Neural Jacobian Fields: Learning Intrinsic Mappings of Arbitrary Meshes

NOAM AIGERMAN, Adobe Research, USA
KUNAL GUPTA*, University of California San Diego, USA
VLADIMIR G. KIM, Adobe Research, USA
SIDDHARTHA CHAUDHURI, Adobe Research, IIT Bombay, India
JUN SAITO, Adobe Research, USA
THIBAULT GROUEIX, Adobe Research, USA

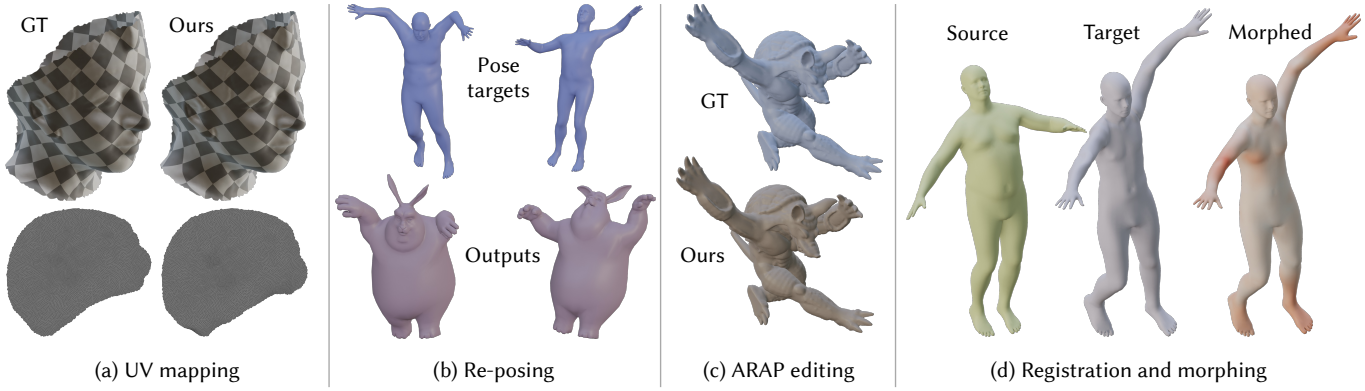

Fig. 1. We present a versatile, detail-preserving and triangulation-agnostic neural framework for learning piecewise-linear mappings of meshes. Using the *exact same core framework*, we show, from left to right: (a) learning to UV-map arbitrary surfaces; (b) re-posing a novel character (the bunny), unseen during training and with a completely different mesh topology, using a model trained only on humans; (c) learning to replicate As-Rigid-As-Possible [Sorkine and Alexa 2007] deformations; and (d) morphing a human shape (green) to a target pose and body shape (blue), without input correspondences.

This paper introduces a framework designed to accurately predict piecewise linear mappings of arbitrary meshes via a neural network, enabling training and evaluating over heterogeneous collections of meshes that do not share a triangulation, as well as producing highly detail-preserving maps whose accuracy exceeds current state of the art. The framework is based on reducing the neural aspect to a prediction of a matrix for a single given point, conditioned on a global shape descriptor. The field of matrices is then projected onto the tangent bundle of the given mesh, and used as candidate jacobians for the predicted map. The map is computed by a standard Poisson solve, implemented as a differentiable layer with cached pre-factorization for efficient training. This construction is agnostic to the triangulation of the input, thereby enabling applications on datasets with varying triangulations. At the same time, by operating in the intrinsic gradient domain of each individual mesh, it allows the framework to predict highly-accurate mappings. We validate these properties by conducting experiments over a broad range of scenarios, from semantic ones such as morphing, registration, and deformation transfer, to optimization-based ones, such as emulating elastic deformations and contact correction, as well as being the first work, to our knowledge, to tackle the task of learning to compute UV parameterizations of arbitrary meshes. The results exhibit the high accuracy of the method as well as its versatility, as it is readily applied to the above scenarios without any changes to the framework.

CCS Concepts: • **Computing methodologies** → **Mesh geometry models**; **Neural networks**; **Mesh models**.

Additional Key Words and Phrases: deformation, UV parameterization, morphing

ACM Reference Format:
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 (July 2022), 17 pages. https://doi.org/10.1145/3528223.3530141

1 INTRODUCTION

Computing mappings between 3D domains is a fundamental concept at the core of graphics and geometry processing, used in a wide range of generative and discriminative tasks, including 3D modeling, deformation, animation, UV-texturing, registration, correspondence-computation, remeshing, simulation, and fabrication.

Many such mapping tasks involve human priors which do not admit compact analytical representations, or otherwise require

*Participated as part of an internship at Adobe.

Authors' addresses: Noam Aigerman, Adobe Research, USA, aigerman@adobe.com; Kunal Gupta, University of California San Diego, USA, k5gupta@eng.ucsd.edu; Vladimir G. Kim, vokim@adobe.com, Adobe Research, USA; Siddhartha Chaudhuri, sidch@adobe.com, Adobe Research, IIT Bombay, India; Jun Saito, jsaito@adobe.com, Adobe Research, USA; Thibault Groueix, groueix@adobe.com, Adobe Research, USA.

0730-0301/2022/7-ART109 $15.00
https://doi.org/10.1145/3528223.3530141

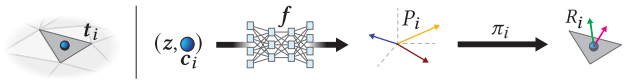

Fig. 2. The neural aspect of our framework is summed up in applying an MLP to a single triangle at a time. The MLP, f, receives as input the centroid feature c_i of the triangle t_i, along with the global code z. It outputs a matrix $P_i \in \mathbb{R}^{3\times3}$, visualized as three vectors in $\mathbb{R}^3$, which is then restricted to its action on the tangent space of the triangle t_i, through the restriction operator π_i, to yield a matrix $R_i \in \mathbb{R}^{3\times2}$, visualized as two vectors in $\mathbb{R}^3$.

computationally-demanding numerical optimization at runtime. Hence, approaching such tasks in a data-driven manner, using predictive tools from modern statistical learning to accommodate semantic priors and approximate optimized solutions in a single forward pass, can have wide impact. Immediate potential applications include modifying 3D shapes to match a target such as an image, inferring high quality UV mappings from examples authored by artists, and integrating with direct optimization to speed-up tasks such as physical simulation.

In order to achieve accurate, high-quality results, it stands to reason to harness deep neural networks, which have proven immensely effective for complex regression tasks, and to learn mappings of 3D triangular meshes, which are arguably the most common 3D representation chosen by artists when fine detail and high accuracy are desired. This also has the benefit of directly integrating into the graphics pipeline, without the need to convert back and forth between different representations.

1.1 Challenges

Inferring mappings of meshes with neural networks is still, to a large extent, an open problem. There are fundamental issues that make 3D surfaces (that is, 2-manifolds immersed in 3D) a harder domain to efficiently learn over than, e.g., 2D images. Methods attempting to predict mappings encounter two main challenges: **(1)** 3D surfaces, as opposed to the standard 2D pixel grid, have significant geometric and topological variation. There is no canonical way to associate corresponding points between a pair of surfaces, and the two instances may differ in their discretization (i.e., have different triangulations). Standard 2D convolutional architectures make assumptions about point ordering and parametrization that do not apply, in general, to collections of 3D surfaces. **(2)** Mappings of surfaces need to be *detail preserving*. That is, fine geometric details should be preserved while mapping the coarse (lower-frequency) structure of the surface through a highly-nonlinear map.

Thus, previous learning-based mesh mapping methods devise specific architectures and mapping spaces to tackle one or both of these challenges. Some methods opt to work with one, fixed triangulation, which prevents their application to scenarios where the runtime triangulation is not given in advance, or where the training data themselves have diverse triangulations. Other methods define deformations over the ambient 3D space [Jiang et al. 2020]. However, ambient fields are not detail-preserving in general, and the burden falls on the network to ensure that the specific predicted fields protect mesh details.

Other methods choose a subset of ambient-space deformations which are detail-preserving by construction, e.g., deformations induced by sparse rigs such as cages [Yifan et al. 2020]. These methods are still not fully shape-aware, but rather rely on the rig inducing a highly restricted deformation space. This in turn implies their success hinges on fitting the rig to the mesh to expose exactly the correct deformation subspace. Further, in cases where the rig is not applicable due to vast differences in shape or topology, the network cannot learn meaningful maps.

In order for a method to map meshes in a truly shape-aware manner, it needs to be defined intrinsically over the manifold surface itself. Non-ML mapping techniques commonly operate in the *gradient domain* of the manifold, by considering the jacobians of the map w.r.t. the intrinsic gradient operator of the mesh. However, incorporating this technique into a neural network revives the first challenge of operating on heterogeneous, non-regular mesh topologies, as the intrinsic gradient operator is defined for each triangle w.r.t. its triangulation-specific tangent spaces.

1.2 Approach

In this work, we aim to tackle both challenges mentioned above, and devise a method for learning *intrinsic, highly-accurate, detail-preserving* mappings, which is provably and experimentally shown to be *triangulation-agnostic*.

Our method is based on the following straightforward observation: we can have a multi-layer perceptron (MLP) make a prediction of an *extrinsic* field over ambient space, unaware of any triangulation, and then, for any desired mesh, we can use the mesh's differential operators to "carve out" an *intrinsic* field of jacobians and compute a mapping of the given mesh from them. By postponing any use of the mesh until *after* the MLP prediction is made, we on one hand enable a mesh-aware, intrinsic loss that supports backpropagation, but on the other hand, prevent the MLP from any ability to use the mesh for inference, thereby forcing it to learn in a manner that cannot rely on a specific triangulation.

Concretely, our neural component (visualized in Figure 2) comprises of the aforementioned MLP, which receives a single point in space as input and produces a 3×3 matrix, thereby inducing a matrix field, in similar fashion to other geometric deep learning methods which make point-wise predictions [Groueix et al. 2018b; Park et al. 2019]. We then define a simple restriction operator which restricts the *extrinsic* field of linear transformations to an *intrinsic* field of linear transformations, defined over the tangent bundle of the given mesh, representing the prediction of candidate jacobians. We further observe that the linear solve of a Poisson system, required to retrieve a continuous map from these projected pseudo-jacobians, can be implemented efficiently as a differentiable layer whose weights can be precomputed for each mesh. This enables us to train and perform inference of maps over a collection of meshes with varying triangulations, in spite of the Poisson linear system varying for each mesh in the dataset.

This construction achieves several goals simultaneously. (1) We make the framework fully agnostic to the triangulation, as the same trained network can be applied to a triangulation of any connectivity, and furthermore it is not biased by any triangulation-specific signal during training. (2) The MLP's prediction is interpreted intrinsically in the mesh's gradient domain, resulting in a detail-preserving, highly accurate map. Furthermore, since most mappings considered

in practical applications vary gradually across the input shapes, their jacobians are low-frequency, smooth signals. As our MLP learns to map points in ambient space to the jacobians, it can exploit this smoothness and learn to reproduce them to high accuracy without exceeding its capacity. (3) Lastly, beyond mesh-agnosticism and accuracy, we also show our method is extremely versatile, as we do not change any aspect of the core framework (other than supplying it an additional input in the form of a pose vector for some of the human retargeting experiments) for any of the experiments in the paper.

We show these three properties of our framework – accuracy, versatility, and triangulation-agnosticism – by exhibiting its capabilities on a large variety of experiments, ranging from morphing, re-posing, reshaping, and deformation transfer of humans, to emulating physical simulations such as non-rigid deformations and collision handling. Lastly, we show that our method is, to the best of our knowledge, the first that can effectively learn to produce UV mappings by training over a heterogeneous collection of meshes.

2 RELATED WORK

Map computation through jacobians. Map computation is a keystone of many central areas of geometry processing and graphics, such as UV parameterization [Sheffer et al. 2007], deformations induced from either physical simulation [Kim and Eberle 2020] or artistic manipulation [Sorkine and Botsch 2009], registration [Bouaziz et al. 2016], and computing polycubes [Tarini et al. 2004]. Several geometry processing methods enable artist-driven mesh-editing through the gradient domain [Lipman et al. 2004; Sorkine et al. 2004; Yu et al. 2004]. Furthermore, mappings are commonly defined by a variational principle as a minimizer of an energy, usually defined in terms of the jacobians, e.g., Dirichlet [Pinkall and Polthier 1993], ARAP [Liu et al. 2008a], LSCM [Lévy et al. 2002], and symmetric Dirichlet [Rabinovich et al. 2017; Smith and Schaefer 2015]. Other methods aim to restrict the jacobians to ensure that they do not invert [Du et al. 2020; Schüller et al. 2013], or additionally bound the distortion of the jacobian [Aigerman and Lipman 2013; Kovalsky et al. 2014; Lipman 2012; Myles and Zorin 2013; Weber and Zorin 2014]. A central algorithm used in these methods is the block-descent local/global algorithm [Liu et al. 2008b] which iteratively applies two steps, a local step that makes a per-triangle update to the jacobians into non-integrable matrices, followed by a global step which harnesses Poisson's equation to find the mapping which best-respects the local update. Our method can be viewed as an adaptation of a single iteration of this algorithm with a local, neural prediction step followed by solving Poisson's equation, along with a construction that enables us to apply it to a collection of meshes with different triangulations.

Data-driven deformations. Deformations play a central role in animation, graphics and physics. Leveraging data-driven methods can enable automating deformation transfer [Gao et al. 2018; Sumner and Popović 2004], inferring deformation spaces which guide asset retrieval [Uy et al. 2020], producing meshes composed of deformable parts [Gao et al. 2019], use image input to reconstruct objects or deform templates [Kanazawa et al. 2016; Wang et al. 2018], and learn deformation subspaces without losing plausibility [Holden et al. 2019].

Rigged deformations. In non-ML context, Skinning-based methods [Fulton et al. 2019; Jacobson et al. 2014] use a rig with handles such as points [Jacobson et al. 2011], skeletons [Kavan et al. 2008], or enclosing cages [Ju et al. 2005; Lipman et al. 2008], which induce weights that propagate rig deformations to the underlying shape, thereby enabling its applicability to arbitrary meshes as well as point clouds. This process heavily relies on the accurate fitting of the rig, which is a subtle task. To automate it, heuristic-driven [Baran and Popović 2007] and neural techniques have been developed to predict and deform various rigs such as skeletal ones [Holden et al. 2015; Li et al. 2021; Xu et al. 2020, 2019], point handles [Jakab et al. 2021; Liu et al. 2021], and cages [Yifan et al. 2020]. This approach can also be employed to use a human model as a rig which deforms their clothing [Liu et al. 2019]. Inaccuracy in the rigging leads to severe artifacts, and the rig further usually restricts the architecture to a certain class of deformations, e.g., articulations for bones. None of these rigging methods is readily applicable to general mapping tasks, such as UV parameterization.

Learning deformations of a single mesh. Many methods focus on a single mesh, or a set of variations of the same mesh [Bogo et al. 2014; Osman et al. 2020; Varol et al. 2017; Zuffi et al. 2017], and define an arbitrary *fixed correspondence* of vertices and/or faces to the entries of a predicted tensor, thereby enabling machine learning algorithms to associate predicted quantities to geometric elements. This enables treating the input and output as general tensors of a homogeneous dataset and applying off-the-shelf tools for data analysis, such as Principal Component Analysis [Anguelov et al. 2005], Gaussian Mixture Models [Bogo et al. 2016], as well as neural networks, which assign per-vertex coordinates [Shen et al. 2021] or offsets from a simpler (e.g., linear) model [Bailey et al. 2020, 2018; Romero et al. 2021; Yin et al. 2021; Zheng et al. 2021]. As they are not shape-aware in the sense discussed in Section 1, such methods often need to add additional regularizers such as ARAP [Sun et al. 2021] or the Laplacian [Kanazawa et al. 2018]. To get around the limitation to a single shape, some methods use a single template to deform partial parts of shapes [Litany et al. 2018].

Similarly to us, [Gao et al. 2018; Tan et al. 2018] also consider jacobians of deformations. They propose to autoencode features extracted from jacobians in order to learn a latent deformation space. However, unlike us, they consider a single mesh, with a fixed assignment of the predicted tensor's entries to the mesh's elements. Thus, they are restricted to only train on this one mesh's triangulation, and cannot be applied to a collection unless the triangulations are in one-to-one correspondence. In contrast, our framework predicts continuous jacobian fields, and thus is readily applicable to heterogeneous collections with diverse triangulations. Furthermore, in Section 4.4 we show that even when restricted to a single mesh, predicting jacobian fields significantly outperforms the fixed-assignment approach.

Discretization agnostic representations. Many works seek to avoid the discrete nature of meshes. Deformation fields are commonly used to generalize across different triangulations and even geometric domains. Specifically, one can learn to predict an implicit

vector field which maps every point in the volume to its new location [Groueix et al. 2018a, 2019; Huang et al. 2020; Jiang et al. 2020; Yang et al. 2021]. This representation acts in a point-wise manner, and is not aware of the underlying surface, thus these maps tend to not preserve local surface details. By operating in the gradient domain our method is detail-preserving by construction. Other methods avoid meshes by defining the surface itself as a mapping of a plane through a neural network [Groueix et al. 2019; Morreale et al. 2021; Williams et al. 2019], or an SDF [Park et al. 2019], however their implicit and non-discrete representation makes it non trivial to manipulate them accurately. In the context of deep-learning, Atlasnet-like works explored leveraging jacobians [Bednarik et al. 2020], however this is to regulate their atlases and does not enable defining mappings of these surfaces. Networks that are agnostic to the discretization of surfaces have recently been studied in [Sharp et al. 2022], however that work is focused on analysis of the surface by designing discretization-invariant diffusion operators, and less fitting for producing deformations of high-resolution surfaces (see also Fig 14 for a comparison of the two methods).

3 APPROACH

We begin describing our method by first elaborating on the main algorithm of our framework, which enables us to predict shape-aware piecewise-linear maps, in a manner completely agnostic to triangulation.

At the prediction stage, we are given as input a mesh $\mathcal{S}$, and a *global code* $\boldsymbol{z}$, which is a vector that is compiled differently for each of the experiments – we elaborate on its computation in Section 4. As an example, $\boldsymbol{z}$ can be the concatenation of the angles at the joints of a skeleton, encoding the desired position to which a given mesh of human should bend. We denote the ground truth map (e.g., the correct mapping of the rest pose human to the desired pose) as Ψ.

Our goal is to design a network that, conditioned on $\boldsymbol{z}$, will predict a map Φ that matches the ground truth Ψ. Furthermore, we set it as our goal to design this network in a way that will enable it to learn and perform inference in complete agnosticism to the source domain's triangulation, but that will still produce a shape-aware, detail-preserving map through the gradient domain of the mesh.

We progress in a bottom-up manner, beginning with a brief review of a few basic concepts and definitions of piecewise-linear mappings needed for our construction, then move on to describing our neural architecture for predicting mesh-agnostic piecewise linear maps, and finish with a discussion on how to train it on a dataset of mappings.

3.1 Preliminaries

We assume the mesh $\mathcal{S}$ is a 2-manifold triangular mesh of a single connected component, embedded in $\mathbb{R}^3$, with vertices $\mathbf{V}$ and triangles $\mathbf{T}$. The *tangent space* at a triangle $\boldsymbol{t}_i \in \mathbf{T}$ is the linear space orthogonal to its normal, denoted T_i. We choose two column vectors which form an oriented orthonormal basis to the tangent space, which we call a *frame*, $\mathcal{B}_i \in \mathbb{R}^{3\times 2}$ of the triangle.

Piecewise linear maps. As is common in geometry processing, we focus on the space of *piecewise-linear mappings* of a given mesh, meaning that the restriction of the map Φ to any triangle $\boldsymbol{t}_i$, denoted $\Phi|_{\boldsymbol{t}_i}$, is affine, i.e., is the sum of a linear map and a constant translation. This is arguably the most common family of maps used when considering mappings of meshes. A piecewise linear mapping Φ of a mesh can be uniquely defined by assigning a new position to each one of the vertices, $\mathbf{V}_i \to \Phi_i$. The mapping is then well-defined also for any point inside a triangle by linearly interpolating the map from the vertices. From now on, we abuse notation and interchangeably use Φ_i to denote the i'th vertex's new position, and consider Φ as a matrix of same dimensions as $\mathbf{V}$.

Jacobians. Analogously to the smooth setting, the jacobian at triangle $\boldsymbol{t}_i$ of the map Φ is a linear transformation of dimensions 3×2 from the triangle's tangent space to $\mathbb{R}^3$, denoted $J_i(x) : T_i \to \mathbb{R}^3$, defined as the linear part of the map restricted to that triangle, $\Phi|_{\mathbf{t}_i}$.

For a given map Φ, the jacobian J_i can be explicitly computed as a 3×2 matrix in the coordinates of the frame $\mathcal{B}_i$ by solving

$$J_i \mathcal{B}_i^T \left[v_k - v_j, v_l - v_j\right] = \left[\phi_k - \phi_j, \phi_l - \phi_j\right] \tag{1}$$

where v_j, v_k, v_l are the triangle's vertices, and ϕ_j, ϕ_k, ϕ_l are their images under Φ. Solving the above linear system yields a linear operator denoted ∇_i, that maps Φ to the jacobian J_i in the basis $\mathcal{B}_i$:

$$J_i = \Phi \nabla_i^T. \tag{2}$$

∇_i is defined as the gradient operator of triangle $\boldsymbol{t}_i$, expressed in the basis $\mathcal{B}_i$. The gradient operator ∇ of the mesh is thus defined as the linear operator mapping Φ to the stack of all jacobians.

Poisson equation. Given an arbitrary assignment of a matrix $M_i \in \mathbb{R}^{3\times 2}$ to each triangle, one can retrieve the map Φ^* whose jacobians $J_i = \Phi^* \nabla_i^T$ are closest to M_i in the least-squares sense, by solving the widely-used *Poisson equation*,

$$\Phi^* = \min_{\Phi} \sum |\boldsymbol{t}_i| \left\| \Phi \nabla_i^T - M_i \right\|^2, \tag{3}$$

where $|\boldsymbol{t}_i|$ is the area of the triangle $\boldsymbol{t}_i$ on the source mesh $\mathcal{S}$.

The solution is obtained by solving the linear system

$$\Phi^* = L^{-1} \mathcal{A} \nabla^T M, \tag{4}$$

where $\mathcal{A}$ is the mesh's mass matrix, $L = \nabla^T \mathcal{A} \nabla$ is the mesh's cotangent Laplacian, and M is the stacking of the input matrices M_i. The solution is well-defined up to a global translation which can be resolved by setting, e.g., $\Phi_0^* = \vec{0}$ and modifying the above equations accordingly.

3.2 Predicting Triangulation-Agnostic, Intrinsic Mappings

We now begin elaborating on our framework. See Figure 3 for the detailed pipeline. Intuitively, one may find analogies between the prediction algorithm and one, *single* iteration of the local-global algorithm widely used in geometry processing, e.g., ARAP [Liu et al. 2008b], with our custom local step (shown in Figure 2) including an MLP and a restriction operator from ambient space into the triangle's tangent space.

The design of our framework should posses two supposedly-conflicting properties: 1) it should be completely agnostic to the triangulation of the meshes, and neither the framework's input, output, nor structure should be modeled with respect to a specific triangulation; 2) it should predict *intrinsic*, gradient-domain mappings so as to accurately preserve details. However, working in

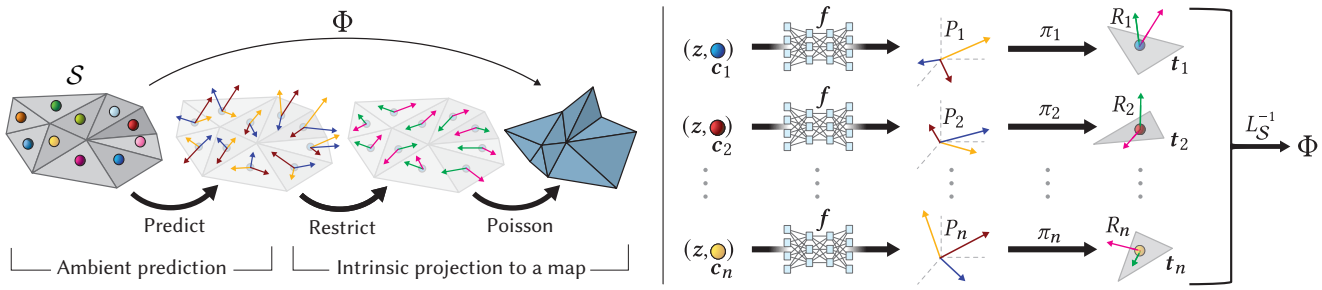

Fig. 3. The main inference algorithm for predicting mappings, depicted once through its geometric action (left) and once through its dataflow (right). For a given mesh $\mathcal{S}$ and a global code $\boldsymbol{z}$, the centroid feature $\boldsymbol{c}_i$ of each triangle $\boldsymbol{t}_i$ of the mesh is concatenated to $\boldsymbol{z}$ and plugged into the MLP $\boldsymbol{f}$ in parallel. The MLP predicts a matrix $P_i \in \mathbb{R}^{3\times3}$ thereby inducing a field of matrices over all triangles. The matrices are then *restricted*, in parallel, via π_i to the tangent spaces of their respective triangles, yielding $R_i \in \mathbb{R}^{3\times2}$. R_i are then plugged to the Poisson solve which computes a map Φ, by solving a linear equation w.r.t. the given mesh's own Laplacian $L_{\mathcal{S}}^{-1}$, intrinsically projecting the network's prediction to a mesh-specific, detail-preserving mapping Φ of the mesh.

the gradient domain entails working in the tangent spaces T_i using gradient operators ∇_i, which are *triangle specific*, entailing a naive approach will couple the method to a specific triangulation.

We address both these challenges in an algorithm consisting of two stages: 1) an MLP, conditioned on the global code $\boldsymbol{z}$, predicts a 3×3 matrix for any given point p, thereby inducing an ambient field of extrinsic matrices. 2) The extrinsic matrix field is reinterpreted and restricted by mesh-specific, basic linear-algebra operations to an *intrinsic* field of 3×2 matrices defined over the tangent spaces T_i of the mesh. The intrinsic matrices are input to the Poisson solve (4) of the mesh, implemented as a differentiable layer over the GPU, yielding a mapping of the mesh through the gradient domain.

Intuitively, the MLP makes its prediction at the end of step 1, and receives only a single triangle centroid each time, thus it cannot make any global inference regarding the triangulation itself based on the input. The only global inference stems from the concatenated code $\boldsymbol{z}$, which in all our experiments does not contain any information about the triangulation itself, rather solely the shape the triangulation represents, ensuring the MLP is blind to the triangulation, thereby making the framework *triangulation agnostic*. Step 2) which transpires outside of the neural component, uses the specific mesh's differential operators along with simple linear algebra to pull the prediction of step 1) into the specific gradient domain of the specific mesh, thereby yielding a shape-aware intrinsic mapping, as desired. Next, we elaborate on each step in sequence.

Step 1: Predicting matrices in $\mathbb{R}^{3\times3}$. In similar fashion to other networks that make per-point predictions [Groueix et al. 2018b; Park et al. 2019] we use an MLP $\boldsymbol{f}$ which receives as input a single point p, concatenated to the global code $\boldsymbol{z}$, and outputs a 3×3 real matrix, thereby inducing a field of matrices over ambient space and not tied to a specific mesh. Given a specific mesh $\mathcal{S}$ to deform, independently for each triangle $\boldsymbol{t}_i$, we feed each of its centroids $\boldsymbol{c}_i$ as the point p, along with the global code $\boldsymbol{z}$, thereby assigning a matrix $P_i \in \mathbb{R}^{3\times3}$ to the triangle $\boldsymbol{t}_i$,

$$P_i = \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{c}_i) \in \mathbb{R}^{3\times3}. \tag{5}$$

Each centroid $\boldsymbol{c}_i$ is represented as a fixed, precomputed vector, consisting of a concatenation of the centroid's 3D position, its normal, and a Wave-Kernel signature [Aubry et al. 2011]. For performance gain, we can of course apply the MLP in parallel on a large batch of centroids over the GPU. As our goal is to define an intrinsic jacobian for each triangle, we next *restrict* each extrinsic matrix to its action on the tangent space of its corresponding triangle.

Step 2: Compute a map from the extrinsic field, using the mesh's intrinsic differential operators. The predicted extrinsic matrix P_i of triangle $\boldsymbol{t}_i$ can be projected to an intrinsic linear map, by considering its *restriction* to the subspace of T_i, expressed in the frame $\mathcal{B}_i$:

$$\pi_i(P_i) \triangleq P_i\mathcal{B}_i. \tag{6}$$

We denote P_i's restriction as $R_i \in \mathbb{R}^{3\times2}$.

With the restricted matrices $\{R_i\}$ at hand, to produce the final map, we plug them into the Poisson system, Eq. (4), defined via the Laplacian and gradient operator of $\mathcal{S}$. Fortunately, we observe that the Poisson solve can be implemented as a custom differentiable layer, as Equation (4) represents a linear transformation, and hence when back-propagating the incoming gradient g, the back-propagation from this linear layer amounts to solving (4) again for a different right hand side input, defined by the gradient.

To solve Equation (4) rapidly, as is desired during consecutive evaluations (e.g., during training or inference), we follow the common practice of computing a decomposition of L into two tridiagonal systems in advance (we use an LU decomposition), during the preprocessing of the data before training. Then, during training or evaluation, the Poisson solve can be quickly obtained by loading the decomposition into GPU memory, and performing backsubstitution on the two tridiagonal systems, on the GPU.

We summarize the inference algorithm in Algorithm 1 and the preprocessing procedure in Algorithm 2.

Algorithm 1 Inference

Input: codeword $\boldsymbol{z}$, source mesh $\mathcal{S}$, $\nabla_{\mathcal{S}}$, LU decomposition of $L_{\mathcal{S}}$
for each centroid $\boldsymbol{c}_i \in C$, in parallel on the GPU, **do**
  Apply $\boldsymbol{f}$ to concatenation of $(\boldsymbol{z}, \boldsymbol{c}_i)$ to get P_i via Eq. (5).
  Restrict P_i to the tangent space via Eq. (6) to get R_i.
end for
Compute Φ via Eq. (4) using $\nabla_{\mathcal{S}}$ and the LU decomposition of $L_{\mathcal{S}}$.
Output: the map Φ assigning a new position to each vertex of $\mathcal{S}$.

Algorithm 2 Mesh Preprocessing

Input: Mesh $\mathcal{S}$ to preprocess.
Compute and store the centroid features $C_{\mathcal{S}} = \{c_i\}$.
Compute and store the frames $\mathcal{B}_{\mathcal{S}} = \{\mathcal{B}_i\}$.
Compute and store the gradient operator $\nabla_{\mathcal{S}} = \{\nabla_i\}$.
Compute and store the Laplacian $L_{\mathcal{S}}$ and its LU-decomposition.

For a collection of meshes, we run Algorithm 2 on each one of them in advance and store all computed data. Then, during training or inference, for each mesh, we load its data and run Algorithm 1.

3.3 Training

Figure 4, bottom, shows the training procedure of the framework. The training is conducted over a dataset of maps, defined via triplets, $\left\{(\mathcal{S}^i, \Psi^i, z^i)\right\}_{i=1}^n$ comprised of a mesh, its corresponding ground-truth mapping, and the global code on which the prediction of the network should be conditioned. In most experiments, z^i is obtained by a PointNet encoder which is trained in tandem with the deformation network, however any encoder or shape representation can be used. During training, we iterate over the triplets and for each triplet, we train the network to predict a map Φ^i of the mesh $\mathcal{S}^i$, conditioned on the global code z^i, using a loss defined w.r.t. the ground-truth map, Ψ^i.

Losses. We optimize for two losses: first, the vertex-vertex loss between the prediction Φ and the ground truth map Ψ,

$$l_{\text{vertex}} = \sum |\mathbf{V}_j| \left\|\Phi_j - \Psi_j\right\|^2, \tag{7}$$

where $|\mathbf{V}_i|$ is the lumped mass around the i'th vertex in $\mathcal{S}$. Since the network's prediction is well-defined up to a global translation (see last paragraph of Section 3.1), we shift both the prediction and the ground truth so that their meshes' center of mass is at the origin, before computing this loss.

We also measure the difference between the restricted predictions $\{R_i\}$ and the ground truth jacobians $J_i = \left\{\Psi \nabla_i^T\right\}$,

$$l_{\text{jacobian}} = \sum |t_j| \left\|R_j - J_j\right\|^2. \tag{8}$$

Our total loss is

$$l_{\text{total}} = 10 \cdot l_{\text{vertex}} + l_{\text{jacobian}}. \tag{9}$$

We use this loss in all experiments. Note that both losses are achieved through the use of the specific mesh's differential operators, after inference. We summarize the training in Algorithm 3.

Algorithm 3 Training epoch

for each mesh and ground-truth map $\mathcal{S}, \Psi$ in the dataset **do**
  Load the LU decomposition and ∇ of $\mathcal{S}$ to GPU memory.
  Encode $\mathcal{S}, \Psi$ into a codeword z. ▷ experiment-specific
  Input $\mathcal{S}$ and z to Algorithm 1 to compute the mapping Φ.
  Compute the loss l_{total}, Eq. (9).
  Optimize parameters of f and encoder via back-propagation.
end for

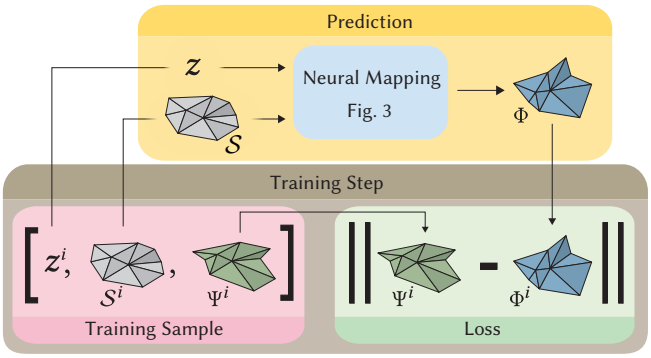

Fig. 4. Top: at inference, the network outputs a mapping Φ of a mesh $\mathcal{S}$ given as input, conditioned on the global code z. Bottom: during training, we iterate over triplets, each consisting of a mesh $\mathcal{S}^i$, groundtruth mapping Ψ^i, and the associated global code z^i, and predict a mapping Φ^i, which is then compared with respect to the ground truth through the $L2$ distance between their vertices and their jacobians. Each training or evaluation sample may have a completely different triangulation without affecting the prediction.

Encoding the global code z and the centroids. The global code z is constructed differently in each of our experiments and we detail it for each of them in Section 4. At a high level, we use two types of encodings: First, we use fixed, pre-given dataset-specific parameters (such as SMPL pose parameters); second, we use Pointnet [Qi et al. 2017] when there's need to encode the shape of either $\mathcal{S}$, or Ψ, or both. In those cases, we sample 1024 points on the mesh and compute a Wave-Kernel Signature [Aubry et al. 2011] of size 50 for each point, and feed those along with the points' 3D position and normal into pointnet, which we train along with the MLP f. Similarly, for each centroid of a triangle fed into the MLP, we attach its 3D coordinates, normal, and a Wave-Kernel Signature of size 50 as one vector.

3.4 Discussion: restriction through the frames $\{\mathcal{B}_i\}$

For a tangent space T_i, there exist infinitely-many orthonormal bases, and we choose one arbitrarily. The frame $\mathcal{B}_i$ changes the representation both of the ground-truth jacobians as well as the predicted R_i, and hence, our framework has to be invariant to the choice of $\mathcal{B}_i$, in order to ensure it does not learn a frame-specific representation which would strictly prevent it from generalizing and learning over collections of meshes. Therefore, we provide a trivial proof in Appendix A showing that our framework is completely invariant to the choice of frames.

Claim. *Our framework is completely invariant to the choice of frames $\{\mathcal{B}_i\}$.*

Additionally, we note that the representation of R_i and the jacobians in frames in the tangent space follows the standard intrinsic definition used in differential geometry. However, the intrinsic restriction operation π_i could alternatively be described extrinsically, on 3×3 matrices, by keeping R_i in world coordinates and simply nullifying its action on the triangle's normal, $R_i = P_i \left(I - \vec{n}_i \vec{n}_i^T\right)$. Both restriction approaches are comparable in the time it takes to perform them. However, in this extrinsic approach the restricted

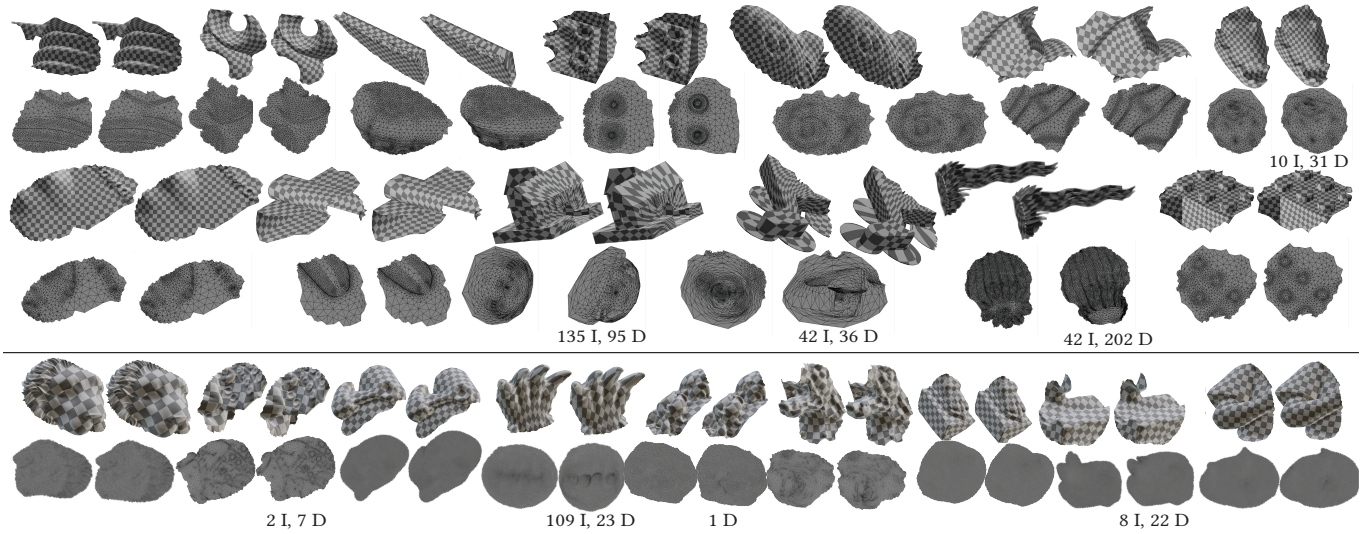

Fig. 5. UV maps generated by our network (right in each pair), compared to SLIM [Rabinovich et al. 2017] (left in each pair). For each example we show the 3D model textured using the UV map, and the 2D UV map below it. In case there were inverted elements, we report their number next to "I", and in case there were triangles with high (> 10) distortion, we report their number next to "D". Top: results on the test set from the dataset generated for this experiment, of patches extracted from Thingi10K [Zhou and Jacobson 2016]. Bottom row: evaluation on a set of meshes generated from other datasets than Thingi10K.

matrix would be a matrix with 9 entries, and a singular value of 0, while in the intrinsic approach the restriction has 6 entries and full rank. We choose the more compact and stable representation.

3.5 Technical Details

We use a 5-layer fully-connected MLP with ReLU activation and group norm [Wu and He 2018] after each layer. Hidden layers are of size 128, with the first layer's input dimensions depending on the size of $\boldsymbol{z}$, and the last layer's output being the 9 entries of P_i. We add the identity matrix to the prediction so that when the MLP outputs the zero matrix the prediction is the identity.

The only encoding we use except for raw parameters like a human's pose parameters, is a PointNet [Qi et al. 2017] encoding of the given shapes, which receives 1024 points sampled uniformally on the mesh, along with their normals and Wave-Kernel Signatures [Aubry et al. 2011] of size 50. We modify PointNet to use group norms as well. Both the MLP and Pointnet are trained simultaneously.

We use PyTorch [Paszke et al. 2019], along with CuPy [Okuta et al. 2017] and *torch-sparse* which are needed to represent the various differential operators and sparse matrices on the GPU. When the dataset we train on encompasses a one-to-many mapping (e.g, when mapping an SMPL model to multiple poses), we train on batches (usually of size 32) that include a single source mesh and multiple target mappings, thereby reusing the differential operators and performing the LU-solve over a batch. We train using the Adam [Kingma and Ba 2015] optimizer, which we initialize with a learning rate of 10^{-3}. Once plateaued, we reduce the learning rate to 10^{-4}, and train until plateauing again.

While we cache the LU-decomposition during preprocessing, we note that the computational overhead of the LU decomposition is within reason for performing it on-the-fly during training and evaluation. This could be useful in scenarios in which the source mesh is assumed to dynamically change during training, such as within a remeshing framework.

Lastly, we note that while symmetric matrices are usually factorized via an LDL decomposition, we could not find an implementation that fit into our differentiable framework, and thus opted to use an LU decomposition using SciPy's SuperLU decomposition.

4 RESULTS

We now turn to evaluate our framework's efficacy in various applications and tests, designed to exhibit its three key traits, namely: detail preservation, mesh-agnosticism, and versatility as a general framework for learning mappings. To show the last trait, we opt to evaluate our framework on a broad range of scenarios, and do not introduce *any* custom modification for any of these experiments, beyond changing the input encoding. We further emphasize that to encode the shapes we solely use a PointNet encoder which we do not consider part of our contribution, as we focus on our ability to decode mappings and not on encoding.

Metrics. We measure error via the L2 distance between a vertex and its ground truth position, after normalizing both the predicted mesh and ground-truth mesh to the unit sphere. We report in Table 2 the average L2 distance summed over vertices and shapes in the dataset. We also report the L2 distance between predicted and ground-truth jacobians, as well as the average angular error on the normals.

4.1 Learning UV parameterizations

Computing UV parameterizations is a fundamental task within the graphics pipeline which, to the best of our knowledge, has not been explored successfully by deep-learning techniques as of now. This

	Distorsion		Flips	
↓	avg. D>10	med. D>10	avg. #I	#I>0 %
Thingi10K				
SLIM (GT)	0.04	0.0	0.0	0
Ours	5.67	0.0	3.44	19
[Sharp et al. 2022]	158.7	132.0	296.3	99.9
UV-generalization				
SLIM (GT)	0.00	0.0	0.0	0
Ours	1.99	0.0	0.774	10
[Sharp et al. 2022]	590.0	513.0	866.5	100.0

Table 1. **UV Distortion.** We report the average (**avg. D>10**) and median (**med. D>10**) number of triangles per mesh whose distortion is higher than 10, as well as the average number of flipped triangles per mesh (**#I**), and the percentage of meshes with at-least one flipped triangle (**#I>0** %). We report these statistics for our prediction, the ground truth UVs generated by SLIM [Rabinovich et al. 2017], and for DiffusionNet [Sharp et al. 2022]. We show results over the test split from the Thingi10K [Zhou and Jacobson 2016] dataset which was generated for training, as well as on a dataset composed of meshes that are not part of the Thingi10K dataset.

in large part is due to this task requiring very high accuracy, as a wrong prediction can cause the embedding to self-overlap, and even a slight perturbation can highly-distort triangles or invert their orientation. Furthermore, this is a task only feasible by a framework which can operate on arbitrary triangulations. Both these hurdles are fit for our method to overcome.

Since a large dataset of artist-authored UV's is not publicly available, we opt to evaluate our network's ability to emulate UV maps computed by an optimization-based algorithm, Scalable Locally Injective Maps [Rabinovich et al. 2017] (SLIM). SLIM is a state-of-the-art algorithm, targeting low distortion, and is guaranteed to output a UV parameterization that has no triangles with inverted orientation, making it an extremely challenging target to replicate via learning.

We created a parameterization benchmark by randomly extracting $100K$ disk-topology patches (SLIM requires disk-topology) by iterating over the meshes in the Thingi10K [Zhou and Jacobson 2016] dataset, sampling random points and growing regions from them to random radii. We then parameterize the patches using SLIM. Since the UV parameterization is invariant to rigid motions of the 3D patch, during data preparation we align the patch to the 2D plane s.t. the distance of its vertices to the plane is minimized. The UV map itself is well-defined up to a rigid motion in the plane, and hence we align it to the patch by solving a 2D procrustes problem which rigidly shifts the UV map. We then train our network by feeding it a global shape description of the patch via a PointNet encoding, and train with the loss l_{total}. In this experiment, we treat the output as 2-dimensional and ignore the output z-coordinate.

Table 1 details statistics on the UV parameterization' quality. We measure the parameterization's distortion via $\max(\sigma_1, 1/\sigma_2)$, where $\sigma_1 \geq \sigma_2$ are the singular values of the jacobian. Figure 5 displays a qualitative comparison of the network's prediction with the ground truth results of SLIM. The top two rows show results on the test set from Thingi10K, while the bottom row shows results on patches extracted from meshes retrieved from other repositories. We denote by **I** and **D** the number of inverted elements and ones with distortion higher than 10, when they exist.

Fig. 6. **Learning to re-pose humans.** We learn the STAR [Osman et al. 2020] dataset's pose parameter space: the network is given a mesh of an arbitrary human (left, green) in an arbitrary pose, along with the parameters of the pose it should be moved to, and re-poses it correctly. Ground truth colored in blue, and predictions w.r.t. error (reddest being 0.05).

As SLIM is an optimization algorithm attaining a minimum of the symmetric Dirichlet energy, it is impossible for us to exceed its performance in the metrics it is optimized for. Nonetheless, we note that our predictions closely-resemble those of SLIM's, and in many cases we manage to yield viable parameterizations with no inversions and possessing relatively-low distortion.

In the bottom row of Figure 5, we further evaluate our network's generalization capabilities by mapping patches extracted from models outside of the Thingi10K dataset, with distinctly different features. The quantitative results in Table 1 prove to be better than those over the Thingi10k test set, but this is mainly due to the two datasets varying in their nature (e.g., more mechanical parts in Thingi10k). Attempting to go beyond this level of detail, or to meshes with highly extruding parts such as hands, proves to be beyond the generalization capabilities of our network which produces highly self-overlapping parameterizations.

Of course, for high-quality production assets, artifacts such as inverted elements are strictly not acceptable. However, we believe that this is a first important step towards applying learning algorithms to the UV-mapping problem. We also note that the trained network can be incorporated as a differentiable layer in a deep-learning pipeline, to incorporate UV-dependent losses in order to, e.g., guide segmentation.

4.2 Re-posing and Registration

Computing deformations of human characters is a challenging task, which demands the computed maps to be highly accurate to capture articulated deformations correctly, while at the same time preserve the fine features of the humans. As in all other experiments, we use a PointNet encoding for capturing the shape of the various humans, and do not modify the network to accommodate specifically for human anatomy in any way.

For this experiment, we leverage two datasets, SMPL [Bogo et al. 2016; Varol et al. 2017] and STAR [Osman et al. 2020]. Both datasets comprise of a parametric model, which receives shape parameters controlling the appearance (e.g., tall or short) of the human, as well as pose parameters, which define the pose the articulated human assumes. We use these parameters to generate datasets of varying human subjects, as well as also use the pose parameters themselves as input to the network in one of the experiments. Since all the

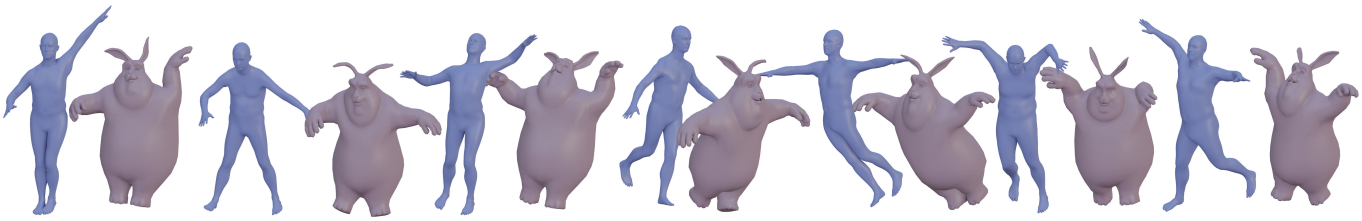

Fig. 7. **Generalization of the re-posing network to Big Buck Bunny.** Our network was trained solely on training samples of human meshes from the STAR [Osman et al. 2020] dataset, and is applied here to the bunny – an unseen mesh, with different triangulation and geometry. We show a STAR mesh in blue, in the ground-truth pose, and the network's reposing of the bunny. The training set consists of one fixed triangulation, which is different than that of the bunny, hence this result can only be achieved by a triangulation-agnostic framework.

Fig. 8. **Learning to register humans.** Our network learns to correctly register any input random SMPL [Bogo et al. 2016] model to itself in arbitrary poses. Source in green, targets in blue, and the deformation colored based on error.

humans are in one-to-one correspondence, we can use their correspondences to compute the ground truth maps during training, while evaluating on different triangulations during evaluation.

Learning to re-pose humans. By training the network to correctly re-pose humans conditioned on pose parameters, we can enable it to control humanoid models. We constructed a dataset of 100K random STAR meshes by sampling the parameter spaces. Each training sample consists of a human with arbitrary shape parameters in an arbitrary initial pose, along with arbitrary target pose parameters, describing the pose the human is to be re-posed to. We supply the network with a PointNet encoding of the initial pose and shape of the human along with pose parameters of the target pose, and optimize the loss l_{total} w.r.t. the ground truth re-posed human. As Table 2 shows, our network is able to map humans into the desired poses with high accuracy. Results of a few examples are shown in Figure 6, with the source mesh shown in green. We (red) are indistinguishable from the ground truth pose (blue), as can be seen by the red color denoting error, with a maximum error below 0.05. Note this is achieved with the network having only a PointNet encoding of 1024 sample-points from the input human, from which it needs to infer both correct shape as well as correct initial pose in order to successfully re-pose it.

With the network fully trained to deform any STAR model w.r.t. pose parameters, we apply it to Big Buck Bunny, a mesh with significantly different features to the training set, and a completely different triangulation. Results are shown in Figure 7, showing the network truly disentangled pose from shape, and can apply the poses to a model which lies far outside of the training set's shape space. Note the preservation of the details of the ears, even though they are a feature that is not present in any training sample of the humans. Likewise, note the natural deformation of the chin at the poses in which Big Buck is looking down.

In terms of triangulation-agnosticism, we note that not only is Big Buck differently-triangulated to the training set, but in this case the network was trained solely on the one specific triangulation which all STAR meshes share, and it is due to the framework's complete agnosticism that it did not overfit to that triangulation and could be applied to other meshes.

On the other hand, training on STAR alone is insufficient for us in order to be able to generalize to any humanoid – our method is highly-sensitive to the encoding of the input shape. As a result, the network is limited to meshes with similar profiles to the humans, and fails when applied to, e.g., the Armadillo. In order for the network to generalize to such case, it would need to be trained on humanoids with a wider range of profiles.

Full and partial registration. We explore the sensitivity of our framework to changes in the triangulation between train and test data, through a synthetic partial-registration experiment. First, we generate a dataset of SMPL [Bogo et al. 2016] humans, by choosing a random identity, and 32 different poses of it, and repeat this for multiple identities until we aggregate 100K samples. We then choose pairs of the same identity in different poses, and train the network by feeding it a PointNet encoding of the two meshes and optimize l_{total}. After the network is fully trained, it can deform a given SMPL model in one pose and register it to another pose. Results are shown in Figure 8. We then modify SMPL meshes from the test set by removing triangles from them to get partial meshes, and then feed each partial mesh as a source, paired with a PointNet encoding of the target. Results are shown in Figure 9. The network produces plausible deformations, and is in general unaffected by the holes, even though it was solely trained on a sphere-topology triangulation

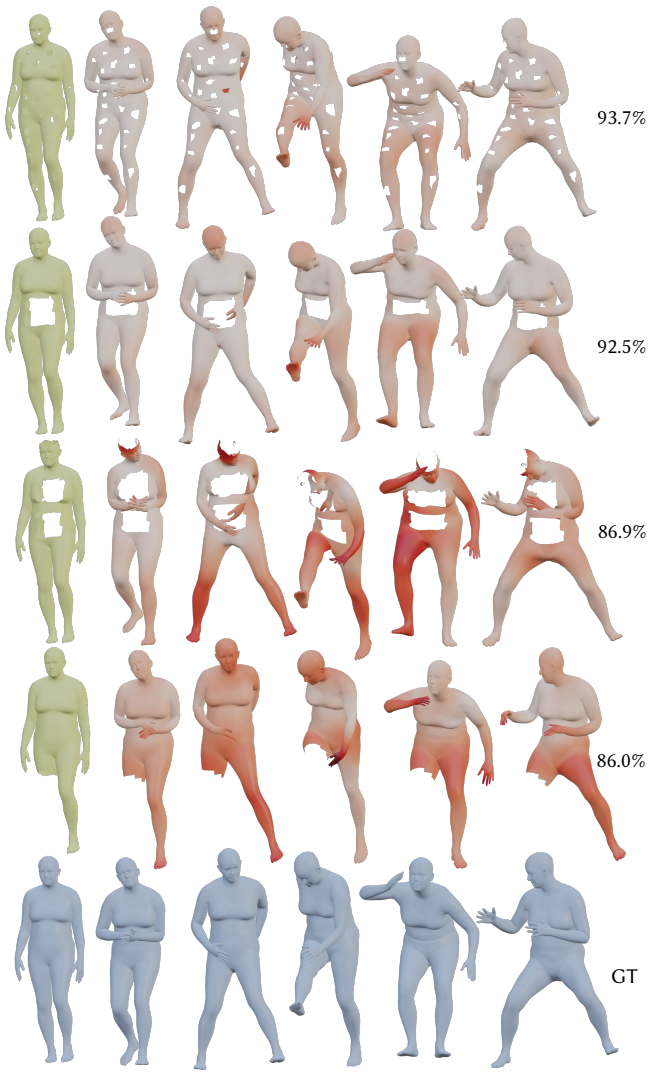

Fig. 9. **Registering partial humans.** A network trained to register full SMPL [Bogo et al. 2016] models to one another manages predicts correct deformations (red) of source models (green) which have parts of them removed. Ground-truth shown in blue.

of the full model. In many cases the network manages to accurately match the partial surface to the full one (we visualize the error w.r.t. the ground truth), and in others fails gracefully by producing an inaccurate but detail-preserving and plausible deformation (note the heads), however there is clear degradation as a result of the removal of triangles. As the predicted jacobian field is conditioned on the PointNet encoding, it seems that the latter is the weak link as during training it has learned to latch on to strong cues of the limbs and head, causing the prediction to be much more sensitive to the absence of a leg than to that of the abdomen.

Morphing humans. In the next experiment, we train the network to morph one human into another, i.e., exactly match the surface of the target according to the ground truth correspondences between

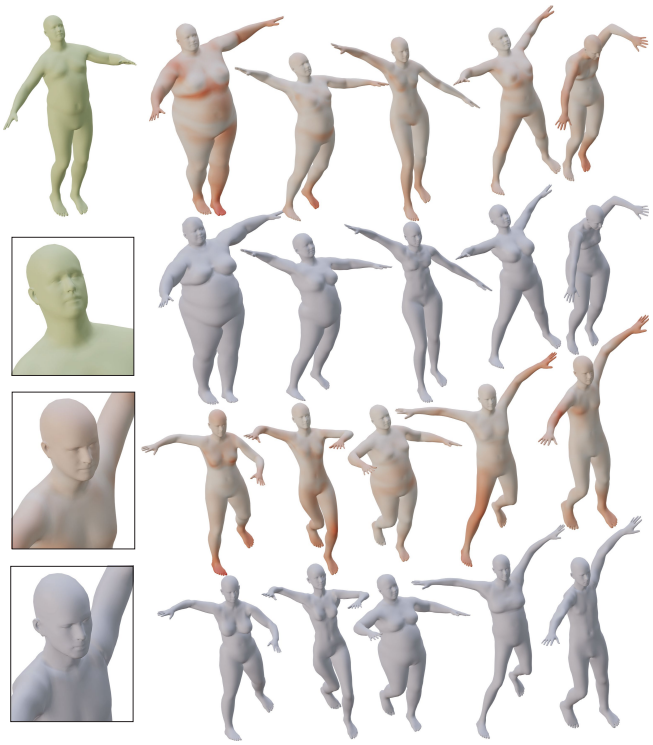

Fig. 10. **Learning to morph humans into other humans.** We train our network to receive Pointnet [Qi et al. 2017] encodings of two random humans in random poses from the STAR [Osman et al. 2020] dataset, and map one into the other. Source mesh is shown in green, targets in blue and the deformations colored w.r.t. error (reddest being 0.1). The network correctly maps the humans, and in places where the pointnet encoding is not descriptive (e.g., the faces), defaults to an accurate detail-preserving deformation of the source.

them (recall these correspondences are not given to the network as input). Each training sample consists of two random human identities in two random poses from the training set. We feed both of their PointNet encodings to the network, and train with the loss l_{total}. Quantitative evaluations over the test set are shown in Table 2. We show a few examples of a single source mapped to multiple different targets in Figure 10. Since the sampled points fed into the PointNet encoder are insufficient to accurately capture the facial features, our deformation network is not aware of the exact facial features of the target mesh and hence does not morph the faces. However, as can be seen in the zoom-ins in Figure 10, in lack of facial information, the intrinsic nature of the framework makes it default into a highly-detail preserving deformation of the head into the target pose, completely preserving the source mesh's face, while morphing the body correctly to the target pose. This could potentially be used by intentionally deleting features of the inputs during training, and forcing the network to "hallucinate" the correct mapping, thereby effectively controlling which details of the original mesh are morphed into the target's, and which ones are preserved.

Discussion: relation to shape correspondence. The experiments in this section exhibit the ability of our network to infer deformation

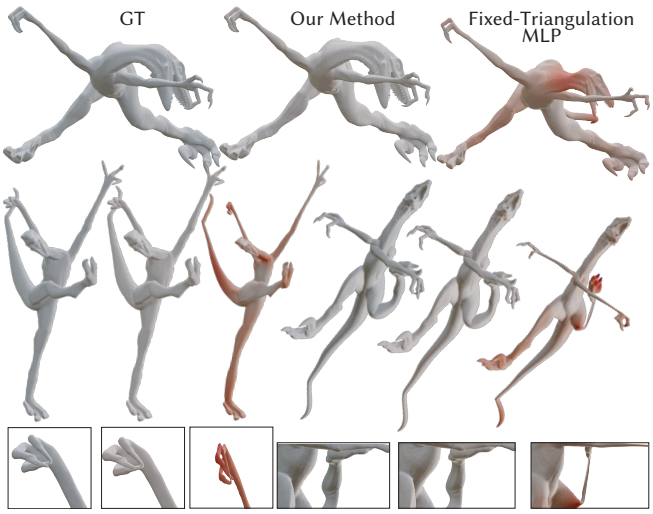

Fig. 11. **Learning volumetric ARAP [Sorkine and Alexa 2007] deformations**. We train our network to predict the boundary-surface deformation induced by volumetric ARAP deformations of a single model, conditioned on the positions of predesignated constraint handles. In each triplet we show, from left to right, the ground truth, us, and the result computed by a fixed-triangulation architecture, similar to [Tan et al. 2018].

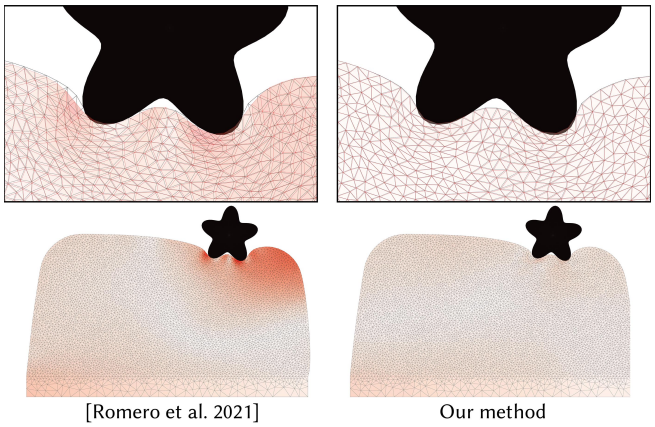

Fig. 12. **Learning collision handling.** We compare our method to [Romero et al. 2021] on collision handling, using their data to train and test. Predictions are overlaid over the ground-truth in the zoom-ins. We achieve slightly higher accuracy, but at higher running times.

spaces of humans. This makes our framework a good candidate for performing shape-correspondence tasks (see [Sahillioğlu 2020] for a recent survey), which we set as an important future direction. Note that there is a subtle difference between reproducing plausible deformations and computing exact correspondences: for deformations, highly-accurate correspondences may still yield implausible deformations when considered as a mapping, e.g., a nose mapped to a cheek will result in a distorted face (alternatively, a plausible deformation cannot be translated to accurate correspondences necessarily, as closest-point matching can yield incorrect correspondences even for plausible deformations). Furthermore, note that we only receive as input 1024 sampled points on the source/target and do not have a dense set of points over which to find dense correspondences.

4.3 Physical Simulation

Data-driven methods are often considered in the context of physical simulation, e.g., for defining a simulation subspace as an alternative to performing direct optimization of the simulation's parameters with respect to the underlying physical model. Our framework can be used as a drop-in general tool in this setting, without any modifications to the architecture to account for the underlying physical model.

Learning volumetric ARAP [Sorkine and Alexa 2007] deformations. We generated a dataset of volumetric ARAP deformations of a model by tetrahedralizing the Raptor model into a tetrahedral mesh consisting of one million elements. We choose 6 fixed handles within the limbs, head and tail of the raptor. Then, we generate 100K training samples, by shifting the handles into random configurations, computing the corresponding ARAP deformation, and saving the boundary triangle mesh, while discarding the tetrahedra. The network is then given as input the 3D positions of the handles, and is trained with the loss l_{total} w.r.t. the ground truth deformation. Quantitative results are reported in Table 2, showing high accuracy. We show a few results in Figure 11. We are able to accurately capture various modes and behaviors of ARAP, such as bending and stretching. ARAP's optimization converges in 10 minutes; our feedforward and linear solve requires approximately a second, using Pytorch and Numpy without any further optimization. We also compare our results to the performance of a fixed-triangulation gradient domain method similar to [Tan et al. 2018]. We discuss it in Subsection 4.4.

Learning collision handling. [Romero et al. 2021] propose an elegant method to train networks to produce non-linear offsets from a linear deformation model so as to account for non-linear behavior due to object collisions, by feeding the network the positions of the collider, along with the handles' positions. We train on their data with the same input as them and optimize for l_{total}, without introducing any changes to our architecture. Results are shown in Figure 12, with a numerical evaluation in Table 2. We achieve slightly more accurate results than Romero, showing our framework is highly versatile. [Romero et al. 2021] is however tailored to be used within simulation frameworks and thus achieve a much higher FPS rate than us. Note that in this case the problem is restricted to 2D, thereby trivializing our restriction operator π_i. In light of this, the experiment solely highlights the efficacy of learning jacobians as a continuous smooth field, instead of a discrete prediction.

4.4 Comparisons

We now move on to comparing our method to other techniques for mapping and deformation.

Neural Cages [Yifan et al. 2020]. Similarly to us, rigging-based methods such as Neural Cages are triangulation-agnostic. Cage-based deformations reduce the deformation space in a manner that

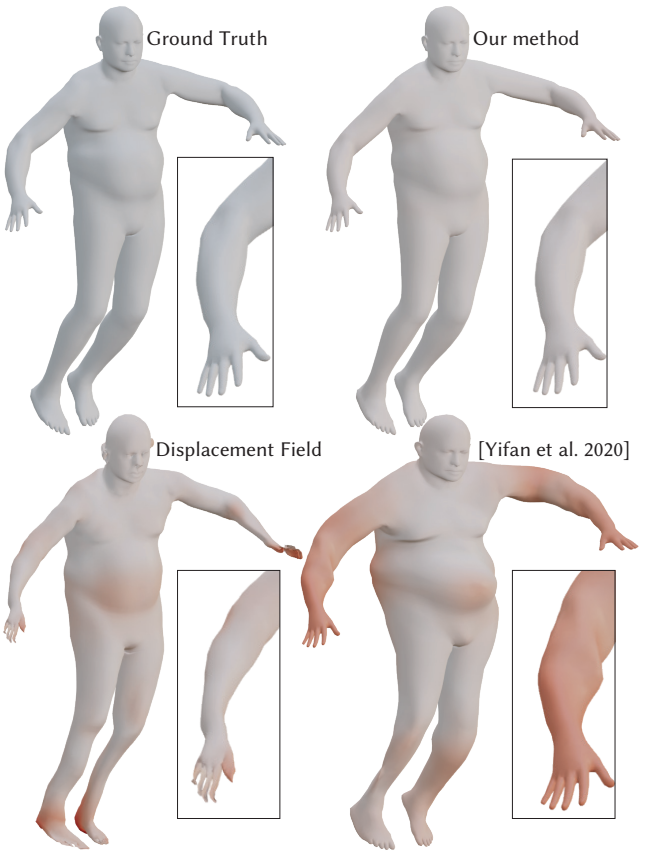

Fig. 13. **Comparison to Neural Cages [Yifan et al. 2020] and direct prediction of a displacement field.** Our method accurately re-poses the source, while both the displacement field approach and Neural Cages distort the shape.

	L2-V ↓	L2-J ↓	L2-N ↓	Hz ↑	Fig.
Applications (all ours)					
UV parameterization					
Thingi10K [Zhou and Jacobson 2016]	4.66	3.68	-	142	5
UV-generalization	3.49	2.59	-	84	5
Humans					
Re-posing STAR [Osman et al. 2020]	1.84	1.55	4.4	93	6
Morphing STAR [Osman et al. 2020]	2.00	2.80	8.2	85	10
Registration SMPL [Bogo et al. 2016]	3.24	3.01	9.9	85	8
Comparisons					
ARAP [Sorkine and Alexa 2007]					
Global MLP baseline	3.66	1.82	13.6	-	11
Ours	**0.64**	**0.43**	**2.7**	-	11
Collision Handling					
[Romero et al. 2021]	0.424	-	-	**980**	12
Ours	**0.398**	**0.17**	-	109	12
Re-posing					
Neural Cages [Yifan et al. 2020]	3.87	1.56	9.6	-	13
Displacement Field	4.21	5.62	11.1	-	13
Ours	**0.84**	**0.83**	2.4	87	13

Table 2. **Quantitative comparisons on various tasks and datasets.** We report the L2 distance between prediction and ground-truth, summed over the mesh's vertices, after normalizing to a unit sphere and scaling by 10^2 (**L2-V**), summer over the Jacobian matrices, scaled by 10 (**L2-J**), and the average angular error on the face normals in degrees (**L2-N**). We also show the number of feed-forward inferences (**Hz**) per second using a single Nvidia V100 and a batch size of 1. All results are reported on unseen data.

excludes highly-oscillatory solutions, but still require a highly accurate prediction of the rig in order to reproduce an artifact-free deformation.

We compare our method to the deformation module of Neural Cages, on a variant of the re-posing experiment from Section 4.2. To accommodate for the fitting of the cage required by [Yifan et al. 2020], we opt to have each source model in the T-pose. Note that in the original paper's humanoid experiments, Yifan et al. do not train the network to fit the cage to the source mesh. Instead, they use given manual annotations to obtain a valid cage at training and at inference time. We follow this setup as well.

To create a good cage for each identity, we first construct a high-quality, enveloping cage over one source in the T-pose. We then use the ground truth vertex correspondences between the STAR models to transfer the MVC weights from the original source to the new source, thereby providing the global optimum to the same fitting described in the original paper. We further confirm that the cages generated in this manner can yield correct deformations matching the source to the target, by overfitting to one example. Note that this gives a significant advantage to Neural Cages since it prevents it from being affected by errors in the cage predictions, which would happen without using the ground truth STAR correspondences. Both ours and their method are then trained over the training set, which consists of pairs of an arbitrary human identity in the T-pose, along with star parameters which describe the pose to deform the human into. We train both methods using the ground truth correspondences between the different STAR models as losses.

As Figure 13 shows, their network struggles to accurately deform the cage correctly for each source, as the cage's deformation needs to be highly accurate itself to not introduce artifacts, with slight variations in it resulting in artifacts in the deformation. This is numerically validated in Table 2. We do note, however, that there exist scenarios in which our intrinsic shape-awareness also has disadvantages: their network, trained on humans, could be applicable to drastically different characters, such as a robot, since the cage warps ambient space and the surfaces embedded in it, without being affected by their intrinsic properties. Our method, on the other hand, would fail on such a model, as intrinsically it is completely different to the training set.

Displacement field. Instead of predicting a field of jacobians, our MLP could directly output displacements for each point in space, similarly to [Huang et al. 2020; Jiang et al. 2020]. This approach is commonly used in mesh-agnostic settings. The limitation of this method lies in it not being shape-aware. This can be observed by noting that for a predicted displacement field, the displacement of a given point is the same, regardless of the shape that is mapped. Thus the prediction itself needs to account for the details of the

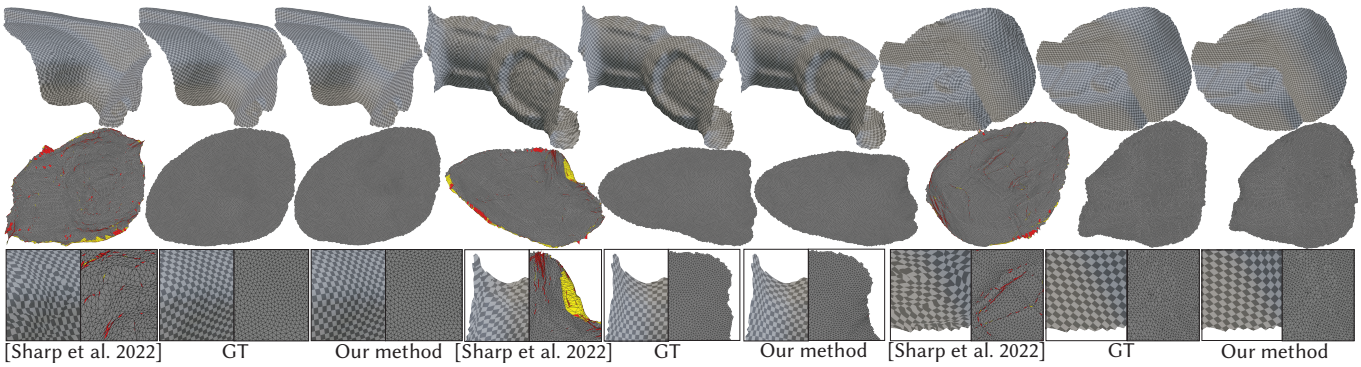

Fig. 14. **Comparison to DiffusionNet [Sharp et al. 2022].** We train DiffusionNet on the UV-learning experiment from Figure 5, and show its predictions on the test set, along with our network's output and the ground truth produced by SLIM [Rabinovich et al. 2017]. We highlight inverted triangles in yellow and triangles with distortion higher than 3 in red. Our output is consistent with SLIM's, and exhibits no high distortion nor inverted triangles.

underlying geometry, resulting in the network struggling to make accurate predictions. In contrast, a single, predicted jacobian field would produce *different*, detail-preserving displacements for the same point in space, for different shapes, as each shape admits a different Laplacian, thereby lifting the burden of the detail preservation from the network. These assertions are verified through qualitative and quantitative comparisons in Figure 13 and Table 2, respectively.

Fixed-triangulation gradient-domain methods. Instead of using our MLP, which receives one triangle and outputs one jacobian at a time, previous works that operate in the gradient domain such as Tan et al. [Tan et al. 2018] and [Gao et al. 2018] use a global MLP which outputs a single, global prediction of a tensor, in our case of size $|T| \times 3 \times 2$, consisting of stacked jacobians. These are then assigned in an arbitrary, consistent order to the triangles. As discussed before, such a construction is fundamentally not applicable to all the experiments shown here, which consist of datasets with more than one triangulation, placing our method and theirs in different categories.

However, putting aside triangulation-agnosticism, we wish to argue for the effectiveness of learning a jacobian *field*, instead of making a prediction of a global tensor. As discussed, the gradient of most maps considered in geometry-processing applications is, relatively, a gradually-varying, low-frequency signal over the domain, ideal for regression and learning. However, using a global MLP completely discards all spatial knowledge. This is a burden on the MLP, causing it to expand its capacity on re-inferring spatial relationships through the fully-connected architecture and the training losses. To exhibit this, we replace our architecture with a global MLP predicting a tensor of jacobians and compare it to ours on learning the Raptor's ARAP deformation space, discussed above. We report the quantitative comparison in Table 2, and visually compare the global MLP's produced deformations to ours (Figure 11), showing in each example from left to right the ground truth, our network's prediction, and the global MLP's. As the raptor consists of 70K triangles, the global MLP lacks the capacity to make an accurate prediction. In contrast, our method leverages the smooth spatial behavior of the jacobians and produces accurate results. Furthermore, our network does not need to increase its capacity with respect to the density of the triangulation, rather only with respect to the underlying granularity of details on the source mesh.

DiffusionNet [Sharp et al. 2022]. Techniques such as DiffusionNet define convolution-like operators to process signals defined over the surface. Similarly to us, they are also triangulation-agnostic and can be applied to arbitrary meshes, which makes them a possible candidate for, e.g., the UV mapping experiment; we compare our performance to theirs on this experiment, by training their method on the same training set as ours and evaluating on the same test sets. The main difference between the two methods lies in ours making a local, per-point prediction through an MLP, along with leveraging Poisson's equation to predict a smooth signal in the gradient domain which guides the mapping; [Sharp et al. 2022] produce a direct global prediction of the output through diffusion operators, and without using Poisson's equation, which "integrates" predicted gradients. Hence, they struggle to make accurate predictions on the delicate UV-mapping task. As shown in Table 1, their predictions exhibit significantly higher distortion, as well as significantly more inverted triangles. We show qualitative comparisons on a few examples, exhibiting these artifacts in Figure 14. We highlight inverted elements in yellow and triangles with distortion higher than 3 in red. Our method's output closely matches the ground truth, exhibits barely any high distortion, and no inverted triangles, while [Sharp et al. 2022] produces a highly-oscillatory output, which does not match the ground truth and exhibits high distortion and inverted elements.

We note that using DiffusionNet to produce input feature-vectors to our MLP instead of the Wave Kernel Signature, or swapping it in place of the PointNet encoder stand as appealing future directions that may leverage the benefits of both approaches.

4.5 Triangulation Agnosticism

While the losses we use during training are derived from the specific triangulation of the meshes, the MLP itself is unaware of anything more than a single triangle during inference, thereby making the network completely agnostic to the triangulation. We validate this claim by applying the network trained for the re-posing experiment,

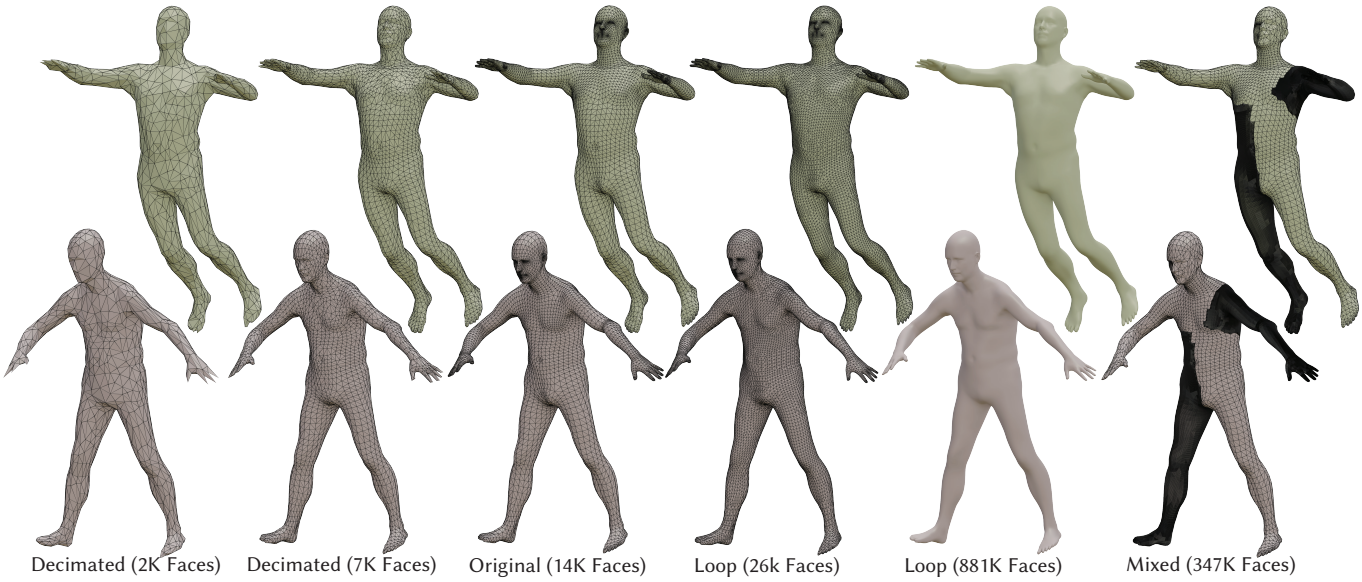

Fig. 15. **Triangulation agnosticism of the framework.** Our network, trained for the re-posing experiment shown in Figure 6, produces the same predictions (bottom row) for different triangulations of the source mesh (top row), even though it was trained on the single, canonical triangulation of the STAR [Osman et al. 2020] dataset (labeled "Original" here).

Section 4.2, on different remeshings of the same surface, from decimated ones to ones achieved via Loop subdivision, as well as a "mixed-technique" model with varying triangulation density. Results are shown in Figure 15. As expected, the network reproduces approximately the same deformation for all the different meshes.

The ability of the network to empirically be almost invariant to the various triangulation can be understood intuitively from a few simple observations: 1) The PointNet encoder used in this experiment to encode the shape is, broadly-speaking, invariant to triangulations since the points fed to it are randomly sampled from the surface; 2) Given the encoding of the shape, the MLP predicts a smooth, gradually-varying field of jacobians, and hence, while different triangulations sample this continuous field in different locations, for dense-enough samples the sampling will converge; 3) the differential operators of the meshes are known (e.g., through Cea's Lemma) to converge as the triangulation is refined, hence the differential operators of the different triangulations produce similar results when applied to the jacobian field.

Lastly, we emphasize the difference between our framework's triangulation-*agnosticism* (lack of knowledge of the triangulation prevents the network from fitting to one triangulation) and a hypothetical complete *invariance* to the triangulation (network is guaranteed to produce exactly same output for any triangulation): our method is FEM-based, and modifying the triangulation modifies the space of piecewise-linear maps, i.e., the space of possible outputs. Therefore, triangulation-invariance is not well-defined for this case.

5 CONCLUSION

The experiments shown in this paper validate that our framework for predicting piecewise-linear mappings of meshes produces highly-accurate and plausible maps on heterogeneous collections of meshes with varying triangulations. They also serve to exhibit the framework's high versatility, as it did not require any modifications for its application to any of the broad range of scenarios considered herein.

Future work. A few important research directions lie ahead. First, in this paper we have focused on the novel deformation module, and hence opted to use a rather-naive approach to encoding input shapes via a PointNet encoder, which we do not consider inherent to our framework. Exploring encoders is an important next step, e.g., using intrinsic ones such as [Sharp et al. 2022], or image-based ones. We are also highly-interested in exploring unsupervised-learning scenarios next, with the immediate candidates being optimizing various distortion and quality measures, or coupling our framework with a visual loss from a neural-renderer.

many of the applications exhibited here are worth revisiting more deeply, tailoring our framework more specifically to them. There are also additional immediate practical applications, e.g., training to map smoothed shapes onto their original geometry, thereby learning a latent space of details which could then be applied to other meshes.

Additionally, one technical consequence of our framework is that it can be used as a differentiable layer, enabling various losses defined in terms of map-predictions. For instance, it is highly enticing to define a differentiable loss for a source mesh, which measures how well does our network map it to a given target, and then optimize the source mesh to reduce that loss and better fit the target. This could be useful for, e.g., optimizing the mesh to accommodate for its desired deformation space. Lastly, we note our approach is directly extendable to tetrahedral-meshes and 2D meshes, in which case the restriction operator is not needed.

. Limitations. While our network is highly effective in computing mappings of triangulated meshes, it relies heavily on discrete differential geometry operators, namely the use of the gradient and Laplacian operators. Hence, in cases where ones are not readily available, such as on point clouds or non-manifold meshes, our network may not be applicable. This may be resolved by using an estimated manifold proxy to one of those operators. Additionally, as the MLP receives positions and normals in cartesian coordinates, it is not invariant to rigid motions nor scaling of the input. This can be resolved by either choosing canonical orientations (e.g., via PCA), augmenting the training data, or using solely intrinsic descriptors and encoders.

Furthermore, we did not explore handling multiple connected components. Our framework is readily applicable to that scenario, by having the MLP predict in addition the matrix P_i a translation vector, and then averaging these translations per each connected component to yield a global translation of each one of them. However, as the pointnet encoder is unaware of topology, the encoding it yields does not enable the network to discern topological information, e.g., the clothes of a human from the human itself, and the training process will fail. This reiterates the necessity of devising an intrinsic encoder of shapes.

Lastly, we note that the current formulation enables us to only produce continuous maps. In case a discontinuous one is desired, e.g., in order to introduce cuts to a UV parameterization, additional means such as masking are needed. We also note that the MLP produces a *continuous* jacobian field, and hence, theoretically, cannot strictly represent a sharp discontinuity in derivatives (e.g., when a flat plane needs to bend by 90 degrees); however, it can predict a smooth approximation of it, and since discrete triangular meshes admit deformation spaces which model sharp discontinuities even when the underlying predicted continuous field (sampled discretely on the mesh) is smooth, we did not experience any such failure cases in practice.

REFERENCES

Noam Aigerman and Yaron Lipman. 2013. Injective and bounded distortion mappings in 3D. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. In *SIGGRAPH*.

Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. 2011. The wave kernel signature: A quantum mechanical approach to shape analysis. In *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE, 1626–1633.

Stephen W Bailey, Dalton Omens, Paul Dilorenzo, and James F O'Brien. 2020. Fast and deep facial deformations. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 94–1.

Stephen W. Bailey, Dave Otte, Paul Dilorenzo, and James F. O'Brien. 2018. Fast and Deep Deformation Approximations. *ACM Transactions on Graphics* 37, 4 (Aug. 2018), 119:1–12. https://doi.org/10.1145/3197517.3201300 Presented at SIGGRAPH 2018, Los Angeles.

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Trans. Graph.* 26, 3 (jul 2007), 72–es. https://doi.org/10.1145/1276377.1276467

Jan Bednarik, Shaifali Parashar, Erhan Gundogdu, Mathieu Salzmann, and Pascal Fua. 2020. Shape reconstruction by learning differentiable surface representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4716–4725.

Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J. Black. 2016. Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image. In *Computer Vision – ECCV 2016 (Lecture Notes in Computer Science)*. Springer International Publishing.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*.

Sofien Bouaziz, Andrea Tagliasacchi, Hao Li, and Mark Pauly. 2016. Modern Techniques and Applications for Real-Time Non-Rigid Registration. In *SIGGRAPH ASIA 2016 Courses* (Macau) *(SA '16)*. Association for Computing Machinery, New York, NY, USA, Article 11, 25 pages. https://doi.org/10.1145/2988458.2988490

Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z Kovalsky, Yajie Yan, Danny M Kaufman, and Tao Ju. 2020. Lifting simplices to find injectivity. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 120–1.

Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. 2019. Latent-space Dynamics for Reduced Deformable Simulation. *Computer Graphics Forum* (2019).

Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L Rosin, Weiwei Xu, and Shihong Xia. 2018. Automatic Unpaired Shape Deformation Transfer. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2018)* 37, 6 (2018), To appear.

Lin Gao, Jie Yang, Tong Wu, Yu-Jie Yuan, Hongbo Fu, Yu-Kun Lai, and Hao Zhang. 2019. SDM-NET: Deep generative network for structured deformable mesh. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–15.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 2018a. 3D-CODED: 3D Correspondences by Deep Deformation. *ECCV* (2018).

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018b. AtlasNet: A Papier-Mache Approach to Learning 3D Surface Generation. *arXiv preprint arXiv:1802.05384* (2018).

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 2019. Deep Self-Supervised Cycle-Consistent Deformation for Few-Shot Shape Segmentation. *SGP* (2019).

Daniel Holden, Bang Chi Duong, Sayantan Datta, and Derek Nowrouzezahrai. 2019. Subspace neural physics: Fast data-driven interactive simulation. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 1–12.

Daniel Holden, Jun Saito, and Taku Komura. 2015. Learning an Inverse Rig Mapping for Character Animation. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '15)*. Association for Computing Machinery, New York, NY, USA, 165–173. https://doi.org/10.1145/2786784.2786788

Jingwei Huang, Chiyu Max Jiang, Baiqiang Leng, Bin Wang, and Leonidas Guibas. 2020. Meshode: A robust and scalable framework for mesh deformation. *arXiv preprint arXiv:2005.11617* (2020).

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

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. 2014. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses*.

Tomas Jakab, Richard Tucker, Ameesh Makadia, Jiajun Wu, Noah Snavely, and Angjoo Kanazawa. 2021. KeypointDeformer: Unsupervised 3D Keypoint Discovery for Shape Control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12783–12792.

Chiyu Jiang, Jingwei Huang, Andrea Tagliasacchi, Leonidas Guibas, et al. 2020. Shapeflow: Learnable deformations among 3d shapes. *arXiv preprint arXiv:2006.07982* (2020).

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. In *ACM Siggraph 2005 Papers*. 561–566.

Angjoo Kanazawa, Shahar Kovalsky, Ronen Basri, and David Jacobs. 2016. Learning 3d deformation of animals from 2d images. In *Computer Graphics Forum*, Vol. 35. Wiley Online Library, 365–374.

Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. 2018. Learning Category-Specific Mesh Reconstruction from Image Collections. In *ECCV*.

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), 1–23.

Theodore Kim and David Eberle. 2020. Dynamic Deformables: Implementation and Production Practicalities. In *ACM SIGGRAPH 2020 Courses* (Virtual Event, USA) *(SIGGRAPH '20)*. Association for Computing Machinery, New York, NY, USA, Article 23, 182 pages. https://doi.org/10.1145/3388769.3407490

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

Shahar Z Kovalsky, Noam Aigerman, Ronen Basri, and Yaron Lipman. 2014. Controlling singular values with semidefinite programming. *ACM Trans. Graph.* 33, 4 (2014), 68–1.

Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least Squares Conformal Maps for Automatic Texture Atlas Generation. In *SIGGRAPH*.

Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. 2021. Learning Skeletal Articulations with Neural Blend Shapes. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1.

Yaron Lipman. 2012. Bounded distortion mapping spaces for triangular meshes. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–13.

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green coordinates. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–10.

Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, David Levin, Christian Rossi, and Hans-Peter Seidel. 2004. Differential coordinates for interactive mesh editing. In *Proceedings Shape Modeling Applications, 2004.* IEEE, 181–190.

Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. 2018. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 1886–1895.

Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. 2008a. A Local/Global Approach to Mesh Parameterization. In *Proceedings of the Symposium on Geometry Processing* (Copenhagen, Denmark) *(SGP '08).* Eurographics Association, Goslar, DEU, 1495–1504.

Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J Gortler. 2008b. A local/global approach to mesh parameterization. In *Computer Graphics Forum*, Vol. 27. Wiley Online Library, 1495–1504.

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

Minghua Liu, Minhyuk Sung, Radomir Mech, and Hao Su. 2021. DeepMetaHandles: Learning Deformation Meta-Handles of 3D Meshes with Biharmonic Coordinates. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 12–21.

Luca Morreale, Noam Aigerman, Vladimir G Kim, and Niloy J Mitra. 2021. Neural Surface Maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 4639–4648.

Ashish Myles and Denis Zorin. 2013. Controlled-distortion constrained global parametrization. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.

Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. 2017. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS).* http://learningsys.org/nips17/assets/papers/paper_16.pdf

Ahmed A A Osman, Timo Bolkart, and Michael J. Black. 2020. STAR: A Sparse Trained Articulated Human Body Regressor. In *European Conference on Computer Vision (ECCV).* 598–613. https://star.is.tue.mpg.de

Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *CVPR* (2019).

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*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Ulrich Pinkall and Konrad Polthier. 1993. Computing Discrete Minimal Surfaces and Their Conjugates. *EXPERIMENTAL MATHEMATICS* 2 (1993), 15–36.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 652–660.

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Transactions on Graphics* 36, 2 (April 2017), 16:1–16:16.

Cristian Romero, Dan Casas, Jesus Perez, and Miguel A. Otaduy. 2021. Learning Contact Corrections for Handle-Based Subspace Dynamics. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)* 40, 4 (2021). http://gmrv.es/Publications/2021/RCPO21

Yusuf Sahillioğlu. 2020. Recent advances in shape correspondence. *The Visual Computer* 36, 8 (2020), 1705–1721.

Christian Schüller, Ladislav Kavan, Daniele Panozzo, and Olga Sorkine-Hornung. 2013. Locally injective mappings. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 125–135.

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. 2022. Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)* 41, 3 (2022), 1–16.

Alla Sheffer, K Hormann, B Levy, M Desbrun, K Zhou, E Praun, and H Hoppe. 2007. Mesh parameterization: Theory and practice. *ACM SIGGRAPPH, course notes* 10, 1281500.1281510 (2007).

Siyuan Shen, Yin Yang, Tianjia Shao, He Wang, Chenfanfu Jiang, Lei Lan, and Kun Zhou. 2021. High-order differentiable autoencoder for nonlinear model reduction. *ACM Transactions on Graphics.*

Jason Smith and Scott Schaefer. 2015. Bijective Parameterization with Free Boundaries. *ACM Trans. Graph.* 34, 4, Article 70 (jul 2015), 9 pages. https://doi.org/10.1145/2766947

Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. In *SGP.*

Olga Sorkine and Mario Botsch. 2009. Interactive Shape Modeling and Deformation. In *EUROGRAPHICS Tutorials.*

Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. 2004. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing.* 175–184.

Robert W Sumner and Jovan Popović. 2004. Deformation transfer for triangle meshes. *ACM Transactions on graphics (TOG)* 23, 3 (2004), 399–405.

Bo Sun, Xiangru Huang, Qixing Huang, Zaiwei Zhang, Junfeng Jiang, and Chandrajit Bajaj. 2021. ARAPReg: An As-Rigid-As Possible Regularization Loss for Learning Deformable Shape Generators. In *ICCV.*

Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. 2018. Variational Autoencoders for Deforming 3D Mesh Models. In *CVPR.*

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-Maps. In *ACM SIGGRAPH 2004 Papers* (Los Angeles, California) *(SIGGRAPH '04).* Association for Computing Machinery, New York, NY, USA, 853–860. https://doi.org/10.1145/1186562.1015810

Mikaela Angelina Uy, Jingwei Huang, Minhyuk Sung, Tolga Birdal, and Leonidas Guibas. 2020. Deformation-aware 3d model embedding and retrieval. In *European Conference on Computer Vision.* Springer, 397–413.

Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. 2017. Learning from Synthetic Humans. In *CVPR.*

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV).* 52–67.

Ofir Weber and Denis Zorin. 2014. Locally injective parametrization with arbitrary fixed boundaries. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–12.

Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. 2019. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 10130–10139.

Yuxin Wu and Kaiming He. 2018. Group normalization. In *Proceedings of the European conference on computer vision (ECCV).* 3–19.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *ACM Trans. on Graphics* 39 (2020).

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. 2019. Predicting Animation Skeletons for 3D Articulated Models via Volumetric Nets. In *2019 International Conference on 3D Vision (3DV).*

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. *NeurIPS.*

Wang Yifan, Noam Aigerman, Vladimir G. Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural Cages for Detail-Preserving 3D Deformations. In *CVPR.*

Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 2021. 3DStyleNet: Creating 3D Shapes with Geometric and Texture Style Variations. In *Proceedings of International Conference on Computer Vision (ICCV).*

Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. 2004. Mesh editing with poisson-based gradient field manipulation. In *ACM SIGGRAPH 2004 Papers.* 644–651.

Mianlun Zheng, Yi Zhou, Duygu Ceylan, and Jernej Barbic. 2021. A Deep Emulator for Secondary Motion of 3D Characters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 5932–5940.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797* (2016).

Silvia Zuffi, Angjoo Kanazawa, David Jacobs, and Michael J. Black. 2017. 3D Menagerie: Modeling the 3D Shape and Pose of Animals. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).*

A INVARIANCE TO THE CHOICE OF FRAMES

Claim. *Our framework is completely invariant to the (arbitrary) choice of frames* $\{\mathcal{B}_i\}$.

This statement follows from two observations:

(1) The basis-dependent, projected jacobians play a role in only two parts of our framework, namely, in the Poisson solve, Eq. (3), and in the loss l_{jacobian}, Eq. (8). In both cases they are used in exactly one type of function, of the form $\|R_i - Q_i\|_F$, where $Q_i = \nabla_i \Psi$ is a ground-truth jacobian of a mapping Ψ, expressed in the same basis $\mathcal{B}_i$ as R_i, and $\|\cdot\|_F$ is the frobenius norm.
(2) The choice of $\mathcal{B}_i$ will not change the value of the expression $\|R_i - Q_i\|_F$. This follows from basic linear algebra which we prove in the lemma below.

Hence the only expression in which the frames appear is invariant to their choice, and therefore our method is completely invariant to the choice of frames.

Lemma. *Let* X, Y *be linear transformations acting on a tangent space* T, $X, Y : T \to \mathbb{R}^3$, *and let* $\mathcal{B}_1, \mathcal{B}_2$ *be two different orthogonal bases for* T. *Let* $X_1, Y_1 \in \mathbb{R}^{3\times 2}$ *(resp.* X_2, Y_2*) be the representation of the linear transformations in the local coordinates of* $\mathcal{B}_1$ *(resp.* $\mathcal{B}_2$*). Then* $\|X_1 - Y_1\|_F = \|X_2 - Y_2\|_F$.

Proof.

$$\|X_1 - Y_1\|_F \overset{FI}{=} \left\|(X_1 - Y_1)\,\mathcal{B}_1^T\mathcal{B}_2\right\|_F =$$

$$\left\|X_1\mathcal{B}_1^T\mathcal{B}_2 - Y_1\mathcal{B}_1^T\mathcal{B}_2\right\|_F \overset{CB}{=} \|X_2 - Y_2\|_F\,,$$

where "FI" is due to the Frobenius norm's invariance to multiplication by orthogonal matrices ($\mathcal{B}_1^T\mathcal{B}_2$ is a 2×2 orthogonal matrix), and "CB" follows from the definition of a change of basis for a linear transformation. □