CageNet: A Meta-Framework for Learning on Wild Meshes

MICHAL EDELSTEIN, Technion - Israel Institute of Technology, Israel
HSUEH-TI DEREK LIU, Roblox & University of British Columbia, Canada
MIRELA BEN-CHEN, Technion - Israel Institute of Technology, Israel

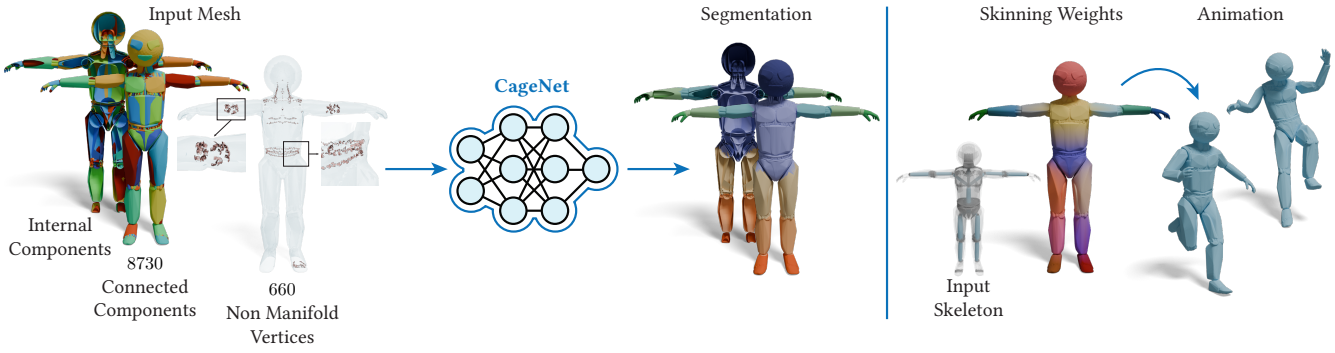

Fig. 1. Our CageNet generalizes a segmentation network pre-trained on nearly-manifold inputs to a mesh in the wild (middle) with multiple connected components, non-manifold elements, and internal structures (left). CageNet also enables training on *wild* meshes, such as predicting skinning weights for animation (right).

Learning on triangle meshes has recently proven to be instrumental to a myriad of tasks, from shape classification, to segmentation, to deformation and animation, to mention just a few. While some of these applications are tackled through neural network architectures which are tailored to the application at hand, many others use generic frameworks for triangle meshes where the only customization required is the modification of the input features and the loss function. Our goal in this paper is to broaden the applicability of these generic frameworks to *"wild" meshes*, i.e. meshes in-the-wild which often have multiple components, non-manifold elements, disrupted connectivity, or a combination of these. We propose a configurable meta-framework based on the concept of *caged geometry*: Given a mesh, a *cage* is a single component manifold triangle mesh that envelopes it closely. *Generalized barycentric coordinates* map between functions on the cage, and functions on the mesh, allowing us to learn and test on a variety of data, in different applications. We demonstrate this concept by learning segmentation and skinning weights on difficult data, achieving better performance to state of the art techniques on wild meshes.

CCS Concepts: • **Computing methodologies** -> **Shape analysis**.

Additional Key Words and Phrases: geometric deep learning, geometry processing, skinning

ACM Reference Format:
Michal Edelstein, Hsueh-Ti Derek Liu, and Mirela Ben-Chen. 2025. CageNet: A Meta-Framework for Learning on Wild Meshes. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '25), August 10–14, 2025, Vancouver, BC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3721238.3730654

Authors' Contact Information: Michal Edelstein, Technion - Israel Institute of Technology, Haifa, Israel, edel.michal@gmail.com; Hsueh-Ti Derek Liu, Roblox & University of British Columbia, Vancouver, Canada, hsuehtil@gmail.com; Mirela Ben-Chen, Technion - Israel Institute of Technology, Haifa, Israel, mirela@cs.technion.ac.il.

SIGGRAPH Conference Papers '25, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1540-2/2025/08
https://doi.org/10.1145/3721238.3730654

1 Introduction

The impact that neural networks have had on geometry processing is unquestionable. The state-of-the-art methods for applications such as shape classification, segmentation, correspondence, deformation and many others, are all using neural networks, either supervised or unsupervised. These methods are much stronger when they are able to leverage the toolbox of discrete geometry processing, including various discrete differential operators such as the Laplace-Beltrami operator. One such example is DiffusionNet [Sharp et al. 2022], which is a general architecture that has been successfully applied to many of the previously mentioned applications.

One obstacle to wider adoption of these approaches, is that meshes "in-the-wild", e.g. meshes that are generated by artists, are often *inconvenient* from the point of view of geometry processing: they have multiple components, can be non-manifold, and often have non-triangular faces. While geometry processing operators can be potentially generalized to handle these cases, this would inevitably require custom modifications for each artifact, making it not scalable to "wild" mesh datasets. Furthermore, multiple-component geometry is specifically difficult to deal with without adding connections that allow information to pass between the components.

We propose CageNet, a meta-framework that fills this gap by enveloping the troublesome geometry with a *cage*, providing a map between the cage and the input geometry using *generalized barycentric coordinates*, and applying a geometry-processing informed neural net on the cage geometry. We use DiffusionNet [Sharp et al. 2022] to demonstrate our framework, due to its wide applicability and easy

setup. We illustrate our framework using two applications: learning mesh segmentation and skinning weights. We achieve comparable performance as the state-of-the-art on a segmentation benchmark consisting of clean mostly-manifold inputs, while obtaining much better generalization to meshes in the wild (see Figure 1). Compared to networks which are *tailored* to computing skinning weights, our method also achieves the lowest error on wild meshes. Thus, using CageNet one benefits from both worlds: a generic network architecture with minimal setup (defining the input features and the loss), which works out-of-the-box on wild meshes, and achieves the performance of specialized networks.

1.1 Related Work

1.1.1 Machine Learning Architectures on Meshes. Developing machine learning architectures for processing meshes has been an active subject [Bronstein et al. 2017]. A key element in these architectures is how to generalize common operators, such as message passing or convolution, for signals defined on the surface. Several works rely on discrete operators (e.g., convolution) defined on vertices [Gong et al. 2019; Lahav and Tal 2020; Lim et al. 2018], edges [Hanocka et al. 2019; Liu et al. 2020], faces [Hertz et al. 2020; Hu et al. 2022], or a mixture of different mesh elements [Milano et al. 2020]. One can also leverage spectral operators, such as the Laplacian, to process scalar [Sharp et al. 2022; Smirnov and Solomon 2021] or vector signals [Gao et al. 2024] on surfaces.

Despite their effectiveness, these architectures either fail to process multiple connected mesh components jointly or *non-manifold* elements, which form a majority of the meshes in existing datasets such as ShapeNet [Chang et al. 2015]. For instance, the convolution stencil of MeshCNN [Hanocka et al. 2019] assumes that each edge only contains two neighboring faces, which is not the case for non-manifold edges. Such a limitation poses a significant challenge to deploying existing architectures to mesh data in the wild (see, e.g., Figure 2). Even if one considers mesh repair before passing it into a neural network, this often leads to the loss of surface attributes [Hu et al. 2020, 2018] or to degraded geometric quality [Huang et al. 2020] (see also the Supplemental). This motivates our CageNet to operate directly on these multi-component, non-manifold meshes by parameterizing volumetric signals with manifold cages.

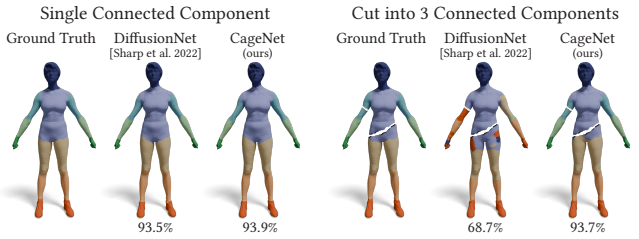

Fig. 2. When the input is a single component manifold mesh (left), the existing method [Sharp et al. 2022] and our CageNet lead to comparable results. However, when the input contains multiple connected components (right), our method results in better generalization. We show the computed segmentation color coded on the mesh, and the resulting accuracy.

1.1.2 Cage-based Geometry Processing. Cages have been utilized for processing geometry, most notably for shape deformation [Ströter et al. 2024]. Additional classic applications are deformation transfer [Ben-Chen et al. 2009; Chen et al. 2010], tracking [Savoye and Franco 2010] and computing "skinning templates" [Ju et al. 2008].

More recently, cages were used in learning-based deformation with learned key point handles [Jakab et al. 2021], learned cages [Yifan et al. 2020]; cages have been combined with NeRFs for editing and deformation transfer [Peng et al. 2022; Xu and Harada 2022], and have been used to deform Gaussian splatting [Huang and Yu 2024]. Our method is different from these approaches, since it provides a *general framework* for learning different tasks on wild meshes.

1.1.3 Geometry Processing on Wild Meshes. Another option for handling difficult meshes, is to address each problem separately. Examples include the non-manifold Laplace Beltrami operator [Sharp and Crane 2020] (see also citations within for additional examples of non-manifold geometry processing), or adding connectivity to handle meshes with multiple connected components [Garland and Heckbert 1997; Zhou et al. 2010], and other approaches such as repairing meshes [Hu et al. 2018]. However, adapting an existing network architecture using such approaches would require a custom solution for each type of mesh defect, re-implementing parts of the network and retraining it. Using our framework, it is possible to use off-the-shelf mesh-based architectures and apply them to in-the-wild meshes directly.

1.2 Contributions

Our contributions are as follows:

- A meta-framework for learning on in-the-wild 3D geometries, such as meshes with interior structures, multiple components, that may be non-manifold.
- Using cages and generalized barycentric coordinates to parameterize volumetric functions for neural networks.
- Demonstrating the applicability of our framework to shape segmentation and computing skinning weights, achieving better performance on wild meshes than state-of-the-art techniques in a generic framework.

2 Method

Mesh-based neural network architectures are often restricted to manifold meshes [Hanocka et al. 2019] or have poor performance on meshes with multiple connected components [Sharp et al. 2022]. CageNet is a neural network architecture that generalizes mesh-based architectures to such *wild meshes*.

Consider a neural network $\mathcal{F}_\Theta$ that maps multi-dimensional input features x defined on the vertices (or faces) of the mesh to a set of output features y. Given a non-manifold or multiple component mesh M, we cannot use $\mathcal{F}_\Theta$ directly, either for training or for testing. Instead, we construct a single component, manifold *cage* denoted by $\widetilde{M}$, that fully encapsulates the model (Sec 2.1). In addition, we define a *differentiable mapping operator* $\mathcal{P}$ that maps functions on $\widetilde{M}$ to functions on M (Sec 2.2). We note that the reverse mapping, from M to $\widetilde{M}$ is *not* required. At training time, we apply $\mathcal{F}_\Theta$ to the cage $\widetilde{M}$, and map the resulting output features to M using $\mathcal{P}$. Thus, the loss function is applied to features on the input M, where the

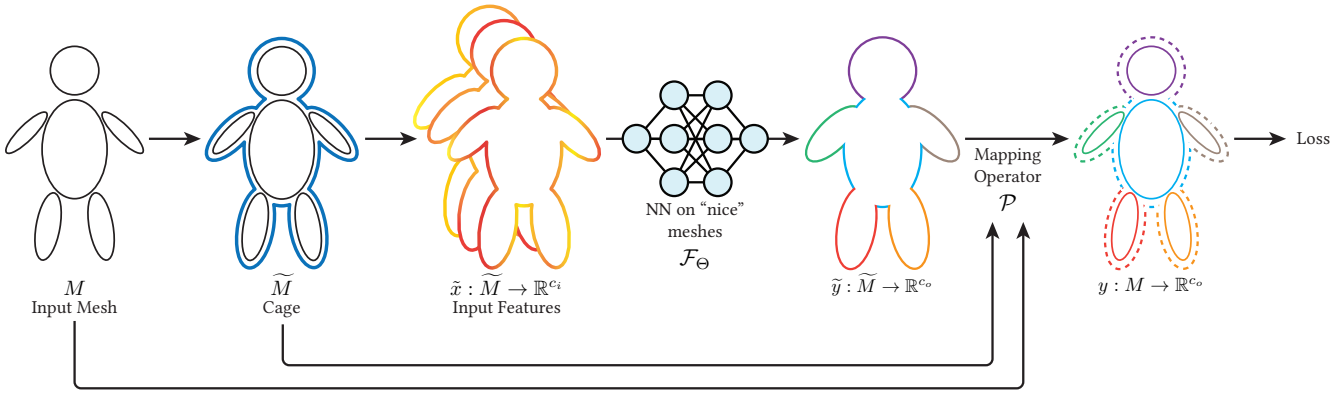

Fig. 3. Given an input mesh M, possibly with multiple components and non-manifold elements, our CageNet constructs a single component, manifold cage $\widetilde{M}$ and the corresponding mapping operator $\mathcal{P}$ which maps signals from $\widetilde{M}$ to M. We then apply a neural network for "nice" meshes $\mathcal{F}_\theta$ on the cage $\widetilde{M}$ (where the input features $\tilde{x}$ are computed). The learned output features $\tilde{y}$ of the cage are projected using $\mathcal{P}$ on the input mesh, yielding the output features y on M, which are used to compute the loss on M, where the training data is provided.

training data is provided (Sec. 2.3). For testing we again apply $\mathcal{F}_\Theta$ to $\widetilde{M}$, and map the output with $\mathcal{P}$, yielding features on the input mesh M (Sec. 2.4). This is illustrated in Figure 3.

2.1 The Cage

Given a mesh M, we require a (1) manifold, (2) single component cage $\widetilde{M}$ such that (3) $\widetilde{M}$ is geometrically "close" to M to minimize the loss of detail. Since we apply our method to large datasets, the cage construction must be (4) fully automatic. These are the 4 necessary requirements for the cage. Additional, nice-to-have properties, are that the cage is (5) topologically equivalent to the input mesh, and (6) that M is in the *interior* of $\widetilde{M}$ for well-behaved cage weights.

Due to the popularity of cages in geometry processing, there exists a myriad of methods for generating them (see e.g. the recent review by Ströter et al. [2024], Sec. 4.2). Unfortunately, none of the publicly available methods fulfills all our necessary constraints. Hence, we opted for a basic approach: we compute the unsigned distance field of the input shape, and extract a ε level set surface with marching cubes (using the implementation by Wang et al. [2022]). To obtain a single component mesh, we first remove internal components (by computing winding numbers for vertices relative to other components [Barill et al. 2018; Sellán et al. 2023]). We repeat the process with a larger offset ε if we obtain multiple disjoint components. The new offset is $\varepsilon_{new} = \varepsilon_{old} + \frac{1}{2} d_{max}$, where d_{max} is the largest distance between two of the disjoint components. This guarantees that the new offset surface has a single component.

Finally, we simplify the result using quadric error edge collapse [Garland and Heckbert 1997] as implemented in MeshLab [Cignoni et al. 2008], to obtain a cage with $\widetilde{m}$ faces. We use $\widetilde{m} = 12K$ for most meshes. If the result does not encapsulate the input mesh (which happens rarely), and an encapsulating cage is required, we add more faces by simplifying to a larger number of faces, up to $\widetilde{m} = 24K$. Figure 4 shows that the process is not sensitive to the exact number $\widetilde{m}$, yielding very similar results at test time for different $\widetilde{m}$ values.

We emphasize that the process is fully automatic, and results in a single component manifold cage that encapsulates the input mesh, thus fulfilling our requirements. This does not, however, guarantee that the cage is topologically equivalent to the input mesh. Figure 6 shows a few resulting cages for different offsets ε. Note that for a large offset, the hands connect to the body, leading to a high genus mesh. We optionally compute multiple cages for each mesh to augment the data, see the segmentation experiment in Sec. 4.

2.2 The Mapping Operator $\mathcal{P}$

The mapping operator $\mathcal{P}$ takes as input the mesh $M = (\mathcal{V}, \mathcal{F})$, the cage $\widetilde{M} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{F}})$ and a function on the cage $\tilde{y} : \widetilde{\mathcal{V}} \to \mathbb{R}$, and computes a function on the input mesh $y : \mathcal{V} \to \mathbb{R}$. We opted for a very simple solution, using a linear map computed using *generalized barycentric coordinates*. Given a point $p \in M$, its generalized barycentric coordinates with respect to the cage $\widetilde{M}$ are $\{\lambda_1(p), \lambda_2(p), ..., \lambda_{\widetilde{n}}(p)\} \in \mathbb{R}^{\widetilde{n}}$, such that $\sum_j \lambda_j = 1$ [Ströter et al. 2024, Sec 2.1]. Here, $\widetilde{n} = |\widetilde{\mathcal{V}}|$. The mapping operator $\mathcal{P}$ is thus defined as:

$$\mathcal{P}(M, \widetilde{M}, \tilde{y}) = C_{\widetilde{M}}(X_M)\tilde{y}, \tag{1}$$

where $X_M \in \mathbb{R}^{n \times 3}$ are the coordinates of the vertices of the input mesh M, with $n = |\mathcal{V}|$. Further, $C_{\widetilde{M}}$ is a matrix of size $n \times \widetilde{n}$, whose (i, j)-th entry is $\lambda_j(p_i)$ where $p_i \in \mathbb{R}^3$ is the embedding of the vertex $v_i \in \mathcal{V}$. Hence, $y = \mathcal{P}(M, \widetilde{M}, \tilde{y})$ contains weighted averages of the values of $\tilde{y}$, where the weights are given by the generalized barycentric coordinates.

Different coordinate functions have different expressivity, and computational complexity. E.g., the Mean Value Coordinates (MVC) by Floater et al. [2003] have a closed form expression, do not require an encapsulating cage and are very efficient to compute. On the other hand, Harmonic [Joshi et al. 2007] and biharmonic [Jacobson et al. 2011] coordinates require an encapsulating cage, do not have a closed form, and require solving a PDE based optimization problem, leading to much higher computational costs. In practice, for PDE

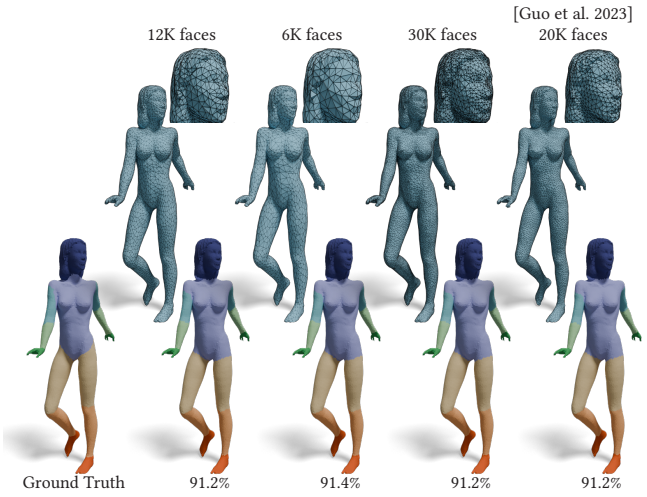

Fig. 4. Our method is robust to different cage resolutions, an advantage inherited from DiffusionNet [Sharp et al. 2022]. We show that the segmentation results are comparable (bottom row) even if the cages have different resolutions (top row). We show the accuracy of the segmentation result compared to the ground truth. The three leftmost cages are generated using our approach, whereas the rightmost cage is generated by Guo et al. [2023].

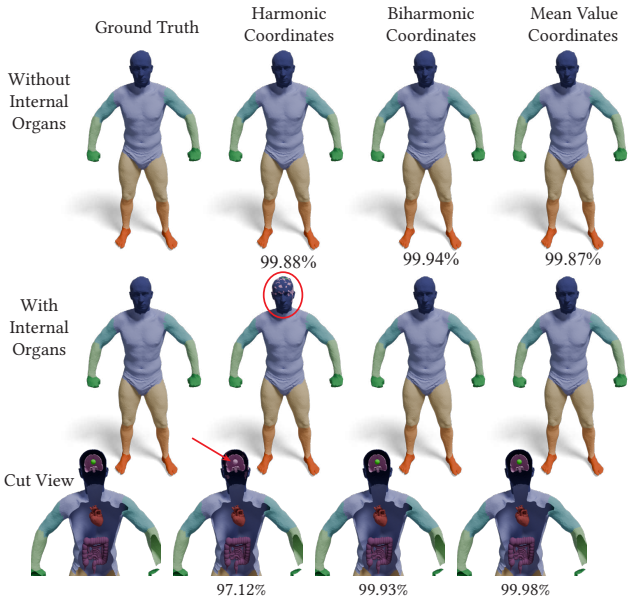

Fig. 5. The choice of coordinates influences the generalization of CageNet. When the input is a single manifold mesh without interior structure (top row), CageNet can overfit the segmentation perfectly using all three coordinate choices. But when the input contains interior components (bottom rows), the harmonic coordinates fail near the brain region (second column), while the biharmonic (third column) and the mean value coordinates (fourth column) still lead to perfect predictions.

based coordinates, we create a tetrahedral mesh of the cage [Hu et al. 2020], solve the PDE using FEM, and linearly interpolate the coordinates from the vertices of each tetrahedron to its interior.

In terms of expressivity, MVC and biharmonic coordinates are similar, whereas harmonic coordinates struggle to represent encapsulated structures with different function values. Since a harmonic function is determined by its boundary values, this is to be expected.

Figure 5 compares these three types of generalized barycentric coordinates for representing a segmentation function. We show the ground truth color coded segmentation, as well as the representation obtained by overfitting CageNet to this function (see Sec 2.3), using the different coordinates. For representing the segmentation of a human (top row), all coordinates are equally good. For a shape with "internal organs" (bottom rows), which are classified differently by the segmentation function, MVC and biharmonic coordinates perform significantly better, while MVC is much faster to compute. Thus, in all our experiments we use MVC.

2.3 CageNet Training

We first compute all the cages and their corresponding generalized barycentric coordinates for all the input meshes.

In our experiments, we use DiffusionNet [Sharp et al. 2022] as the underlying network due to its robustness to cage tessellation. Our framework, however, is general, and other mesh-based networks could be used instead.

We use the default DiffusionNet input features, which are computed on the input cages. We elaborate on the features when describing the specific experiments in Section 3. We emphasize that the loss is computed *on the input data* directly, by mapping the cage output features computed by the network to the input meshes using the mapping operator $\mathcal{P}$. When the labeling data is provided on the edges or the faces (for example, for segmentation), we first map from the cage to the input mesh vertices, then average to the edges or faces, and apply the last activation as appropriate.

2.4 CageNet Testing

The cage and barycentric coordinates are computed on the input mesh. Then the network is tested as usual, with the output signal again mapped to the source mesh using the mapping operator $\mathcal{P}$.

CageNet can generalize from training on "nice" meshes, to testing on "wild" meshes. For example, in Section 4 we train CageNet on the human segmentation benchmark [Maron et al. 2017], where all the models are single component meshes. In Figure 1 we test the trained network on a model which is non-manifold, has internal structures and multiple components. Indeed, the network generalizes well to this type of "inconvenient" (though abundant in the wild) meshes.

Figure 4 demonstrates that the method is not sensitive at test time to the cage resolution. Specifically, we test the segmentation network trained in Sec 4 using cages with different resolutions generated with our method, as well as a cage generated by the method of Guo et al. [2023]. We note that the segmentation results are very similar, regardless of the cage used.

3 Experimental Results

We apply CageNet to two applications: segmentation and computation of skinning weights. We used the authors' implementation of DiffusionNet [Sharp et al. 2022], and implemented CageNet in PyTorch [Paszke et al. 2019]. All training was done on a single machine with NVIDIA RTX A4500 GPU (20GB GPU memory) and 128GB system memory. We elaborate on the training data, input features, loss, hyper-parameters and evaluation metrics for each application separately.

4 Human segmentation

Semantic segmentation [Gao et al. 2023] is one of the most basic shape processing tasks, and as such is addressed by many different methods. We focus here on the *Human Segmentation* dataset by Maron et al. [2017], due to its popularity as a benchmark.

4.1 Implementation Details

4.1.1 Input features and hyper-parameters. We use a very similar setup as DiffusionNet's original segmentation setup, using HKS features. The only different hyper-parameters are the network width (64 instead of 128), and the initial learning rate (0.0005 instead of 0.001). We provide the full list of hyper-parameters in the supplemental material for completeness.

4.1.2 Loss. We apply a cross-entropy loss on vertex labels averaged to the faces (as does DiffusionNet). In our case, to obtain the vertex labels we use the operator $\mathcal{P}$ to map the cage functions to functions on the vertices of the input mesh.

4.1.3 Evaluation Metrics. As is standard, we use the segmentation accuracy as the evaluation metric.

4.2 Results

In Table 1 we show that our method achieves the same results as DiffusionNet on *nice* single-component manifold meshes. Indeed, since our method is based on DiffusionNet, it cannot be expected to surpass it. In contrast, when testing on a broken mesh as in Figure 2, our method generalizes much better than DiffusionNet. Here, we remove some faces from one of the meshes in the dataset to generate a mesh with multiple connected components. This allows us to compute the accuracy, since we have the ground truth data for the remaining faces. We also show our robustness to wild meshes by testing on a *"souped"* dataset. We split each model in the test dataset into a triangle soup and randomly inverted half of the face normals. We obtain 91.7% accuracy, the same as for the clean dataset.

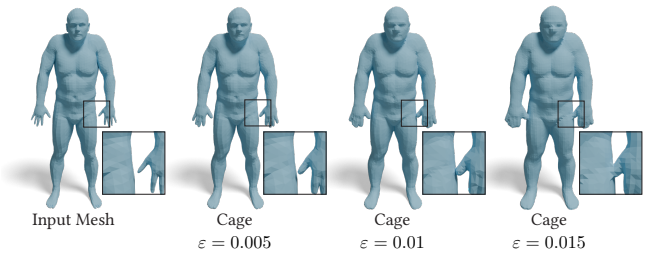

Fig. 6. We augment the training data by generating multiple cages with different offset values. This increases the robustness of CageNet against topological noise as the cage topology may change with different offsets, such as the hand region in this example.

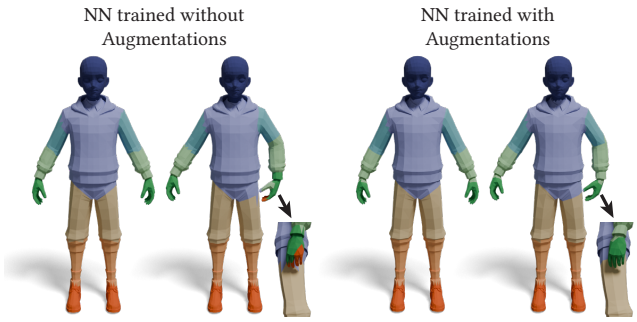

Fig. 7. We compare the segmentation results on a wild mesh that does not belong to the training dataset, when the network is trained without (left) and with (right) cage offset augmentation (see Sec. 4.3). In each pair, the mesh on the right has problematic regions (the left hand) which leads to different cage topologies for different offsets, whereas the mesh on the left does not. The mesh on the left yields good segmentation results with or without augmentation (see e.g. the ground truth segmentation in Fig. 4). However, the mesh on the right leads to erroneous segmentation on the left hand without augmentation, which is resolved when adding the augmentation.

As another experiment, in Figure 8 we test on *wild mesh* inputs, which have multiple connected components. Here as well our method leads to better generalization than the baselines. Figure 1 shows another example, where we test on a mesh which has multiple components, interior parts and non-manifold elements, and achieve good generalization. For the wild meshes ground truth is not available, so we do not compute accuracy. A qualitative comparison is possible with the ground truth segmentation of one of the meshes from the training dataset, shown on the top left. We show additional results on wild meshes in the Supplemental material.

4.3 Cage offset augmentation

For each training mesh we generate cages at offsets 0.005, 0.01, 0.015, and 0.02 (models are normalized in the unit box), see Figure 6. At each epoch, for each mesh, we randomly choose one of the generated cages. For testing we use the smallest training offset. Note that while each additional cage increases preprocessing time during training, it does not affect the training loop time. In addition, it does not affect inference time, as testing is performed with a single cage offset. Figure 7 shows an example of testing using a net that was trained without (left) and with (right) cage offset augmentation. Note the correction obtained for the segmentation of the left hand of the model when the augmentation is used. The results on the human segmentation dataset without the augmentation are given in Table. 1. Indeed, the accuracy without the augmentation is lower.

5 Skinning Weights

Skinning is one of the most popular methods for real-time shape deformation [Jacobson et al. 2014]. A core ingredient is the creation

Table 1. We report the segmentation results on the manifold mesh dataset [Maron et al. 2017]. Our CageNet does not deteriorate the performance of existing methods on clean meshes, getting the same accuracy (%) as the DiffusionNet [Sharp et al. 2022] where our method is based on. In contrast, we can achieve better generalization to wild inputs in Figure 7.

Method	Acc.	Method	Acc.
GCNN [Masci et al. 2015]	86.4	MeshWalker [Lahav et al. 2020]	92.7
ACNN [Boscaini et al. 2016]	83.7	CGConv [Yang et al. 2021]	89.9
Toric Cover [Maron et al. 2017]	88.0	FC [Mitchel et al. 2021]	92.5
PointNet++ [Qi et al. 2017]	90.8	DiffusionNet [Sharp et al. 2022]	91.7
MDGCNN [Poulenard et al. 2018]	88.6	MeshMAE [Liang et al. 2022]	90.0
DGCNN [Wang et al. 2019]	89.7	SubdivNet [Hu et al. 2022]	93.0
SNGC [Haim et al. 2019]	91.0	SieveNet [Yuan et al. 2023]	93.2
PFCNN [Yang et al. 2020]	91.5	CageNet (ours) w/o aug.	90.4
HSN [Wiersma et al. 2020]	91.1	CageNet (ours)	91.7
PD-MeshNet [Milano et al. 2020]	86.9	CageNet (ours) - *"Souped"* dataset	91.7

of *skinning weights* to bind vertices with skeletons to drive deformation. Some existing techniques offer a semi-automatic interface to aid in the creation [Bang and Lee 2018; Ma et al. 2024]. Some rely on fully automatic approaches to optimize for smoothed and localized skinning weights based purely on geometric information [Dionne and de Lasa 2013; Jacobson et al. 2011; Wang and Solomon 2021]. However, these techniques may struggle to create weights that are aligned with, for instance, semantics. This issue has been motivating the development of several learning-based techniques, including [Liu et al. 2019; Ma and Zhang 2023; Mosella-Montoro and Hidalgo 2022; Ouyang and Feng 2020; Pan et al. 2021; Xu et al. 2020], to capture semantic information by learning from data, using neural nets *tailored* to this problem. Our approach, on the other hand, is, to our best knowledge, the first to treat this problem (albeit with a known constant skeleton) in a general mesh-based network.

We evaluate the performance on skinning weight generation on a dataset of 753 artist-created biped characters (see supplemental, Figure 1) obtained from the Roblox platform, where we hold out 75 meshes as the test set. Meshes in the dataset share the same pose, orientation and skeleton topology.

5.1 Implementation Details

5.1.1 Input features and hyper-parameters. We use a cage offset of $\epsilon = 0.01$. The cage vertex locations and the *volumetric geodesic distance* from the cage vertices to the skeleton's bones are the input features to our network, as suggested in Xu et al. [2020]. Our network uses DiffusionNet's default hyper-parameters, with a 128-width. After obtaining output features on the cage, we use the cage coordinates to map the results to the input mesh. As skinning weights need to be positive and sum up to 1, we apply the softmax activation to the weights at each vertex to satisfy these constraints.

5.1.2 Loss. Our loss function consists of a KL divergence term, an L_p loss to encourage sparsity, and a symmetry loss to encourage symmetric weights for symmetric meshes. Thus our loss is:

$$\mathcal{L} = \mathrm{KL}(\hat{s}, s) + \lambda_p L_p(\hat{s}) + \lambda_{\mathrm{sym}} \mathrm{Sym}(\hat{s}, s), \tag{2}$$

where s are the ground truth skinning weights, $\hat{s}$ are the predicted skinning weights and λ_p and λ_{sym} balance the different terms. We use $p = 0.3, \lambda_p = 0.1, \lambda_{\mathrm{sym}} = 0.05$.

The skinning weights per vertex are effectively a distribution with respect to the bones (since they are positive and sum to 1). Hence we use the KL divergence to measure the error of the predicted weights w.r.t. ground truth, as in [Liu et al. 2019]. We compute the KL divergence per vertex, and then average over the vertices.

The skinning weights are expected to be *sparse*, thus each vertex should be affected by a small number of bones. The L_p loss, with $0 < p \leq 1$ is used in classic optimization methods to encourage sparsity [Bruckstein et al. 2009]. While $p = 1$ is a common choice [Tibshirani 1996], it is known that smaller p values are more beneficial for sparsity. We find that $p = 0.3$ leads to the best results. The L_p-norm of the skinning weights is computed per vertex, and then averaged over the vertices.

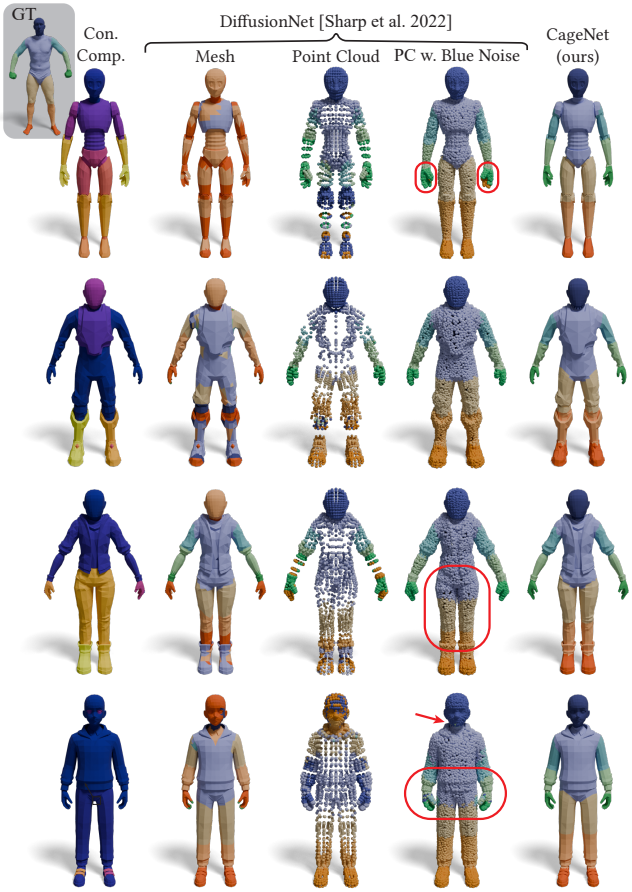

Fig. 8. Segmentation results on wild biped characters, consisting of multiple connected components visualized using component-wise colors (first column). DiffusionNet [Sharp et al. 2022] trained only on a single component mostly-manifold mesh dataset [Maron et al. 2017] fails to generalize, regardless of using mesh connectivity (second column), vertex point cloud connectivity (third column), or uniformly sampled [Yuksel 2015] point connectivity (fourth column). In contrast, our CageNet (fifth column) generalizes to wild characters to produce desired segmentation results. These in-the-wild models do not have ground truth segmentation. The ground truth on one of the meshes from the training dataset is shown for comparison on the top left.

The symmetry loss is designed to improve the symmetry of the predicted weights in areas where both the mesh and the ground truth weights are symmetric. Hence, it measures the symmetry error of the predicted weights in these regions. For simplicity, we consider only extrinsic symmetry with respect to the yz plane, which is compatible with our consistently oriented dataset.

5.1.3 Evaluation Metrics. We use a few evaluation metrics, following Xu et al. [2020]. (1) The average L_1 loss compares the predicted and ground truth weights in the L_1 norm. (2) Precision and recall measure the success of finding the *influential* (weight larger than 0.0001) bones of a vertex, and the F_1 score is the harmonic mean of precision and recall, representing both in one metric. (3) We measure the normalized vertex distance between an animation frame generated by the predicted v.s. the ground truth weights, and report the average and the maximum. The vertex distances are evaluated on 8 artist-created animations (see the supplementary video).

5.2 Results

5.2.1 Quantitative Results. We evaluate our method by comparing the predicted skinning weights against different baselines, including BBW [Jacobson et al. 2011], GeoVoxel [Dionne and de Lasa 2013], and RigNet [Xu et al. 2020]. For BBW, we use the authors implementation, where we first tetrahedralize the mesh using fTetWild [Hu et al. 2020]. For GeoVoxel, we use Maya's implementation [Autodesk, Inc. 2024]. We show the results both with the default parameters, and with the recommended parameters in [Xu et al. 2020]. For both of these methods, we remove meshes with joints that are outside the body, which are about 5% of the test set.

RigNet and our CageNet are trained on 90% randomly selected meshes from our dataset (see Supplementary Figure 1) and evaluated on the test set. We present our results using our predicted skinning ("Ours") and with sparsity enforced by thresholding values below 0.2 to zero ("Ours-S").

The results are reported in Table 2. We note that for the average L_1 distance, which measures the error of the predicted weights, our method achieves the best results (with and without enforcing sparsity). The statistical F_1 score, which combines both the precision and recall of finding the influential bones, is also the best for our approach. As expected, enforcing sparsity by thresholding increases precision and reduces recall, however the combined F_1 measure shows that the trade-off is reasonable and not much is lost. In terms of vertex distance of the generated animation frames, we achieve the best average distance (tied with RigNet). The best maximal distance is achieved by BBW, with the rest of the methods achieving mostly similar results. BBW wins on this metric, as it is the only method that is not affected by regions of the mesh that are close in Euclidean distance but far in geodesic distance.

To summarize, our method shows very good results, comparable or surpassing existing techniques. The closest to our performance is RigNet, which is *tailored* for computing skinning weights (and thus for example can also handle different skeleton structures), whereas we achieve these results using a *generic* network.

5.2.2 Qualitative Results. We show qualitative results in Figure 12. We show for each model the rest pose and skeleton (left most column), and the ground truth weights. The weights are visualized by setting a different color for each bone, and combining these colors using the weights. In addition, we show the ground truth for one animation frame from a set of user-generated animations. The animation frame is computed using Linear Blend Skinning (LBS) [Jacobson et al. 2014] with the ground truth skinning weights. For each method we show the computed skinning weights, and the corresponding animation frame computed with LBS and the computed weights. The animation frame is color-coded with the pointwise normalized vertex displacement error. Note that in all cases our error is considerably lower than the other methods, with RigNet arguably as the closest competitor. We provide the full animation sequence in the supplemental video and the average normalized vertex error per frame in the Supplemental.

In addition to baseline comparisons, in Figure 9 we demonstrate our resilience to multi-component meshes by decomposing a mesh into a triangle soup (second row), as well as adding vertex noise to the triangle soup (third row), and flipping randomly the orientation of some of the faces (fourth row). For each case we show the mesh, the cage, the ground truth weights, predicted weights, and color coded L_1 pointwise error, as well as the average error. We note that for both types of noise the predicted result is very similar to the predicted result on the original mesh.

In Figure 13 we apply our method to wild AI meshes generated by [Xiang et al. 2024] and Meshy, and compare it with the baselines. These meshes do not have ground truth, hence we show only the resulting weights, and a corresponding animation frame. The weights can be qualitatively compared to the ground truth weights of the Roblox dataset in Figure 12, since we used the same skeleton (which

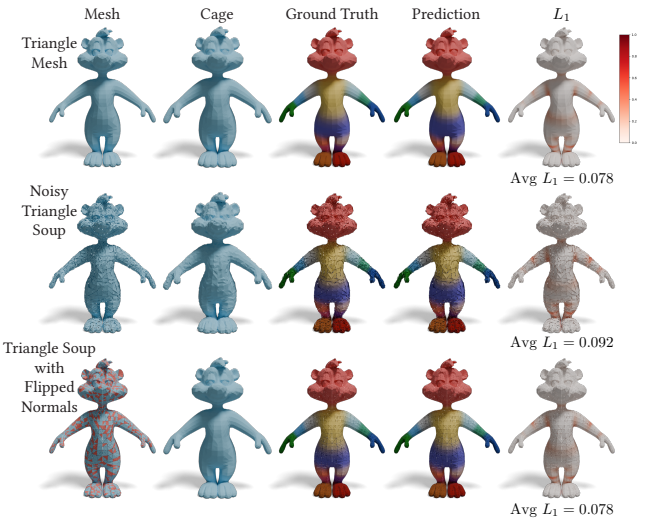

Fig. 9. Generalization to triangle soups. We train on a single-component triangle mesh (top) and test on a triangle soup with vertex noise (middle) and with flipped triangles (bottom). Note that our method leads to the desired skinning weight prediction (4-th column) with low error (5-th column).

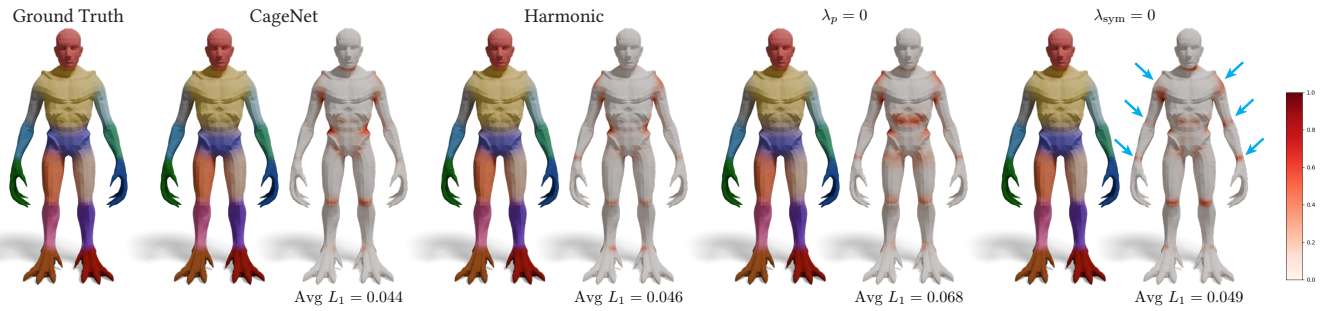

Fig. 10. Our ablation study on the skinning weight experiment shows that our choice of coordinates and incorporating the Lp and the symmetry loss terms lead to more accurate predictions. Specifically, removing the L_p loss increases the error significantly, since weights are no longer sparse. Removing the symmetry loss leads to non-symmetric weights (visible in the non-symmetric error) in addition to a higher error term.

was manually posed to fit the generated mesh). We note that our results are most similar semantically to the ground truth weights. While RigNet leads to good results as well, artifacts are visible for all three models. See the video for the animation sequence.

We also show additional results on wild meshes, and an experiment demonstrating the cage topology effect on the results in the Supplementary.

5.2.3 Ablation. In Figure 10 we modify different parts of our algorithm for this application. For each option we show the predicted weights and the L_1 error. We use harmonic weights instead of MVC, which results in a minor increase in the L_1 error (third column). We remove the L_p loss leading to a significant increase in the L_1 error, as well as smoothing of the error, and its distribution in a larger area, as expected (fourth column). Finally, we remove the symmetry loss leading again to an increase in the L_1 loss, as well as loss of symmetry of the weights, which can be seen by the loss of symmetry of the L_1 error function (fifth column).

Additionally, Table 3, shows the quantitative ablation results on the Roblox dataset. We note that the full configuration, with MVC and all the loss parts, achieves the best average L_1 loss, closely followed by the harmonic coordinates (as in Figure 10). We conjecture that even though the dataset has interior parts, for which harmonic coordinates do not work as well), for skinning weights we expect the interior parts to deform similarly to the exterior parts, thus allowing for a good weight approximation by harmonic coordinates.

Table 2. We compare our predicted skinning weights vs. existing baselines: bounded biharmonic weights (BBW [Jacobson et al. 2011]), GeoVoxel ([Dionne and de Lasa 2013] GV with default parameters, GV* with parameters recommended by Xu et al. [2020]), and RigNet [Xu et al. 2020]. We report the following metrics: average L_1 error (↓), Precision (↑), Recall (↑), F1-score (↑), average vertex distance (↓), and maximum vertex distance (↓).

Method	Avg L_1	Prec.	Recall	F_1	Avg D.	Max D.
BBW	0.575	45.80	97.06	56.58	0.009	**0.410**
GV	0.687	42.14	**99.99**	55.54	0.009	0.658
GV*	0.645	75.11	89.24	76.80	0.008	0.7046
RigNet	0.153	**98.11**	87.17	90.41	**0.002**	0.697
Ours	**0.124**	88.70	98.78	**91.72**	**0.002**	0.692
Ours-S	0.135	98.09	88.85	91.53	**0.002**	0.697

Naturally, these are much more expensive to compute, so we provide this example for illustrative purposes only. The F_1 score is similar for all configurations, with the exception of removing the L_p loss. In this case, the precision reduces significantly, as we falsely identify many more bones as influential (since the weights are not as sparse), leading to more false positives. The average and maximal vertex distance error on the animated frames is also similar between the different configurations. The smallest maximal distance is obtained by removing the L_p loss and reducing sparsity, leading to a more uniform error distribution and smaller maximal error.

6 Limitations, Future Work and Conclusion

Meshes that have self-intersections (Figure 11 left) are problematic for CageNet if the regions that intersect have different labels or feature values. Since our features are volumetric scalar *functions* on the interior of the cage, they cannot be multi-valued at a given point. Using multi-valued functions (e.g., polynomial roots represented by the polynomial coefficients) may alleviate this restriction in future work. Despite the cage-offset augmentation (Figure 7), our approach is still somewhat sensitive to topological inconsistencies due to cage generation (such as the extreme case in Figure 11 right). Future improvements on cage generation, such as adaptive offsets or a topologically robust method (e.g., [Zint et al. 2024]), could further improve the stability of CageNet. We demonstrate the efficacy of our CageNet using DiffusionNet [Sharp et al. 2022] as an example. Future explorations on combining our approach with different network architectures and with other generalized barycentric coordinates (e.g., [de Goes and Desbrun 2024; Thiery et al. 2024]) could further boost CageNet's performance, and improve its efficacy in applications (e.g., generalizing to different skeleton topologies).

Table 3. We show an ablation study with different cage coordinates and loss terms, using the same metrics as in Table 2. See the text for details.

Ablation	Avg L_1	Prec.	Recall	F_1	Avg D.	Max D.
CageNet	**0.124**	88.70	98.78	91.72	**0.002**	0.692
Harmonic	0.127	89.21	98.54	91.92	**0.002**	0.705
$\lambda_p = 0$	0.135	59.10	**99.93**	71.15	**0.002**	**0.688**
$\lambda_{\text{sym}} = 0$	0.133	**89.47**	98.49	**92.06**	**0.002**	0.748

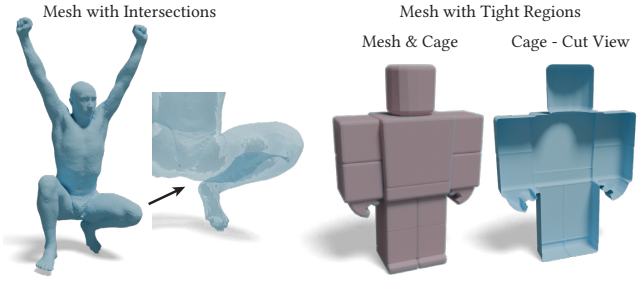

Fig. 11. Limitations. CageNet learns parametrized volumetric *functions*, and thus it cannot represent multiple values at the same spatial position. Thus, intersections in the input mesh (left) will be problematic if labeled differently. Additionally, if the mesh has large areas in close proximity (right), the cage may envelope them together, restricting the diversity of functions representable in these areas.

To conclude, we have shown that caging wild meshes is beneficial for data-driven shape processing with neural networks. We believe that CageNet can find many additional applications, and inspire future work in the realm of robust geometry processing.

Acknowledgments

Mirela Ben Chen acknowledges the support of the Israel Science Foundation (grant No. 1073/21). We thank Weiqi Shi for collecting and providing the Roblox dataset. We also use models from Turbosquid, the Windows 3D Library, and Sketchfab, and we thank the respective creators for making their models publicly available. We acknowledge the Blender community for making their software publicly available [Blender Foundation 2025; Liu 2018].

References

Autodesk, Inc. 2024. *Autodesk Maya.* https://www.autodesk.com Software.

Seungbae Bang and Sung-Hee Lee. 2018. Spline Interface for Intuitive Skinning Weight Editing. *ACM Trans. Graph.* 37, 5 (2018), 174. doi:10.1145/3186565

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

Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. 2009. Spatial deformation transfer. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* 67–74.

Blender Foundation. 2025. Blender – a 3D Modelling and Rendering Package. http://www.blender.org

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. *Advances in neural information processing systems* 29 (2016).

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.* 34, 4 (2017), 18–42.

Alfred M Bruckstein, David L Donoho, and Michael Elad. 2009. From sparse solutions of systems of equations to sparse modeling of signals and images. *SIAM review* 51, 1 (2009), 34–81.

Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. 2015. *ShapeNet: An Information-Rich 3D Model Repository.* Technical Report arXiv:1512.03012 [cs.GR]. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

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

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, Guido Ranzuglia, et al. 2008. Meshlab: an open-source mesh processing tool.. In *Eurographics Italian chapter conference*, Vol. 2008. Salerno, Italy, 129–136.

Fernando de Goes and Mathieu Desbrun. 2024. Stochastic Computation of Barycentric Coordinates. *ACM Trans. Graph.* 43, 4 (2024), 42:1–42:13.

Olivier Dionne and Martin de Lasa. 2013. Geodesic voxel binding for production character meshes. In *The ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '13, Anaheim, CA, USA, July 19-21, 2013*, Jinxiang Chai, Yizhou Yu, Theodore Kim, and Robert W. Sumner (Eds.). ACM, 173–180. doi:10.1145/2485895.2485919

Michael S. Floater. 2003. Mean value coordinates. *Comput. Aided Geom. Des.* 20, 1 (2003), 19–27. doi:10.1016/S0167-8396(03)00002-5

Alexander Gao, Maurice Chu, Mubbasir Kapadia, Ming C. Lin, and Hsueh-Ti Derek Liu. 2024. An Intrinsic Vector Heat Network. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024.* OpenReview.net.

Lin Gao, Yu Liu, YuXiang Liu, XiaoYa Cheng, JueLin Zhu, and JingYi Wang. 2023. Large-scale 3D Mesh Data Semantic Segmentation: A Survey. In *2023 9th International Conference on Big Data and Information Analytics (BigDIA).* IEEE, 81–89.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques.* 209–216.

Shunwang Gong, Lei Chen, Michael M. Bronstein, and Stefanos Zafeiriou. 2019. SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019.* IEEE, 4141–4148.

Jia-Peng Guo, Wen-Xiang Zhang, Chunyang Ye, and Xiao-Ming Fu. 2023. Robust Coarse Cage Construction With Small Approximation Errors. *IEEE Transactions on Visualization and Computer Graphics* (2023).

Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. 2019. Surface networks via general covers. In *Proceedings of the IEEE/CVF international conference on computer vision.* 632–641.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Trans. Graph.* 38, 4 (2019), 90:1–90:12.

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep geometric texture synthesis. *ACM Trans. Graph.* 39, 4 (2020), 108.

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Junxiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. 2022. Subdivision-based Mesh Convolution Networks. *ACM Trans. Graph.* 41, 3 (2022), 25:1–25:16.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Trans. Graph.* 39, 4 (2020), 117.

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

Jiajun Huang and Hongchuan Yu. 2024. GSDeformer: Direct Cage-based Deformation for 3D Gaussian Splatting. *arXiv preprint arXiv:2405.15491* (2024).

Jingwei Huang, Yichao Zhou, and Leonidas J. Guibas. 2020. ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups. *CoRR* abs/2005.11621 (2020). arXiv:2005.11621 https://arxiv.org/abs/2005.11621

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78. doi:10.1145/2010324.1964973

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 *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021.* Computer Vision Foundation / IEEE, 12783–12792.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph.* 26, 3 (2007), 71. doi:10.1145/1276377.1276466

Tao Ju, Qian-Yi Zhou, Michiel Van De Panne, Daniel Cohen-Or, and Ulrich Neumann. 2008. Reusable skinning templates using cage-based deformations. *ACM Transactions on Graphics (ToG)* 27, 5 (2008), 1–10.

Alon Lahav and Ayellet Tal. 2020. MeshWalker: deep mesh understanding by random walks. *ACM Trans. Graph.* 39, 6 (2020), 263:1–263:13.

Yaqian Liang, Shanshan Zhao, Baosheng Yu, Jing Zhang, and Fazhi He. 2022. Meshmae: Masked autoencoders for 3d mesh data analysis. In *European Conference on Computer Vision.* Springer, 37–54.

Isaak Lim, Alexander Dielen, Marcel Campen, and Leif Kobbelt. 2018. A Simple Approach to Intrinsic Correspondence Learning on Unstructured 3D Meshes. In *Computer Vision - ECCV 2018 Workshops - Munich, Germany, September 8-14, 2018, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11131)*, Laura Leal-Taixé and Stefan Roth (Eds.). Springer, 349–362.

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

Hsueh-Ti Derek Liu. 2018. *Blender Toolbox.* https://github.com/HTDerekLiu/BlenderToolbox

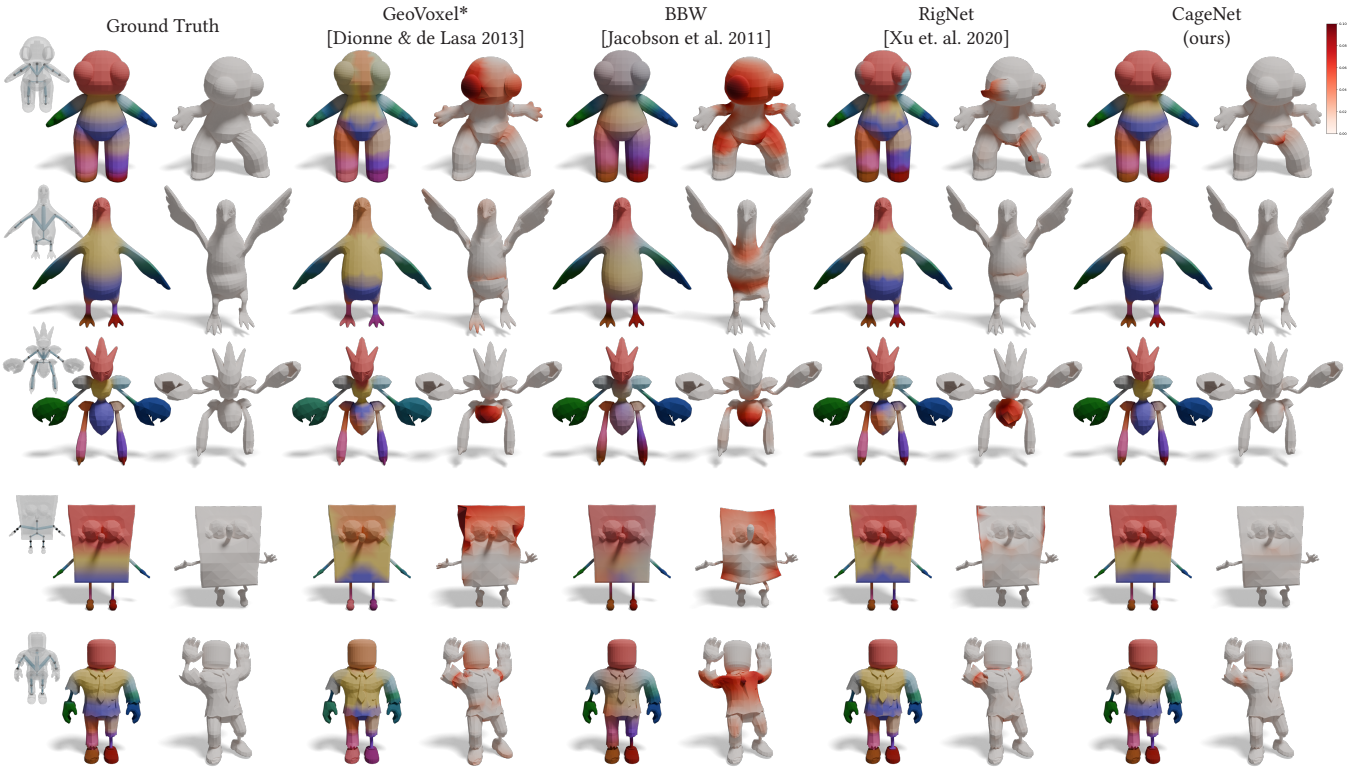

Fig. 12. Comparison against baselines, each column shows the skinning weights (left) and an animation frame resulting from these weights, color coded with the corresponding normalized vertex displacement error (right). Here GeoVoxel* uses the parameters recommended by Xu et al. [2020]. Our approach achieves better skinning weight predictions compared to the baselines, leading to less artifacts in the resulting animation frame. Please refer to the supplementary video for the full animation sequences.

Fig. 13. Our method can be applied to AI-generated meshes, such as [Xiang et al. 2024]. We show qualitative comparisons of the generated skinning weights (left) and a resulting deformation frame (right) on these characters. Here GeoVoxel* uses the parameters recommended by Xu et al. [2020]. As the ground truth weights are not available here, compare with the dataset ground truth in Figure 12. We note that our approach leads to weights which are most similar to the ground truth, with the least artifacts in the deformed frame. Please refer to the supplementary video for the full animation sequences.

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.

Jing Ma, Jituo Li, and Dongliang Zhang. 2024. EasySkinning: Target-oriented skinning by mesh contraction and curve editing. *Comput. Graph.* 124 (2024), 104049. doi:10.1016/J.CAG.2024.104049

Jing Ma and Dongliang Zhang. 2023. TARig: Adaptive template-aware neural rigging for humanoid characters. *Comput. Graph.* 114 (2023), 158–167. doi:10.1016/J.CAG.2023.05.018

Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph.* 36, 4 (2017), 71:1–71:10.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*. 37–45.

Francesco Milano, Antonio Loquercio, Antoni Rosinol, Davide Scaramuzza, and Luca Carlone. 2020. Primal-Dual Mesh Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.).

Thomas W Mitchel, Vladimir G Kim, and Michael Kazhdan. 2021. Field convolutions for surface cnns. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 10001–10011.

Albert Mosella-Montoro and Javier Ruiz Hidalgo. 2022. SkinningNet: Two-Stream Graph Convolutional Neural Network for Skinning Prediction of Synthetic Characters. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*. IEEE, 18572–18581. doi:10.1109/CVPR52688.2022.01804

Xuming Ouyang and Cunguang Feng. 2020. Autoskin: Skeleton-based human skinning with deep neural networks. In *Journal of Physics: Conference Series*, Vol. 1550. IOP Publishing, 032163.

Xiaoyu Pan, Jiancong Huang, Jiaming Mai, He Wang, Honglin Li, Tongkui Su, Wenjun Wang, and Xiaogang Jin. 2021. HeterSkinNet: A Heterogeneous Network for Skin Weights Prediction. *Proc. ACM Comput. Graph. Interact. Tech.* 4, 1 (2021), 10:1–10:19. doi:10.1145/3451262

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. 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: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035.

Yicong Peng, Yichao Yan, Shengqi Liu, Yuhao Cheng, Shanyan Guan, Bowen Pan, Guangtao Zhai, and Xiaokang Yang. 2022. Cagenerf: Cage-based neural radiance field for generalized 3d deformation and animation. *Advances in Neural Information Processing Systems* 35 (2022), 31402–31415.

Adrien Poulenard and Maks Ovsjanikov. 2018. Multi-directional geodesic neural networks via equivariant convolution. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–14.

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.

Yann Savoye and Jean-Sébastien Franco. 2010. Cage-based tracking for performance animation. In *Asian Conference on Computer Vision*. Springer, 599–612.

Silvia Sellán, Oded Stein, et al. 2023. gptyoolbox: A Python Geometry Processing Toolbox. https://gpytoolbox.org/.

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. 2022. DiffusionNet: Discretization Agnostic Learning on Surfaces. *ACM Trans. Graph.* 41, 3 (2022), 27:1–27:16.

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

Dmitriy Smirnov and Justin Solomon. 2021. HodgeNet: learning spectral geometry on triangle meshes. *ACM Trans. Graph.* 40, 4 (2021), 166:1–166:11.

Daniel Ströter, Jean-Marc Thiery, Kai Hormann, Jiong Chen, Qingjun Chang, Sebastian Besler, Johannes Sebastian Mueller-Roemer, Tamy Boubekeur, André Stork, and Dieter W Fellner. 2024. A Survey on Cage-based Deformation of 3D Models. In *Computer Graphics Forum*. Wiley Online Library, e15060.

Jean-Marc Thiery, Élie Michel, and Jiong Chen. 2024. Biharmonic Coordinates and their Derivatives for Triangular 3D Cages. *ACM Trans. Graph.* 43, 4 (2024), 138:1–138:17.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58, 1 (1996), 267–288.

Peng-Shuai Wang, Yang Liu, and Xin Tong. 2022. Dual octree graph networks for learning adaptive volumetric shape representations. *ACM Trans. Graph.* 41, 4 (2022), 103:1–103:15. doi:10.1145/3528223.3530087

Yu Wang and Justin Solomon. 2021. Fast quasi-harmonic weights for geometric data interpolation. *ACM Trans. Graph.* 40, 4 (2021), 73:1–73:15. doi:10.1145/3450626.3459801

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)* 38, 5 (2019), 1–12.

Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. 2020. CNNs on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (ToG)* 39, 4 (2020), 92–1.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. 2024. Structured 3D Latents for Scalable and Versatile 3D Generation. *arXiv preprint arXiv:2412.01506* (2024).

Tianhan Xu and Tatsuya Harada. 2022. Deforming Radiance Fields with Cages. In *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXXIII (Lecture Notes in Computer Science, Vol. 13693)*, Shai Avidan, Gabriel J. Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer, 159–175.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: neural rigging for articulated characters. *ACM Trans. Graph.* 39, 4 (2020), 58. doi:10.1145/3386569.3392379

Yuqi Yang, Shilin Liu, Hao Pan, Yang Liu, and Xin Tong. 2020. PFCNN: Convolutional neural networks on 3D surfaces using parallel frames. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13578–13587.

Zhangsihao Yang, Or Litany, Tolga Birdal, Srinath Sridhar, and Leonidas Guibas. 2021. Continuous geodesic convolutions for learning on 3d shapes. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 134–144.

Wang Yifan, Noam Aigerman, Vladimir G Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. 2020. Neural cages for detail-preserving 3d deformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 75–83.

Shengchao Yuan, Yishun Dou, Rui Shi, Bingbing Ni, and Zhong Zheng. 2023. SieveNet: Selecting Point-Based Features for Mesh Networks. *arXiv preprint arXiv:2308.12530* (2023).

Cem Yuksel. 2015. Sample Elimination for Generating Poisson Disk Sample Sets. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2015)* 34, 2 (2015), 25–32. doi:10.1111/cgf.12538

Kun Zhou, Weiwei Xu, Yiying Tong, and Mathieu Desbrun. 2010. Deformation Transfer to Multi-Component Objects. *Comput. Graph. Forum* 29, 2 (2010), 319–325.

Daniel Zint, Zhouyuan Chen, Yifei Zhu, Denis Zorin, Teseo Schneider, and Daniele Panozzo. 2024. Topological Offsets. *arXiv preprint arXiv:2407.07725* (2024).