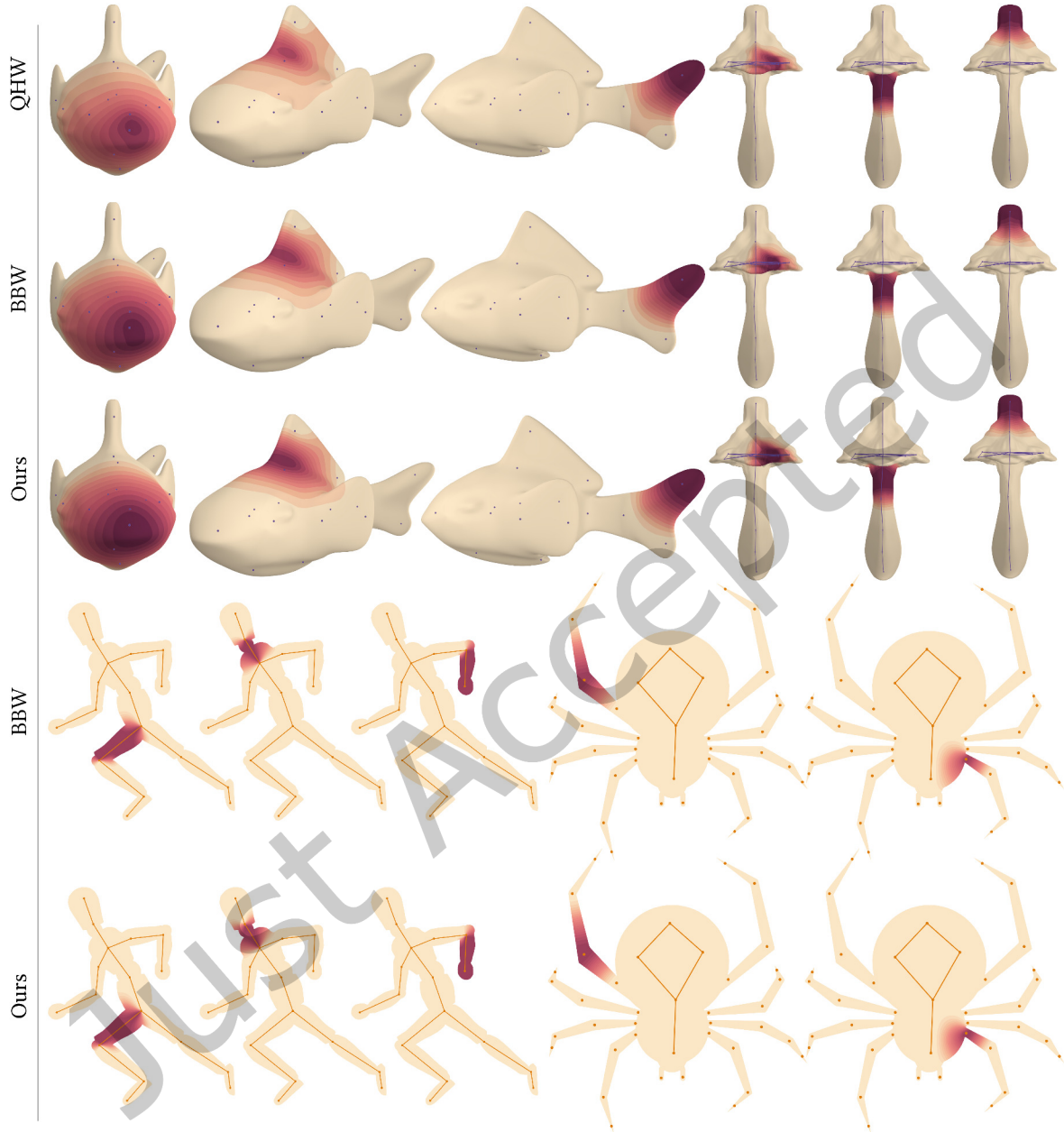




Fig. 20. Comparison of our results to quasi-harmonic weights QHW [Wang and Solomon 2021] and bounded biharmonic weights (BBW) [Jacobson et al. 2011]. QHW and BBW rely on tetrahedralization in 3D or triangulation in 2D, while our method requires only the boundary triangle mesh or polygon. All methods produce similar-looking weights as they are solving the same problem. Some minor differences are possible due to differences in discretization and enforcement of boundary conditions.

## Acknowledgments

We thank Yu Wang for helping us compare with quasi-harmonic weights, and to Enrique Rosales for the permission to use the Piggybank and Bunny meshes. Thank you to Ahmed Mahmoud, David Palmer, and Silvia Sellán for help with proof reading.

The MIT Geometric Data Processing Group acknowledges the generous support of Army Research Office grants W911NF2010168 and W911NF2110293, of Air Force Office of Scientific Research award FA9550-19-1-031, of National Science Foundation grant CHS-1955697, from the CSAIL Systems that Learn program, from the MIT–IBM Watson AI Laboratory, from the Toyota–CSAIL Joint Research Center, from a gift from Adobe Systems, and from a Google Research Scholar award.

The MIT Scene Representation Group acknowledges the generous support of the National Science Foundation under Grant No. 2211259, of the Singapore DSTA under DST00OECI20300823 (New Representations for Vision), of the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior/ Interior Business Center (DOI/IBC) under 140D0423C0075, of the Amazon Science Hub, and of IBM.

Oded Stein acknowledges the generous support of the SNF Early Postdoc Mobility fellowship and of the National Science Foundation (award #2335493).



## References

Marc Alexa, Markus Gross, Mark Pauly, Hanspeter Pfister, Marc Stamminger, and Matthias Zwicker. 2004. Point-based computer graphics. In *ACM SIGGRAPH 2004 Course Notes* (Los Angeles, CA) *(SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 7–es. doi:10.1145/1103900.1103907

Seungbae Bang and Sung-Hee Lee. 2018. Spline interface for intuitive skinning weight editing. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 1–14.

Ilya Baran and Jovan Popović. 2007. Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG)* 26, 3 (2007), 72–1–72–8.

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph.* 37, 4, Article 43 (July 2018), 12 pages. doi:10.1145/3197517.3201337

Mario Botsch and Leif Kobbelt. 2004. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 630–634.

Nithin Chalapathi, Yiheng Du, and Aditi S. Krishnapriyan. 2024. Scaling physics-informed hard constraints with mixture-of-experts. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=u3dX2CEIZb

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2023. Implicit Neural Spatial Representations for Time-dependent PDEs. In *International Conference on Machine Learning*.

Simin Chen, Zhixiang Liu, Wenbo Zhang, and Jinkun Yang. 2024. A Hard-Constraint Wide-Body Physics-Informed Neural Network Model for Solving Multiple Cases in Forward Problems for Partial Differential Equations. *Applied Sciences* 14, 1 (2024).

Xu Chen, Yufeng Zheng, Michael J Black, Otmar Hilliges, and Andreas Geiger. 2021. SNARF: Differentiable forward skinning for animating non-rigid neural implicit shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11594–11604.

Aditya Chetan, Guandao Yang, Zichen Wang, Steve Marschner, and Bharath Hariharan. 2023. Accurate Differential Operators for Hybrid Neural Fields. arXiv:2312.05984 [cs.CV]

Ronald R. Coifman and Stéphane Lafon. 2006. Diffusion maps. *Applied and Computational Harmonic Analysis* 21, 1 (2006), 5–30. doi:10.1016/j.acha.2006.04.006 Special Issue: Diffusion Maps and Wavelets.

Lorenzo Diazzi, Daniele Panozzo, Amir Vaxman, and Marco Attene. 2023. Constrained Delaunay Tetrahedrization: A Robust and Practical Approach. *ACM Trans. Graph.* 42, 6, Article 181 (dec 2023), 15 pages. doi:10.1145/3618352

Olivier Dionne and Martin de Lasa. 2013. Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 173–180.

Olivier Dionne and Martin de Lasa. 2014. Geodesic binding for degenerate character geometry using sparse voxelization. *IEEE Transactions on Visualization and Computer Graphics* 20, 10 (2014), 1367–1378.

Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. 2022. Neural Networks with Physics-Informed Architectures and Constraints for Dynamical Systems Modeling. arXiv:2109.06407 [cs.LG]

Ana Dodik, Oded Stein, Vincent Sitzmann, and Justin Solomon. 2023. Variational Barycentric Coordinates. *ACM Transactions on Graphics* (2023). doi:10.1145/3618403

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. 2000. Incorporating Second-Order Functional Knowledge for Better Option Pricing. In *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp (Eds.), Vol. 13. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2000/file/44968aece94f667e4095002d140b5896-Paper.pdf

Mark Gillespie, Nicholas Sharp, and Keenan Crane. 2021. Integer Coordinates for Intrinsic Geometry Processing. *ACM Trans. Graph.* 40, 6 (2021). doi:10.1145/3478513.3480522

Robert A Gingold and Joseph J Monaghan. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society* 181, 3 (1977), 375–389.

Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. 2020. Implicit geometric regularization for learning shapes. *Proceedings of the 37th International Conference on Machine Learning* (2020).

Henrik Halen and K Hayward. 2021. Global illumination based on surfels. In *Proc. ACM SIGGRAPH Symp. Interactive 3D Graph. Games.* 1399–1405.

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TriWild: robust triangulation with curve constraints. *ACM Trans. Graph.* 38, 4, Article 52 (jul 2019), 15 pages. doi:10.1145/3306346.3323011

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 39, 4, Article 117 (July 2020), 18 pages. doi:10.1145/3386569.3392385

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral Meshing in the Wild. *ACM Trans. Graph.* 37, 4, Article 60 (jul 2018), 14 pages. doi:10.1145/3197517.3201353

Alec Jacobson et al. 2021. gptoolbox: Geometry Processing Toolbox. http://github.com/alecjacobson/gptoolbox.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 30, 4 (2011), 78:1–78:8.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine. 2013. Robust Inside-Outside Segmentation using Generalized Winding Numbers. *ACM Trans. Graph.* 32, 4 (2013).

Alec Jacobson and Daniele Panozzo. 2017. libigl: prototyping geometry processing research in C++. In *SIGGRAPH Asia 2017 Courses* (Bangkok, Thailand) *(SA '17)*. Association for Computing Machinery, New York, NY, USA, Article 11, 172 pages. doi:10.1145/3134472.3134497

Alec Jacobson, Elif Tosun, Olga Sorkine, and Denis Zorin. 2010. Mixed finite elements for variational surface modeling. In *Computer graphics forum*, Vol. 29. Wiley Online Library, 1565–1574.

Alec Jacobson, Tino Weinkauf, and Olga Sorkine. 2012a. Smooth shape-aware functions with controlled extrema. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1577–1586.

Alec Jacobson, Tino Weinkauf, and Olga Sorkine. 2012b. Smooth Shape-Aware Functions with Controlled Extrema. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing)* 31, 5 (2012), 1577–1586.

Wenzel Jakob. 2022. nanobind: tiny and efficient C++/Python bindings. https://github.com/wjakob/nanobind.

Doug L James and Christopher D Twigg. 2005. Skinning mesh animations. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 399–407.

Timothy Jeruzalski, David IW Levin, Alec Jacobson, Paul Lalonde, Mohammad Norouzi, and Andrea Tagliasacchi. 2020. NiLBS: Neural inverse linear blend skinning. *arXiv:2004.05980* (2020).

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 71–1–71–9.

Yash Kant, Aliaksandr Siarohin, Riza Alp Guler, Menglei Chai, Jian Ren, Sergey Tulyakov, and Igor Gilitschenski. 2023. Invertible Neural Skinning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8715–8725.

Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4 (2008), 105.

Ladislav Kavan, Peter-Pike Sloan, and Carol O'Sullivan. 2010. Fast and efficient skinning of animated meshes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 327–336.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (2023).

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

Muhammed Kocabas, Jen-Hao Rick Chang, James Gabriel, Oncel Tuzel, and Anurag Ranjan. 2023. HUGS: Human gaussian splats. *arXiv:2311.17910* (2023).

Binh Huy Le and Zhigang Deng. 2012. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–10.

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–10.

Binh Huy Le and JP Lewis. 2019. Direct delta mush skinning and variants. *ACM Trans. Graph.* 38, 4 (2019), 113–1.

Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. 2021. Learning skeletal articulations with neural blend shapes. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhouyingcheng Liao, Vladislav Golyanik, Marc Habermann, and Christian Theobalt. 2023. VINECS: Video-based Neural Character Skinning. *arXiv:2307.00842* (2023).

Lijuan Liu, Youyi Zheng, Di Tang, Yi Yuan, Changjie Fan, and Kun Zhou. 2019. Neuroskinning: Automatic skin binding for production characters with deep graph networks. *ACM Transactions on Graphics (ToG)* 38, 4 (2019), 1–12.

Songming Liu, Hao Zhongkai, Chengyang Ying, Hang Su, Jun Zhu, and Ze Cheng. 2022. A Unified Hard-Constraint Framework for Solving Geometrically Complex PDEs. In *Advances in Neural Information Processing Systems*, Vol. 35. 20287–20299.

Lu Lu, Raphaël Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G. Johnson. 2021. Physics-Informed Neural Networks with Hard Constraints for Inverse Design. *SIAM Journal on Scientific Computing* 43, 6 (2021), B1105–B1132. arXiv:https://doi.org/10.1137/21M1397908 doi:10.1137/21M1397908

Leon B Lucy. 1977. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal* 82 (1977), 1013–1024.

Jing Ma and Dongliang Zhang. 2023. TARig: Adaptive template-aware neural rigging for humanoid characters. *Computers & Graphics* (2023).

Qianli Ma, Jinlong Yang, Michael J Black, and Siyu Tang. 2022. Neural Point-based Shape Modeling of Humans in Challenging Clothing. In *International Conference on 3D Vision (3DV)*. IEEE, 679–689.

Miles Macklin. 2022. Warp: A High-performance Python Framework for GPU Simulation and Graphics. https://github.com/nvidia/warp. NVIDIA GPU Technology Conference (GTC).

Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT)* 8, 2 (5 June 2019), 1–30. http://jcgt.org/published/0008/02/01/

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2023. Boundary Value Caching for Walk on Spheres. *ACM Trans. Graph.* 42, 4 (2023).

Arvind T. Mohan, Nicholas Lubbers, Misha Chertkov, and Daniel Livescu. 2023. Embedding hard physical constraints in neural network coarse-graining of three-dimensional turbulence. *Phys. Rev. Fluids* 8 (2023), 17 pages. Issue 1.

Albert Mosella-Montoro and Javier Ruiz-Hidalgo. 2022. SkinningNet: Two-stream graph convolutional neural network for skinning prediction of synthetic characters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18593–18602.

E. A. Nadaraya. 1964. On Estimating Regression. *Theory of Probability & Its Applications* 9, 1 (1964), 141–142. arXiv:https://doi.org/10.1137/1109020 doi:10.1137/1109020

Xuming Ouyang and Cunguang Feng. 2020. AutoSkin: Skeleton-based Human Skinning with Deep Neural Networks. In *Journal of Physics: Conference Series*, Vol. 1550. IOP Publishing, 032163.

Xiaoyu Pan, Jiancong Huang, Jiaming Mai, He Wang, Honglin Li, Tongkui Su, Wenjun Wang, and Xiaogang Jin. 2021. HeterSkinNet: A Heterogeneous Network for Skin Weights Prediction. In *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, Vol. 4. Association for Computing Machinery.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics* (August 2010).

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-style-high-performance-deep-learning-library.pdf

Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross. 2003. Shape modeling with point-sampled geometry. *ACM Trans. Graph.* 22, 3 (jul 2003), 641–650. doi:10.1145/882262.882319

Matt Pharr, Wenzel Jakob, and Greg Humphreys. 2016. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* (3rd ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 1266 pages.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707.

Ruben Rodriguez-Torrado, Pablo Ruiz, Luis Cueto-Felgueroso, Michael Cerny Green, Tyler Friesen, Sebastien Matringe, and Julian Togelius. 2021. Physics-informed attention-based neural network for solving non-linear partial differential equations. arXiv:2105.07898 [cs.LG]

Enrique Rosales, Jafet Rodriguez, and Alla Sheffer. 2019. SurfaceBrush: From Virtual Reality Drawings to Manifold Surfaces. *ACM Transaction on Graphics* 38, 4 (2019). doi:10.1145/3306346.3322970

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains. *ACM Trans. Graph.* 39, 4 (2020).

Rohan Sawhney, Bailey Miller, Ioannis Gkioulekas, and Keenan Crane. 2023. Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions. *ACM Trans. Graph.* 42, 4 (2023).

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-Free Monte Carlo for PDEs with Spatially Varying Coefficients. *ACM Trans. Graph.* XX, X (2022).

Nicholas Sharp and Keenan Crane. 2020a. A laplacian for nonmanifold triangle meshes. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 69–80.

Nicholas Sharp and Keenan Crane. 2020b. You Can Find Geodesic Paths in Triangle Meshes by Just Flipping Edges. *ACM Trans. Graph.* 39, 6 (2020).

Nicholas Sharp, Mark Gillespie, and Keenan Crane. 2021. Geometry Processing with Intrinsic Triangulations. (2021).

Hang Si. 2015. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* 41, 2, Article 11 (feb 2015), 36 pages. doi:10.1145/2629697

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proceedings of the Conference on Neural Information Processing Systems*.

Oded Stein, Eitan Grinspun, Max Wardetzky, and Alec Jacobson. 2018. Natural boundary conditions for smoothing in geometry processing. *ACM Transactions on Graphics (TOG)* 37, 2 (2018), 1–13.

N. Sukumar and Ankit Srivastava. 2022. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering* 389 (2022), 114333.

J-M Thiery and Elmar Eisemann. 2018. ARAPLBS: Robust and efficient elasticity-based optimization of weights and skeleton joints for linear blend skinning with parametrized bones. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 32–44.

Elif Tosun. 2008. *Geometric modeling using high-order derivatives*. Ph. D. Dissertation. New York University.

Jane Tournois, Noura Faraj, Jean-Marc Thiery, and Tamy Boubekeur. 2024. Tetrahedral Remeshing. In *CGAL User and Reference Manual* (5.6.1 ed.). CGAL Editorial Board. https://doc.cgal.org/5.6.1/Manual/packages.html#PkgTetrahedralRemeshing

Ingo Wald. 2020. OWL - A Productivity Library for OptiX. http://owl-project.github.io

Kevin Wampler. 2016. Fast and reliable example-based mesh IK for stylized deformations. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 1–12.

Yu Wang and Justin Solomon. 2021. Fast quasi-harmonic weights for geometric data interpolation. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Geoffrey S. Watson. 1964. Smooth Regression Analysis. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)* 26, 4 (1964), 359–372. http://www.jstor.org/stable/25049340

Jane Wu, Zhenglin Geng, Hui Zhou, and Ronald Fedkiw. 2020. Skinning a parameterization of three-dimensional space for neural network cloth. *arXiv:2006.04874* (2020).

Shangzhe Wu, Ruining Li, Tomas Jakab, Christian Rupprecht, and Andrea Vedaldi. 2023. MagicPony: Learning articulated 3d animals in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8792–8802.

Chuhua Xian, Shuo Jin, and Charlie CL Wang. 2018. Efficient $C^2$-weighting for image warping. *IEEE Computer Graphics and Applications* 38, 1 (2018), 59–76.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 641–676.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: neural rigging for articulated characters. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 58–1.

Ze Yang, Shenlong Wang, Sivabalan Manivasagam, Zeng Huang, Wei-Chiu Ma, Xinchen Yan, Ersin Yumer, and Raquel Urtasun. 2021. S3: Neural shape, skeleton, and skinning fields for 3d human modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13284–13293.

Fangcheng Zhong, Kyle Fogarty, Param Hanji, Tianhao Wu, Alejandro Sztrajman, Andrew Spielberg, Andrea Tagliasacchi, Petra Bosilj, and Cengiz Oztireli. 2024. Neural fields with hard constraints of arbitrary differential order. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 992, 26 pages.



## A Implementation Details

Our algorithm is primarily implemented in PyTorch [Paszke et al. 2019], with the exception of closest-point queries in Section 4.3 and geometry-aware radius queries in Section 4.1. For these, we provide our own low-level CUDA implementations.

For the geometry-aware kernel queries, we first build an SPH hash-grid in CUDA, with hash-grid cell sizes equal to the kernel radius. For each query point, we loop over all of the points in the the neighboring cells. If a

point falls within the kernel radius, we trace a ray between it and the query point, as explained in Section 4.1. For the ray-tracing itself, we rely on hardware-acceleration via the Optix [Parker et al. 2010] and OWL [Wald 2020] libraries. In practice, this means that the kernel queries are implemented as *ray-generation* shaders in Optix parlance. The closest point-queries rely on the Warp library [Macklin 2022], which internally builds a bounding-volume hierarchy over the boundary mesh.

We make the CUDA kernels interoperable with PyTorch by relying on `nanobind` [Jakob 2022]. We use the fast generalized winding numbers implementation from `libigl` [Barill et al. 2018; Jacobson and Panozzo 2017].

Before optimizing, we rescale meshes to fit into the $[0, 1]^d$ unit box. We generate $\mathcal{X}$ by uniformly sampling the interior of the mesh, with an initial budget of $2^{11}$ points in 2D and $2^{13}$ points in 3D. If a mesh of the interior is unavailable, we rely on fast winding number and perform rejection sampling until the target budget is reached. Our initial learning rate is set to 0.2. Each iteration, we sample $M = 2^{12}$ points in 2D. Since $\mathcal{X}$ is larger in 3D compared to 2D, using the same number of optimization samples would become prohibitively slow. For this reason, we use fewer samples in 3D, $M = 2^{11}$.

We train for a total of 6000 steps, and double the size of $\mathcal{X}$ a total of 3 times at regular intervals throughout the optimization. Every time we upsample the representation, we lower the learning rate by a factor of $\frac{1}{3}$.

We found it important to lower the $\beta_2$ parameter of Adam [Kingma and Ba 2014] to 0.8. A higher value of $\beta_2$ results in the optimizer slowing down if the gradient changes direction. This is useful in high-variance scenarios, but when optimizing variational energies, this is the opposite of what we want; in our scenario, gradients change directions often as boundary conditions propagate.

The sizes of the boundary condition parameters have been determined through our ablation study in Figure 11. We set the value of the radii for the Neumann, Lagrange, and Dirichlet conditions equal to half the initial value of $\sigma$. We update the radius of the Lagrange and Dirichlet conditions as we upscale the point cloud.

*Evaluation on the Boundary.* When trying to evaluate the weights *on* the boundary of a shape, the imprecision of the ray-tracing hardware can produce noisy results. Since it is highly unlikely that a random point will land close enough to the boundary for this to matter, this does not impact the quality of the weights. This means that our optimization remains unmodified. At evaluation time, we evaluate our weights *on* the boundary using a few simple heuristics that work well in all of our tests.

To evaluate the kernel at a vertex $\boldsymbol{y}$, we compute the kernel weights between $\boldsymbol{y}$ and $\boldsymbol{x} \in \mathcal{X}$, but we trace rays between $\boldsymbol{y} - 10^{-5}\boldsymbol{n_y}$ and $\boldsymbol{x}$. In other words, we nudge $\boldsymbol{y}$ slightly into the shape for the purposes of ray tracing alone.

In some cases, a vertex can be classified as outside of the shape by the generalized winding number, and/or completely occluded from any point $\boldsymbol{x} \in \mathcal{X}$ (e.g., a vertex slightly protrudes through a different triangle, common in VR ribbon drawings or triangle soups). For all such vertices, we flood-fill their values from their neighbors using an averaging operation. In the rare case that there are still any vertices with no weights, we run our kernel with no ray tracing.

## B Smoothing Function

Here we include definitions of the smoothing functions we rely on. As defined by Dugas et al. [2000], the `softplus` function is:

$$\texttt{softplus}(t) = \log(1 + \exp(t)). \tag{14}$$

*Lagrange Condition Mollifier.* In using $w$ to construct $\widetilde{\boldsymbol{e}}$ we must ensure that the partition of unity property (1.2) remains conserved, recognizing that the $\varepsilon$-neighborhoods of handles often overlap, e.g., because two bones share

a vertex. Denoting by $d_i(\boldsymbol{x})$ the distance of a sample $\boldsymbol{x}$ to the $i^{\text{th}}$ handle *within* the $\varepsilon$-neighborhoods, we write:

$$\widetilde{e}_i(\boldsymbol{x}) \coloneqq \max_{j=1\ldots K} \left\{ w\big(d_j(\boldsymbol{x})\big) \right\} \frac{w(d_i(\boldsymbol{x}))}{\sum_{j=1}^{K} w\big(d_j(\boldsymbol{x})\big)}. \tag{15}$$

For a point $\boldsymbol{x}$ *outside* of all $\varepsilon$-neighborhoods, we define $\widetilde{e}_i(\boldsymbol{x}) \coloneqq 0$ to avoid division by zero.

In the case where $\boldsymbol{x}$ is within a $\varepsilon$-neighborhood of exactly one handle, (15) simplifies to $\widetilde{e}_i(\boldsymbol{x}) = w(d_i(x))$. If $\boldsymbol{x}$ happens to be inside multiple overlapping $\varepsilon$-neighborhoods, we use the bump function value of the closest handle and distribute it among all neighboring handles in proportion to the value of their respective bump function at $\boldsymbol{x}$. As a concrete example, if a point is equidistant to two different handles such that both of their bump functions equal 0.5, both $\widetilde{e}_i$ will be assigned the value 0.25, ensuring (1.2) still holds.

There are many possible *bump functions* one could choose—for example, in all our experimets we use:

$$w'(t) = \begin{cases} 0 & t < 0 \\ 1 & t > 1 \\ \frac{\exp(-\frac{1}{t})}{\exp(-\frac{1}{t})+\exp(-\frac{1}{1-t})} & \text{otherwise,} \end{cases} \tag{16}$$
$$w(t) = 1 - w'\left(\frac{t^2}{\varepsilon^2}\right).$$

Here, $w'$ has the convenient property that all of its higher-order derivatives disappear at 0 and 1; that way, the derivatives of $w'$ do not interfere with our energy computation when applying the chain rule.

Received 11 December 2024; revised 14 June 2025; accepted 1 September 2025