Variational Barycentric Coordinates

ANA DODIK, MIT CSAIL, USA
ODED STEIN, University of Southern California and MIT CSAIL, USA
VINCENT SITZMANN, MIT CSAIL, USA
JUSTIN SOLOMON, MIT CSAIL, USA

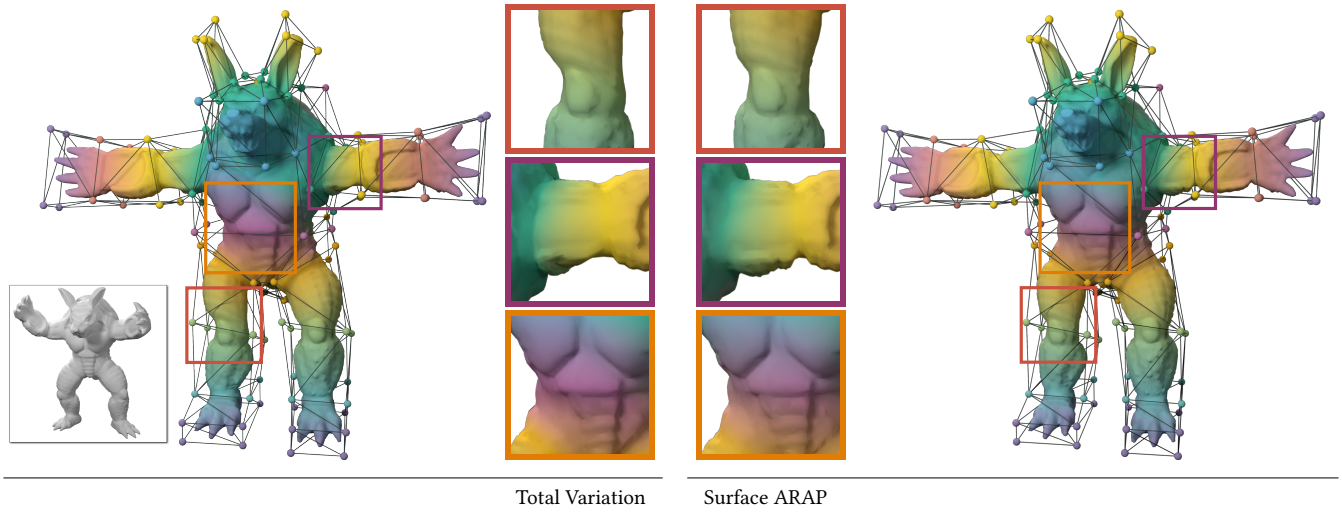

Fig. 1. Our method allows us to optimize various energies resulting in different sets of barycentric coordinates. This figure shows the result of using different sets of coordinates on the tasks of color interpolation and mesh deformation. Minimizing *total variation* (TV) results in deformations that have an undesirable rubbery appearance. However, our formulation allows us to address this issue by minimizing *deformation-aware* energies, such as the *as-rigid-as-possible* (ARAP) energy.

We propose a variational technique to optimize for generalized barycentric coordinates that offers additional control compared to existing models. Prior work represents barycentric coordinates using meshes or closed-form formulae, in practice limiting the choice of objective function. In contrast, we directly parameterize the continuous function that maps any coordinate in a polytope's interior to its barycentric coordinates using a neural field. This formulation is enabled by our theoretical characterization of barycentric coordinates, which allows us to construct neural fields that parameterize the entire function class of valid coordinates. We demonstrate the flexibility of our model using a variety of objective functions, including multiple smoothness and deformation-aware energies; as a side contribution, we also present mathematically-justified means of measuring and minimizing objectives like total variation on discontinuous neural fields. We offer a practical acceleration strategy, present a thorough validation of our algorithm, and demonstrate several applications.

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

Authors' addresses: Ana Dodik, anadodik@mit.edu, MIT CSAIL, USA; Oded Stein, University of Southern California and MIT CSAIL, USA; Vincent Sitzmann, MIT CSAIL, USA; Justin Solomon, MIT CSAIL, USA.

0730-0301/2023/12-ART255
https://doi.org/10.1145/3618403

Additional Key Words and Phrases: barycentric coordinates, neural fields, geometry processing, deformation, inverse problem, partial differential equations, geometric variational problem.

ACM Reference Format:
Ana Dodik, Oded Stein, Vincent Sitzmann, and Justin Solomon. 2023. Variational Barycentric Coordinates. *ACM Trans. Graph.* 42, 6, Article 255 (December 2023), 16 pages. https://doi.org/10.1145/3618403

1 INTRODUCTION

Generalized barycentric coordinates are used to interpolate functions defined on the vertices of a polytope to its interior, such as to evaluate a function known only on the surface of a shape inside its volume. They are also often used for *cage-based deformation*, with the polytope serving as a *deformation cage*. In this setting, a user deforms a mesh by moving the vertices of a low-resolution cage that surrounds it. The positions of the transformed cage vertices are then interpolated into its interior, and in particular onto the surface of the mesh inside of it. Figure 1 demonstrates how generalized barycentric coordinates can interpolate data and deform shapes using a cage.

The barycentric coordinates associated to each point in the interior of the cage can be understood as a set of averaging weights, one per cage vertex. These weights satisfy a number of mathematical constraints: they must be non-negative, sum to one, and have a prescribed expectation. This final *reproduction constraint* distinguishes

barycentric coordinates from more general *skinning weights*; to ensure that a linear deformation of the cage vertices leads to a linear deformation field inside the cage, every point in the interior must equal the linear combination of the cage vertices with its barycentric coordinates as coefficients. Moreover, we might expect barycentric weights to be smooth and/or local, although these considerations may be understood as objectives rather than hard constraints.

Existing methods for computing barycentric coordinates typically fall into two categories. Classically, a number of barycentric coordinate functions are expressed in closed-form [Ju et al. 2005; Lipman et al. 2007] or via relatively simple algorithms. These coordinates are fast to evaluate but often add assumptions on the cage vertices (e.g., convexity) to satisfy the required properties of barycentric coordinates; moreover, they are inflexible in the sense that they each provide a single means of computing coordinates rather than allowing users to optimize for coordinates best suited for a given application. More recently, Joshi et al. [2007] and Zhang et al. [2014] pose the computation of generalized barycentric coordinates as a convex optimization problem. This approach promises global satisfaction of the constraints defining barycentric coordinate functions and suggests the possibility of optimizing for customized barycentric weights for a given application or artistic intention, but the current models rely on a mesh-based discretization and optimization technique customized to a specific smoothness objective functions.

In this paper, we introduce *variational barycentric coordinates* (VBCs), a flexible mesh-free framework that allows us to optimize for generalized barycentric coordinates given a boundary cage and a differentiable objective. VBCs are built on a simple mathematical observation, namely that all generalized barycentric coordinates can be expressed as weighted averages of simplex barycentric coordinates of all the simplices generated by connecting cage vertices. With this observation in place, we can represent the set of generalized barycentric coordinates for a given cage using the machinery of neural networks, which allow us to efficiently operate in the high-dimensional space of all possible simplices. Beyond total variation, we can then optimize using customized objectives tailored to different tasks and applications.

VBCs offer several departures from and advantages over classical generalized barycentric coordinate functions. Most importantly, our formulation is general and operates in the space of valid barycentric coordinates by construction. We offer a differentiable representation of the *entire* function class of barycentric coordinate functions, as well as some examples for the possible design choices one could make by incorporating different optimization objectives. We also offer a mathematically justified means of computing and optimizing the total variation of our model in the presence of possible discontinuities.

Furthermore, our formulation is defined in terms of the cage vertices, allowing us to support cages composed of meshes, triangle soups, or point clouds. When it is acceptable to restrict to fewer degrees of freedom at the cost of excluding some possible coordinate functions, we offer a practical modification to the algorithm that not only enforces locality but also significantly reduces the memory and compute required.

To demonstrate the benefits of our approach, we provide a thorough qualitative evaluation as well as comparisons with previous work. We additionally offer a quantitative analysis of the different performance trade-offs introduced by our approach.

Contributions. In summary, we introduce:

- a mathematical formulation that expresses generalized barycentric coordinates as convex combinations of simplex coordinates,
- a computational model that uses our formulation to constrain neural networks to the function space of barycentric coordinates,
- heuristics that help make our model practically tractable while simultaneously enforcing a notion of locality in our coordinates,
- a way of approximating and optimizing several common smoothness energies—such as total variation and the Dirichlet energy—in the context of discontinuous neural fields,
- experiments that demonstrate how our coordinates can be combined with familiar *deformation-aware* energies or used to solve inverse deformation problems, and
- a thorough evaluation and comparison that confirms the validity and practical usefulness of our model on a variety of 2D and 3D shapes.

2 RELATED WORK

2.1 Generalized Barycentric Coordinates

Many methods have been proposed to compute generalized barycentric coordinates. A complete survey is outside the scope of our discussion; we refer the reader to existing surveys [Floater 2015; Hormann and Sukumar 2017] for a comprehensive introduction. Here, we mention a few particularly relevant works to our formulation.

Barycentric coordinates go back at least as far as Möbius [Coxeter 1969, p. 217], who used them to define a coordinate system on the inside of a triangle, parametrized by the triangle vertex positions. They have since been generalized to polygons [Floater 2003] and higher dimensions [Floater et al. 2005; Ju et al. 2005].

In graphics and geometry processing, barycentric coordinates are used for interpolation and deformation. The cage polygon/polyhedron parameterizes motions of a complicated interior domain in 2D/3D [Chen et al. 2010; Deng et al. 2020; Hormann and Sukumar 2008; Huang et al. 2006; Li and Hu 2013; Lipman et al. 2007, 2008; Weber et al. 2009]. Beyond these basic applications, barycentric coordinates are a part of methods for distance computation [Rustamov et al. 2009], image registration [Weistrand and Svensson 2015], mesh generation [Gregson et al. 2011], finite elements [Wicke et al. 2007], and subdivision [Liu et al. 2020].

Later works on generalized barycentric coordinates use the geometry of the domain's interior to produce coordinates that are aware of local distances; this approach yields more natural-looking animations [Joshi et al. 2007]. Unlike early attempts to define generalized barycentric coordinate functions, many of these works pose computation of the coordinates as an optimization problem over the space of possible coordinate functions. The recent Local Barycentric Coordinates [Zhang et al. 2014] are geometry-aware barycentric coordinates obtained via minimizing an energy containing the Total Variation (TV) of the coordinate functions. This method was recently accelerated by Tao et al. [2019]. Wang et al. [2015] optimize the Laplacian energy with a modified boundary

term to produce cage-free barycentric coordinates with simple control vertices instead of control cages. Stein et al. [2018] construct cage-free barycentric coordinates using the Hessian energy with natural boundary conditions, later generalizing the method to curved surfaces [Stein et al. 2020].

In general, closed-form expressions of geometry-aware, locally supported barycentric coordinates are difficult to obtain. Anisimov et al. [2017] provide closed-form locally-supported barycentric coordinates using a Delaunay triangulation of the cage polygon, but their method does not support polyhedra in 3D.

Lastly, Floater [1997] formulates one simple choice of barycentric coordinates in the context of mesh parameterizations as a special case of the formulation presented in §4.1. Besides the different application domain, Floater's formulation does not support polytopes and is not guaranteed to satisfy the necessary properties on concave polygons (see Figure 3 for a failure case). Importantly, our method relies on a neural representation together with an optimization procedure, whereas Floater [1997] uses a closed-form formula. This is a key component of our method, as one of our main goals is to enable users to tune weights using arbitrary objectives.

2.2 Neural Networks for Interpolation and Deformation

Recent work has demonstrated the potential of treating fully-connected networks as continuous, memory-efficient representations of general functions, shape parts, objects, or scenes by mapping each coordinate to a value stored at that coordinate. These networks are commonly referred to as *neural fields* [Xie et al. 2022]. Our method parametrizes barycentric coordinates using neural fields.

In geometry processing, neural fields have been used to parameterize distributions over shape boundary vertices for Linear Blend Skinning (LBS) [Jeruzalski et al. 2020]; note that LBS weights are *not* the same as barycentric coordinates, as they do not satisfy the reproduction property. Neural fields can also be used to forgo mesh discretizations entirely and perform geometry processing tasks on the field directly [Yang et al. 2021]. Yifan et al. [2020] use a neural network to learn cage-based shape deformations. Neural representations are also popular to solve physics problems formulated as PDEs on a variety of geometries [Li et al. 2021; Raissi et al. 2019; Rao et al. 2021; Sukumar and Srivastava 2022]. Deforming Neural Radiance Fields (NeRFs) is a popular use-case for cage-based deformation [Peng et al. 2022; Yuan et al. 2022], with applications such as modeling dynamic human bodies [Peng et al. 2021] or reconstructions of scenes with deformation [Park et al. 2021].

Beyond neural fields, Tan et al. [2018] use variational autoencoders to model mesh deformation. Luo et al. [2020] model linear elasticity using neural networks. Jiang et al. [2020] learn deformations by learning flows between shapes. Chentanez et al. [2020] model deformations on triangle meshes using convolutional neural networks. Aigerman et al. [2022] learn intrinsic mappings between meshes using neural networks.

3 PRELIMINARIES

We will begin by introducing the broad mathematical definition of generalized barycentric coordinates for an arbitrary polytope cage $\mathcal{P} \subset \mathbb{R}^d$ with K boundary vertices $\mathcal{V}(\mathcal{P}) = \{\boldsymbol{v}_i \mid 1 \le i \le K\}$. We also include a brief review of the special case of barycentric coordinates of triangles and tetrahedra in this section, as we will be using them as a building block for our model.

We write *barycentric coordinate functions* for a polytope $\mathcal{P}$ as functions $\alpha_i : \mathcal{P} \to \mathbb{R}_+$ and denote by $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_K]^\top$ the corresponding vector-valued barycentric coordinate function. For $\boldsymbol{\alpha}$ to be useful as interpolation weights, the definition of barycentric coordinates includes a number of constraints that the functions must satisfy [Floater 2015; Floater et al. 2006]. In particular, a function $\boldsymbol{\alpha}$ is said to be *valid* if it fulfills the barycentric coordinate constraints for all $\boldsymbol{x} \in \mathcal{P}$:

- Non-negativity. $\alpha_i(\boldsymbol{x}) \ge 0$;
- Partition of unity. $\sum_i \alpha_i(\boldsymbol{x}) = 1$;
- Reproduction. $\sum_i \boldsymbol{v}_i \alpha_i(\boldsymbol{x}) = \boldsymbol{x}$; and
- Lagrange property $\alpha_i(\boldsymbol{v}_j) = \delta_{ij}$,

where δ_{ij} is the Kronecker delta. Previous work has additionally identified *locality*—the idea that coordinates should be non-zero only within a small neighborhood of their vertices—as an optional desirable property for barycentric coordinates [Zhang et al. 2014].

As an example use case for generalized barycentric coordinates, suppose we deform the polytope vertices $\boldsymbol{v}_i$ to new positions $\boldsymbol{v}'_i$. We can extend this deformation to interior points $\boldsymbol{x}$ via the following map:

$$\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x};\ \mathcal{P}') = \sum_{i=1}^{K} \alpha_i(\boldsymbol{x}) \boldsymbol{v}'_i. \tag{1}$$

Thanks to the Lagrange property, we have $\varphi_{\boldsymbol{\alpha}}(\boldsymbol{v}_i) = \boldsymbol{v}'_i$ for all boundary vertices i. Moreover, thanks to the reproduction property, when $\boldsymbol{v}_i = \boldsymbol{v}'_i$ for all boundary vertices i, the map becomes the identity: $\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \boldsymbol{x}$.

Simplex barycentric coordinates. For triangles and tetrahedra, this definition leads to unique barycentric coordinates. In particular, for a triangle in the plane, $\mathcal{T} \subset \mathbb{R}^2$ with vertices $\mathcal{V}(\mathcal{T}) = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\}$, and a point $\boldsymbol{x} \in \mathcal{P}$, the triangle barycentric coordinates are the solution to the linear system

$$\begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \\ 1 & 1 & 1 \end{bmatrix} \boldsymbol{\alpha}(\boldsymbol{x}; \mathcal{T}) = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}. \tag{2}$$

As long as $\boldsymbol{x}$ is inside the triangle and the $\boldsymbol{v}_i$ are affinely independent, the solution is unique and satisfies the desired constraints.

In the general case of an arbitrary cage, the linear system above usually is underdetermined, and therefore, the barycentric coordinate functions are not unique. Instead, they lay somewhere in the feasible set defined by the constraints. The feasible set $\mathcal{A}$ of generalized barycentric coordinates at a point $\boldsymbol{x}$ is a $(K-d-1)$-dimensional simplex. To ensure that our computed coordinates always lie in $\mathcal{A}$, we need to optimize in the constrained space of feasible coordinate functions rather than the larger space of all possible smooth functions.

4 VARIATIONAL BARYCENTRIC COORDINATES

We are now ready to introduce our formulation of generalized barycentric coordinates. We begin by introducing the optimization problem at the heart of our formulation in §4.1 and then offer a computational representation of the function space of valid barycentric coordinates in §4.2. Additionally, we show that not only are

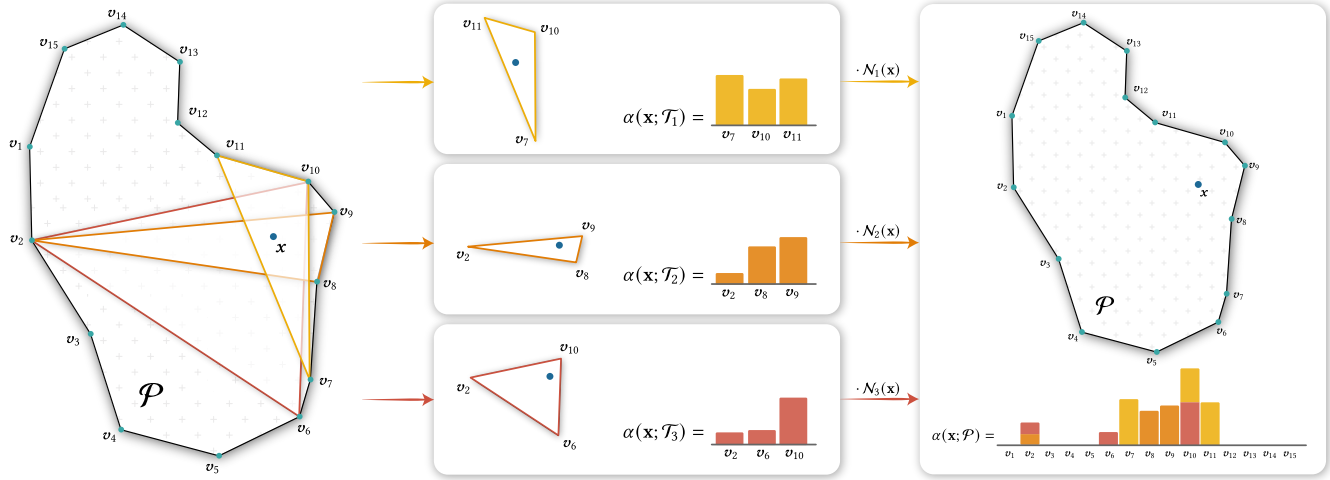

Fig. 2. A 2D illustration of our model. We connect triplets of polygon vertices into non-degenerate triangles. For each triangle that contains $\boldsymbol{x}$ in its interior, we compute the triangle barycentric coordinates of $\boldsymbol{x}$. We define the polygon barycentric coordinates as the convex combination of triangle barycentric coordinates, with coefficients given by a differentiable parametric function $\mathcal{N}(\boldsymbol{x})$.

all of the coordinates produced by our method within the feasible set (Proposition 4.2), but also our method allows us to represent *all possible* valid barycentric coordinates for any cage (Proposition 4.1).

4.1 Model

We define variational barycentric coordinates as the minimizer of an energy functional F under the relevant constraints

$$\min_{\alpha_{i\in[1,K]}} \sum_{i=1}^{K} \int_{\mathcal{P}} F[\alpha_i(\boldsymbol{x}), \nabla\alpha_i(\boldsymbol{x}), \Delta\alpha_i(\boldsymbol{x})]\, \mathrm{d}V(\boldsymbol{x}), \tag{3}$$

$$\text{s.t.} \quad \alpha_i(\boldsymbol{x}) \geq 0\,, \ \forall\alpha_i, \boldsymbol{x}, \qquad \text{(non-negativity)} \tag{3.1}$$

$$\textstyle\sum_i \alpha_i(\boldsymbol{x}) = 1\,, \ \forall\boldsymbol{x}, \qquad \text{(partition of unity)} \tag{3.2}$$

$$\alpha_i(\boldsymbol{v}_j) = \delta_{ij}\,, \ \forall\alpha_i, \boldsymbol{v}_j \qquad \text{(Lagrange property)} \tag{3.3}$$

$$\textstyle\sum_i \alpha_i(\boldsymbol{x})\boldsymbol{v}_i = \boldsymbol{x}\,, \ \forall\boldsymbol{x}, \qquad \text{(reproduction)} \tag{3.4}$$

where δ_{ij} is the Kronecker delta, and V is the volume form of $\mathcal{P}$. Some choices for F have historically included the Dirichlet energy Joshi et al. [2007] and the weighted total variation Zhang et al. [2014]. Because our method is constrained to the set of valid barycentric coordinates by construction, optimizing for different objectives is simply a matter of using the correct energy functional. We will see some other possible choices for F in §5.

4.2 Decomposition of Barycentric Coordinates

Deep learning architectures provide expressive function approximators, but barycentric coordinates require enforcing the constraints in (3)—which are not typically satisfied by generic neural network parameterizations. While constraints in Equations 3.1, 3.2, and 3.3 can be satisfied by normalizing and carefully reweighing the network output, building the reproduction property from Equation 3.4 into a neural network architecture poses a nontrivial challenge. In this section, we will offer a theoretical characterization of barycentric coordinate functions that is amenable to neural network representations.

Recall from §3 that the barycentric coordinates of a simplex in $\mathbb{R}^d$—e.g. triangles in $\mathbb{R}^2$, or tetrahedra in $\mathbb{R}^3$—are uniquely determined under mild conditions and can be computed in closed form. Due to this fact, our method relies on simplices as its basic building blocks. For the sake of simplicity, we will restrict the following discussion to 2-dimensional polygonal cages.

Our method decomposes the cage into a large number of non-degenerate overlapping *virtual triangles* that share their vertices with the cage:

$$\mathbb{T} := \{\mathcal{T} \mid \mathcal{V}(\mathcal{T}) \subseteq \mathcal{V}(\mathcal{P}), \mathrm{Vol}(\mathcal{T}) \neq 0\}. \tag{4}$$

Let $\mathcal{T} \in \mathbb{T}$ be a virtual triangle with vertices $\{\boldsymbol{v}_l, \boldsymbol{v}_m, \boldsymbol{v}_n\} \subseteq \mathcal{V}(\mathcal{P})$. Then, the *cage barycentric coordinates due to* $\mathcal{T}$ can be written in terms of the corresponding triangle barycentric coordinates:

$$\alpha_i(\boldsymbol{x}; \mathcal{P}, \mathcal{T}) = \begin{cases} \alpha_i(\boldsymbol{x}; \mathcal{T}), & \text{if } \boldsymbol{v}_i \in \{\boldsymbol{v}_l, \boldsymbol{v}_m, \boldsymbol{v}_n\}, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

This formula states that for every $\boldsymbol{x}$ inside of $\mathcal{T}$, the triangle coordinates of $\mathcal{T}$ are also valid cage coordinates as they inherit the necessary constraints.

In practice, multiple triangles typically overlap any given $\boldsymbol{x}$. Therefore, we define the set of *valid triangles at* $\boldsymbol{x}$ to be all virtual triangles in $\mathbb{T}$ that cover $\boldsymbol{x}$:

$$\mathbb{T}_{\boldsymbol{x}} := \{\mathcal{T} \in \mathbb{T} \mid \boldsymbol{x} \in \mathcal{T}\}. \tag{6}$$

To combine the coordinates due to the different virtual triangles, we have to assign each triangle $\mathcal{T}_j \in \mathbb{T}_{\boldsymbol{x}}$ a spatially varying weight, $w_j(\boldsymbol{x})$. We can then define the full cage barycentric coordinates as

a convex combination of the different triangle weights:

$$\boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P}) = \sum_{j=1}^{|\mathbb{T}_{\boldsymbol{x}}|} w_j(\boldsymbol{x})\, \boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P},\mathcal{T}_j). \tag{7}$$

As we will show in Proposition 4.2, so long as $w_j(\boldsymbol{x}) \geq 0$ and $\sum_j w_j(\boldsymbol{x}) = 1$, this operation retains all of the necessary properties such that $\boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P})$ always remain valid barycentric coordinates.

The triangle weights are degrees of freedom, and we are free to choose any set of weights for each $\boldsymbol{x}$ such that the weights are valid coefficients of a convex combination. An equivalent way of phrasing this is that the weights w_j form a *categorical probability distribution* over the elements of $\mathbb{T}_{\boldsymbol{x}}$. Therefore, we can model the weights using any representation that maps points on the interior of the cage to categorical distributions over the set of valid triangles. Luckily, this is a standard problem in machine learning, and we can use a neural network as the triangle weights: $w_j := \mathcal{N}_j$, where $\mathcal{N} : \mathbb{R}^d \to \Delta(\mathbb{T}_{\boldsymbol{x}})$, and $\Delta(\mathbb{T}_{\boldsymbol{x}})$ is the probability simplex over $\mathbb{T}_{\boldsymbol{x}}$.

This construction leads to parameterization of variational barycentric coordinates:

$$\boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P}) = \sum_{j=1}^{|\mathbb{T}_{\boldsymbol{x}}|} \mathcal{N}_j(\boldsymbol{x})\, \boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P},\mathcal{T}_j). \qquad \text{(VBC)} \tag{8}$$

Please refer to Figure 2 for an illustrative explanation and to §6.1 for more details on the computational model.

As we will see in the proof of Proposition 4.2, the formula above automatically satisfies *most* of the properties of barycentric coordinates, namely non-negativity, partition of unity, and reproduction. The one missing property is the Lagrange property. To enforce this property, we have to introduce a slight modification to our formulation. While there are multiple ways of addressing this challenge, we opted for one that is simple to implement, as it requires no modifications to the architecture, while reducing computational complexity (see §6.3). In particular, we remove all triangles from $\mathbb{T}$ that contain another cage vertex in their interior. In other words, we prune triangles containing other vertices of $\mathcal{P}$ in their interiors:

$$\widehat{\mathbb{T}} := \{\mathcal{T} \in \mathbb{T} \mid \forall \boldsymbol{v} \in \mathcal{V}(\mathcal{P}) : \boldsymbol{v} \in \mathcal{T} \implies \boldsymbol{v} \in \mathcal{V}(\mathcal{T})\}. \tag{9}$$

The rest of our formulation remains identical, and we simply operate on $\widehat{\mathbb{T}}$ instead of on $\mathbb{T}$. Now, when we evaluate (8) at $\boldsymbol{v} \in \mathcal{V}(\mathcal{P})$, we are taking a convex combination of barycentric coordinates exclusively from triangles with $\boldsymbol{v}$ as a vertex, all of which fulfill the Lagrange property.

The proposed modification limits the types of coordinate functions that our model can represent, in contrast with Proposition 4.1 below. It would have been possible to solve this problem differently, e.g. by reweighting the network outputs to take into account the distance to each cage vertex or by adding a penalty term to the optimization. However, these approaches introduce unnecessary complexity compared to pruning. Moreover, from a practical standpoint, there are few scenarios where having large triangles that cover other vertices is desirable behavior; see Figure 3. Lastly, as we will see in §6, pruning helps in reducing the computational demands of our algorithm.

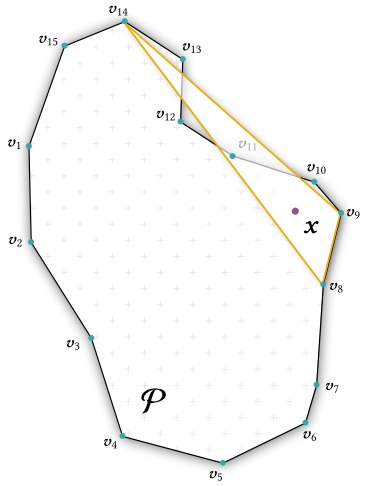

Fig. 3. To satisfy the Lagrange property, we prune away triangles which contain polygon vertices in their interior. These triangles introduce long-range dependencies across polygon boundaries, which is considered undesirable [Zhang et al. 2014]. The illustrated triangle results in the Lagrange property not being enforced at $\boldsymbol{v}_{11}$, while introducing a long-range dependency between $\boldsymbol{x}$ and $\boldsymbol{v}_{14}$. Therefore, pruning these kinds of triangles not only enforces the Lagrange property, but also encourages locality.

4.3 Analysis

Next, we discuss the theoretical properties of our formulation. In particular, we will show that it produces correct barycentric coordinates by construction, and less obviously, that it can represent *all* barycentric coordinate functions.

Proposition 4.1 (Universality). *Given a cage $\mathcal{P}$, any barycentric coordinate function $\boldsymbol{\alpha}$ can be decomposed into a convex combination of simplex barycentric coordinates of all possible simplices that can be formed by combining the cage vertices.*

Proof. Take a point $\boldsymbol{x} \in \mathcal{P}$. By definition, $\boldsymbol{\alpha}(\boldsymbol{x})$ lies in a convex polytope defined by the non-negativity constraint (3.1) (K linear inequality constraints), the partition of unity constraint (3.2) (1 linear constraint), and the reproduction constraint (3.4) (d linear constraints).

The corners of such a polytope cut out by linear inequality constraints are given by intersecting K constraint planes at a time. In this case, there are d reproduction constraints, and there is one partition of unity constraint; this means that each corner intersects at least $K - (d+1)$ non-negativity constraint planes. This argument shows that $\boldsymbol{\alpha}(\boldsymbol{x})$ contains at most $K - (K - (d+1)) = d+1$ nonzero values.

By the argument above, the corners of the constraint polytope are exactly simplex barycentric coordinates drawn from subsets of vertices of $\mathcal{P}$. Since our constraints are linear, the constraint polytope is contained within the convex hull of its corners, completing the proof. □

A less formal way of phrasing the proposition is that our construction is a *universal barycentric coordinate approximator*. Intuitively,

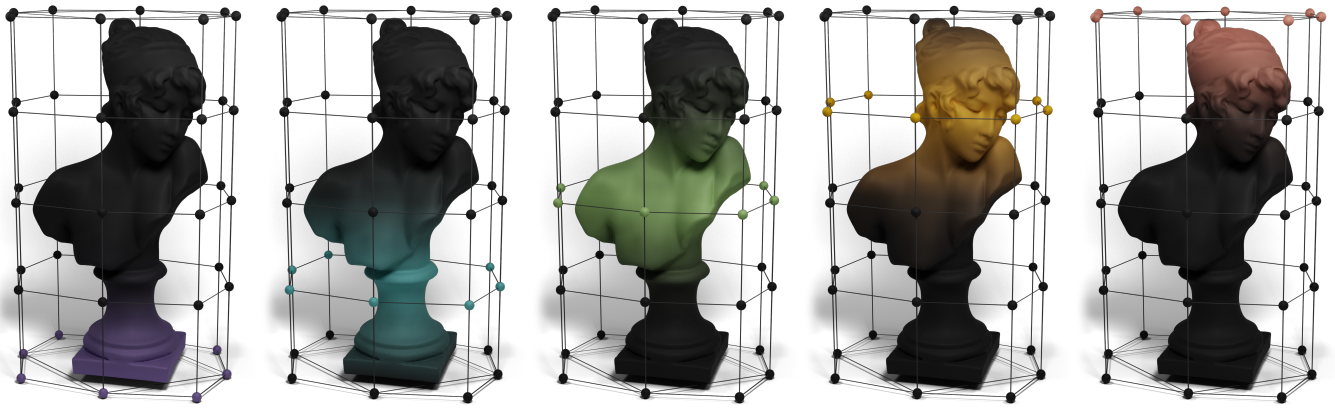

Fig. 4. Variational barycentric coordinates produce smooth-looking color interpolations from the colored cage vertices onto the Bust of Sappho mesh. The coordinates in this Figure were optimized using the weighted TV objective from Section 5.2.

we have shown that the constraint polytope of barycentric coordinates is exactly the space in which our network operates, $\Delta(\mathbb{T}_{\boldsymbol{x}})$. This result shows that our construction captures the fundamental underlying structure of the function space of barycentric coordinates. For completeness, we also verify a converse result:

Proposition 4.2 (Validity). *As long as the network $\mathcal{N}(\boldsymbol{x})$'s outputs are non-negative and sum to one, variational barycentric coordinates satisfy the barycentric coordinate constraints by construction.*

Proof. Non-negativity and partition of unity are fairly intuitive to verify. Since each set of simplex coordinates $\boldsymbol{\alpha}(\boldsymbol{x};\mathcal{P},\mathcal{T})$ is non-negative and sums up to one, a convex combination of them will have the same properties. As discussed near Equation 9, the Lagrange property follows from an identical argument.

Next, we need to address the reproduction property. We know from Equation 2 that for any individual simplex $\mathcal{T}_j \in \mathbb{T}_{\boldsymbol{x}}$,

$$\sum_{i=1}^{d+1} \alpha_i(\boldsymbol{x};\,\mathcal{P},\mathcal{T}_j)\,\boldsymbol{v}_i = \boldsymbol{x}. \tag{10}$$

Taking a convex combination over the barycentric coordinates of all $\mathcal{T}_j \in \mathbb{T}_{\boldsymbol{x}}$ retains the reproduction property,

$$\sum_{j=1}^{|\mathbb{T}_{\boldsymbol{x}}|} \mathcal{N}_j(\boldsymbol{x}) \sum_{i=1}^{d+1} \alpha_i(\boldsymbol{x};\mathcal{P},\mathcal{T}_j)\,\boldsymbol{v}_i = \sum_{j=1}^{|\mathbb{T}_{\boldsymbol{x}}|} \mathcal{N}_j(\boldsymbol{x})\,\boldsymbol{x} = \boldsymbol{x}, \tag{11}$$

which completes the proof. □

5 OPTIMIZATION OBJECTIVES

The key benefit of our method compared to previous work is its flexibility with respect to the optimization objective. In this section, we will first discuss how to optimize commonly used first-order smoothness energies: total variation, weighted total variation, and the Dirichlet energy. Additionally, we offer two examples of alternative optimization energies to demonstrate the degree of control over the final look of the animation offered by formulation.

5.1 Total Variation

Minimizing total variation (TV) ensures both *piecewise-smoothness* as well as *locality* [Joshi et al. 2007; Zhang et al. 2014]. However, our formulation is inherently non-smooth due to the discontinuities along the boundaries of virtual triangles. As a consequence, minimizing TV requires additional care. Specifically, the familiar definition of total variation [Zhang et al. 2014],

$$\mathrm{TV}(\boldsymbol{\alpha}) = \sum_{i=1}^{K} \mathrm{TV}(\alpha_i) = \sum_{i=1}^{K} \int_{\mathcal{P}} \|\nabla \alpha_i\|_2 \,\mathrm{d}\boldsymbol{x}, \tag{12}$$

is inadequate for our model as it fails to account for the difference in function values across the discontinuities.

Instead, we begin with the more general dual formulation of total variation [Chambolle et al. 2010]:

$$\mathrm{TV}(\alpha_i) = \sup_{\phi \in C^\infty} \left\{ -\int_{\mathcal{P}} \alpha_i \,\mathrm{div}\phi \,\mathrm{d}\boldsymbol{x} \;\middle|\; \forall \boldsymbol{x} \in \mathcal{P} :\; \|\phi\| \le 1 \right\}. \tag{13}$$

Applying integration by parts to this formulation on a domain with no discontinuities and smooth functions α_i recovers Equation 12. Following Zhang et al. [2020], we can apply a similar strategy to our problem, taking special care to account for the boundary conditions along discontinuities.

The discontinuities in our formulation partition the domain into N disjoint sets, $\mathcal{P} = \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_N$, with a total of M possible discontinuities, $\partial\mathcal{P}_1, \ldots, \partial\mathcal{P}_M$, between them. As already alluded to, applying integration by parts to Equation 13 incurs a boundary term for each discontinuity, $\partial\mathcal{P}_j$. Denoting with α_i^+ and α_i^- the values of α_i on either side of the discontinuity, the final formulation of total variation for our problem is

$$\mathrm{TV}(\alpha_i) = \sum_{j=1}^{N} \int_{\mathcal{P}_j} \|\nabla \alpha_i\|_2 \,\mathrm{d}\boldsymbol{x} + \sum_{j=1}^{M} \oint_{\partial\mathcal{P}_j} |\alpha_i^+ - \alpha_i^-| \,\mathrm{d}\boldsymbol{x}. \tag{14}$$

This version of TV makes it apparent that naively applying automatic differentiation or a finite-difference estimator to the model in Equation 8 would not work, as neither approach correctly captures the boundary integral. At a first glance, it might seem necessary

to manually keep track of the discontinuities as well as α_i^+ and α_i^- during the optimization. However, in §6.2 we will introduce a simple mollification strategy motivated by our total variation formulation in Equation 14 that will allow us to approximate of this energy with a standard finite-difference estimator.

5.2 Weighted Total Variation

A key component of *local barycentric coordinates* [Zhang et al. 2014] is the inclusion of a distance-based weighting of the objective energy,

$$\mathrm{TV}(\boldsymbol{\alpha}) = \sum_{i=1}^{K} \int_{\mathcal{P}} \Psi(d(\boldsymbol{x}, \boldsymbol{v}_i)) \, \|\nabla \alpha_i\|_2 \, \mathrm{d}\boldsymbol{x}, \tag{15}$$

where $d(\boldsymbol{x}, \boldsymbol{v}_i)$ is the geodesic distance between $\boldsymbol{x}$ and $\boldsymbol{v}_i$, and ψ is a user-specified function. As a consequence of the acceleration strategy explained in §6.3, the geodesic distance to a cage vertex $\boldsymbol{v}_i$ is equivalent to the Euclidean distance in all regions where its coordinates α_i are non-zero. This allows us to choose $d(\boldsymbol{x}, \boldsymbol{v}_i)$ to be the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{v}_i$, doing away with geodesic distance solvers and making our method completely grid-free.

In our experiments, we use $\Psi(t) = c + (1-c)t^2$, with $c = 10^{-1}$, as a way of blending between standard TV and square-weighted TV. Using only the square-weighted TV would artificially set the smoothness energy near the vertex to close-to-zero. In practice, this means that the optimization procedure would not minimize the discontinuities between virtual triangle edges close to the vertex.

5.3 Dirichlet Energy

The Dirichlet energy was first proposed as an objective for barycentric coordinates by Joshi et al. [2007]. For a smooth function u, it is the integral of its squared gradient over the domain, $\int \|\nabla u\|^2 dx$. Since our functions are not smooth and can be discontinuous over the boundaries $\partial \mathcal{P}_j$, we account for this by introducing additional finite differences over element boundaries to our formulation as a way of mimicking the gradient at these discontinuities. We define the Dirichlet energy for our formulation as

$$\mathrm{Dir}(\alpha_i) \coloneqq \sum_{j=1}^{N} \int_{\mathcal{P}_j} \|\nabla \alpha_i\|_2^2 \, \mathrm{d}\boldsymbol{x} + \sum_{j=1}^{M} \oint_{\partial \mathcal{P}_j} |\alpha_i^+ - \alpha_i^-|^2 \, \mathrm{d}\boldsymbol{x}. \tag{16}$$

While this is a heuristic approximation of the Dirichlet energy, we find that it yields useful results (Figure 11). Barycentric coordinates obtained with our formulation of $\mathrm{Dir}(\alpha_i)$ exhibit similar fall-off behavior as Dirichlet coordinates computed by discretizing the entire domain (Figure 16).

5.4 As-Rigid-As-Possible Energy

Energies such as total variation or the Dirichlet energy do not account for the final look of the deformations that result from using a set of barycentric coordinates. For example, coordinates produced by minimizing TV often produce elastic-looking deformations, which can be undesirable (see Figures 1 and 5 for examples). To address this, our approach allows us to fine-tune already optimized weights by using deformation-aware objectives.

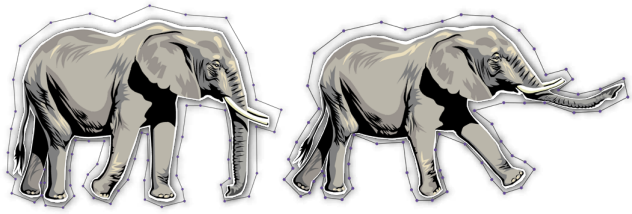

Fig. 5. Variational barycentric coordinates produce smooth looking 2D deformations, as shown on the Elephant mesh. The shown result was obtained by using variational barycentric coordinates optimized using the weighted TV objective from §5.2.

In this task, we are given a 3D model in its rest pose, $\mathcal{S}$, a cage $\mathcal{P}$ surrounding that model, and a deformation of the cage, $\mathcal{P}'$. The optimization variables are the coordinate functions, $\boldsymbol{\alpha}$ in Equation 1, and the optimization generates weights so that the map $\varphi_{\boldsymbol{\alpha}}$ minimizes the as-rigid-as-possible (ARAP) energy restricted to the surface of the deformed interior mesh, $\mathcal{S}'$ [Chao et al. 2010; Igarashi et al. 2005; Sorkine and Alexa 2007]. The continuous ARAP energy is defined as,

$$\mathrm{ARAP}(\varphi_{\boldsymbol{\alpha}};\ \mathcal{P}') = \int_{\mathcal{S}} \left\| d\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x};\ \mathcal{P}') - \mathrm{proj}_{\mathrm{SO}(3)}\, d\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x};\ \mathcal{P}') \right\|_F^2 \mathrm{d}\boldsymbol{x}, \tag{17}$$

where $d\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x};\ \mathcal{P}')$ is the push-forward of $\varphi_{\boldsymbol{\alpha}}$ at $\boldsymbol{x}$, and $\mathrm{proj}_{\mathrm{SO}(3)}$ the projection onto SO(3). Following Sorkine and Alexa [2007], we have only focused on the surface ARAP energy, i.e. the energy of the deformation between $\mathcal{S}$ and $\mathcal{S}'$. We leave volumetric deformation energies [Abulnaga et al. 2023] between $\mathcal{P}$ and $\mathcal{P}'$ for future work.

Since $\mathcal{S}$ is a mesh, we compute the discrete ARAP energy at each vertex [Sorkine and Alexa 2007] and use it to fine tune a set of variational barycentric coordinates. Specifically, the discrete ARAP energy (assuming uniform Laplacian weights) at an interior vertex $\boldsymbol{x}_i$ of $\mathcal{S}$ is defined as

$$\mathrm{ARAP}_{\boldsymbol{x}_i}(\varphi_{\boldsymbol{\alpha}}) \approx \sum_{j \in n(i)} \left\| (\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_i) - \varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_j)) - R_i(\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_F^2, \tag{18}$$

where $n(i)$ is the set of its neighbors, and R_i the best approximating rotation to $d\varphi(\boldsymbol{x}_i)$.

During training, we uniformly randomly sample vertices, and, for each sampled vertex, we randomly sample an adjacent edge. A single-sample Monte Carlo estimator of the energy of a given sampled vertex is given by:

$$\widetilde{\mathrm{ARAP}}_{\boldsymbol{x}_i}(\varphi_{\boldsymbol{\alpha}}) \approx \frac{1}{|n(i)|} \left\| (\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_i) - \varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_j)) - R_i(\boldsymbol{x}_i - \boldsymbol{x}_j) \right\|_F^2. \tag{19}$$

Our final loss is constructed by averaging this single-sample estimator for all sampled vertices.

As a slight modification to the approach by Sorkine and Alexa [2007], instead of using SVD to find R_i for each vertex, we store each vertex's rotation as a variable and jointly optimize both R_i and $\varphi_{\boldsymbol{\alpha}}$ using stochastic gradient descent. In our implementation, we store rotations in the SO(3) matrix logarithm format. Before running the main optimization loop, we initialize the rotations by manually finding the best-approximating rotations to the initial $d\varphi_{\boldsymbol{\alpha}}$.

5.5 Inverse Deformation Energy

Variational barycentric coordinates can be used to solve inverse deformation problems: given a 3D model in its rest pose, $\mathcal{S}$, a cage $\mathcal{P}$ surrounding that model, and a deformation of the model, $\mathcal{S}'$, we look for a deformation of the cage, $\mathcal{P}'$, such that the deformation induced by $\varphi_{\boldsymbol{\alpha}}(\mathcal{S};\ \mathcal{P}')$ best approximates $\mathcal{S}'$. Common use-cases are finding the deformation of a human body model to best fit a registered 3D scan or finding a cage-based deformation that best approximates the result of a physics simulation as a way of amortizing compute. Our model allows us to tackle inverse deformation problems by jointly optimizing our coordinates and the deformed cage vertex positions. We demonstrate the usefulness of VBC for the inverse physics simulation problem in Figure 12 .

Given a set of uniformly randomly drawn samples on the surface of the undeformed mesh, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N) \in \mathcal{S}$, and the corresponding points on the deformed mesh $(\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_N) \in \mathcal{S}'$, we can compute the mean absolute distance to measure the error between the deformation produced by the optimized map and the target mesh. In addition, we add a regularization term that encourages the norm of the deformed mesh Laplacian to be as close as possible to that of the ground-truth mesh. For each sample $\boldsymbol{x}_i$, we compute the norm of the Laplacian at the neighboring triangle vertices and interpolate them onto $\boldsymbol{x}_i$. Our inverse deformation energy is given by

$$\begin{aligned}\mathrm{I}(\mathcal{S}, \mathcal{S}') = \frac{A(\mathcal{S})}{N} \sum_{i=1}^{N} &\left\|\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_i) - \boldsymbol{x}'_i\right\|_2 \\ &+\lambda\left(\left\|\Delta\varphi_{\boldsymbol{\alpha}}(\boldsymbol{x}_i)\right\|_2 - \left\|\Delta\boldsymbol{x}'_i\right\|_2\right)^2,\end{aligned} \tag{20}$$

where $\Delta\boldsymbol{x}$ denotes the interpolated vertex Laplacian at $\boldsymbol{x}$, $A(\mathcal{S})$ denotes the area of $\mathcal{S}$ and λ is a user-specified regularization parameter that we keep fixed at 10^{-4}.

We optimize Equation 20 using a two step procedure. Starting from a weighted TV barycentric coordinate network, we first find the closest-approximating global rotation and scale for the deformed cage using stochastic gradient descent on the energy in Equation 20 During, this stage, we keep the deformed cage's local-frame vertex positions and the network weights fixed. Once a suitable global rotation and translation of the deformed cage is found, we jointly optimize the deformed cage's local-frame vertices and fine-tune the barycentric coordinate network. The exact details of the optimization setup for the experiment in Figure 12 are presented in Section 7.4.

6 COMPUTATIONAL MODEL

Having covered the mathematical underpinnings of our model, in this section we detail the practical aspects of its implementation.

6.1 Neural Network Model

We model the convex combinations of triangle barycentric coordinates using a neural field that maps points on the interior, $\boldsymbol{x}$ to categorical probability distributions over *valid* virtual triangles in 2D, i.e. tetrahedra in 3D. To this end, we construct the last network layer to as many outputs as the *total* number of triangles in $\mathbb{T}$, regardless of whether they contain $\boldsymbol{x}$ or not. The network outputs are mapped to a positive value using the `softplus` activation function. We zero out the probabilities for all triangles which do not contain the queried interior point before finally normalizing the distribution.

At first glance, this approach results in a potentially large network output layer, as the total number of virtual simplices scales with $O(K^3)$ in 2D and $O(K^4)$ in 3D. As a remedy, we will introduce a simplex pruning strategy in §6.3, which reduces the computational complexity of our method to $O(K)$ for most common scenarios and makes our method tractable for more complex cages.

We use a feed-forward network with 5 hidden layers of width 256 and a `LeakyReLU` [Maas 2013] activation function. Before feeding the interior coordinates into the network, we first encode them with the *hash-grid encoding* [Müller et al. 2022], with 16 levels, 4 features per level, and `smoothstep` interpolation. We train the network using the Adam optimizer Kingma and Ba [2014]. We include the remaining experimental parameters in Table 1. We have not made a significant effort to tune the architecture or the hyperparameters of our network, nor have we used any acceleration data structures to accelerate the process of finding which triangles contain a given interior point.

6.2 Smoothing Discontinuities

Explicitly sampling the discontinuities and computing α_i^+ and α_i^- in Equation 14 would add significant additional complexity to our approach. Instead, we introduce a comparatively simple mollification approach, which allows us to estimate both the interior and the boundary terms in Equations 14 and 16 with an off-the-shelf finite-difference estimator over $\mathcal{P}$.

Assume f is zero everywhere on $\mathbb{R}^2$ except within a triangle. One way of thinking about f is as a smooth function multiplied by the 0-1 indicator function of the triangle. In essence, our approach replaces the indicator function with an appropriate mollifier, such that the resulting smooth surrogate, $f^*_{r,\delta}$, approaches f as $\delta \to \infty$. By carefully choosing the smoothing function, we can ensure that $\int \|\nabla f^*_{r,\delta}\|_2$ approaches the TV formulation in Equation 14. Once our model is optimized, we disable the mollification to ensure that the barycentric coordinate constraints are satisfied during inference.

To accomplish this, we define a *smoothing radius* r around each discontinuity, inside of which we rapidly decay the indicator function to zero, and then smooth it using a *scaled logistic* function. Denoting by $d(\boldsymbol{x})$ the signed distance of a point $\boldsymbol{x}$ to the boundary of a triangle, we define the a ramp function which increases from linearly from -1 to 1 within the smoothing radius,

$$R_r(\boldsymbol{x}) = \begin{cases} \frac{d(\boldsymbol{x})}{r} & \text{if } d(\boldsymbol{x}) \le |r|, \\ 1 & \text{if } d(\boldsymbol{x}) > r, \\ -1 & \text{if } d(\boldsymbol{x}) < -r. \end{cases} \tag{21}$$

Finally, we smooth the ramp function using a *scaled logistic function*, $\sigma_\delta(\ x) = \frac{1}{1+\exp\{-\delta x\}}$, where δ is a user-chosen sharpness parameter,

$$f^*_{r,\delta}(\boldsymbol{x}) = \frac{\sigma_\delta(R_r(\boldsymbol{x})) - \sigma_\delta(-1)}{\sigma_\delta(1) - \sigma_\delta(-1)} f(\boldsymbol{x}). \tag{22}$$

We normalize and recenter σ_δ to ensure continuity at $d(\boldsymbol{x}) = \pm r$. This function has a C^1 discontinuity, which does not appear to affect our optimization procedure in practice; we leave to future

work design of a spline-based smoothing function with higher order continuity at $d(x) = \pm r$. Our approach is illustrated in Figure 6.

Choice of smoothing function. In our scenario, there are multiple indicator functions meeting at the shared edges of virtual triangles. The smoothing function we choose has to have the correct limiting behavior in this situation. The sum of two or more logistic functions centered around the same point is again a logistic function, whose limit converges to a step function. Moreover, the derivative of a logistic function becomes a Dirac delta as $\delta \to \infty$, whose integral equals exactly the difference on either side of the function, as needed to capture the second term in Equation 14.

Implementation details. Naturally, we only want to perform this type of smoothing on the interior virtual edges, but not at the boundary of $\mathcal{P}$. Therefore, it is either necessary to explicitly keep track of which edges are part of the boundary, or to exclude samples closer than r to the boundary from the optimization. We opted for the second approach as it appears not to yield noticeable artifacts as long as r is small enough and the cage faces distant enough from the interior mesh.

There is a natural trade-off when picking the values of r and δ. If r is too small and δ too large, we might have trouble sampling the discontinuities during optimization. If δ is too small, we blur the gradient function too much and possibly optimize for the wrong result. The concrete parameters used in our experiments are included in Table 1.

Finite-difference estimator. We use a standard central difference estimator and keep the spacing constant at $h = 2.5 \cdot 10^{-2}$. There appears to be a similar trade-off when it comes to the size of h as with the value of r: if we make h too small, we amplify the noise due to the network and the hash-grid encoding, but if we make it too large, we increase the bias in the estimated gradient.

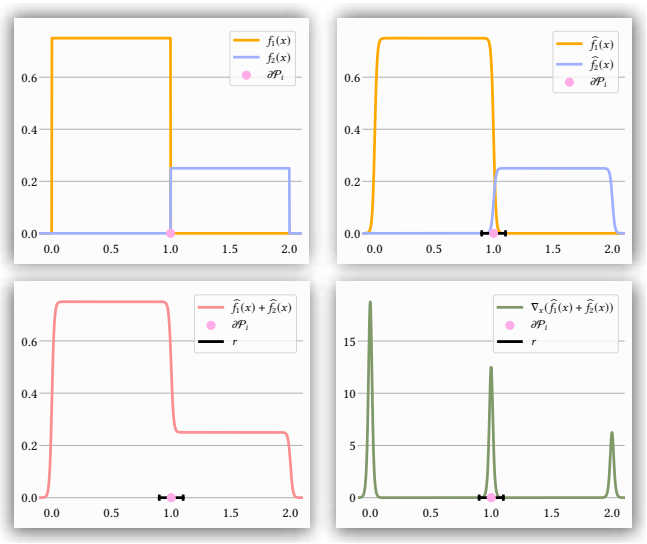

Fig. 6. An illustration of our method for computing total variation with a finite difference estimator. The figure in the top left shows two functions, f_1 and f_2, which are non-zero over two different line segments (i.e. 1D simplices). The sum of these two functions has a 1D discontinuity at the vertex that is shared between the two line segments, marked with $\partial\mathcal{P}_i$. The top right figure shows the effect of our smoothing term with radius r on each of the two functions. The figure in the bottom left demonstrates that the sum of the individually smoothed functions is equivalent to smoothing the summed function. Lastly, the figure in the bottom right shows the finite differences gradient estimator computed on the smoothed function. We can see that as we increase r, the gradient tends to a Dirac delta and the discontinuities are correctly accounted for.

6.3 Accelerating The Model

So far, the practical utility of our theoretical formulation has been limited by the fact that the number possible virtual triangles grows with $O(K^3)$. This presents a problem for two reasons—not only does it significantly increase the necessary compute, it also means that the representational power of the neural field quickly becomes insufficient as K becomes larger. We alleviate both issues by using a simple strategy for pruning triangles, which allows us to limit the number of triangles to a constant multiple of the number of vertices. Not only does such a strategy make our method tractable, it is also *locality-enforcing*, meaning that it builds a notion locality into our model as a hard constraint.

In §4.2, we presented a simple modification to our formulation that ensures that the necessary constraints are satisfied by eliminating all virtual triangles which contain a cage vertex in their interior. For reasons discussed in Figure 3, we have found this to be a useful heuristic and therefore expand on the idea by completely eliminating all virtual triangles that cross the boundary of the shape. We do this by densely sampling the ambient space around our shape and discarding all virtual triangles that contain one of the samples.

The actual pruning heuristic is designed with locality of coordinates in mind. For each vertex, we first find all virtual triangles that have that vertex as a corner. To enforce locality, we discard triangles whose extent is not limited to the local neighborhood of the vertex. Specifically, we sort the triangles by the length of their longest edge, and keep the M_p shortest ones, where M_p is a user-specified parameter. We will refer to this as the *min-longest-edge* heuristic. Note that this pruning strategy does not replace the one in §9 and Figure 3. As an example, in cages with three co-linear boundary vertices, we must ensure that no virtual triangle contains the middle of the three boundary vertices on one of its edges.

In certain edge cases, this heuristic can result in parts of the interior not being covered by any virtual triangles. To ensure coverage, we sample the cage interior—either the interior volume, or the surface of an interior mesh—and check if any of the samples have no virtual triangles containing it. If this happens, we proceed to find all virtual triangles that contain the interior point and have the closest cage vertex as a corner, and then use the min-longest-edge heuristic to choose M_c triangles.

We present the steps of our pruning approach in more detail in Algorithm 1. The parameter values used in our experiments are included in Table 1. We also include relevant statistics in Table 2 to offer a more concrete picture of the computational savings.

Algorithm 1: Virtual Simplex Pruning

Input: Cage $\mathcal{P}$, set of all virtual triangles $\mathbb{T}$, no. pruned triangles per cage vertex M_p, no. coverage triangles per interior point M_c.
Result: Pruned set of virtual triangles $\widehat{\mathbb{T}}^{\text{prune}}$.

```
# Remove degenerate triangles and triangles not inside P:
X_out := sample_outside(P, N_out)
T̂ := {T ∈ T | not_degenerate(T) and (∀x ∈ X_out : x ∉ T)}

# Prune using the min longest edge heuristic:
T̂^pruned := ∅
for v_i in V(P):
    # Find triangles with v_i as a vertex:
    T̂_i := {T ∈ T̂ | v_i ∈ V(T)}
    # Find M_p triangles with shortest longest edge:
    T̂^pruned := T̂^pruned ∪ subsample(T̂_i, M_p, "min-longest-edge")

# Ensure that the interior is covered with triangles:
X_in := sample_inside(P, N_in)
for x in X_in:
    # Check if x is already covered by a triangle:
    if {T ∈ T̂^pruned | x ∈ T} ≠ ∅:
        continue
    # Find triangles that connect to the closest vertex:
    T̂_i := {T ∈ T̂ | closest_vertex(x) ∈ V(T)}
    T̂^pruned := T̂^pruned ∪ subsample(T̂_i, M_c, "min-longest-edge")
```

Discussion. The coverage step in Algorithm 1 potentially increases the computational complexity beyond a constant factor of the number of cage vertices. However, if the interior mesh is known a priori—which is often the case—we only need to ensure that the vertices of the interior mesh are covered virtual triangles. Doing this reduces the number of virtual triangles down to a constant factor of the sum of cage and interior mesh vertices. We consider this scenario as an edge case as it only occurred in a limited region of one cage mesh from our test data, namely the Armadillo mesh.

7 RESULTS

In this section, we demonstrate the effectiveness and robustness of variational barycentric coordinates. First, in §7.1 we present a series of experiments and quantitative results which illustrate the practical behavior of our implementation. Having validated the method, in §7.2 we show qualitative results of our method and, in §7.3, compare it to previous work. Lastly, §7.4 demonstrates the key benefit of our formulation, its ability to optimize for deformation-aware barycentric coordinates.

We built our implementation using PyTorch [Paszke et al. 2019], as well as the Tiny CUDA NN [Müller 2021] library. All experiments were performed on a desktop computer with an Intel Xeon E5-2630 v3 CPU, 32 GB of RAM memory and a Nvidia TITAN Xp GP102 with 12 GB of VRAM. Note that, despite using the Tiny CUDA NN library, we were not able to take full advantage of it due to the lack of necessary hardware components in our GPU.

Table 1. Experimental model parameters. We use two sets of parameters, one for 2D experiments, and one for 3D.

Parameter	Value (2D)	Value (3D)
Smoothing sharpness (δ)	3000	1000
Smoothing radius (r)	$5 \cdot 10^{-3}$	$8 \cdot 10^{-3}$
Max. simplices per cage vertex (M_p)	28	80
Max. simplices per interior point (M_c)	5	5
Training steps	2000	3000
Learning rate	10^{-3}	$5 \cdot 10^{-4}$
Batch size	3000	2000

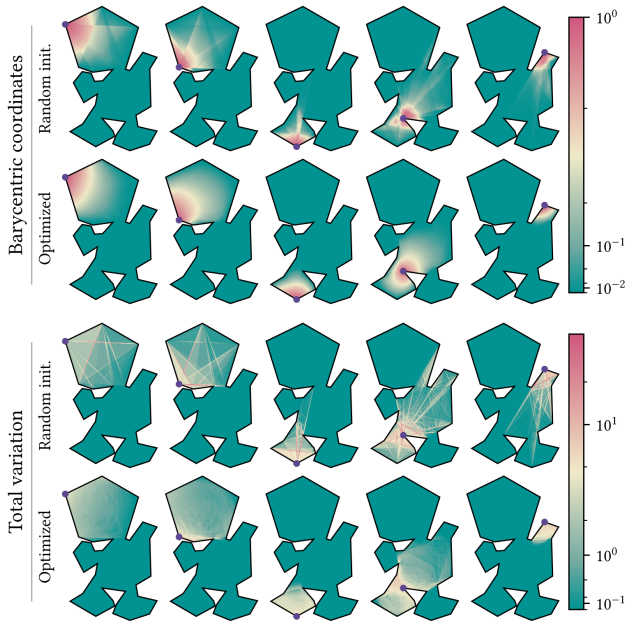

Fig. 7. Visualizing the 2D variational barycentric coordinates as well as the associated TV energy before and after optimization on the Gecko mesh.

7.1 Validation

In this section, we discuss the different practical outcomes of our implementation. We scale each cage mesh to the unit square, (resp. unit cube) while maintaining the aspect ratio of its bounding box. This allows us to have only two sets of parameters across all of our test meshes: one for 2D and one for 3D.

To determine suitable parameters for the discontinuity smoothing in §6.2, as well as the number of virtual simplices M_p, we performed two ablation studies. An excerpt of the results is shown in Figure 8, with the entire ablation study available in the supplemental material. In summary, we want to make δ as large as possible to best approximate Equation 14, while still keeping it small enough as to not cause numerical issues. As for the number of virtual simplices, we require a large enough value of M_p for sufficient representation

	Star	Gecko	Woody	Elephant	Blue Monster	Horse	Bust of Sappho	Hand	Armadillo
Cage vertices	12	34	26	64	100	51	40	92	110
Possible simplices	220	5984	2600	41664	16170	249900	91390	2794155	5773185
Interior simplices	68	388	738	1923	5255	3247	83958	35462	56438
Used simplices	68	388	403	803	1469	1072	1308	3432	4153
Training time (min)	1.03	1.36	1.37	1.99	3.09	11.88	15.45	35.32	43.05
Inference time (ms)	3.64	4.49	4.66	7.17	11.41	42.70	56.78	136.26	169.66

Table 2. Experimental statistics. As a way of quantitatively characterizing our pruning heuristic, we include in the table for a subset of cage meshes the number of vertices, the total number of possible virtual simplices, the number of non-degenerate simplices fully contained in the interior of the cage, as well as the final number of simplices used in the model. We also include the training times, as well as the inference times for a batch size of 2000.

power. Interestingly, the quality seems to plateau after a certain number of virtual triangles, practically justifying the min-longest-edge-heuristic. Table 1 contains the concrete algorithm parameters we used for all experiments.

Each 3D training sample has a lower probability of being in the smoothing radius of each discontinuity, making the training signal in those areas sparser and more noisy. Therefore, we found it useful to use a wider smoothing function in 3D compared to 2D. In general, we also require more virtual simplices per cage vertex in 3D compared to 2D to produce smooth coordinate function. This is to be expected, as there are far more possible virtual tetrahedra than there are triangles. The parameter which governs the number of simplices per interior point in the scenario where our initial pruning heuristic fails to cover the entire domain, M_c, remained unused in all of our 2D experiments. We nonetheless suggest leaving it as a non-zero value to ensure against corner cases.

Table 2 presents the quantitative behavior of our algorithm, including statistics related to our pruning heuristic from Section 6.3. While the number of all possible simplices increases with the number of cage vertices and the dimension of ambient space, our heuristics keep the final number of simplices in our model to a more reasonable number.

While the training and inference time increase somewhat with the number of cage vertices, the main factor influencing timing is dimension (2D vs. 3D). We attribute this to several factors: the inference time is slower due to a larger final network layer, a naive 3D central difference estimator requires 2 additional network evaluations compared to a 2D one, and we require a smaller batch size with more training steps and a smaller learning rate to accomodate the increased GPU memory requirements due to the two additional network evaluations each training step.

Figure 7 visually compares the variational barycentric coordinates and associated TV energy of a randomly initialized network to those of a TV-optimized network. The total variation of the randomly initialized network is focused primarily on the discontinuities between virtual triangles. These are largely optimized-away, demonstrating the effectiveness of the smoothing approach from §6.2.

In this experiment, the *full* total variation, $TV(\boldsymbol{\alpha}) = \sum_i TV(\alpha_i)$, of the optimized network is non-zero on the entire domain. This

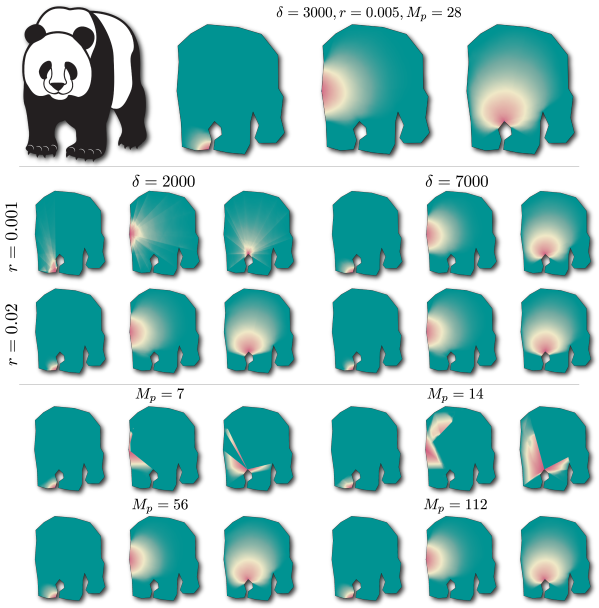

Fig. 8. A subset of the results on the Panda mesh from our ablation study. We find larger values of the sharpness parameter δ to perform better as they help better approximate Equation 14. The visual smoothness of the results deteriorates again after a certain point, presumably due to numerical issues caused by large gradient values associated with larger values of δ. Making the smoothing radius r too small compared to the sharpness parameter δ results in visual artifacts. We hypothesize that this is due to discontinuities caused by the ramp function cutoff in Equation 21 which otherwise become numerically negligable when δ is large enough compared to r. For the ablation study of the min-longest-edge heuristic, we set $M_c = 0$, disabling the part of the algorithm that ensures interior coverage, and varied the maximum number of virtual triangles per cage vertex, M_p. As expected, small values of M_p are not enough to faithfully represent the coordinate functions in the entire shape, and after a certain point, adding additional virtual triangles does not affect the results visually.

demonstrates a tension between barycentric coordinate constraints

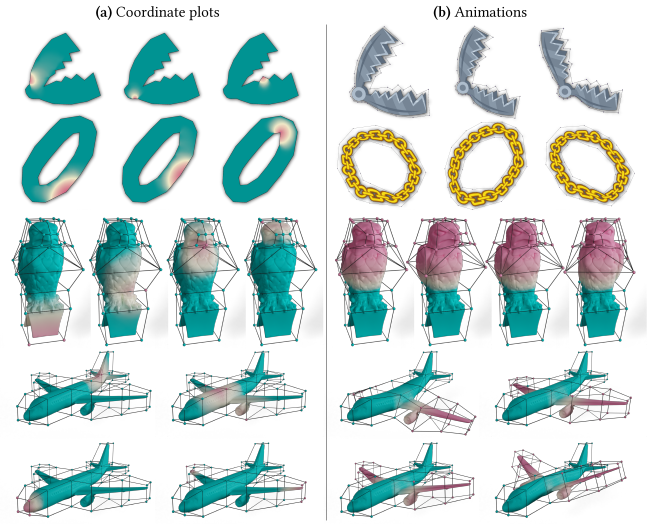

Fig. 9. Figure (a) shows the visualization of weighted TV coordinates on 2D and 3D meshes. Figure (b) shows excerpts from smooth animations generated by inbetweening cage deformations.

and smoothness energies: Because all barycentric coordinate functions are convex combinations of simplex barycentric coordinates, their gradients are also complex combinations of the corresponding simplex gradients towards the cage vertex.

7.2 Qualitative Results

In this section, we provide several visualizations of variational barycentric coordinates in both 2D and 3D and demonstrate their usefulness for cage-based deformation.

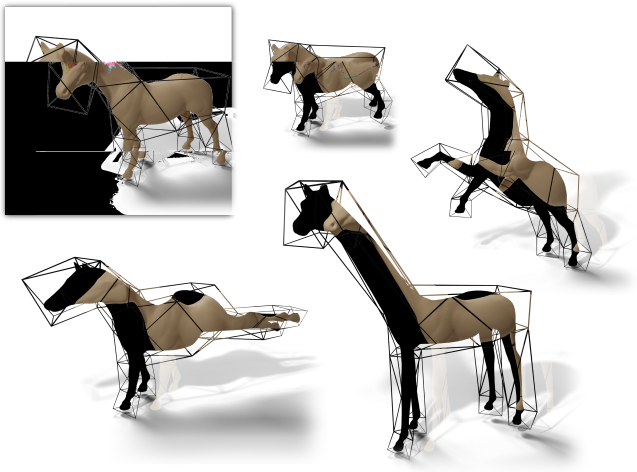

Fig. 10. We demonstrate the effect of using TV-optimized variaitonal barycentric coordinates to perform various deformation of the Horse mesh. The undeformed mesh is shown on the top left.

Figure 4 demonstrates how our coordinates can be used to smoothly interpolate values such as color from the vertices of a 3D cage into its interior. Similar to the 2D example in Figure 7, the TV variational barycentric coordinates produce smooth-looking function interpolations.

Figure 11 depicts an experiment where we compare the barycentric coordinate functions obtained by minimizing the Dirichlet energy as opposed to total variation. The total variation coordinates tend to be significantly more localized compared to the Dirichlet ones, leading to narrower regions of influence. The flexibility of our formulation with respect to the optimization objective allows the user to choose the set of coordinate functions which best suits their particular use-case.

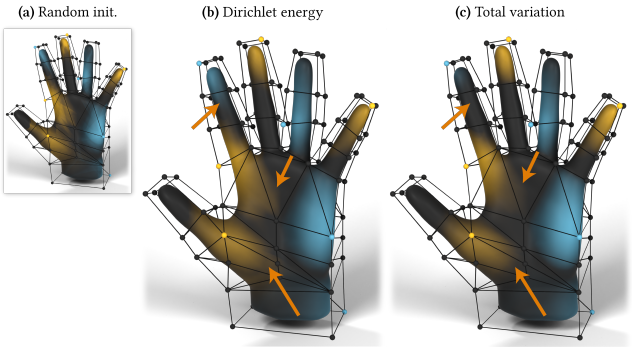

Fig. 11. Comparing different sets of barycentric coordinate functions in 3D. Left to right, we show coordinates from a randomly initialized network prior to training, our Dirichlet coordinates, as well as our weighted TV coordinates. The Dirichlet-optimized coordinate functions tend to be blurrier and have a wider region of influence compared to the TV-optimized ones.

Lastly, Figures 5, 9 and 10 demonstrate the usefulness of our coordinates for the common use-case of 2D and 3D cage-based deformation. As evidenced by our experiments, the deformations appear smooth, and the deformations due to individual cage vertices do not exhibit unnecessarily global influence, in part due to the pruning heuristic from Section 6.3. We refer the reader to the supplemental material for the animation videos.

7.3 Comparison

Figures 16 and 13 compare our Dirichlet, TV, and weighted TV coordinates with other available methods. All produce broadly similar weights that fulfill that barycentric conditions, but differences exist. Our weighted TV results are qualitatively similar to Local Barycentric Coordinates (LBC), which also uses a weighted TV energy. Weighted TV coordinates have more local support (no small values far away from the cage vertex), and a harsher fall-off of basis functions (the area of large values near the vage vertex is larger). This effect is slightly more pronounced for our weighted TV results compared to LBC's. Moreover, unlike LBC, our TV and weighted TV methods do not require discretization of the domain. MVC generates particularly nonlocal results: while the coordinates' red values in our weighted TV function can be seen to almost partition the shape, MVC coordinates are red only extremely close to the cage

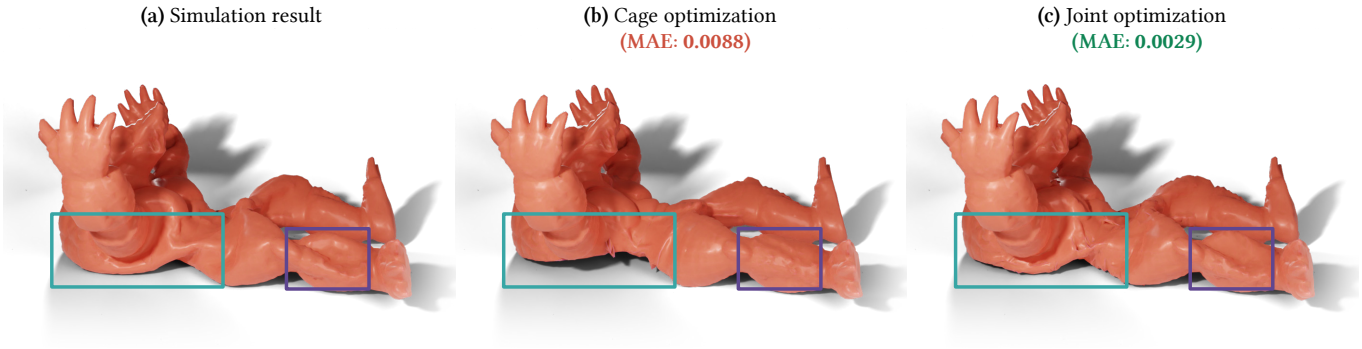

Fig. 12. Our method can be used to approximate non-linear deformations such as the soft-body simulation result shown in Figure (a). Simply using the standard TV coordinate and optimizing the cage leads to an unnatural result, visible artifacts and a large per-vertex approximation error, as visible in Figure (b). However, if we also allow the weights of our network to change during the optimization, we can achieve a much more natural looking result, decreasing the error by two thirds—see Figure (c).

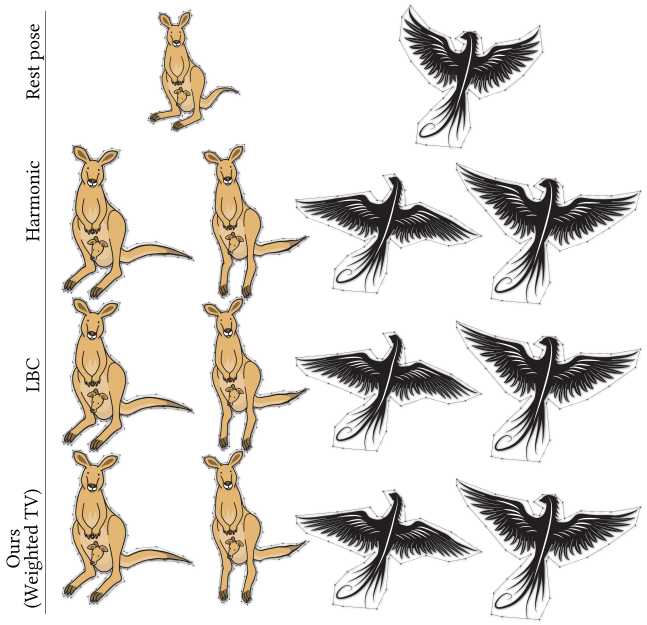

Fig. 13. Select frames taken from smooth animations produced by our method compared to those produced by previous work (full videos available in the supplemental material).

vertex, and all blend into each other in the interior of the shape. The Dirichlet and MVC coordinate functions have rounder isolines than the TV-based methods. This is a consequence of the order of the exponent in the energy's norm. Our Dirichlet coordinates exhibit maximum overall smoothness and smaller areas with large function values, while having larger supports, without the need to discretize the shape's interior. Our non-weighted TV coordinates' behavior is in between the Dirichlet and weighted TV coordinate functions. The resulting coordinates are not as local as the weighted TV coordinates, but they are smoother, and they are more local than the Dirichlet coordinates, but not as smooth.

7.4 Deformation-Aware Coordinates

A distinguishing feature of our formulation is its ability to optimize different objective functions within the space of barycentric coordinate functions. Already, in Section 7.2, we demonstrated several results with two such functions: the total variation from Equations 14 and 15, and the Dirichlet energy from Equation 16. Beyond these two generic energies, in this section we test the *as-rigid-as-possible* energy introduced in Section 5.4, as well as the inverse deformation energy from Section 5.5.

As-rigid-as-possible coordinates. Figures 1 and 14 show examples of using the ARAP energy from Section 5.4 to optimize for coordinates that produce less elastic-looking deformations. In Figure 1, the initial result was obtained by using weighted TV coordinates to deform the Armadillo mesh. Fine-tuning the coordinates with the surface ARAP energy resulted in fewer artifacts and more rigid-looking deformations, as evidenced by the highlighted regions of the figure.

Figure 14 demonstrates how the ARAP energy can be useful in producing rigid and smooth looking deformations (refer to the supplemental material for videos). To generate the animation, we manually deformed the cage at a set of key-frames and inbetweened the cage vertices for the rest of the frames. We used the key-frame with the most extreme deformation of the cage as $\mathcal{P}'$ when optimizing the energy in Equation 18. Nonetheless, we see that the entire animation is affected by the ARAP coordinates, making it look overall more rigid.

In our implementation, we found it necessary to use a significantly larger learning rate for the parameters of the local rotations compared to the network weights (0.1 and 10^{-3}, respectively). We trained the model for a total of 1200 steps, decaying the learning rate by a factor of 0.8 every 150 steps.

Inverse deformation. Our model is fully differentiable, allowing us to solve inverse deformation problems. In Figure 12 we created

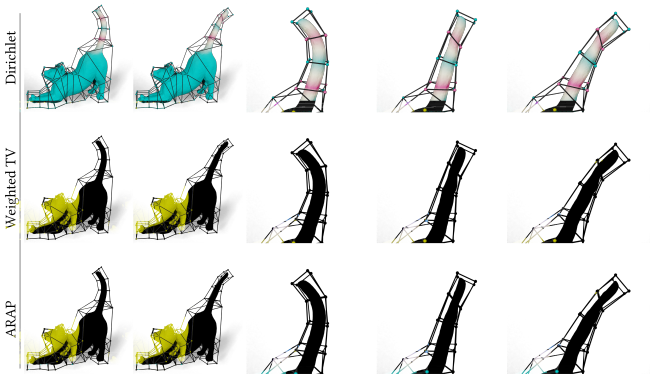

Fig. 14. Visualizing the 3D variational barycentric coordinates during an animation generated by inbetweening deformations. The Dirichlet coordinates have a more non-local influence compared to the Weighted TV objective, resulting in less abrupt bends near the tip of the tail. Nonetheless, we are able to make the Weighted TV deformation more rigid looking by minimizing the ARAP objective. Note that, even though the distortion was minimized only between the undeformed shape and the shape at the 36^{th} frame, the ARAP optimization improves the look of the entire animation.

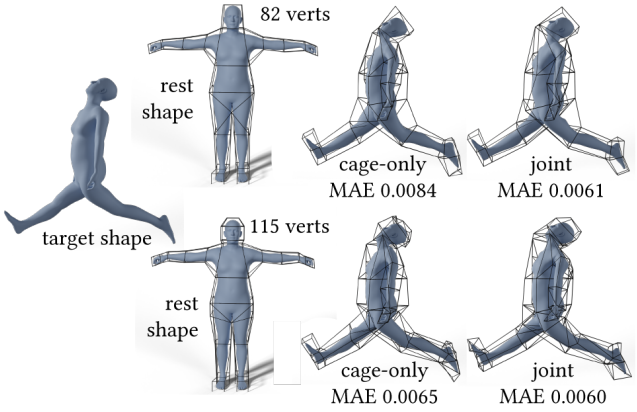

Fig. 15. Increasing the cage resolution by 40% results in an overall decrease in approximation error. Notably, even when using lower-resolution cage, the joint optimization leads to a 37% error reduction.

a high-resolution soft-body physics simulation in Blender to use as the target for optimization. As shown by our results, optimizing the positions of the cage vertices alone is insufficient to faithfully reproduce the simulation result. By comparison, jointly optimizing the weights of the network produces fewer artifacts and achieves a significantly lower reproduction error. Once optimized, these coordinates could be used to transfer the deformations due to a physics simulation from low-resolution meshes onto high-resolution ones similar to Sacht et al. [2015] or onto the same mesh after an editing operation that changed the mesh topology.

In Figure 15, we show the effect of increasing the cage resolution by 40% during inverse optimization of a human body mesh [Loper et al. 2015]. As expected, the additional degrees of freedom introduced by the new cage vertices result in a better fit to the target shape for both the cage-only and the joint cage-and-coordinates optimization. However, as evidenced by the the mean absolute vertex distance, jointly optimizing for the coordinates alongside the cage vertices improves the results significantly even on the lower resolution cage. This implies that the additional degrees of freedom in a cage are not as useful in the presence of coordinates that are better suited for a specific use-case, further validating our deformation-aware approach to designing generalized barycentric coordinates.

In these experiments, we fine-tuned the network for a total of 10000 steps for the Armadillo mesh, and 15000 steps for the Human mesh, with an initial learning rate of $5 \cdot 10^{-4}$. During the first 6000 steps, we decayed the learning rate by a factor of 0.8 every 200 steps. We used a larger learning rate of $5 \cdot 10^{-3}$ for the cage rotation, scale, and vertex positions.

8 DISCUSSION AND CONCLUSION

Alternative deformation-aware energies. We have demonstrated how to incorporate the surface as-rigid-as-possible energy into our formulation as a proof-of-concept. This work represents the first step towards other deformation-aware barycentric coordinates. For example, all of the deformation-aware energies we worked with are restricted to the surface of an interior mesh. Given that the deformation field is volumetric, a reasonable alternative would be to use a volumetric deformation energy, such as the ones enumerated by Abulnaga et al. [2023]. Since our model is grid-free, it would be necessary to find a way of computing the volumetric deformation gradient required by volumetric deformation energies.

Alternative inverse problems. In addition to the inverse deformation problem presented in Figure 12, our method can help tackle other inverse problems. For example, our experiment requires the existence of dense vertex-to-vertex correspondences between two meshes. This points to a natural direction for future work, namely solving inverse cage-based deformation problems while relying only on sparse correspondences. Pushing this idea further, one might consider combining our inverse deformation framework with an inverse rendering loss, similar to Peng et al. [2021], thus completely eliminating the need for 3D reconstructions or correspondences. As an alternative future direction, one could consider amortizing a physics simulation over multiple frames—or possibly even over an entire dataset—to create coordinates specific to a single class of shapes.

Performance. We see multiple viable avenues for improving the performance of our model. It seems likely that a stochastic finite-difference estimator would reduce the memory and compute cost of computing first-order smoothness energies. Furthermore, our algorithm requires us to find all valid simplices for every interior sample in a brute-force manner. Future work might instead consider applying an acceleration data structure to improve the computational complexity of this step. Lastly, while we already use Tiny CUDA NN [Müller 2021] in our implementation, we were unable to exploit the fully-fused neural networks offered by the library due to the limitations of our hardware.

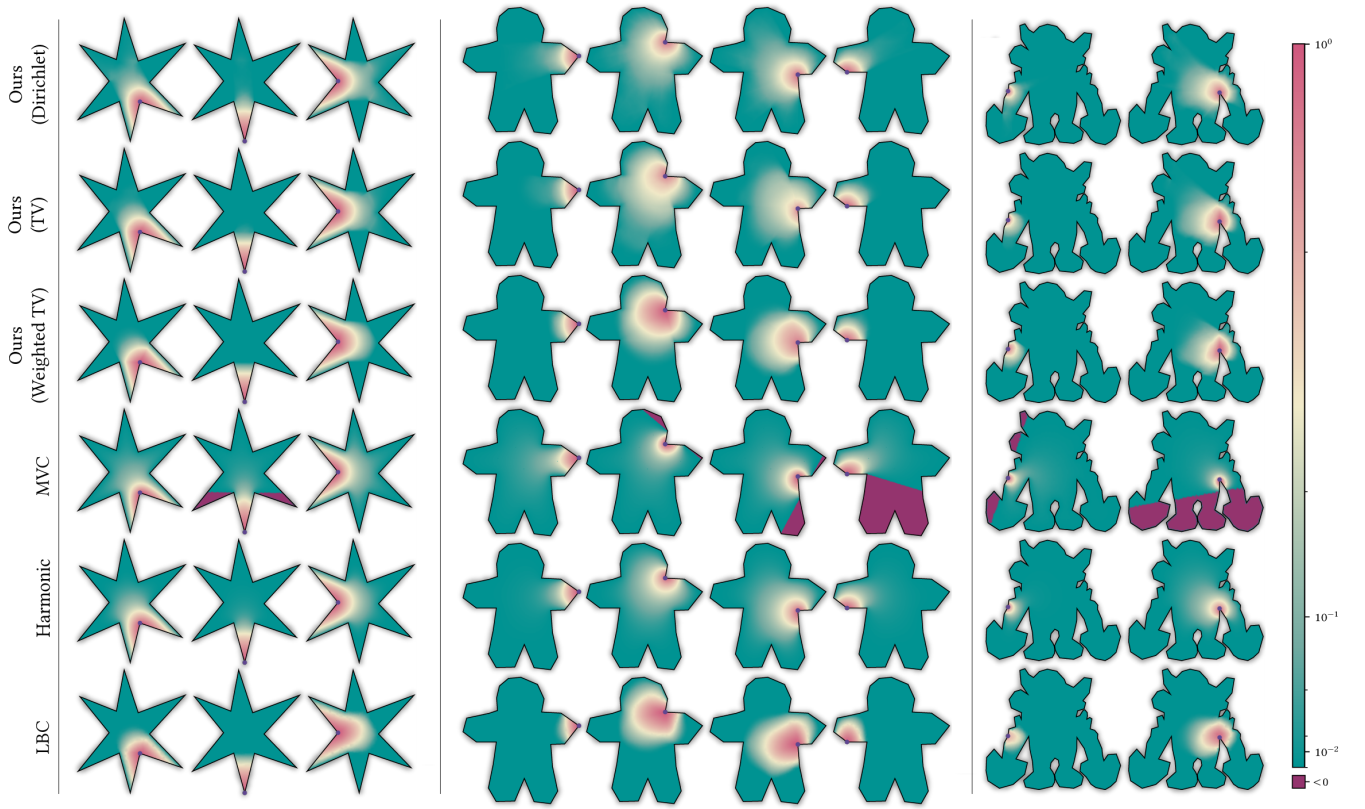

Fig. 16. Our Dirichlet, TV, and weighted TV coordinate functions, compared to the basis functions produced by Mean Value Coordinates [Floater 2003; Ju et al. 2005], Harmonic Coordinates [Joshi et al. 2007], and Local Barycentric Coordinates [Zhang et al. 2014] and on three different shape.

Limitations. Because of the pruning heuristic from Section 6.3, our coordinates are limited to a star-shaped neighborhoods of cage vertices. In practice, this means that a slight bend of a straight cage into a concavity can change the coordinates. To address this, one could consider not eliminating virtual simplices that cross the cage boundary, but rather modifying the pruning heuristic to be based on the geodesic distance between simplex vertices instead of edge lengths. However, our method would then no longer be grid-free due to the geodesic distance solver.

Conclusion. Our work departs from existing methods for generalized barycentric coordinates through a fresh theoretical perspective and computational approach. By relying on the machinery of neural fields, we realize a practical algorithm and demonstrate its usefulness for a number of applications. Our work represents a key step toward practical deformation-aware generalized barycentric coordinates.

ACKNOWLEDGMENTS

We thank Prof. Mirela Ben-Chen for discussion and feedback during the course of this project.

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 support of the National Science Foundation under grant 2211259, the Singapore DSTA under DST00OECI20300823, the Amazon Science Hub, the Toyota Research Institute, and the MIT-IBM Watson AI Laboratory.

REFERENCES

S Mazdak Abulnaga, Oded Stein, Polina Golland, and Justin Solomon. 2023. Symmetric Volume Maps: Order-Invariant Volumetric Mesh Correspondence with Free Boundary. *ACM Transactions on Graphics (TOG)* 42, 3 (2023).

Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. 2022. Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes. *ACM Trans. Graph.* 41, 4, Article 109 (2022), 17 pages.

Dmitry Anisimov, Daniele Panozzo, and Kai Hormann. 2017. Blended barycentric coordinates. *Computer Aided Geometric Design* 52-53 (2017), 205–216.

Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. 2010. *An Introduction to Total Variation for Image Analysis.* De Gruyter, Berlin, New York, 263–340. https://doi.org/doi:10.1515/9783110226157.263

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A Simple Geometric Model for Elastic Deformations. *ACM Trans. Graph.* 29, 4, Article 38 (jul 2010), 6 pages. https://doi.org/10.1145/1778765.1778775

Lu Chen, Jin Huang, Hanqiu Sun, and Hujun Bao. 2010. Cage-based deformation transfer. *Computers & Graphics* 34, 2 (2010), 107–118.

Nuttapong Chentanez, Miles Macklin, Matthias Müller, Stefan Jeschke, and Tae-Yong Kim. 2020. Cloth and Skin Deformation with a Triangle Mesh Based Convolutional Neural Network. *Computer Graphics Forum* 39, 8 (2020), 123–134.

Harold Scott MacDonald Coxeter. 1969. *Introduction to Geometry.* John Wiley & Sons.

Chongyang Deng, Qingjun Chang, and Kai Hormann. 2020. Iterative coordinates. *Computer Aided Geometric Design* 79 (2020), 101861.

Michael S. Floater. 1997. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design* 14, 3 (1997), 231–250. https://doi.org/10.1016/S0167-8396(96)00031-3

Michael S. Floater. 2003. Mean value coordinates. *Computer Aided Geometric Design* 20, 1 (2003), 19–27.

Michael S. Floater. 2015. Generalized barycentric coordinates and applications. *Acta Numerica* 24 (2015), 161–214. https://doi.org/10.1017/S0962492914000129

Michael S. Floater, Kai Hormann, and Géza Kós. 2006. A general construction of barycentric coordinates over convex polygons. *Advances in Computational Mathematics* 24 (2006), 311–331.

Michael S. Floater, Géza Kós, and Martin Reimers. 2005. Mean value coordinates in 3D. *Computer Aided Geometric Design* 22, 7 (2005), 623–631.

James Gregson, Alla Sheffer, and Eugene Zhang. 2011. All-Hex Mesh Generation via Volumetric PolyCube Deformation. *Computer Graphics Forum* 30, 5 (2011), 1407–1416.

K. Hormann and N. Sukumar. 2008. Maximum Entropy Coordinates for Arbitrary Polytopes. In *Proceedings of the Symposium on Geometry Processing* (Copenhagen, Denmark) *(SGP '08)*. 1513–1520.

K. Hormann and N. Sukumar (Eds.). 2017. *Generalized Barycentric Coordinates in Computer Graphics and Computational Mechanics.* CRC Press, Boca Raton, FL.

Jin Huang, Xiaohan Shi, Xinguo Liu, Kun Zhou, Li-Yi Wei, Shang-Hua Teng, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2006. Subspace Gradient Domain Mesh Deformation. In *ACM SIGGRAPH 2006 Papers*. 1126–1134.

Takeo Igarashi, Tomer Moscovich, and John F. Hughes. 2005. As-Rigid-as-Possible Shape Manipulation. *ACM Trans. Graph.* 24, 3 (jul 2005), 1134–1141. https://doi.org/10.1145/1073204.1073323

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

Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, and Leonidas J Guibas. 2020. ShapeFlow: Learnable Deformation Flows Among 3D Shapes. In *Advances in Neural Information Processing Systems (NeurIPS 2020)*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. 9745–9757.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3 (2007), 71–es.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.* 24, 3 (2005), 561–566.

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

Xian-Ying Li and Shi-Min Hu. 2013. Poisson Coordinates. *IEEE Transactions on Visualization and Computer Graphics* 19, 2 (2013), 344–352.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2021. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*.

Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin. 2007. GPU-Assisted Positive Mean Value Coordinates for Mesh Deformations. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Barcelona, Spain) *(SGP '07)*. 117–123.

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green Coordinates. *ACM Trans. Graph.* 27, 3 (2008), 1–10.

Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020. Neural Subdivision. *ACM Trans. Graph.* 39, 4, Article 124 (2020), 16 pages.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34, 6 (Oct. 2015), 248:1–248:16.

Ran Luo, Tianjia Shao, Huamin Wang, Weiwei Xu, Xiang Chen, Kun Zhou, and Yin Yang. 2020. NNWarp: Neural Network-Based Nonlinear Deformation. *IEEE Transactions on Visualization and Computer Graphics* 26, 4 (2020), 1745–1759.

Andrew L. Maas. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models.

Thomas Müller. 2021. *tiny-cuda-nn.* https://github.com/NVlabs/tiny-cuda-nn

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. 2021. Nerfies: Deformable Neural Radiance Fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5865–5874.

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

Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Xiaowei Zhou, and Hujun Bao. 2021. Animatable Neural Radiance Fields for Modeling Dynamic Human Bodies. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 14314–14323.

Yicong Peng, Yichao Yan, Shenqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. 2022. CageNeRF: Cage-based Neural Radiance Fields for Generalized 3D Deformation and Animation. In *Thirty-Sixth Conference on Neural Information Processing Systems*.

M. Raissi, P. Perdikaris, and G.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.

Chengping Rao, Hao Sun, and Yang Liu. 2021. Physics-Informed Deep Learning for Computational Elastodynamics without Labeled Data. *Journal of Engineering Mechanics* 147, 8 (2021), 04021043.

R. M. Rustamov, Y. Lipman, and T. Funkhouser. 2009. Interior Distance Using Barycentric Coordinates. In *Proceedings of the Symposium on Geometry Processing* (Berlin, Germany) *(SGP '09)*. 1279–1288.

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested Cages. *ACM Transactions on Graphics (TOG)* 34, 6 (2015).

Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *Geometry Processing*, Alexander Belyaev and Michael Garland (Eds.). The Eurographics Association. https://doi.org/10.2312/SGP/SGP07/109-116

Oded Stein, Eitan Grinspun, Max Wardetzky, and Alec Jacobson. 2018. Natural Boundary Conditions for Smoothing in Geometry Processing. *ACM Trans. Graph.* 37, 2, Article 23 (2018), 13 pages.

Oded Stein, Alec Jacobson, Max Wardetzky, and Eitan Grinspun. 2020. A Smoothness Energy without Boundary Distortion for Curved Surfaces. *ACM Trans. Graph.* 39, 3, Article 18 (2020), 17 pages.

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.

Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. 2018. Variational Autoencoders for Deforming 3D Mesh Models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jiong Tao, Bailin Deng, and Juyong Zhang. 2019. A fast numerical solver for local barycentric coordinates. *Computer Aided Geometric Design* 70 (2019), 46–58.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear Subspace Design for Real-Time Shape Deformation. *ACM Trans. Graph.* 34, 4, Article 57 (2015), 11 pages.

Ofir Weber, Mirela Ben-Chen, and Craig Gotsman. 2009. Complex Barycentric Coordinates with Applications to Planar Shape Deformation. *Computer Graphics Forum* 28, 2 (2009), 587–597.

Ola Weistrand and Stina Svensson. 2015. The ANACONDA algorithm for deformable image registration in radiotherapy. *Medical Physics* 42, 1 (2015), 40–53.

Martin Wicke, Mario Botsch, and Markus Gross. 2007. A Finite Element Method on Convex Polyhedra. *Computer Graphics Forum* 26, 3 (2007), 355–364.

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.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. In *Advances in Neural Information Processing Systems (NeurIPS 2021)*, Vol. 34. 22483–22497.

Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural Cages for Detail-Preserving 3D Deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 75–83.

Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. 2022. NeRF-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18353–18364.

Juyong Zhang, Bailin Deng, Zishun Liu, Giuseppe Patanè, Sofien Bouaziz, Kai Hormann, and Ligang Liu. 2014. Local Barycentric Coordinates. *ACM Trans. Graph.* 33, 6, Article 188 (2014), 12 pages.

Paul Zhang, Josh Vekhter, Edward Chien, David Bommes, Etienne Vouga, and Justin Solomon. 2020. Octahedral Frames for Feature-Aligned Cross Fields. *ACM Trans. Graph.* 39, 3, Article 25 (apr 2020), 13 pages. https://doi.org/10.1145/3374209