Fast Quasi-Harmonic Weights for Geometric Data Interpolation

YU WANG and JUSTIN SOLOMON, Massachusetts Institute of Technology, USA

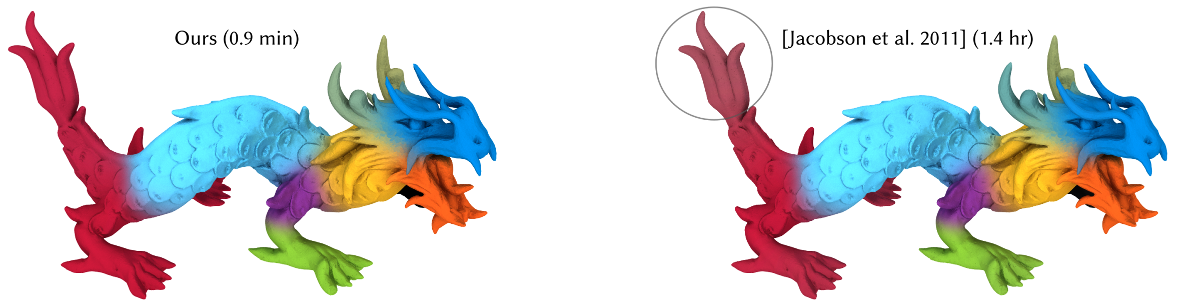


Fig. 1. Color interpolation using our skinning weights and bounded biharmonic weights [Jacobson et al. 2011]. Note the latter weights (and thus the blended color) have a local extrema at the tail of the Asian dragon. Our weights are free from local extrema and have a slightly lower smoothness energy. More importantly, our weights are many orders of magnitude faster to compute. Previous methods can take hours on this model, which has more than one million tetrahedra. Our method takes less than a minute, with potential for further speedups.

We propose *quasi-harmonic weights* for interpolating geometric data, which are orders of magnitude faster to compute than state-of-the-art. Currently, interpolation (or, skinning) weights are obtained by solving large-scale constrained optimization problems with explicit constraints to suppress oscillative patterns, yielding smooth weights only after a substantial amount of computation time. As an alternative, our weights are obtained as minima of an unconstrained problem that can be optimized quickly using straightforward numerical techniques. We consider weights that can be obtained as solutions to a parameterized family of second-order elliptic partial differential equations. By leveraging the maximum principle and careful parameterization, we pose weight computation as an inverse problem of recovering optimal anisotropic diffusivity tensors. In addition, we provide a customized ADAM solver that significantly reduces the number of gradient steps; our solver only requires inverting tens of linear systems that share the same sparsity pattern. Overall, our approach achieves orders of magnitude acceleration compared to previous methods, allowing weight computation in near real-time.

CCS Concepts: • **Computing methodologies** → **Animation**.

Additional Key Words and Phrases: skinning animation, inverse problem, optimal control, partial differential equations, geometric variational problem.

ACM Reference Format:
Yu Wang and Justin Solomon. 2021. Fast Quasi-Harmonic Weights for Geometric Data Interpolation. *ACM Trans. Graph.* 40, 4, Article 73 (August 2021), 15 pages. https://doi.org/10.1145/3450626.3459801

1 INTRODUCTION

Skinning is perhaps the simplest and most popular method for shape deformation. The difficulty of drawing skinning weights by hand, however, motivates automatic computation of skinning weights as a critical way to make the animation pipeline more efficient. Beyond their original application in animation, however, automatic skinning weight computation has found application in graphics pipelines beyond animation as a generic tool to interpolate geometric and physical quantities smoothly across geometric domains.

The problem of computing skinning weights amounts to designing a *partition of unity*, or set of functions that sums to 1 at every point on the domain, that satisfies a few properties. There is one skinning function per control handle of a deforming shape, which typically equals 1 at the handle and decays as we move farther away; this function indicates the influence of displacing that handle on the deformation of the rest of the shape. Other desirable properties for skinning weights include smoothness, nonnegativity, and a lack of local optima away from the handles.

Typically, automatic skinning weights are computed as solutions to certain geometric variational problems. Linear partial differential equations (PDEs) yield weights that are fast to compute, such as harmonic weights [Joshi et al. 2007]. These weights, however, have undesirable properties for animation, notably spike artifacts and nonsmoothness at control handles. Higher-order PDEs like the biharmonic equation [Botsch and Kobbelt 2004] remove some issues while introducing new ones, such as oscillation, negative values, and local extrema [Jacobson et al. 2011]. Hence, a popular method, bounded biharmonic weights [Jacobson et al. 2011, 2012b], adds explicit constraints to the problem to prevent these artifacts, yielding a quadratic programming formulation that is considerably slower to solve. This prohibits applications where fast iteration or interactive feedback are desired (see e.g. [Wang et al. 2015]).

As an alternative to the methods above, we present an algorithm that extracts high-quality skinning weights orders of magnitude faster than past work. Rather than relying on explicit constraints, we optimize in the family of second-order elliptic PDEs parameterized by positive definite tensor fields, which can be interpreted as spatially-varying anisotropic diffusivity. The maximum principle

Authors' address: Yu Wang, wangyu9@mit.edu; Justin Solomon, jsolomon@mit.edu, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA, 02139, USA.

0730-0301/2021/8-ART73
https://doi.org/10.1145/3450626.3459801

holds for elliptic PDEs, automatically guaranteeing that our weights are positive and without local extrema.

Our method is built on the key insight that second-order elliptic PDEs provide a *rich* parametric family of weights that automatically satisfy all necessary constraints (§4.3). Within this family, we simply search for the minimizer of a conventional smoothness energy. This leads to a nonlinear but unconstrained optimization problem.

We propose a customized ADAM optimizer to solve the problem. Typical gradient-based optimizers used in geometry processing, such as L-BFGS, take substantially more steps to converge for our problem. In contrast, our customized ADAM optimizer is extremely efficient, only requiring tens of gradient steps. The major computational cost of our method is to solve 10 to 20 sparse linear systems, each with the same sparsity pattern. We can accelerate solution of these sparse linear systems using symbolic pre-factorization.

Our method takes seconds to find optimal weights on difficult examples, compared to the constrained optimizations in previous methods that could take hours. Our weights are visually similar to the state-of-the-art with orders-of-magnitude speedup, as confirmed on typical examples in skinning animations (see §7).

2 RELATED WORK

Interpolating data along geometric domains is a ubiquitous task in graphics, geometry processing, and manifold data analysis. Applications in these areas encounter the problem of determining a set of weight functions such that each function corresponds to a control handle (or data point) on the domain, indicating the spatial influence of the handle; the weights are then used to propagate data at the control handles to the remainder of the domain. These weights are known as *skinning weights* due to their use in character animation.

2.1 Applications of Skinning Weights

The main application of skinning weights is to determine a shape deformation from handle displacements. See [Jacobson et al. 2014] for an extensive survey of skinning-based animation; here we mention a few exemplary methods.

Linearly interpolating affine transformation matrices, known as linear blend skinning (LBS) [Badler and Morris 1982; Magnenat-Thalmann et al. 1988], is a simple and effective model for shape deformation and is widely integrated in graphics pipelines. For skeleton-based character animation, however, LBS can suffer from candy wrapper artifacts, which are avoided by dual quaternion skinning (DQS) [Kavan et al. 2008]. Lie group structure has also been explored to interpolate transformations [Alexa 2002; Bansal and Tatu 2019]. Given the rest pose and skinning weights, Le and Hodgins [2016] optimize the center of rotation for each vertex.

In semi-supervised learning, Zhu et al. [2003] use harmonic interpolation for label propagation by solving a Laplacian system. Their interpolation method—as well as numerous follow-up techniques—can be understood as an application of harmonic skinning weights to higher-dimensional data.

2.2 Computing skinning weights

When they are not painted manually by artists, skinning weights can be computed purely based on the geometry of the input shape (discussed below) or learned from exemplary frames of animated sequences [James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012, 2014; Wampler 2016]. The latter is applicable only to limited scenarios with relevant data, so in this paper we focus on the former, computing *geometric skinning weights*.

Harmonic weights. Solutions to harmonic equations can be used as interpolation weights [Joshi et al. 2007; Zhu et al. 2003]. Harmonic weights, however, are only C^0 at control handles. In fact, harmonic weights for point handles are mesh resolution-dependent and not well-defined continuously; this is analogous to the fact that the harmonic distance to a point is singular and ill-defined, motivating the biharmonic distance [Lipman et al. 2010].

Heat diffusion weights. Baran and Popović [2007] use solutions to the heat equation as skinning weights. Again, these weights are not smooth at control handles (see e.g. [Jacobson et al. 2011]). Despite the drawbacks, due to their simplicity heat diffusion weights have been integrated in graphics software tools like Maya and Blender.

Biharmonic weights. Solutions to the biharmonic equation can also be used as skinning weights [Botsch and Kobbelt 2004]. Biharmonic weights are smooth but unbounded and oscillative: They can be negative or greater than one, and they possibly have local extrema away from handles. Triharmonic weights demonstrate similar behaviors and tend to have more oscillations [Jacobson et al. 2010; Tosun 2008].

Bounded biharmonic weights (BBW). Jacobson et al. [2011] minimize the biharmonic energy with explicit constraints to prevent negative weights. In follow-up work, Jacobson et al. [2012b] derive sufficient constraints to further prevent local extrema. As a popular geometric skinning method, bounded biharmonic weights have been used in many applications, such as physical simulation and multi-grid methods [Chen et al. 2016; Xian et al. 2019].

Other methods. Bang and Lee [2018] propose an interactive spline interface to edit weights. Thiery and Eisemann [2018] simultaneously optimize skeletons and associated weights. Liu et al. [2019] learn the mesh-skeleton binding patterns from pairs of mesh and weights provided by artists. The recent direct delta mush skinning applies Laplacian smoothing on top of linear blend skinning and requires only binary rigid binding weights [Le and Lewis 2019]. Jacobson et al. [2012a] achieve fast reduced simulation and dynamics within the subspaces spanned by linear blend skinning. Popular industry solutions include ngSkinTools [Makauskas 2013] for painting weights and geodesic voxel binding [Dionne and de Lasa 2013], which has been integrated in Maya.

2.3 Barycentric coordinates

Another family of deformation methods updates vertex positions as weighted sums of control handle positions; the corresponding weights are known as *barycentric coordinates*. See [Nieto and Susín 2013] for a survey.

Barycentric control handles are typically specified as cages. Popular cage-based coordinates include mean value coordinates [Floater 2003; Hormann and Floater 2006; Ju et al. 2005], harmonic coordinates [Joshi et al. 2007], Green coordinates [Lipman et al. 2008], maximum entropy coordinates [Hormann and Sukumar 2008], Poisson coordinates [Li and Hu 2012], and locally barycentric coordinates [Zhang et al. 2014]. These barycentric coordinates, however, cannot be used as skinning weights: They have to be used together with a cage enclosing but being apart from the shape, since they are only C^0 at cage vertices.

Some barycentric coordinates do not rely on a bounding cage, allowing for isolated vertices or skeletons within the domain. Thin plate splines [Bookstein 1989] are popular to interpolate data in Euclidean space and have been generalized to geometric domains e.g. by minimizing a modified bi-Laplacian energy [Wang et al. 2015] or Hessian energy [Stein et al. 2018]. These coordinates, however, can be oscillative and negative, making them unsuitable for use as skinning weights. Herholz et al. [2017] combine the computation of a Voronoi diagram of control handles with natural neighbor coordinates [Sibson and Barnett 1981]; the resulting coordinates, while being sparse, are comparably slower to compute than bounded biharmonic weights. Yan and Schaefer [2019] modify and extend the Floater-Hormann-Kós family of barycentric coordinates to non-convex polygons.

3 BACKGROUND AND PRELIMINARIES

3.1 Problem Setup

Consider a geometric domain Ω with m control handles $\mathbf{c}_1, \mathbf{c}_2, \dots \mathbf{c}_m$; each control handle is a region on the domain Ω. We assume each handle $\mathbf{c}_j$ is associated with a weight function $w_j(\cdot)$, indicating its spatial influence across the domain. For a smooth function $f : \Omega \to \mathbb{R}$ whose value is known only at $\mathbf{c}_1, \mathbf{c}_2, \dots \mathbf{c}_m$, we would like to approximate $f(\mathbf{x})$ as

$$f(\mathbf{x}) \approx \sum_{j=1}^{m} w_j(\mathbf{x}) f(\mathbf{c}_j) \tag{1}$$

at any $\mathbf{x} \in \Omega$. The values of $f(\mathbf{c}_1), f(\mathbf{c}_2), \dots, f(\mathbf{c}_m)$ are propagated to every other point via the weight functions $w_1(\cdot), w_2(\cdot), \dots, w_m(\cdot)$. For example, in linear blend skinning, $f(\cdot)$ is an affine transformation whose value at each control handle is specified by the user at runtime; typically the skinning weights $w_j(\cdot)$ are computed ahead of time. In this paper, we restrict to volumetric domains $\Omega \subseteq \mathbb{R}^d, d \in \{2, 3\}$, with smooth boundary, although in principle it is straightforward to extend to the case of a general manifold Ω.

Our task is to find a set of weight functions $w_j(\cdot)$ such that (1) gives high quality interpolation. Ideally, the $w_j(\cdot)$'s should be smooth functions of $\mathbf{x}$ and should sum to 1 at every $\mathbf{x}$, that is, $\sum_j w_j(\mathbf{x}) = 1 \ \forall \mathbf{x}$, so that $f(\mathbf{x})$ is a weighted average of the values $f(\mathbf{c}_j)$. More specifically, several properties are desirable for $w_j(\cdot)$ to produce high-quality skinning animations:

$$\begin{array}{rll}
\text{(Lagrange)} & \forall \mathbf{c}_i : w_j(\mathbf{c}_i) = \delta_{ij} & i, j = 1, \dots, m \\
\text{(Partition of unity)} & \forall \mathbf{x} : \displaystyle\sum_{j=1}^{m} w_j(\mathbf{x}) = 1 & \\
\text{(Nonnegativity)} & \forall \mathbf{x} : w_j(\mathbf{x}) \geq 0 & j = 1, \dots, m \\
\text{(No local extrema)} & \forall \mathbf{x} : w_j(\mathbf{x}) \text{ not a local extremum} & j = 1, \dots, m
\end{array} \tag{2}$$

where $\delta_{ij} = 1_{(i=j)}$ is the Kronecker delta function.

The Lagrange property ensures hard interpolation in which (1) reproduces the value of $f(\mathbf{c}_j)$ at $\mathbf{x} \in \mathbf{c}_j$. In the context of linear blend skinning, the Lagrange property allows the user to specify affine transformations at handles exactly, the partition of unity property ensures that the skinning method is translationally invariant, and the nonnegativity and no-local-extrema properties avoid counterintuitive skinning behavior [Jacobson et al. 2011, 2012b]. Given the desiderata above, our main task is to compute a set of smooth weight functions $w_j(\cdot)$ satisfying all constraints in (2).

3.2 PDE-based Weights

Harmonic weights provide a simple way to obtain skinning weights satisfying many considerations above [Joshi et al. 2007; Zhu et al. 2003]. They are obtained by solving Laplace equation with assorted boundary conditions:

$$\begin{array}{rll}
\Delta w_j(\mathbf{x}) = 0 & \forall j = 1, \dots, m \\
w_j(\mathbf{c}_i) = \delta_{ij} & \forall i, j = 1, \dots, m \\
\frac{\partial}{\partial \mathbf{n}} w_j(\mathbf{x}) = 0 & \forall \mathbf{x} \in \partial\Omega \backslash \{\mathbf{c}_k\}_{k=1}^m, \forall j = 1, \dots, m.
\end{array} \tag{3}$$

When $\{\mathbf{c}_k\}_{k=1}^m \subseteq \partial\Omega$, e.g., in cage-based deformation [Joshi et al. 2007], harmonic weights become generalized barycentric coordinates and are termed *harmonic coordinates*. The partition of unity, nonnegativity, and no-local-extrema properties hold for free for harmonic weights, thanks to properties of harmonic functions.

Harmonic weights solve the following optimization problem:

$$\begin{array}{rl}
\min_w & \sum_{j=1}^m \int_\Omega \|\nabla w_j(\mathbf{x})\|^2 \\
\text{s.t.} & w_j(\mathbf{c}_i) = \delta_{ij} \quad \forall i, j = 1, \dots, m
\end{array} \tag{4}$$

Using this variational formulation, we can see harmonic weights use Dirichlet energy as a smoothness measure.

For skinning animation, control handles are often isolated points or line segments. Continuous solutions to (3), however, are ill-defined in this case. Discrete solutions can be obtained by inverting a Laplacian matrix, but the solutions are resolution-dependent with spike artifacts at control points or bone edges. Being only C^0 at control handles, harmonic weights are not suitable for skinning weights, although they *can* be used for harmonic coordinates since the cage is placed away from the deforming shape.

Higher-order functionals and PDEs can make the weight function well-defined and C^1 in handle neighborhoods. In computer vision and image processing, thin plate splines [Bookstein 1989] use biharmonic functions for interpolating data over $\mathbb{R}^2$. When the domain is only a sub-region of $\mathbb{R}^2$, biharmonic weights [Botsch and Kobbelt 2004] can be used, solving the following optimization problem:

$$\begin{array}{rl}
\min_w & \sum_{j=1}^m \int_\Omega \|\Delta w_j(\mathbf{x})\|^2 \\
\text{s.t.} & w_j(\mathbf{c}_i) = \delta_{ij} \quad \forall i, j = 1, \dots, m
\end{array} \tag{5}$$

Similarly, one can resort to even higher-order smoothness energies such as tri-harmonic energies [Jacobson et al. 2010; Tosun 2008]. Another variant of biharmonic weights is to use higher-order boundary conditions [Stein et al. 2018; Wang et al. 2015] such that the resulting weights are also generalized barycentric coordinates. Higher-order energies promote smoothness at handles, but the weights can become oscillative and negative.

To remove negative weights, Jacobson et al. [2011] propose the following optimization problem:

$$\begin{array}{lll} \min_w & \sum_{j=1}^{m} \int_\Omega \|\Delta w_j(\mathbf{x})\|^2 & \\ \text{s.t.} & w_j(\mathbf{c}_i) = \delta_{ij} & \forall i,j = 1,\dots,m \\ & \sum_{j=1}^m w_j(\mathbf{x}) = 1 & \forall \mathbf{x} \in \Omega \\ & w_j(\mathbf{x}) \geq 0 & \forall j = 1\dots,m. \end{array} \tag{6}$$

Jacobson et al. [2012b] further add topological constraints to prevent local extrema, and Zhang et al. [2014] propose a variant with an L^1 loss. Solving this convex quadratic program requires active set and conic programming solvers that are orders of magnitude slower than solving the linear system used by harmonic weights (see e.g. [Wang et al. 2015]).

3.3 Anisotropic Laplacian and Elliptic PDE

The homogeneous, isotropic Laplacian operator $\Delta_{\mathbb{R}^d} = \sum_{i=1}^d \frac{\partial^2}{\partial x_i^2}$ is extensively used in physics and geometry processing. A more generic version is the *anisotropic Laplacian*, which has spatially-varying parameters that weight derivatives along different directions [Gilbarg and Trudinger 2015].

More precisely, an anisotropic Laplacian operator is specified by a *conductivity tensor field* $\mathbf{A}(\mathbf{x}) : \Omega \to \mathbb{R}^{d\times d}$, where $d \in \{2,3\}$ is the dimension of Ω, such that $\mathbf{A}(\mathbf{x}) \succeq 0$, i.e. $\mathbf{A}(\mathbf{x})$ is positive semi-definite (p.s.d.) for any $\mathbf{x} \in \Omega$. We use $a_{ji}(\mathbf{x})$ to denote the entry at position (j,i) in the matrix $\mathbf{A}(\mathbf{x})$.

The anisotropic Laplacian $\Delta^{\mathbf{A}}$ with the associated co-derivative operator $\nabla^{\mathbf{A}}_{\mathbf{n}}$ generalizes the usual Laplacian $\Delta_{\mathbb{R}^d}$ with the normal derivative operator $\nabla_{\mathbf{n}} = \mathbf{n} \cdot \nabla$ as follows:

$$\Delta^{\mathbf{A}} = \sum_{i,j=1}^{d} \frac{\partial}{\partial x_j}\left[a_{ji}(\mathbf{x})\frac{\partial}{\partial x_i}\right] = \nabla \cdot [\mathbf{A}(\mathbf{x})\nabla],$$

$$\nabla^{\mathbf{A}}_{\mathbf{n}} = \sum_{i,j=1}^{d} \mathbf{n}_j \left[a_{ji}(\mathbf{x})\frac{\partial}{\partial x_i}\right] = \mathbf{n}^\top[\mathbf{A}(\mathbf{x})\nabla].$$

Similar to the relationship between (3) and (4), the variational problem

$$\begin{array}{ll} \min_{u(\cdot)} & \int_\Omega \nabla u(\mathbf{x})^\top \mathbf{A}(\mathbf{x}) \nabla u(\mathbf{x}) \\ \text{s.t.} & u(\mathbf{x}) = h(\mathbf{x}) \quad \forall \mathbf{x} \in \mathbf{c} \end{array} \tag{7}$$

is equivalent to the PDE system

$$\begin{array}{rl} \Delta^{\mathbf{A}} u(\mathbf{x}) = 0 & \\ \nabla^{\mathbf{A}}_{\mathbf{n}} u(\mathbf{x}) = 0 & \forall \mathbf{x} \in \partial\Omega/\mathbf{c} \\ u(\mathbf{x}) = h(\mathbf{x}) & \forall \mathbf{x} \in \mathbf{c}. \end{array} \tag{8}$$

4 QUASI-HARMONIC WEIGHTS

We now introduce our new model for computing skinning weights on geometric domains, which we term *quasi-harmonic weights.* At a high level, these weights are obtained by optimizing a smoothness energy (5) over the space of harmonic weights generated by anisotropic Laplacians (described in §3.3); our variables are the anisotropy tensors $\mathbf{A}(\mathbf{x})$. This section introduces our model from a smooth perspective; §5 then introduces a discretization suitable for triangle/tetrahedral meshes.

4.1 A Parametric Family of Weights

By replacing the Laplacian in (3) with the anisotropic Laplacian operator, we obtain *quasi-harmonic weights* by solving the following linear problem:

$$\begin{array}{rl} \Delta^{\mathbf{A}} w_j(\mathbf{x}) = 0 & \forall j = 1,\dots,m \\ w_j(\mathbf{c}_i) = \delta_{ij} & \forall i,j = 1,\dots,m \\ \nabla^{\mathbf{A}}_{\mathbf{n}} w_j(\mathbf{x}) = 0 & \forall \mathbf{x} \in \partial\Omega/\mathbf{c}, \forall j = 1,\dots,m. \end{array} \tag{9}$$

We say that the resulting weight functions $w_1(\cdot), w_2(\cdot), \dots, w_m(\cdot)$ are *generated* by the tensor field $\mathbf{A}(\cdot)$, denoted $w = g(\mathbf{A})$.

As a direct generalization of harmonic weights, elliptic PDE weights also satisfy all the desirable properties:

Proposition 4.1. *Solutions to* (9) *satisfy all the properties listed in* (2) *as long as* $\mathbf{A}(\cdot)$ *is uniformly positive definite: for some* $\delta > 0$, $\mathbf{A}(\mathbf{x}) \succeq \delta \mathbf{I}\ \forall \mathbf{x}$.

Proof. The weights are nonnegative and have no local extrema, as a direct result of the celebrated *maximum principle* (see e.g. [Gilbarg and Trudinger 2015]), which holds for second-order elliptic PDEs. The partition of unity property also holds since (9) is a linear PDE with boundary conditions $w_j(\mathbf{c}_i) = \delta_{ij}$, such that $\forall i : \sum_j w_j(\mathbf{c}_i) = 1$. □

Quasi-harmonic weights comprise a large parametric family

$$\mathcal{F} = \{g(\mathbf{A}) \mid \mathbf{A}(\cdot) : \Omega \to \mathbb{R}^{d\times d} \text{ uniformly positive definite}\} \tag{10}$$

that all satisfy the properties listed in (2). Surprisingly, beyond including the classical harmonic weights, we find weights in this family that look closer to bounded biharmonic weights and other alternatives.

4.2 Proposed Model

Given the observations above, our algorithm searches for weights within the family $\mathcal{F}$. That is, our weights are generated by an unknown p.s.d. anisotropy tensor field $\mathbf{A}(\cdot)$, chosen to optimize a smoothness functional $E(\cdot)$:

$$\inf_{\mathbf{A}} E(w) \text{ s.t. } w = g(\mathbf{A}).$$

Restricting our weights to be generated by some tensor field is the key step in designing our fast method for computing weights. In particular, this restriction allows us to transform the problem into an *unconstrained optimization*, thanks to Proposition 4.1.

In more detail, we consider the optimization problem:

$$\begin{array}{lll} \inf_{\mathbf{A}} & \sum_{j=1}^m \int_\Omega \|\Delta w_j(\mathbf{x})\|^2 & \\ \text{s.t.} & w_j(\mathbf{c}_i) = \delta_{ij} & \forall i,j = 1,\dots,m \\ & \Delta^{\mathbf{A}} w_j(\mathbf{x}) = 0 & \forall \mathbf{x} \in \Omega, \forall j = 1,\dots,m \\ & \nabla^{\mathbf{A}}_{\mathbf{n}} w_j(\mathbf{x}) = 0 & \forall \mathbf{x} \in \partial\Omega, \forall j = 1,\dots,m \\ & \mathbf{A}(\mathbf{x}) \succeq 0 & \forall \mathbf{x} \in \Omega \end{array} \tag{11}$$

In words, our problem finds the quasi-harmonic weights w_j with the lowest biharmonic smoothness energy. This is a variational problem with *PDE constraints.* As justified in §4.3, the PDE constraints serve as alternatives to the partition of unity, nonnegativity and no-local-extrema constraints in previous methods. This new formulation is capable of removing singularities in the derivatives of harmonic weights by adjusting the anisotropy tensors: The coefficients $\mathbf{A}(\mathbf{x})$ provide additional degrees of freedom allowing to "swap" singularities in $\nabla w(\mathbf{x})$ to the coefficients $\mathbf{A}(\mathbf{x})$ when necessary.

Although the formulation (11) may look complicated, it is straightforward to solve in practice. In particular, the weight functions $w_j(\cdot)$ are just intermediate variables that can be eliminated entirely, since they are uniquely determined by the tensor field $\mathbf{A}(\cdot)$; this leaves an unconstrained problem whose free variables are the tensors $\mathbf{A}(\cdot)$. In subsequent sections, we show this problem can be very efficiently solved.

Remark (Parameter identification). *To the best of our knowledge, problem* (11) *has not been previously studied in theory or in practice. But, we briefly discuss a related formulation known as* distributed parameter identification, *which has been studied in inverse problems and optimal control, defined as follows:*

$$\begin{array}{rll} \min_{\mathbf{A}} & \int_\Omega \|w(\mathbf{x}) - u(\mathbf{x})\|^2 & \\ s.t. & \textit{some boundary conditions} & \\ & \Delta^{\mathbf{A}} w(\mathbf{x}) = 0 & \forall \mathbf{x} \in \Omega \\ & \mathbf{A}(\mathbf{x}) \geq 0 & \forall \mathbf{x} \in \Omega \end{array} \tag{12}$$

where $u(\cdot)$ is a prescribed function. Similar to our formulation (11), *in problem* (12), *$w(\cdot)$ is completely determined by $\mathbf{A}(\cdot)$. Hence, the optimization is over $\mathbf{A}(\cdot)$, while $w(\cdot)$ can be thought of as an auxiliary variable. Problem* (12) *is a well-posed* inverse problem *[Kirsch 2011]: When $u(\cdot)$ is the solution to a second-order PDE,* (12) *tries to recover the true tensor field $\mathbf{A}(\cdot)$ that generates $u(\cdot)$. Distributed parameter identification is related to our problem in that it considers nearly identical PDE constraints. It also sheds light on the representation capacity of our parametric family $\mathcal{F}$.*

Convexity of (11). While we are unable to verify convexity of the problem (11), in practice we find that our formulation does not have noticeable local optima. In our experiments, we observed that our method always converges to the same weights, regardless of initialization. We conjecture that our problem may have hidden convex structure, the identification of which we leave for future work. In fact, certain variants of the parameter identification problem [Hinze and Quyen 2016]—which have the same PDE constraints but a different objective to our problem—are proven to be convex.

4.3 Representation Capacity of Quasi-harmonic Weights

Searching in the parametric family $\mathcal{F}$ is the key difference between our fast weight computation scheme and past work. To justify this restriction, we need to verify that the family $\mathcal{F}$ is suitably rich. The following representation theorem indicates that $\mathcal{F}$ contains the key functions of interest to skinning:

Theorem 4.2 ([Richter 1981], Main Theorem). *For a connected bounded domain $\Omega \subset \mathbb{R}^2$ and function $w : \Omega \to \mathbb{R}$ satisfying*

$$\inf_{\mathbf{x} \in \Omega} \max(|\nabla w(\mathbf{x})|, \Delta w(\mathbf{x})) > 0, \tag{13}$$

there exists some $\mathbf{A}(\mathbf{x}) = a(\mathbf{x})\mathbf{I} \geq 0$ such that $\nabla \cdot [\mathbf{A}(\mathbf{x}) \nabla w(\mathbf{x})] = 0$.

Theorem 4.2 suggests that when there are two control handles (i.e., one independent weight function) with associated smooth skinning weight functions $\{w(\mathbf{x}), 1 - w(\mathbf{x})\}$ such that (13) holds, there exists some $\mathbf{A}(\cdot)$ generating them.

For example, given a target weight function $w(\mathbf{x})$, if we know $\forall \mathbf{x} \in \Omega : |\nabla w(\mathbf{x})| > \epsilon$ for some $\epsilon > 0$ then we can conclude that $w(\cdot)$ can be generated by some $\mathbf{A}(\cdot)$; otherwise, it is often easy to modify $w(\cdot)$ to a visually similar function $w_\epsilon(\cdot)$ such that $\forall \mathbf{x} \in \Omega : |\nabla w_\epsilon(\mathbf{x})| > \epsilon$ for small ϵ, provided that $w(\cdot)$ has no local extrema.

The construction in Theorem 4.2 only uses one degree of freedom for $\mathbf{A}(\mathbf{x})$, sufficient to handle the case with two control handles. We conjecture that $\mathbf{A}(\mathbf{x}) \in \mathbb{R}^{d \times d}$ can reproduce up to to $d(d+1)/2 + 1$ prescribed weights, when exploiting all degrees of freedom. That is, if we know each vertex is supported by no more than 4 handles in the 2D case (7 handles in 3D), there exists some tensor field $\mathbf{A}(\mathbf{x})$ generating the weights. Aligning theory to practice, in skinning animation the most common case is that each skin vertex is controlled by 2 skeleton handles; it is common to avoid having a vertex controlled by too many handles.

In summary, existing theory suggests that restricting the weights within the parametric family $\mathcal{F}$ does not limit the representation capacity. While we do not have proofs covering all scenarios, experiments in §7 confirm that our weights have similar smoothness energies to previous methods, suggesting our space of skinning weights is not compromised noticeably.

Tensor field $\mathbf{A}(\cdot)$—a change of metric. While the tensor field $\mathbf{A}(\cdot)$ is commonly understood as the spatially varying diffusivity in the parameter identification setup, it also has a clear geometric interpretation. Our model (11) can be understood as first modifying the metric locally according to $\mathbf{A}(\cdot)$, computing harmonic weights on the modified shape, and then pulling back the weights onto the original shape. So, $\mathbf{A}(\cdot)$ represents a change of metric that is aware of the control handles.

5 DISCRETIZATION

We use a simple yet convenient discretization of our continuous problem (11). We assume the (volumetric) geometric domain Ω has been discretized as a triangle (tetrahedral) mesh with n vertices and f triangles (tetrahedra).

5.1 Relevant Matrices and Operators

First, we discretize the anisotropic Laplacian $\Delta_{\mathbf{A}}$. We use a piecewise constant discretization for the anisotropy tensor field represented by a matrix $\mathbf{A} \in \mathbb{R}^{df \times df}$ $(d = 2, 3)$ satisfying

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \textsc{diag}(\mathbf{a}_{00}) & \textsc{diag}(\mathbf{a}_{01}) \\ \textsc{diag}(\mathbf{a}_{10}) & \textsc{diag}(\mathbf{a}_{11}) \end{bmatrix} \quad (d = 2) \\ \mathbf{A} &= \begin{bmatrix} \textsc{diag}(\mathbf{a}_{00}) & \textsc{diag}(\mathbf{a}_{01}) & \textsc{diag}(\mathbf{a}_{02}) \\ \textsc{diag}(\mathbf{a}_{10}) & \textsc{diag}(\mathbf{a}_{11}) & \textsc{diag}(\mathbf{a}_{12}) \\ \textsc{diag}(\mathbf{a}_{20}) & \textsc{diag}(\mathbf{a}_{21}) & \textsc{diag}(\mathbf{a}_{22}) \end{bmatrix} \quad (d = 3) \end{aligned} \tag{14}$$

where $\mathbf{a}_{ij} \in \mathbb{R}^{f \times 1}$ $(i, j \in \{0, \ldots, d-1\})$ contains the element at position (i, j) of the local tensor for every face; $\textsc{diag}(\cdot)$ expands a vector into a diagonal matrix. That is, the anisotropy tensor at the

j-th triangle (tetrahedron) is

$$\begin{bmatrix}(\mathbf{a}_{00})_j & (\mathbf{a}_{01})_j \\ (\mathbf{a}_{01})_j & (\mathbf{a}_{11})_j\end{bmatrix} \qquad (d=2)$$
$$\begin{bmatrix}(\mathbf{a}_{00})_j & (\mathbf{a}_{01})_j & (\mathbf{a}_{02})_j \\ (\mathbf{a}_{01})_j & (\mathbf{a}_{11})_j & (\mathbf{a}_{12})_j \\ (\mathbf{a}_{02})_j & (\mathbf{a}_{12})_j & (\mathbf{a}_{22})_j\end{bmatrix} \qquad (d=3).$$

We have already enforced the symmetry conditions $\mathbf{a}_{10} = \mathbf{a}_{01}$ and $\mathbf{a}_{20} = \mathbf{a}_{02}$.

$\mathbf{A}$ is a $d \times d$ block matrix with $f \times f$ diagonal blocks. For convenience, we also stack the nonzero coefficients of $\mathbf{A}$ into a vector $\mathbf{a}$ such that:

$$\mathbf{a} = \begin{bmatrix}\mathbf{a}_{00} \\ \mathbf{a}_{10} \\ \mathbf{a}_{11}\end{bmatrix} \in \mathbb{R}^{3f\times 1} \qquad (d=2)$$
$$\mathbf{a} = \begin{bmatrix}\mathbf{a}_{00} \\ \mathbf{a}_{10} \\ \mathbf{a}_{11} \\ \mathbf{a}_{20} \\ \mathbf{a}_{21} \\ \mathbf{a}_{22}\end{bmatrix} \in \mathbb{R}^{6f\times 1} \qquad (d=3) \tag{15}$$

We use $\mathbf{a} = \textsc{flatten}(\mathbf{A})$ to denote the conversion from the sparse matrix format in (14) to the compact vector format in (15). For now we do not require $\mathbf{A}$ to be p.s.d.; we introduce this constraint in §5.4.

Denote $\mathbf{M} \in \mathbb{R}^{n\times n}$ as the per-vertex mass matrix, $\mathbf{M}_f \in \mathbb{R}^{f\times f}$ as the per-face mass matrix (a diagonal matrix stacking the area/volume of each triangle/tetrahedron), and $\mathbf{G} \in \mathbb{R}^{df\times n}$ as the usual discrete gradient operator using piecewise linear bases, with

$$\mathbf{G} = \begin{bmatrix}\mathbf{G}_x \\ \mathbf{G}_y\end{bmatrix} \in \mathbb{R}^{2f\times n} \quad (d=2), \qquad \mathbf{G} = \begin{bmatrix}\mathbf{G}_x \\ \mathbf{G}_y \\ \mathbf{G}_z\end{bmatrix} \in \mathbb{R}^{3f\times n} \quad (d=3),$$

where $\mathbf{G}_x, \mathbf{G}_y, \mathbf{G}_z$ are discrete approximations of $\frac{\mathrm{d}}{\mathrm{d}x}, \frac{\mathrm{d}}{\mathrm{d}y}, \frac{\mathrm{d}}{\mathrm{d}z}$, respectively. Recall that

$$\mathbf{L} = \mathbf{G}^\top \begin{bmatrix}\mathbf{M}_f & \\ & \mathbf{M}_f\end{bmatrix} \mathbf{G} \qquad (d=2)$$
$$\mathbf{L} = \mathbf{G}^\top \begin{bmatrix}\mathbf{M}_f & & \\ & \mathbf{M}_f & \\ & & \mathbf{M}_f\end{bmatrix} \mathbf{G} \qquad (d=3),$$

where $\mathbf{L}$ is the cotangent Laplacian matrix. Analogously, the anisotropic Laplacian $\Delta_\mathbf{A} = \nabla \cdot [\mathbf{A}(\mathbf{x})\nabla]$ is discretized by the matrix $\mathbf{G}^\top\mathbf{A}\mathbf{G}$.

Next, we introduce notation for the weights and boundary conditions. Assume there are m control handles $\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m$; each control handle $\mathbf{c}_j$ consists of a few vertices that are connected in the mesh. The skinning weights are discretized as the matrix $\mathbf{W} \in \mathbb{R}^{n\times m}$, where $\mathbf{W}_{ij}$ is the weight of the j-th control handle for the i-th vertex of the mesh. Denote $M = \sum_{j=1}^m \textsc{size}(\mathbf{c}_j)$ to be the total number of vertices of the control handles; $M \geq m$ since each control handle contains at least one vertex.

Rows in $\mathbf{W}$ that correspond to control handle vertices are known: Denote by $\mathbf{R} \in \mathbb{R}^{n\times M}$ the binary selection matrix such that $\mathbf{R}_{:j}^\top$ selects rows for the vertices representing the j-th control handle, and denote by $\mathbf{B} \in \mathbb{R}^{M\times m}$ the boundary condition matrix, such that $\mathbf{R}^\top\mathbf{W} = \mathbf{B}$. $\mathbf{B}$ is the known part of the weights matrix $\mathbf{W}$, so $\mathbf{B}_{ij}$ is non-zero only if the i-th vertex is part of control handle $\mathbf{c}_j$.

Similarly, denote by $\mathbf{S} \in \mathbb{R}^{n\times(M-m)}$ the binary selection matrix such that $\mathbf{S}^\top$ selects rows that correspond to vertices that are not control handles, and $\mathbf{U} \in \mathbb{R}^{(n-M)\times m}$ as the unknown part of $\mathbf{W}$, such that $\mathbf{S}^\top\mathbf{W} = \mathbf{U}$. Together we have $\mathbf{W} = \mathbf{S}\mathbf{U} + \mathbf{R}\mathbf{B}$ and

$$\text{(known part)} \quad \mathbf{R}^\top\mathbf{W} = \mathbf{B}$$
$$\text{(unknown part)} \quad \mathbf{S}^\top\mathbf{W} = \mathbf{U}.$$

5.2 Discretized Optimization Problem

With above definitions, the Lagrange property $w_j(\mathbf{c}_i) = \delta_{ij}$ in (11) becomes the discrete equation $\mathbf{R}^\top\mathbf{W} = \mathbf{B}$, and the PDE constraint $\Delta^\mathbf{A} w_j(\Omega) = 0$ with $\nabla_\mathbf{n}^\mathbf{A} w_j(\partial\Omega/\mathbf{c}) = 0$ becomes $\mathbf{S}^\top[\mathbf{G}^\top\mathbf{A}\mathbf{G}]\mathbf{W} = 0$. This amounts to applying the finite element method with piecewise-linear bases.

We discretize the continuous problem in (11) as follows:

$$\begin{aligned} \min_\mathbf{A} \quad & \tfrac{1}{2}\textsc{tr}(\mathbf{W}^\top\mathbf{Q}\mathbf{W}) \\ \text{s.t.} \quad & \mathbf{R}^\top\mathbf{W} = \mathbf{B} \\ & \mathbf{S}^\top\mathbf{W} = \mathbf{U} \\ & [\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}]\mathbf{U} + [\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{R}]\mathbf{B} = 0, \end{aligned} \tag{16}$$

where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a quadratic form measuring the smoothness of the weights. For example, $\mathbf{Q}$ can discretize the usual bilaplacian energy by taking $\mathbf{Q} := \mathbf{L}\mathbf{M}^{-1}\mathbf{L}$ [Jacobson et al. 2011]. In §5.4 we use a (re-)parameterization of $\mathbf{A}$ that easily ensures positive-definiteness.

Note the (unknown) weights $\mathbf{U}$ are uniquely determined by $\mathbf{A}$ via

$$\mathbf{U} = -[\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}]^{-1}\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{R}\mathbf{B} \tag{17}$$

Substituting this expression into (16) yields an unconstrained optimization over $\mathbf{A}$ (or—equivalently—its flattened counterpart $\mathbf{a}$).

5.3 Differentiating the Unconstrained Problem

To apply optimization algorithms, we need to differentiate the objective of (16) with respect to the unknown tensors $\mathbf{A}$. We derive relevant expressions below.

To start, we differentiate $\mathbf{U}$ w.r.t. $\mathbf{a}$ by differentiating the third constraint of (16). Consider an infinitesimal deviation $\delta\mathbf{A}$ from $\mathbf{A}$, such that $\delta\mathbf{A}$ has the same size and sparsity pattern as $\mathbf{A}$. The resulting deviation of $\mathbf{U}$, denoted $\delta\mathbf{U}$, satisfies

$$\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}\delta\mathbf{U} + \mathbf{S}^\top\mathbf{G}^\top\delta\mathbf{A}\mathbf{G}\mathbf{S}\mathbf{U} + \mathbf{S}^\top\mathbf{G}^\top\delta\mathbf{A}\mathbf{G}\mathbf{R}\mathbf{B} = 0,$$

or equivalently,

$$\delta\mathbf{U} = -[\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}]^{-1}\mathbf{S}^\top\mathbf{G}^\top\delta\mathbf{A}\mathbf{G}[\mathbf{S}\mathbf{U} + \mathbf{R}\mathbf{B}].$$

Vectorizing this expression shows

$$\frac{\partial\mathbf{U}_{:j}}{\partial\mathbf{a}} = -[\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}]^{-1}\mathbf{S}^\top\mathbf{G}^\top\textsc{sp}(\mathbf{G}\mathbf{S}\mathbf{U}_{:j} + \mathbf{G}\mathbf{R}\mathbf{B}_{:j}) \in \mathbb{R}^{n\times\frac{d(d+1)}{2}f}$$

In this expression, we use a new operator $\textsc{sp}(\cdot)$, chosen to satisfy the expression $\mathbf{A}\mathbf{b} = \textsc{sp}(\mathbf{b})\textsc{flatten}(\mathbf{A})$. For $\mathbf{b} = \begin{bmatrix}\mathbf{b}_0 & \mathbf{b}_1\end{bmatrix}^\top \in \mathbb{R}^{2f\times 1}$ with $\mathbf{b}_0, \mathbf{b}_1 \in \mathbb{R}^{f\times 1}$, we define

$$\textsc{sp}(\mathbf{b}) := \begin{bmatrix}\textsc{diag}(\mathbf{b}_0) & \textsc{diag}(\mathbf{b}_1) & \\ & \textsc{diag}(\mathbf{b}_0) & \textsc{diag}(\mathbf{b}_1)\end{bmatrix}$$

For $\mathbf{b} = \begin{bmatrix}\mathbf{b}_0 & \mathbf{b}_1 & \mathbf{b}_2\end{bmatrix}^\top \in \mathbb{R}^{3f\times 1}$ with $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^{f\times 1}$, we define

$$\textsc{sp}(\mathbf{b}) := \begin{bmatrix}\textsc{diag}(\mathbf{b}_0) & \textsc{diag}(\mathbf{b}_1) & & \textsc{diag}(\mathbf{b}_2) & & \\ & \textsc{diag}(\mathbf{b}_0) & \textsc{diag}(\mathbf{b}_1) & & \textsc{diag}(\mathbf{b}_2) & \\ & & & \textsc{diag}(\mathbf{b}_0) & \textsc{diag}(\mathbf{b}_1) & \textsc{diag}(\mathbf{b}_2)\end{bmatrix}$$

Finally, we can derive a formula for $\frac{\partial E}{\partial \mathbf{a}}$. Defining the objective $E(\mathbf{W}) := \frac{1}{2}\textsc{tr}(\mathbf{W}^\top \mathbf{Q}\mathbf{W})$ and substituting (17), we have

$$E = \frac{1}{2}\textsc{tr}(\mathbf{U}^\top\mathbf{S}^\top\mathbf{Q}\mathbf{S}\mathbf{U}) + \textsc{tr}(\mathbf{B}^\top\mathbf{R}^\top\mathbf{Q}\mathbf{S}\mathbf{U}) + \cancelto{\textsc{const.}}{\frac{1}{2}\textsc{tr}(\mathbf{B}^\top\mathbf{R}^\top\mathbf{Q}\mathbf{R}\mathbf{B})}$$
$$= \sum_{j=1}^{m} \frac{1}{2}\mathbf{U}_{:j}^\top\mathbf{S}^\top\mathbf{Q}\mathbf{S}\mathbf{U}_{:j} + \mathbf{B}_{:j}^\top\mathbf{R}^\top\mathbf{Q}\mathbf{S}\mathbf{U}_{:j} + \textsc{const.} \tag{18}$$

Hence,

$$\frac{\partial E}{\partial \mathbf{a}} = \sum_{j=1}^{m} \underbrace{[\mathbf{U}_{:j}^\top\mathbf{S}^\top\mathbf{L}\mathbf{M}^{-1}\mathbf{L}\mathbf{S} + \mathbf{B}_{:j}^\top\mathbf{R}^\top\mathbf{L}\mathbf{M}^{-1}\mathbf{L}\mathbf{S}]}_{\partial E/\partial \mathbf{U}_{:j}} \left[\frac{\partial \mathbf{U}_{:j}}{\partial \mathbf{a}}\right]$$
$$= \sum_{j=1}^{m} [\mathbf{W}_{:j}^\top\mathbf{L}\mathbf{M}^{-1}\mathbf{L}\mathbf{S}] \left[\frac{\partial \mathbf{U}_{:j}}{\partial \mathbf{a}}\right] \tag{19}$$

where

$$\frac{\partial \mathbf{U}_{:j}}{\partial \mathbf{a}} = -[\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}]^{-1}\mathbf{S}^\top\mathbf{G}^\top\textsc{sp}(\mathbf{G}\mathbf{S}\mathbf{U}_{:j} + \mathbf{G}\mathbf{R}\mathbf{B}_{:j})$$

Deriving $\frac{\partial E}{\partial \mathbf{a}}$ involves differentiating a matrix inverse, but as shown above, evaluating $\frac{\partial E}{\partial \mathbf{a}}$ involves the same sparse linear system matrix $\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}$ that we solved to obtain the weights. Once the matrix has been factored, at each iteration two back substitutions are required: One for computing current weights and the other for evaluating the gradients. In §6.2, we will discuss reusing matrix factorization in details.

5.4 Tensor Field Parameterization

Our final task is to discretize the anisotropy matrix $\mathbf{A}$ in (14) such that the anisotropy tensor (matrix) at the j-th triangle (tetrahedron) is p.s.d. for $j = 1, 2, ..., f$. That is, we need the set of 2×2 or 3×3 p.s.d. matrices. There are many ways to solve this problem; we provide one simple one below for completeness.

For a semidefinite 2×2 matrix, we can explicitly write out its Cholesky factorization

$$\begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} a & 0 \\ b & c \end{bmatrix} \begin{bmatrix} a & b \\ 0 & c \end{bmatrix} = \begin{bmatrix} a^2 & ab \\ ab & b^2 + c^2 \end{bmatrix} \succeq 0.$$

Similarly, in 3D, we write

$$\begin{bmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{bmatrix} \begin{bmatrix} a & b & d \\ 0 & c & e \\ 0 & 0 & f \end{bmatrix} = \begin{bmatrix} a^2 & ab & ad \\ ab & b^2 + c^2 & bd + ce \\ ad & bd + ce & d^2 + e^2 + f^2 \end{bmatrix} \succeq 0$$

We optimize for the elements (a, b, c) or (a, b, c, d, e, f) of the Cholesky factors rather than for the semidefinite tensor elements directly; differentiating the objective of our problem with respect to these unknowns is a simple application of the chain rule to (19). This parameterization p.s.d. by construction, and any p.s.d. matrix can be written in this form thanks to the Cholesky factorization.

Although the continuous maximum principle holds for elliptic PDEs, a discrete maximum principle may not necessarily hold for our problem on meshes. Note even discrete harmonic weights can be negative when there are obtuse angles if using the cotangent Laplacian [Wardetzky et al. 2007]. Without a discrete maximum principle, the resulting weights can be negative or have spurious local extrema, which vanish when refining the mesh.

Many papers propose discretizations with a discrete maximum principle [Droniou and Potier 2011], such as using nonlinear FEM [Liska and Shashkov 2008; Lu et al. 2014]. While these methods are more complex than is needed our application, they provide critical insights. In particular, we know that the mesh quality and the largest condition number of anisotropy tensors are two factors determining the difficulty of ensuring a discrete maximum principle. Hence, a simple practical approach to avoid failure of the discrete maximum principle is to bound the condition number of the anisotropy tensors. For example, the following parameterization specifies a *cone* of p.s.d. tensors with bounded condition number:

$$\begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} a^2 + \kappa(b^2 + c^2) + \epsilon & ab \\ ab & b^2 + c^2 + \kappa a^2 + \epsilon \end{bmatrix},$$

where $\kappa > 0$ and $\epsilon \geq 0$ are constants. The following bound on the condition number of the tensor holds:

$$\textsc{cond}\left(\begin{bmatrix} E & F \\ F & G \end{bmatrix}\right) \leq \max\left(\kappa, 1 + \frac{2}{\kappa}\right).$$

In 3D, the formula becomes

$$\begin{bmatrix} a^2 + \kappa(b^2+c^2+d^2+e^2+f^2) + \epsilon & ab & ad \\ ab & b^2+c^2+\kappa(a^2+d^2+e^2+f^2) + \epsilon & bd + ce \\ ad & bd + ce & d^2+e^2+f^2+\kappa(a^2+b^2+c^2) + \epsilon \end{bmatrix}.$$

We choose $\kappa = 0.2, \epsilon = 10^{-4}$ in our experiments. The anisotropy tensors are initialized as identity matrices multiplied by triangle areas (tetrahedron volumes).

Denote $\boldsymbol{\theta}$ as the final parameters, i.e. the collection of LU coefficients a, b, c or $(a, b, c, d, e, f$ in 3D) for each triangle (tetrahedron) in a column vector. By the chain rule we have $\frac{\partial E}{\partial \boldsymbol{\theta}} = \frac{\partial E}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \boldsymbol{\theta}}$, where $\frac{\partial \mathbf{a}}{\partial \boldsymbol{\theta}}$ is the Jacobian matrix of our parameterization. In practice, we find a further change of variables works slightly better $(a \leftarrow \frac{1}{a}, b \leftarrow \frac{b}{a}, c \leftarrow \frac{c}{a}, d \leftarrow \frac{d}{a}, e \leftarrow \frac{e}{a}, f \leftarrow \frac{f}{a})$.

5.5 Differentiable Projection

Our discrete weights $\mathbf{W}$ can be mildly negative due to violations of the discrete maximum principle. While mildly negative weights do not typically cause problems in practice, for fair comparison to previous methods that strictly prohibit negative weights, we introduce a simple modification to our method that yields nonnegative weights.

To eliminate negative weights, for each vertex i we project its weights $\{\mathbf{W}_{i,1}, \mathbf{W}_{i,2}, ...\mathbf{W}_{i,m}\}$ back onto the probability simplex $\mathbb{P}^m$. We choose a projection mapping $\mathbf{p} : \mathbb{R}^m \rightarrow \mathbb{P}^m$ as a composition of two functions: $\mathbf{p}(w_1, ..., w_m) = \mathbf{n}(f(w_1), ..., f(w_m))$. The thresholding function $f : \mathbb{R} \rightarrow \mathbb{R}_+$ is defined as:

$$f(x) = \begin{cases} 0, & \text{for } x < \epsilon_1 \\ -\frac{\epsilon_1+\epsilon_2}{(\epsilon_2-\epsilon_1)^3}(x - \epsilon_1)^3 + \frac{\epsilon_1+2\epsilon_2}{(\epsilon_2-\epsilon_1)^2}(x - \epsilon_1)^2, & \text{for } \epsilon_1 \leq x \leq \epsilon_2 \\ x, & \text{for } \epsilon_2 < x \leq 1, \end{cases} \tag{20}$$

designed to have a continuous derivative. $f(\cdot)$ is applied elementwise; only weights less than ϵ_2 are modified. We choose $\epsilon_1 = 0, \epsilon_2 = 10^{-4}$ so most weights are unaffected.

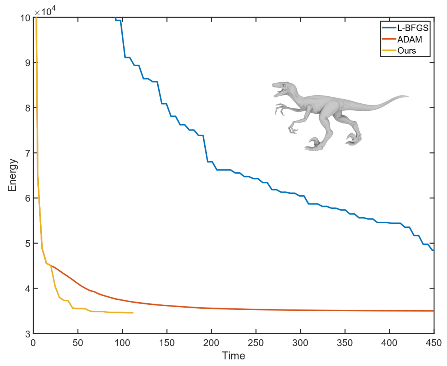


Fig. 2. Comparison of convergence speeds using different algorithms on a typical example. The curve for our method coincides with ADAM until resetting the momentum. ADAM converges much faster than L-BFGS (history size set to 10) but has a long tail of slow convergence. Our customized ADAM removes this long tail and converges quickly.

The normalization function $\mathbf{n} : \mathbb{R}_+^m \to \mathbb{P}^m$ is:

$$\mathbf{n}_i(w_1, ..., w_m) = \frac{w_i}{w_1 + w_2 ... + w_m}. \tag{21}$$

where $1 \leq i, j \leq m$.

We can compute our weights in an "end-to-end" fashion: Our objective function becomes $E(\mathbf{p}(\mathbf{W}))$, which directly measures the smoothness of the projected weights. The projection $\mathbf{p}$ is fully differentiable, so the gradients can be easily back-propagated.

6 OPTIMIZATION

Having explained our model and how it can be differentiated, our next task is to provide efficient unconstrained optimization techniques for obtaining weights in practice.

6.1 A Customized ADAM Optimizer

Since we can evaluate the gradient $\frac{\partial E}{\partial \theta}$, in principle we can apply any gradient-based algorithm. In practice, we find that using an appropriate optimizer can greatly reduce the number of iterations required. Specifically, we propose a customized ADAM solver, which allows convergence within as few as tens of gradient steps, contributing to the efficiency of our method.

ADAM [Kingma and Ba 2014] is a variant of gradient descent that incorporates a momentum term normalized by its second moment. It maintains adaptive estimates of the normalized momentum for each optimized parameter and is extremely simple to implement. Despite being a default optimizer in deep learning and stochastic gradient descent, ADAM is not a typical choice in geometry processing. For our problem, we hypothesize ADAM is particularly suitable due to its simplicity and capacity to handle parameters whose scales vary substantially, which is the case for the anisotropy tensor.

As a small customization to ADAM, we add a restarting strategy that resets the accumulated momentum estimates to zero every T iterations; we find $T \geq 4$ typically works well. We choose the base learning rate of 0.1 in our experiments and decay the learning rate by half in every T iterations. This simple modification considerably accelerates the convergence of ADAM for our problem.

Table 1. Timing (in seconds) of symbolic & numerical factorization and back substitution per column. The statistics suggest that reusing the symbolic factorization can lead to an order of magnitude speedup for solving linear systems.

Mesh	#Vertices	Symbolic	Numerical	Back Sub.
Dragon	1187670	9.37	0.602	0.015
Bunny	89947	2.05	0.410	0.006
Beast	106158	3.09	0.16	0.0037

Figure 2 shows typical convergence curves for our problem. L-BFGS converges slowly, typical requiring hundreds to thousands iterations. Vanilla ADAM is extremely effective in the first few iterations but starts to progress very slowly. It has a long tail of convergence which still needs hundreds of iterations. We find that, by resetting the momentum estimates, ADAM makes rapid progresses again. We simply use a fixed number of iterations $k = 10, 20$, which works well for examples in the paper. It is possible to come up with an adaptive stop criteria based on e.g. the gradient norm.

A second advantage of our ADAM-based optimizer is that it does not need the expensive line search step required by L-BFGS and variants. For our problem, the overhead of line search is even more costly than solving linear systems. So, our optimizer not only requires fewer iterations, but also is cheaper per iteration.

6.2 Symbolic Pre-Factorization

Solving the Laplacian system in (17) is the major computational cost in each gradient step of our method. Fortunately, using an appropriate implementation with symbolic pre-factorization effectively makes it 10× faster to solve a sparse linear system.

Since the sparsity pattern of (a sub-block of) the discrete anisotropic Laplacian, $\mathbf{S}^\top\mathbf{G}^\top\mathbf{A}\mathbf{G}\mathbf{S}$, is fixed, we can perform symbolic factorization once. In each iteration we reuse the symbolic factorization, only performing one numerical factorization and two back-substitutions (for weights computing and gradient evaluation). This leads to significant speedup: In our implementation, symbolically factorizing the cotangent Laplacian for a mesh with 122902 vertices takes 2.16 seconds, and the subsequent numerical factorization takes only 0.20 seconds, using the CHOLMOD module in SuiteSparse [Davis et al. 2015]; see Table 1 for more examples.

7 EVALUATION

In this section, we compare our method with state-of-the-art algorithms in terms of timing and weight quality. Experiments confirm that our method enjoys all the desiderata of previous weights and is much faster.

7.1 Baseline

For clarity, we refer to the original version of bounded biharmonic weights solved in (6) as BBW. A fast approximation employed by Jacobson et al. [2011] is to solve for each $w_j(\mathbf{x})$ individually without the partition of unity constraint and then re-normalize the weights *a posteriori*; it has been reported that the resulting weights produce visually indistinguishable deformations and that the energy gap with

Table 2. Comparison of energies for weights obtained by different methods. Energies have been normalized by that of BBWA. #Ele is the number of triangles/tetrahedra. #Hdl is the number of control handles. "*" indicates missing data points when the Mosek solver used by previous methods fails. Numerical factorization is responsible for only a small fraction of the cost per iteration, suggesting a large room for further improvement.

Example			Smoothness Energy				Time (Sec.)			
Mesh	#Hdl.	#Ele.	BBWA	MBBW	Ours, $k=20$	$k=10$	BBWA	MBBW	Ours ($k=10$)	num. fact.
Beast	15	443442	1	1.001	0.997	1.016	707.8	840.6	12.6	0.16
Bunny	14	531392	1	0.997	1.001	1.026	2430.6	11111.3	18.0	0.41
Raptor	15	367966	1	1.002	0.992	1.051	640.1	720.3	14.9	0.11
Elephant	17	516858	1	0.996	0.987	1.008	1092.3	2379.5	14.4	0.16
Dragon	17	1187670	1	*	0.992	1.023	5022.1	*	54.8	0.60
Image	27	6952	1	0.995	0.984	1.005	7.78	17.70	0.18	0.0009
Brick	2	78529	1	1.000	0.981	1.084	17.8	53.7	1.21	0.03
Tibiman	16	84125	1	0.991	0.986	0.995	91.9	274.1	1.80	0.025

the full BBW is often negligible. We refer to this fast approximation as BBWA.

BBW can be two orders of magnitude slower to compute than BBWA due to the partition of unity constraint, so we primarily compare with BBWA. Note BBWA no longer minimizes the biharmonic energy over all possible the non-negative weights due to renormalization, so our weights can have a slightly lower biharmonic energy for some examples.

We refer to the monotonic bounded biharmonic weights [Jacobson et al. 2012b] as MBBW. MBBW also uses the fast approximation that first solves for each individual weight function without the partition of unity constraint and re-normalizes afterwards. After re-normalization, these weights are not guaranteed to be free from local extrema; in practice, however, this theoretical concern does not appear to be an issue.

7.2 Performance

Timing. The major computational cost of our method is solving tens of linear systems with the same sparsity pattern. Thanks to the efficient optimizer (§6.1) and sparse linear solver (§6.2), timings of our method are comparable to linear problems even though our problem is nonlinear. This is substantially cheaper than the conic programming and active set solvers used in previous methods.

Table 2 reports the performance of our method and variants of BBW in details. As shown in the table, tetrahedralizing a fine mesh can easily lead to a mesh with hundreds of thousands of vertices and tetrahedra, on which the computation time of BBW and variants is very long. As an example, on the Dragon shape, variants of BBW take few hours; in contrast, each iteration of our method currently takes only few seconds, out of which one numerical factorization takes only 0.60 seconds. Our method usually obtains weights with an equal or lower energy than that of BBWA in 10 to 20 iterations.

Although our implementation is already orders of magnitude faster than previous methods, our method still has a large room for further improvement. Take the Beast shape, for example: in each iteration, numerical factorization ($0.16s$) and back substitution ($0.055s$) take only a small fraction out of the total time per iteration ($1.26s$). The majority of execution time is spent on simple linear-algebraic operations such as computing the weights gradients $\mathbf{GW}$—"local computations" per triangle/tetrahedron. These parts are currently straightforwardly implemented in CPU but are easily parallelizable. Implementing these parts on the GPU and/or as shaders likely will lead to further speedups, which we leave for future work.

7.3 Quality

Our method produces weights that are visually similar to previous methods. Figure 3 visualizes weights computed on a brick using different methods. These weights look almost identical. In fact the largest pointwise difference between our weights and BBWA & MBBW is only 0.0039 & 0.0038, resp., and our weights have a slightly lower energy. This example represents a scenario in which our framework, in theory, should be able to reproduce previous weights exactly (see §4.3).

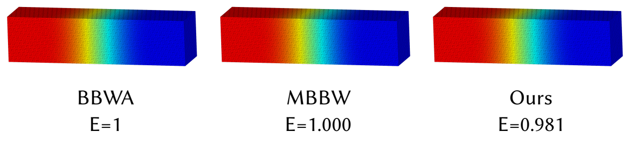


Fig. 3. Weights computed on a brick shape.

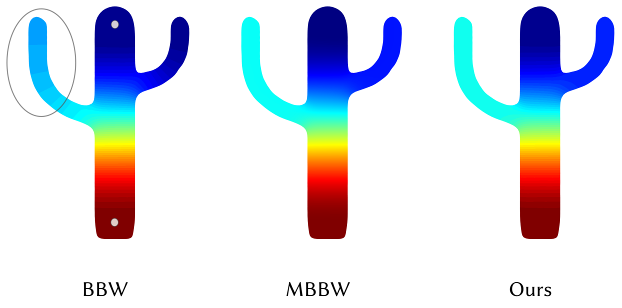


Fig. 4. BBW, MBBW, and our weights, computed on a cactus shape with two handles placed at its top and bottom. BBW has local extrema at the left arm of the cactus, which is not the case for both MBBW and ours.

In general, the visual appearance of our weights is similar but not always identical to previous methods. As an extreme example, Figure 5 shows the weights computed on an image editing setup

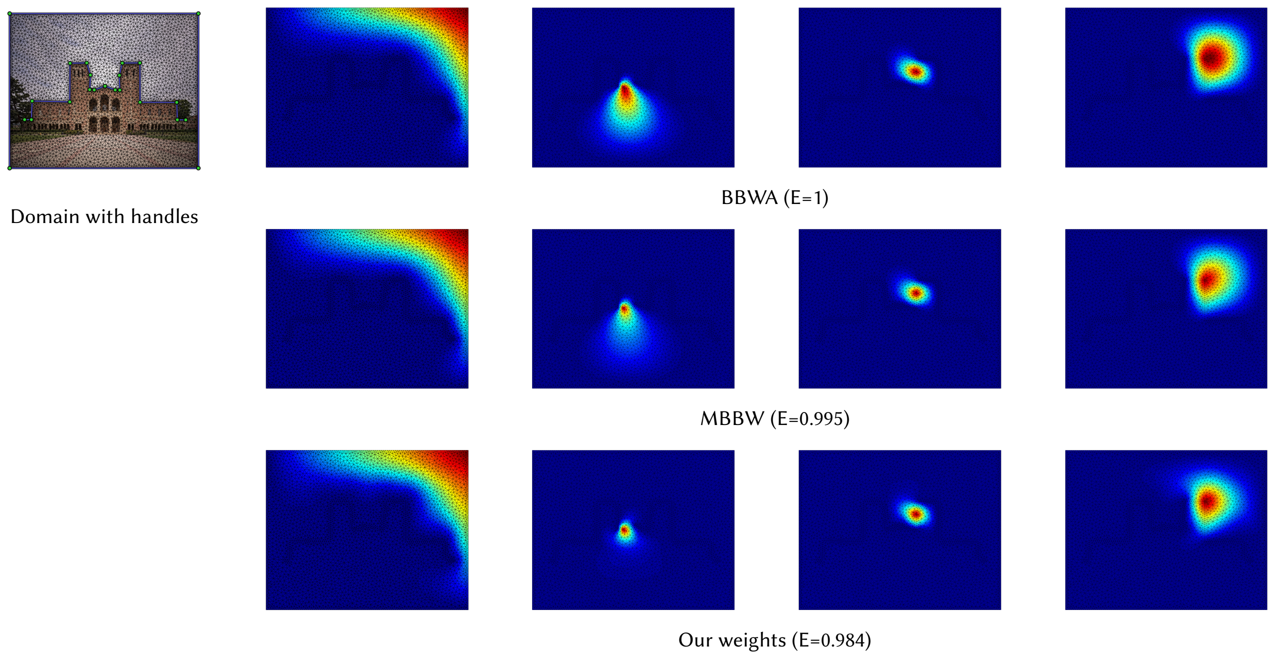


Fig. 5. Visualization of BBWA, BBWM, and our weights, computed on an image editing setup. Despite the presence of a large number of handles, our weights have a smoothness energy similar to previous methods.

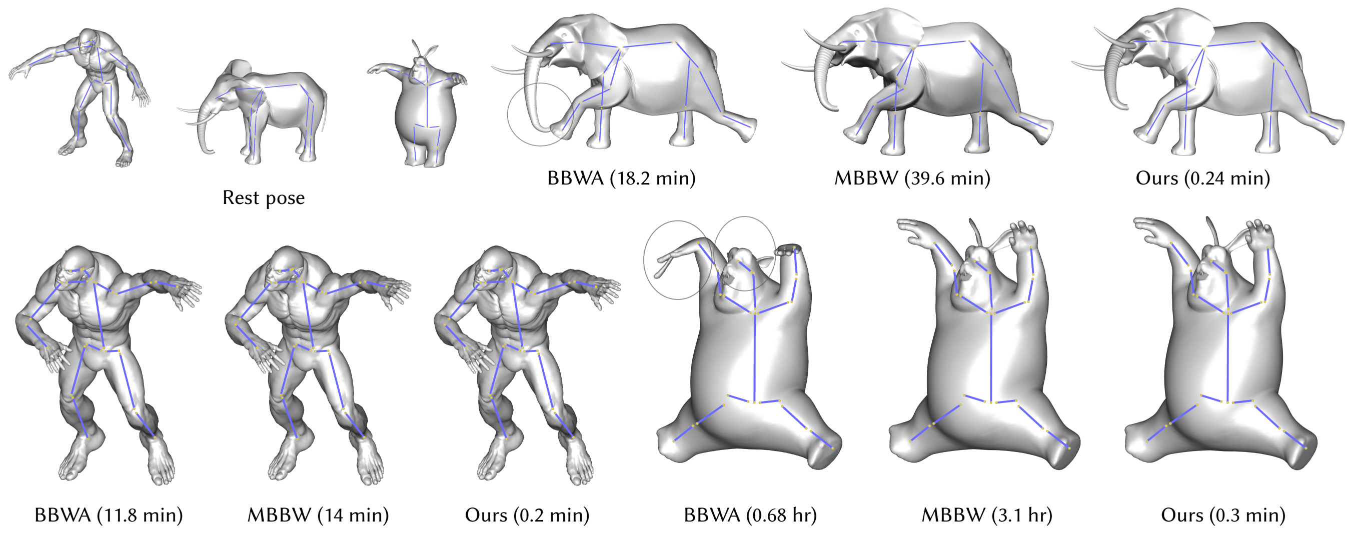


Fig. 6. Linear blend skinning using different weights. Due to the lack of monotonicity, the BBWA weights of the Bunny's arm are not exactly 1 at the hand. Transforming the shoulder affects the hand, leading to noticeable artifacts. This is not the case for MBBW and our weights. Similar artifacts are visible at the Elephant's trunk using BBWA.

with tens of handles. Figures 7 and 8 visualize our weights on common examples encountered in skinning animation. Our method consistently obtains weights with a similar visual appearance. For an intuitive illustration, Figure 1 and 9 show color interpolation using different weights.

The differences in weights are usually subtle, making it hard to tell which one is better. Nonetheless, the smoothness of the weight functions, as measured by the biharmonic energy, is our ultimate criterion for weight quality. Comparisons using this measure are summarized in Table 2. We report the smoothness energies of BBWA,

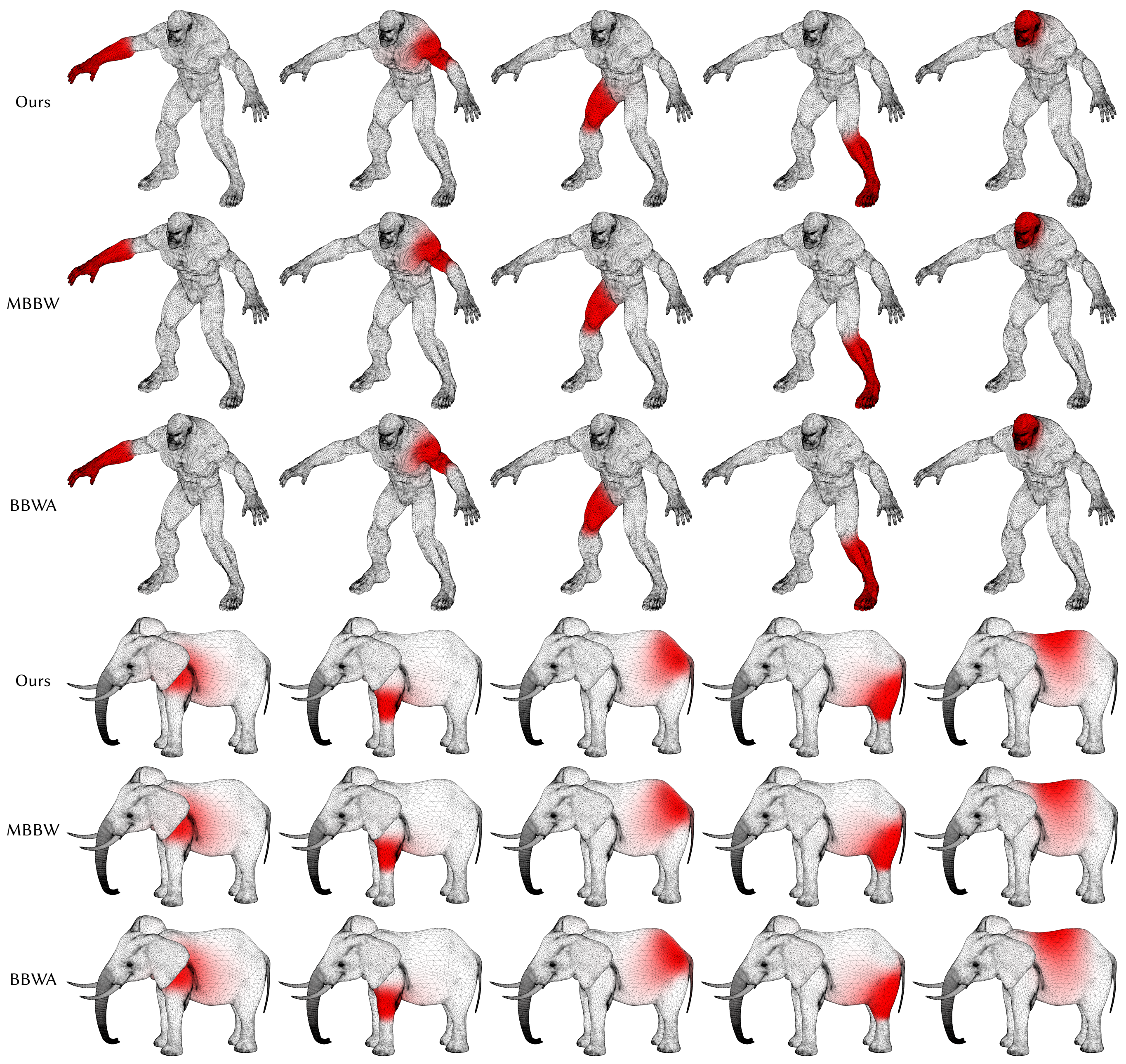


Fig. 7. Comparison of weights, computed on the Beast and Elephant shapes.

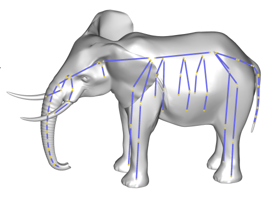
MBBW, and ours, divided by that of BBWA for ease of evaluation. It should not be a surprise that some relative energies of BBWA are less than 1: Recall that the energies for BBW, BBWA, MBBW, and ours, when fully optimized, should satisfy the inequalities

$$E_{BBW} \leq E_{BBWA} \leq E_{MBBW},$$
$$E_{BBW} \leq E_{ours}.$$

In practice, these relations can be mildly violated due to varieties in the stopping criteria.

Table 2 confirms that the optimal weights in our framework of quasi-harmonic weights always have a smoothness energy similar to previous methods.

Multiple control handles. Although the theoretical analysis in §4.3 is directly applicable only to cases with a small number of control handles, we observe our model does not overly restrict the



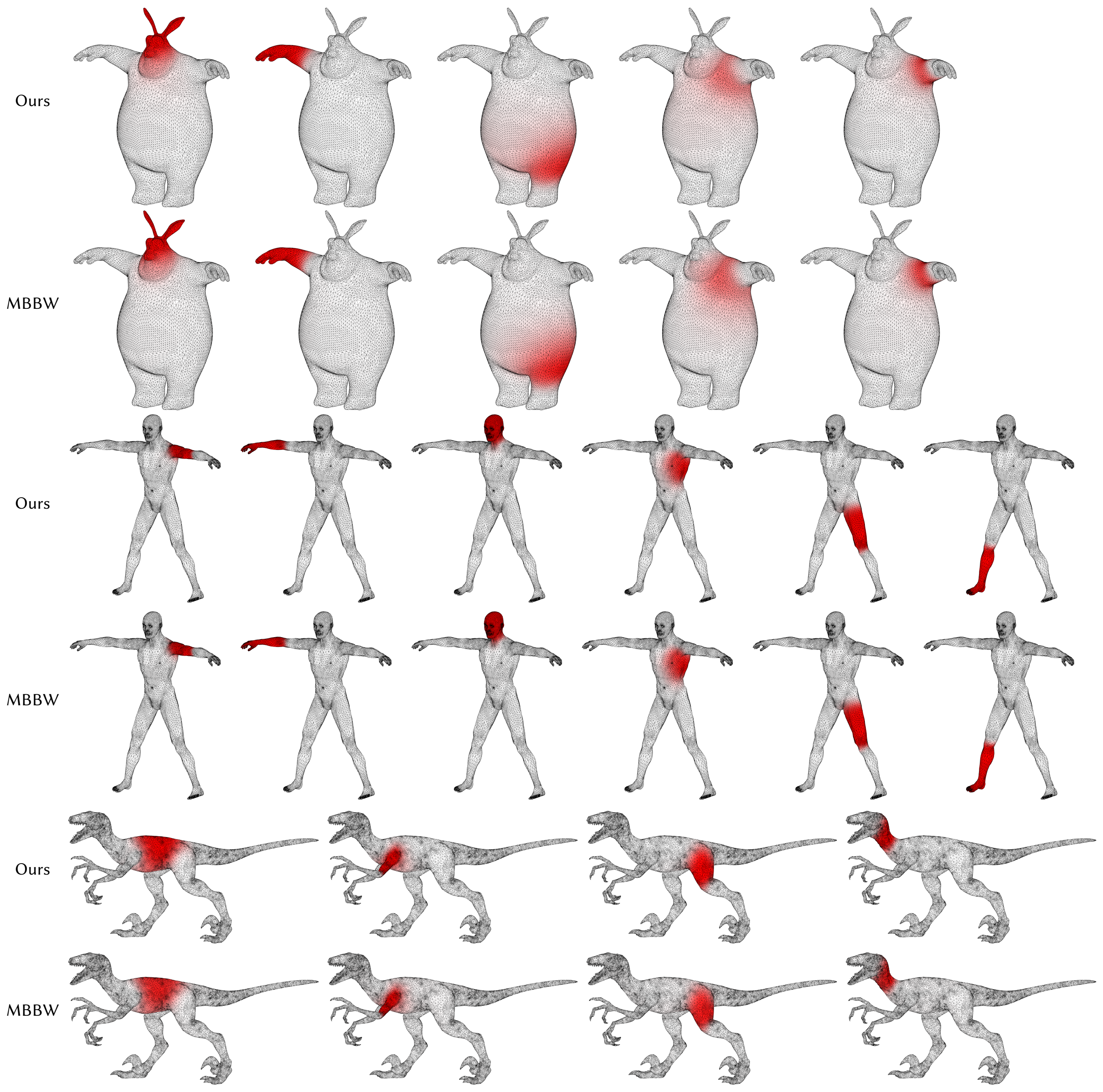


Fig. 8. Visualizations of our weights computed on the Bunny, Tibiman, and Raptor shapes.

space of weights, regardless of the number of handles. In particular, E_{ours} can often be lower than E_{BBWA} and/or E_{MBBW}, even for meshes with a large number of control handles. All examples in the paper already have a reasonable number of control handles. As an extreme example, in the elephant model when the number of control handles is increased to 54 as shown in the embedded Figure, we have $E_{ours}/E_{BBWA} = 0.9915$—our method still yields slightly smoother weights.

No local extrema. The visual appearances of BBW and MBBW are similar in many cases, since the no-local-extrema constraint is often not active. When the constraint is active and does make a difference, our weights have a closer visual appearance to MBBW

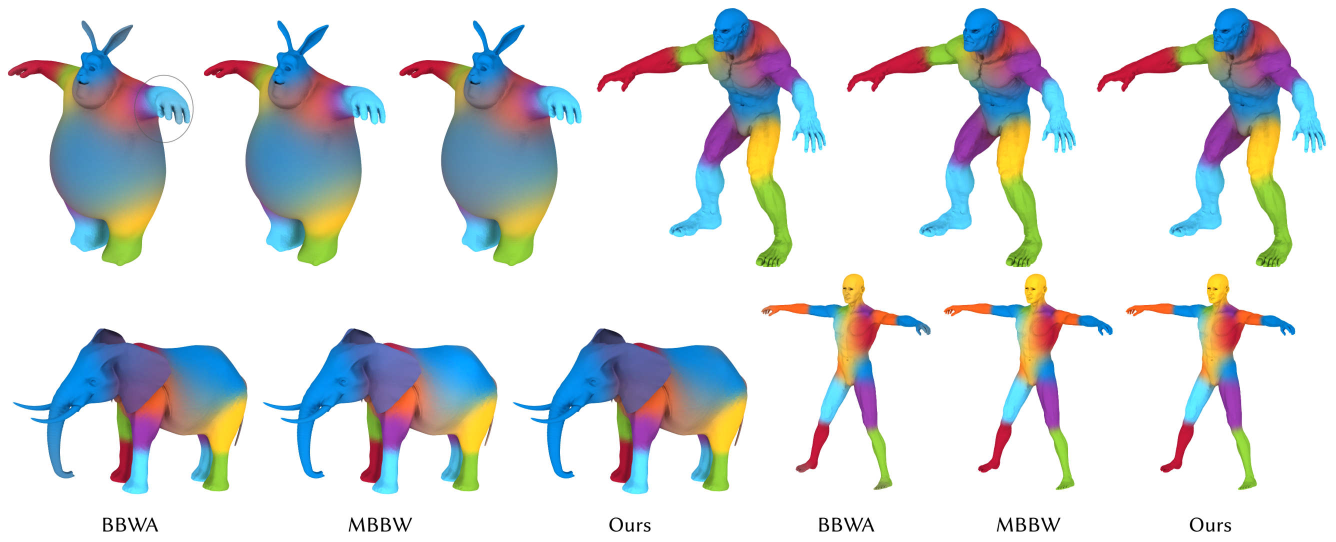


Fig. 9. Color interpolation using different weights. Usually a similar color blending is obtained. An exception is the Bunny—BBWA has a local extrema at the hand.

than BBW. Figure 4 shows an example. Our weights are almost identical to MBBW, both of which have no local extrema away from handles. BBW, however, has a local maximum at the left arm of the cactus. Figure 6 shows LBS using different weights. BBWA, due to the lack of monotonicity, has weights that are not exactly 1 at the Bunny's hand and Elephant's trunk. This results in artifacts for large transformations. Both our method and MBBW remove this issue.

Since our method minimizes the same energy as BBW and variants, the weights—and thus the resulting deformation—are often similar to each other and sometimes even identical. The difference between MBBW and ours, only reflecting the nuance in handling the monotonicity constraints, is usually negligible. In Figure 10, we show a scenario where the difference is the most visible. The Bunny shape is deformed using linear blend skinning with both weights. Note the difference at the Bunny's hip and leg.

7.4 Tensor Field Visualization

In Figure 11, we visualize patterns of the anisotropy tensor $\mathbf{A}(\mathbf{x})$ for the optimal weights. We show the first principal directions and condition numbers. The principal directions form a vector field that usually starts from one handle and ends at another handle or the domain boundary. Recall that our method penalizes gradients along the principal direction, by a factor of the condition number.

7.5 Implementation Details

We use SuiteSparse [Davis et al. 2015] for sparse linear algebra. For dense linear algebra we use Armadillo [Sanderson 2010] and TensorFlow [Abadi et al. 2016] (CPU only). Our results are collected on a Linux machine with an Intel i9-7900X CPU. For comparison with BBW and variants, we use the source code provided by the authors, using the fastest option with Mosek and conic programming solvers.

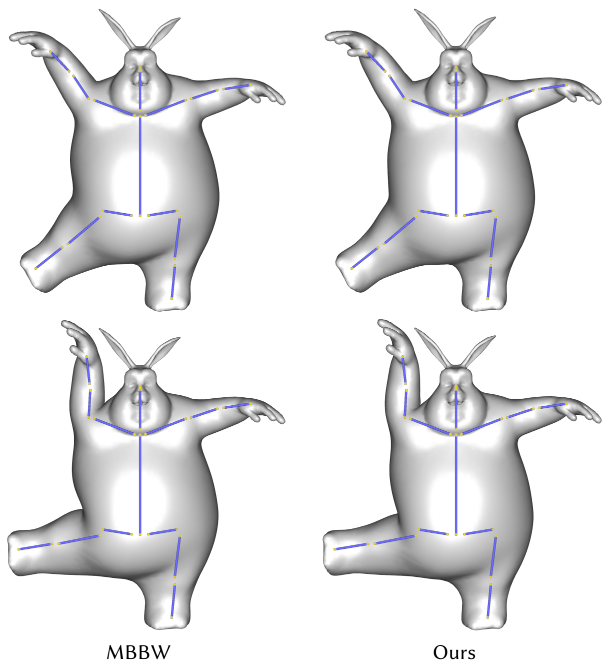


Fig. 10. Linear blend skinning using MBBW and our weights at two poses.

In our experiments, we initialize the anisotropy tensors as identity matrices multiplied by triangle areas (tetrahedron volumes), and thus our initial weights are harmonic weights, the same as the initial guess used by BBW and MBBW for fair comparison. In practice, we find initializing the anisotropy as the inverse quadratic distance to the nearest control handle can improve the convergence speed,

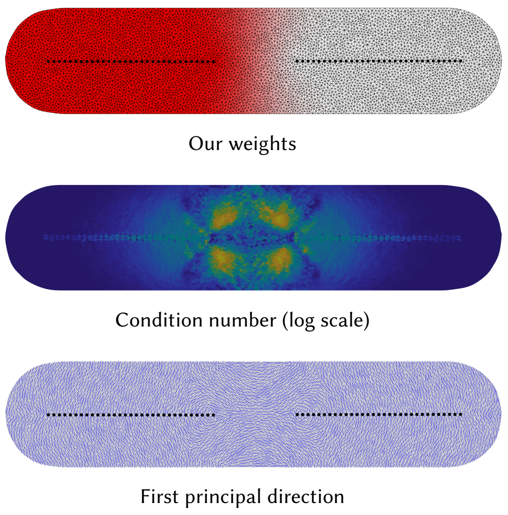


Fig. 11. Visualization of our weights, as well as the condition numbers and first principal directions of the optimal tensor field.

which can be easily obtained using, e.g., the heat method [Crane et al. 2013].

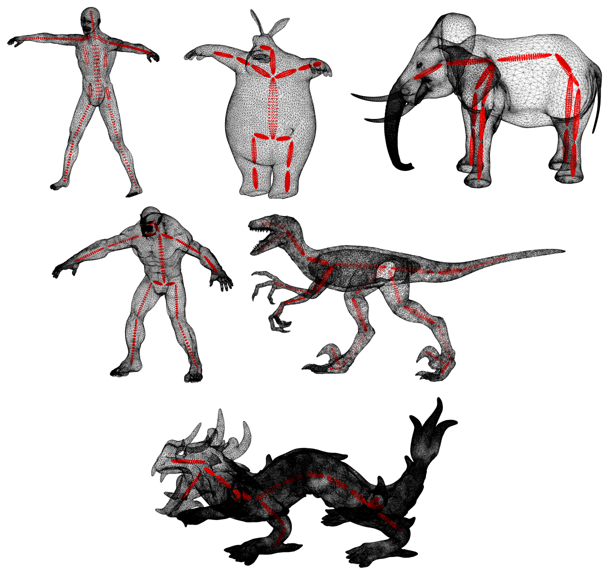


Fig. 12. Visualization of meshes and handles.

We use Triangle [Shewchuk 1996] and Tetgen [Si 2015] to generate triangle and tetrahedral meshes, respectively. To solve anisotropic Laplacian systems accurately, our method prefers adaptive meshing that has a higher resolution around control handles, since the scale of the anisotropy tensor varies a lot around handles. Fortunately, this is automatically the case when using meshing softwares Triangle and Tetgen. Figure 12 shows the mesh and sampled points on the bone handles that are provided to Tetgen: A dense enough sampling on the bone, as shown in the figure, implicitly leads to meshes with a higher resolution around them.

7.6 Limitations

Our method relies on the ability to faithfully solve anisotropic Laplace equations. Unlike the optimization approach [Jacobson et al. 2011], which guarantees nonnegative weights for arbitrary meshes, our method should avoid using highly irregular meshes with many obtuse and sharp triangles/tetrahedra, on which the discrete maximum principle fails more easily.

The method proposed by Jacobson et al. [2011] leads to a convex optimization problem. It is not clear if our formulation (11) is convex or if it can be converted to a convex problem. For future work, it would be interesting to study this variational problem theoretically.

8 CONCLUSION AND FUTURE WORK

Our work provides a new approach for the problem of variational weight computation. Our formulation is theoretically justified, transforming the problem into an unconstrained optimization problem. Combined with a customized ADAM optimizer and a fast sparse solver, our method is much faster than past alternatives, allowing weight computation in near real-time. Fast weight computation opens up many opportunities for future work, such as the joint optimization of handle placements and weights.

While our method is designed for a concrete problem of variational weight computation, the approach potentially can be applied to a broad range of problems. Our approach can be viewed as searching for the optimal surface metric or geometric operator under which certain loss function or variational regularizer is optimized; similar techniques may be applied to geometric deep learning and computational design.

We choose a quadratic smoothness energy for simplicity and ease of comparison, but theoretically our method can use any nonlinear energy as long as it is differentiable. It would be interesting to explore other smoothness measures. For practical purposes, our method also can be further accelerated and improved: For example, it is possible to replace the direct linear solver with iterative solvers, similar to how heat diffusion weights are implemented and made available in commercial software. For further speedup and simplification, it would be interesting to directly design a tensor field that yields high quality weights.

ACKNOWLEDGMENTS

We thank Oded Stein and Paul Zhang for proofreading. The MIT Geometric Data Processing group acknowledges the generous support of Army Research Office grant W911NF2010168, of Air Force Office of Scientific Research award FA9550-19-1-031, of National Science Foundation grant IIS-1838071, 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, from an MIT.nano Immersion Lab/NCSOFT Gaming Program seed grant, and from the Skoltech–MIT Next Generation Program.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

Marc Alexa. 2002. Linear combination of transformations. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 380–387.

Norman I Badler and Mary Ann Morris. 1982. Modelling flexible articulated objects. In *Proc. Computer Graphics' 82, Online Conf.* 305–314.

Seungbae Bang and Sung-Hee Lee. 2018. Spline Interface for Intuitive Skinning Weight Editing. *ACM Transactions on Graphics (TOG)* 37, 5 (2018), 174.

Sumukh Bansal and Aditya Tatu. 2019. Affine interpolation in a lie group framework. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 71.

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

Fred L. Bookstein. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on pattern analysis and machine intelligence* 11, 6 (1989), 567–585.

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

Xiang Chen, Changxi Zheng, and Kun Zhou. 2016. Example-based subspace stress analysis for interactive shape design. *IEEE transactions on visualization and computer graphics* 23, 10 (2016), 2314–2327.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in heat: A new approach to computing distance based on heat flow. *Transactions on Graphics* 32, 5 (2013), 152.

Timothy A Davis et al. 2015. SuiteSparse: A suite of sparse matrix software. *URL http://faculty. cse. tamu. edu/davis/suitesparse. html* (2015).

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.

Jérôme Droniou and Christophe Le Potier. 2011. Construction and convergence study of schemes preserving the elliptic local maximum principle. *SIAM J. Numer. Anal.* 49, 2 (2011), 459–490.

Michael S Floater. 2003. Mean value coordinates. *Computer aided geometric design* 20, 1 (2003), 19–27.

David Gilbarg and Neil S Trudinger. 2015. *Elliptic partial differential equations of second order*. springer.

Philipp Herholz, Felix Haase, and Marc Alexa. 2017. Diffusion diagrams: Voronoi cells and centroids from diffusion. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 163–175.

Michael Hinze and Tran Nhan Tam Quyen. 2016. Matrix coefficient identification in an elliptic equation with the convex energy functional method. *Inverse problems* 32, 8 (2016), 085007.

Kai Hormann and Michael S Floater. 2006. Mean value coordinates for arbitrary planar polygons. *ACM Transactions on Graphics (TOG)* 25, 4 (2006), 1424–1441.

Kai Hormann and Natarajan Sukumar. 2008. Maximum entropy coordinates for arbitrary polytopes. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1513–1520.

Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. 2012a. Fast automatic skinning transformations. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–10.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78–1.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. 2014. Skinning: real-time shape deformation. In *ACM SIGGRAPH 2014 Courses*. ACM, 24.

Alec Jacobson, Elif Tosun, Olga Sorkine, and Denis Zorin. 2010. Mixed finite elements for variational surface modeling. In *Computer Graphics Forum*, Vol. 29. 1565–1574.

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

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

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.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 561–566.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2008. Geometric skinning with approximate dual quaternion blending. *ACM Transactions on Graphics (TOG)* 27, 4 (2008), 105.

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

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

Andreas Kirsch. 2011. *An introduction to the mathematical theory of inverse problems*. Vol. 120. Springer Science & Business Media.

Binh Huy Le and Zhigang Deng. 2012. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 199.

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 84.

Binh Huy Le and Jessica K Hodgins. 2016. Real-time skeletal skinning with optimized centers of rotation. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 37.

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

Xian-Ying Li and Shi-Min Hu. 2012. Poisson coordinates. *IEEE Transactions on visualization and computer graphics* 19, 2 (2012), 344–352.

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green coordinates. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 78.

Yaron Lipman, Raif M Rustamov, and Thomas A Funkhouser. 2010. Biharmonic distance. *Transactions on Graphics* 29, 3 (2010), 27.

Richard Liska and Mikhail Shashkov. 2008. Enforcing the discrete maximum principle for linear finite element solutions of second-order elliptic problems. *Commun. Comput. Phys.* 3, 4 (2008), 852–877.

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), 114.

Changna Lu, Weizhang Huang, and Jianxian Qiu. 2014. Maximum principle in linear finite element approximations of anisotropic diffusion–convection–reaction problems. *Numer. Math.* 127, 3 (2014), 515–537.

Nadia Magnenat-Thalmann, Richard Laperrire, and Daniel Thalmann. 1988. Joint-dependent local deformations for hand animation and object grasping. In *In Proceedings on Graphics interfaceâĂŹ88*. Citeseer.

Viktoras Makauskas. 2013. ngSkinTools. https://www.ngskintools.com

Jesús R Nieto and Antonio Susín. 2013. Cage based deformations: a survey. In *Deformation models*. Springer, 75–99.

Gerard R Richter. 1981. An inverse problem for the steady state diffusion equation. *SIAM J. Appl. Math.* 41, 2 (1981), 210–221.

Conrad Sanderson. 2010. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. (2010).

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Workshop on Applied Computational Geometry*. Springer, 203–222.

Hang Si. 2015. TetGen, a Delaunay-based quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software (TOMS)* 41, 2 (2015), 1–36.

Robin Sibson and Vic Barnett. 1981. Interpreting multivariate data. *A brief description of natural neighbor interpolation* (1981), 21–36.

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), 23.

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. Citeseer.

Kevin Wampler. 2016. Fast and reliable example-based mesh ik for stylized deformations. *ACM Transactions on Graphics (TOG)* 35, 6 (2016), 235.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 57.

Max Wardetzky, Saurabh Mathur, Felix Kälberer, and Eitan Grinspun. 2007. Discrete Laplace operators: no free lunch. In *Symposium on Geometry processing*. 33–37.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. 2019. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 162.

Zhipei Yan and Scott Schaefer. 2019. A Family of Barycentric Coordinates for Co-Dimension 1 Manifolds with Simplicial Facets. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 75–83.

Juyong Zhang, Bailin Deng, Zishun Liu, Giuseppe Patanè, Sofien Bouaziz, Kai Hormann, and Ligang Liu. 2014. Local barycentric coordinates. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 188.

Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 912–919.