# **Supplementary Material**
# CageNeRF: Cage-based Neural Radiance Field for Generalized 3D Deformation and Animation

**Yicong Peng**[1] **Yichao Yan**[2*] **Shenqi Liu**[2] **Yuhao Cheng**[2]
**Shanyan Guan**[2] **Bowen Pan**[3] **Guangtao Zhai**[1*] **Xiaokang Yang**[2]
[1] Institute of Image Communication and Network Engineering, Shanghai Jiao Tong University
[2] MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
[3] Alibaba Group
{jack-sparrow,yichaoyan,lsqlsq,chengyuhao,shyanguan,zhaiguangtao,xkyang}@sjtu.edu.cn
bowen.pbw@alibaba-inc.com

## A Derivation of Mean Value Coordinate

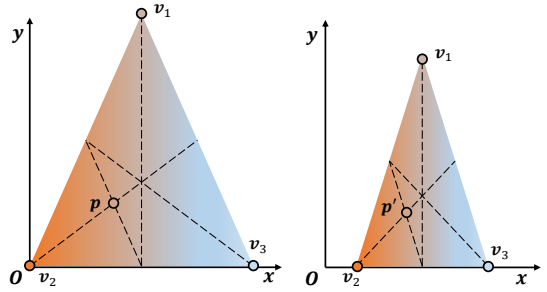


Figure 1: **MVC and cage deformation in 2D case.** MVC defined by basis vertices $\mathbf{v}_{1,2,3}$ are visualized through color. By translating the basis vertices to novel positions point $\mathbf{p}$ is shifted to $\mathbf{p}'$ while maintaining its MVC.

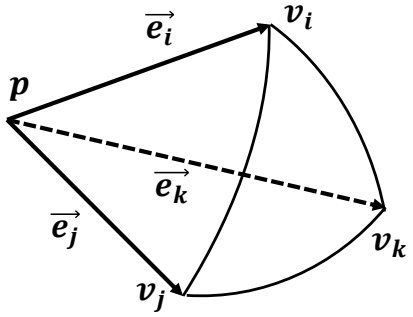


Figure 2: **MVC extended to 3D scenario via unit sphere projection.** $\mathbf{v}_{i,j,k}$ are projected points on the unit sphere with center $\mathbf{p}$ from $\hat{\mathbf{v}}_{i,j,k}$ of closed polygon mesh $\Omega$, and $\vec{\mathbf{e}}_{i,j,k}$ are unit normals with origin $\mathbf{p}$.

Mean value coordinates (MVC) were first introduced in [2] as a way of representing points using convex combinations of vertices from a polygon. It was later extended to work with 3D closed meshes [3, 7]. Mean value coordinates encode the position of a given point with respect to a set of predefined vertices. The relationship between the MVC of a given point and the predefined vertices is analogous to that between a vector and its corresponding basis in linear algebra, i.e., the predefined vertices act as the basis in MVC space. Figure 1 illustrates the cage-based deformation using MVC coordinates in the 2D scenario. Given the predefined vertices $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$, which act as the basis in the MVC space, we can express arbitrary points inside the enclosed triangle as linear combinations of these vertices. For example, the midpoint of line segment $\overline{\mathbf{v}_1\mathbf{v}_2}$ can be expressed as $\frac{1}{2} \cdot \mathbf{v}_1 + \frac{1}{2} \cdot \mathbf{v}_2$ and hence having the MVC of $(0.5, 0.5, 0)$. In Figure 1, we visualize the MVC of points inside the triangle by using different colors, and provide the Cartesian plane with origin $\Omega$ as the reference coordinate system. Compared to the left, basis vertices are translated to novel positions. All points inside the enclosed triangle are shifted and thus have different coordinates in the Cartesian coordinate

*Corresponding authors

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

system. Notice that while the position of an arbitrary point $\mathbf{p}$ is shifted to $\mathbf{p}'$ after the deformation, its MVC remain unchanged, i.e., $\mathbf{p}$ and $\mathbf{p}'$ share the same color.

In general, given an arbitrary point $\mathbf{p}$ inside the convex kernel $\mathbf{K}$ of a polygon $\Omega$ with $n$ vertices, we can calculate the MVC by constructing a series of non-negative functions $\phi_1, \phi_2, \cdots, \phi_n : \mathbf{K} \to \mathbb{R}$, such that:

$$\sum_{i=1}^{n} \phi_i(\mathbf{p}) = 1 \quad \text{and} \quad \sum_{i=1}^{n} \phi(\mathbf{p})\mathbf{v}_i = \mathbf{p}, \tag{1}$$

where $\mathbf{v}_i$ is the $i$-th ordered vertex of $\Omega$. The MVC is defined by:

$$\phi_i = \frac{w_i}{\sum_{j=1}^{n} w_j}, \quad w_i = \frac{1}{r_i}(\tan\frac{\alpha_i}{2} + \tan\frac{\alpha_{i-1}}{2}), \tag{2}$$

where $\alpha_i$ is the angle $\angle \mathbf{v}_i\mathbf{p}\mathbf{v}_{i+1}$, and $r_i(\mathbf{p}) = \|\mathbf{v}_i - \mathbf{p}\|$. To extend this case to the 3D scenario, given a closed polygon mesh $\Omega$, its vertices $\hat{\mathbf{v}}_i \in \Omega$ are first projected to a unit sphere with $\mathbf{p}$ at its center. As shown in Figure 2, $\{\vec{\mathbf{e}}_i, \vec{\mathbf{e}}_j, \vec{\mathbf{e}}_k\}$ are unit vectors from $\mathbf{p}$ to the projected vertices $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ on the unit sphere. The MVC of the point $\mathbf{p}$ takes the same form as Equation 1, except that $\phi_i$ is defined by:

$$\phi_i = \frac{w_i}{\sum_{j=1}^{n} w_j}, \quad w_i = \frac{1}{r_i} \sum_{\mathbf{v}_i \in \mathbf{T}} \mu_{i,\mathbf{T}}, \tag{3}$$

where $T$ is the spherical triangle enclosed by $\{\mathbf{v}_i, \mathbf{v}_j, \mathbf{v}_k\}$ and $\mu_{i,\mathbf{T}}$ is given by:

$$\mu_{i,\mathbf{T}} = \frac{\beta_{jk} + \beta_{ij}\vec{\mathbf{n}}_{ij} \cdot \vec{\mathbf{n}}_{jk} + \beta_{ki}\vec{\mathbf{n}}_{ki} \cdot \vec{\mathbf{n}}_{jk}}{2\vec{\mathbf{e}}_i \cdot \vec{\mathbf{n}}_{jk}}, \quad \text{and} \quad \beta_{rs} = \angle \mathbf{v}_r\mathbf{p}\mathbf{v}_s \tag{4}$$

$$\vec{\mathbf{n}}_{rs} = \frac{\vec{\mathbf{e}}_r \times \vec{\mathbf{e}}_s}{\|\vec{\mathbf{e}}_r \times \vec{\mathbf{e}}_s\|}. \tag{5}$$

# B Datasets

We train our framework on different datasets and evaluate CageNeRF on several tasks. In the following section, we provide a detailed description of the employed datasets. We will make the synthetic data publicly available.

For the task of **Geometric Editing via Deformation Transfer**, we use a synthetic dataset of an office chair, which act as the static object inside the canonical space, to train our neural renderer. We also employ the sampled objects from ShapeNet [1] as deformation targets and comparison baseline to evaluate the editing results. Our synthetic dataset of the office chair is created from a polygon mesh with colored texture using 3D render software. The dataset contains around 1.2K $360^\circ$ inward-facing multiview images with corresponding masks and camera parameters. Some samples of the multiview images and masks are shown in Figure 3. As for the camera parameters, we store the camera rotation $\mathbf{R} \in \mathbb{R}^{3\times3}$, camera translation $\mathbf{T} \in \mathbb{R}^{1\times3}$, and the camera intrinsic parameters $\mathbf{K} \in \mathbb{R}^{3\times3}$. Since we use the same camera for the entire render process, the intrinsic parameters are shared across all the images, while the rotation and translation vary with the rendering view.

In the experiment of **Neural Animation**, we use two types of datasets: 1) a synthetic dataset of animated objects driven by linear blend skinning, and 2) a real-world dataset, i.e., Human3.6M [6], which is often utilized as a benchmark for human body synthesis. The synthetic datasets contain two characters, one contains a comic character Hulk, and the other contains a swimming blue whale. The deformable meshes of both objects are fully rigged with corresponding skeletons and blend weights. Both objects are capable of producing short animation clips using liner blend skinning as driving method. Similar to the synthetic dataset used in **Geometric Editing**, we render 1.2K $360^\circ$ inward-facing multiview images of the animatable objects at the starting frame. The deformed polygon meshes of the following frames are also extracted as the animation target for our cage deformer module to generate new deformation fields. The Human3.6M dataset was captured by 4 synchronized inward-facing cameras from different viewpoints. Every section of this dataset captures a short clip of a posing performer with 2D and 3D joint annotations and per-frame masks. For each frame in this clip, a SMPL [9] human model is estimated based on the annotations and masks for its


Figure 3: **Samples from synthetic dataset.** We render the inward-facing 360° multiview images along with corresponding masks. The masks are used during ray sampling on the training image to improve sample efficiency.

position, rotation, and joint locations. This estimated SMPL model, similar to the animated mesh in our synthetic dataset, is used in our cage deformer to generate deformations on the radiance field. We provide additional results on synthesizing human body on different subjects in Figure 4.


Figure 4: **Additional results in comparison with AniNeRF.** We render various poses of different subjects in Human 3.6M dataset and achieve comparable results with the state-of-the-art human synthesis method. Note that our framework does not require additional training or linear blend skinning information when rendering novel poses.

As for **Pose Reenactment**, we also create a synthetic dataset in the same manner as the office chair dataset used in **Geometry Editing**. We use a static robot model with colored texture, instead of an office chair, to create this dataset. We sample our reenactment targets from the Surreal [4] dataset.

# C Implementation Details

## C.1 Training Configuration

We train our network on a single Nvidia RTX3090 GPU for 500 epochs, depending on the complexity of the geometry and texture. The overall training process takes around 22 hours for a dataset with 1K multi-view images. We take ADAM optimizer [8] with an exponential learning rate scheduler. We set our initial learning rate to $5e-4$ and our learning rate decays by a factor of 10 for every 500 epochs. We implement our entire framework using PYTORCH [11] deep learning library. It roughly takes 12G GPU memory to train our framework. Our framework achieves the average test-time rendering speed of 2.071 seconds per frame with the resolution of $800 \times 800$.

## C.2 Network Structure

Different from the original NeRF [10], which optimizes two radiance fields, i.e., a coarse and a fine model, the neural render in our framework only optimizes a single multi-layer perceptron (MLP) network. We sample 128 points along each camera ray during volumetric rendering. We also provide a trainable latent code to compensate for the texture and normal variances of the object inside the canonical space. The structure of the neural renderer is shown in Figure 5. Our neural renderer has a similar structure to the original NeRF network except for the additional latent code input. We take inspiration from [14, 5] and design our cage deformer in a similar encoder-decoder manner. We use the PointNet++ [12] with multi-scale grouping strategy to extract the feature from 3D objects in our encoder. For the decoder, we adopt an MLP network to predict the vertex-wise offset for the input cage base on the extracted feature.

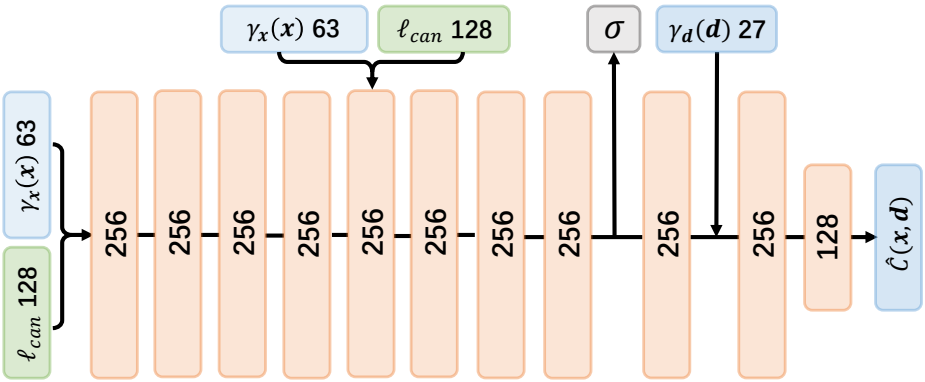


Figure 5: **Network structure of the Neural Renderer module.** Our neural renderer adopts a structure similar to the vanilla NeRF save for the additional latent code input. Numbers inside the colored block indicate the dimension of the input or the linear layer. $\gamma(\cdot)$ is the embedding function. The neural renderer takes in concatenated input of embedded position $\mathbf{x}$ and latent code $l_{can}$ and outputs density $\sigma$ and estimated color $\hat{C}$.

# D Additional Experiment Analysis

## D.1 Impact of the Network Structure

As mentioned in our paper, we design our framework in a decoupled manner, the underlying structure of our neural renderer module, which learns the implicit representation of a static object in canonical space, can be replaced with other designs without affecting the integrity of our framework. To validate this point, we prepare two neural renderers with different structures and carry out experiments for the task of neural animation. Our neural renderer adopts a

Table 1: **Impact of network structure in render quality measured in PSNR.**

| | Hulk | Whale | H36M |
|---|---|---|---|
| **NeRF** | 34.672 | 35.154 | 22.605 |
| **Ours** | 35.280 | 35.857 | 23.583 |

modified NeRF structure, as described in Section C.2, with additional latent code to represent the appearance of the object inside the canonical space, while the other uses the vanilla version of NeRF. Table 1 shows the impact of different network structures on the rendering quality across three datasets. When using the vanilla NeRF structure as the neural renderer, the rendering quality slightly drops w.r.t. PSNR. The introduced additional latent code is effective in improving the rendering quality for both the synthetic and real-world datasets.

### D.2 Impact of Mesh Accuracy

Our cage-based deformation method utilizes a geometry proxy extracted from the learned radiance field as a 3D mask. The quality of this geometry proxy affects the point sampling process during volumetric rendering. To examine its influence on our framework, we apply deformation transfer to our canonical radiance field using the MVC field generated from the ground truth mesh and the mesh reconstructed by the signed distance function [13]. As shown in Figure 6, while the reconstructed mesh has minor artifacts in terms of smoothness and geometric detail, the overall shape and structure match the ground truth. The artifacts and dilation in certain areas such as the armrest causes blurred and distorted texture. This result shows the accuracy of geometry proxy has a noticeable effect in rendering quality. In the future, one possible direction to further improve the flexibility of CageNeRF is to learn geometry directly from multi-view images.

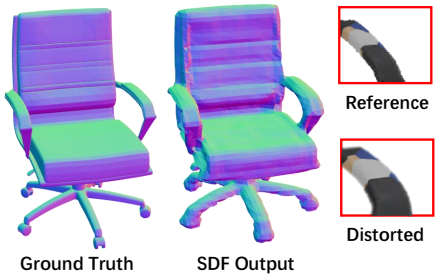


Figure 6: **Comparison between the ground truth and mesh extracted using SDF.**

### D.3 Functionality of Explicit Mesh

The explicit mesh extracted in our method has the following functionalities:

**Enhance Deformation Capability.** While our framework supports explicit deformation by manually editing the shape of the canonical cage, we want to enhance the deformation capability further. Neural Cage [14] offers impressive performance with explicit mesh in tasks like deformation transfer and pose reenactment. By incorporating an explicit mesh into our framework, we are able to transfer the capability of Neural Cage, i.e., a mesh-based method, to neural radiance field.

**Increase Sampling Efficiency.** As mentioned in our paper Section **??**, the canonical radiance field is sparsely occupied by the render target. Restricting the sample points inside the vicinity space of the render target can effectively reduce the sample points down to $10\%$ of the original number required in volume rendering.

**Provide Robust Correspondence.** Our framework handles deformation by first learning a static radiance field inside the canonical space. However, this canonical space is not always directly accessible. For example, the Human 3.6M dataset only contains posing human subjects captured by multi-view cameras, the canonical space, i.e., "T-pose" images of the target is not given. This requires first estimating the deformed cages of the targets in various poses and performing cage-based deformation backward to transform posing targets to canonical "T-pose". An explicit mesh can be used to generate correct MVC field which enables us to robustly establish this backward deformation. Cage-based deformation can express the forward deformation correctly, i.e., from the canonical space to the deformed space, but not always vice versa. In some cases, both Neural Cages and our cage deformer module will output a deformed cage with overlapping faces or collapsed vertices. Naively taking the MVC value calculated in such deformed cages will induce artifacts, since points that reside in such regions cannot be correctly projected into the canonical space. This property can be interpreted as: CBD using a cage with a self-overlapping structure is similar to applying a matrix that is not full rank, and hence not invertible. We provide backward deformation results on SMPL model using cage with and without self-overlapping structures in. Additionally, if the cage vertices are manually edited without overlapping faces and collapsed vertices, we can generate MVC field directly based on this deformed cage, i.e., without the need of an explicit mesh, since this cage deformation is invertible from deformed space to canonical space.

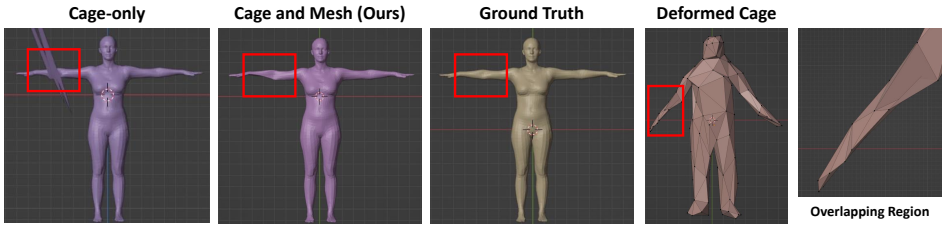


Figure 7: **Comparison of inverse cage-based deformation w/o an explicit mesh.** We perform a backward deformation a SMPL model using a self-overlapping cage. The "A-pose" cage resides in the deformed space, and the "T-pose" SMPL is the object in canonical space. As is shown in the top row, backward deformation using only the cage will produce artifacts such as spikes and outlier vertices, while using an explicit mesh can avoid this artifact. The difference between our backward deformation result and the ground truth can be compensated by the sampling vicinity $\epsilon$.

## E Broader Impacts

As shown in our experiments, our work is capable of producing high-fidelity results of edited and animated 3D objects with 2D images as its input. Our framework may find application in 3D animation, digital human creation, high fidelity 3D object editing. Especially, the deformation method used in our framework is category-agnostic, our framework can be applied in cross-domain animation retargeting and reenactment.

## References

[1] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[2] Michael S. Floater. Mean value coordinates. *CAGD*, 20(1):19–27, 2003.

[3] Michael S. Floater, Géza Kós, and Martin Reimers. Mean value coordinates in 3d. *CAGD*, 22(7):623–631, 2005.

[4] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 3d-coded: 3d correspondences by deep deformation. In *ECCV*, volume 11206, pages 235–251, 2018.

[5] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. A papier-mâché approach to learning 3d surface generation. In *CVPR*, pages 216–224, 2018.

[6] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE TPAMI*, 36(7):1325–1339, 2014.

[7] Tao Ju, Scott Schaefer, and Joe D. Warren. Mean value coordinates for closed triangular meshes. *ACM TOG*, 24(3):561–566, 2005.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[9] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: a skinned multi-person linear model. *ACM TOG*, 34(6):248:1–248:16, 2015.

[10] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, pages 405–421, 2020.

[11] 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. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019.

[12] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, pages 5099–5108, 2017.

[13] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NIPS*, pages 27171–27183, 2021.

[14] Yifan Wang, Noam Aigerman, Vladimir G. Kim, Siddhartha Chaudhuri, and Olga Sorkine-Hornung. Neural cages for detail-preserving 3d deformations. In *CVPR*, pages 72–80, 2020.