CageNeRF: Cage-based Neural Radiance Field for Generalized 3D Deformation and Animation

Anonymous Author(s)
Affiliation
Address
email

Abstract

While implicit representations have achieved high-fidelity results on 3D rendering, deforming and animating the learned implicit field remain a challenging task. Existing works typically leverage specific 3D model as deformation prior, such as SMPL for human animation. However, the category-specific prior dependency limits them to generalize to other objects. In this work, we propose a novel framework for deforming and animating the neural radiance field learned on *arbitrary* objects. The key insight is that we introduce a cage-based representation as deformation prior, which is category-agnostic. Specifically, the deformation is performed based on an enclosing cage with sparsely defined vertices inside the rendering space, where each point is projected into a novel position based on the barycentric interpolation of the deformed cage vertices via weight functions. In this way, we transform cage into a generalized constraint, which is able to deform and animate arbitrary target while preserving geometry details. Based on extensive experiments, we demonstrate the effectiveness of our framework in the task of geometry editing, object animation and deformation transfer.

1 Introduction

Editing and manipulating 3D object is a fundamental task for generating immerse contents, which enables numerous applications in VR/AR. Although recent advances in implicit neural rendering, such as neural radiance field (NeRF) [30], are able to produce photo-realistic modeling of static objects, how to edit the 3D content remains a challenging task. In this paper, we focus on learning a general deformation field for NeRF, enabling the deformation and animation of general objects.

Editing explicit 3D representation, such as point cloud and mesh, has been well studied. Point cloud and deformable mesh [7, 3, 14] store the geometry information in 3D space, which are directly editable and are widely utilized in computer graphics for modeling dynamic objects [31]. Cage-based deformation [21, 22, 26], for example, has been used to animate polygon mesh, where per-vertex deformation can be generated by translations prescribed on a coarse cage. While mesh and point cloud representations are relatively flexible to edit and animate, it is difficult to produce geometrically accurate and visually-faithful complex objects and scenes. Additional data such as normal maps and textures are usually required to achieve high-quality rendering results.

Recent works on implicit neural rendering, such as NeRF, employ multi-layer perceptron (MLP) as a universal approximator [16, 37], allowing them to approximate a continuous volumetric scene function to reconstruct high-fidelity results with only sparsely sampled images. Therefore, it is highly desirable if the neural rendering models can be also applied to model dynamic objects. However, optimizing a deformable radiance field end-to-end for dynamic objects like human body is a highly under-constrained problem. To address this issue, when modeling dynamic objects, most works optimize inside a deformation-invariant canonical space with data-dependent priors, such as

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

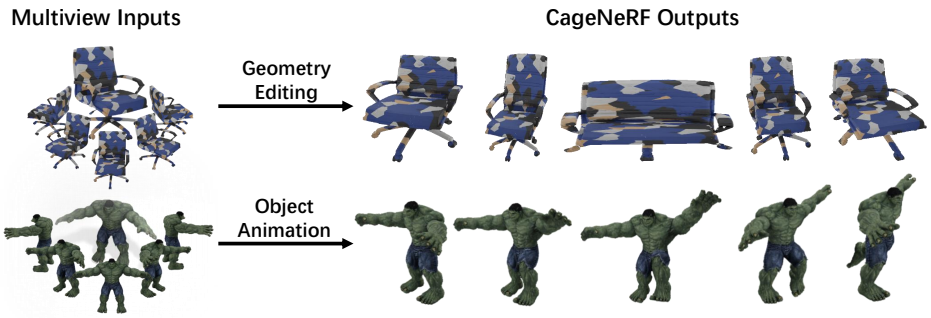

Figure 1: Given multiview images of an object, our method learns an implicit deformable 3D representation, which is used to deform and animate the object.

3DMM [4] for face deformation [1, 15], and SMPL [27] for human animation [35, 33]. While using data-dependent priors when modeling specific dynamic objects proved to be effective in providing the stable correspondence between the object structure and the implicit field, it also limits the model's capability to generalize for broader applications.

To address this issue, we propose a novel framework which focuses on deforming and animating general objects represented by neural radiance fields, as shown in Figure 1. We take inspiration from the vanilla cage deforming approaches [21, 45] and extend them to work with implicit neural radiance fields. However, directly applying cage deformation into implicit field is non-trivial, due to the following issues. i) Traditional cage deformation methods work on polygon mesh with finite vertices and faces, while implicit radiance field is a continuous 3D volume. ii) Cage-based deformation transfer requires explicit geometry of the target, which is not directly accessible in the implicit field. iii) It is computationally expensive to densely sample and discretize the implicit field. To address these issues, we propose to estimate a coarse geometry of the target and generate a discrete field which supports bi-directional mapping between the canonical space and the deformed space, while keeping the radiance field continuous. Moreover, we further design a novel sampling strategy to reduce the computational cost while maintaining the render quality.

The advantages of our framework are three-fold. First, by utilizing category-agnostic cage-based deformation, our framework does not rely on data-dependent priors and can be generalized to work with a wide rage of objects. Second, since cage-based deformation is a self-contained method, once the model is trained on the static scene, no additional training is required to apply deformation. Third, both cage deformation and volumetric rendering operate on the Euclidean space, we can design our framework in a decoupled manner as long as we keep the underlying coordinate system consistent between the two modules. This increases the flexibility and versatility of our framework. Moreover, as a well-developed method in computer graphics, cage-based deformation can be used in various tasks, which further expands the application scenario of our framework. We demonstrate the effectiveness of our framework in geometry editing, object animation, and deformation transfer.

2 Related Work

Novel View Synthesis. Traditional methods for novel view synthesis rely on explicit representations of 3D scenes. In these methods, 3D objects are modeled by meshes, point clouds [47, 38], etc, and then novel views are generated by shading and rendering. However, high-fidelity textures and well-designed 3D models are required for photo-realistic rendering, which makes the procedure highly expensive and inefficient. Recently, NeRF [30] has achieved impressive performance on novel view synthesis, where 3D objects are represented by implicit functions which map positions to densities and colors. A number of follow-up works have made further improvements to NeRF to yield synthesis results with higher quality [50, 2, 49, 43]. Although these NeRF-based implicit methods are good at representing static scenes, it is still challenging to model dynamic scenes.

Explicit Geometry Deformation. Modeling deformable 3D objects is a fundamental task in computer graphics and computer vision, and triangle meshes are most commonly utilized due to their flexibility in manipulation. Traditional methods achieve this by deforming a fixed set of vertex points and then

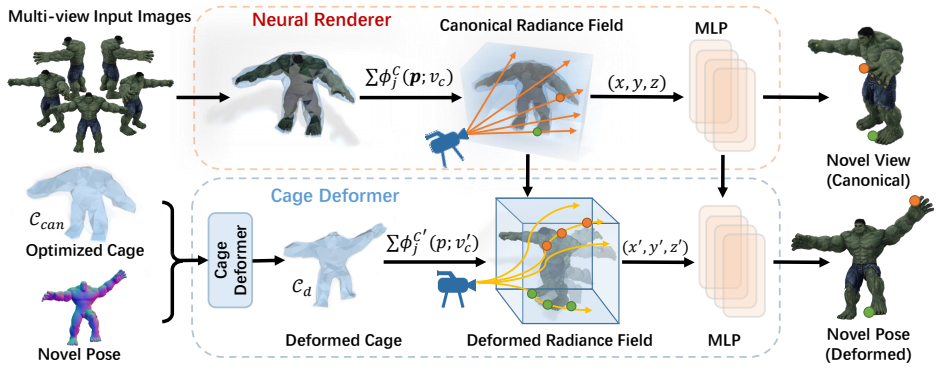

Figure 2: **Overview of CageNeRF.** CageNeRF consists of two modules: (1) a neural rendering module, which learns a radiance field from multi-view input images in the canonical space; (2) a cage deformer module that produce general cage deformation to manipulate the learned radiance field. CageNeRF deforms the implicit radiance field in an end-to-end manner.

interpolating values among them [23, 44], such as skinning on skeletons [18, 29] or cages-based methods [22, 39, 52]. Along with the advances of large-scale datasets [6, 8, 48], learning-based methods are proposed by analyzing the prior principle of mesh deformation [10, 45, 11, 20]. Wang et al. [45] proposed a differentiable cage-based deformation module to synthesize shape variations and perform deformation transfer. However, these deformation methods can be only applied to explicit models, e.g., meshes. It is expensive to generate high-fidelity textures for photo-realistic rendering, and these methods cannot be directly applied to implicit models.

Implicit Geometry Deformation. Compared to explicit representations that only discretely control the vertex of meshes, the deformation of implicit geometry is more challenging, since the space is represented with a continuous function. Recently, various approaches build different deformation fields on implicit representations for animation [32, 19, 40, 41]. However, the generalization abilities of these methods are limited with respect to novel poses and different objects. Referring to the widely concerned digital humans, a vast range of methods are proposed for the deformation of the implicit face [13, 53, 15] and body [35, 9, 36, 34]. However, these methods heavily rely on parametric models as priors, such as 3DMM [5, 25] and SMPL [27], which are difficult to transfer to non-humanoid objects. In this paper, we propose a cage-based implicit geometry deformation method, which utilizes a sparse neural cage as guidance to build the deformation field for NeRF. The proposed CageNeRF is a general algorithm which is applicable to a wide variety of objects.

3 Methodology

We propose a novel framework, dubbed CageNeRF, for editing and animating arbitrary 3D objects by seamlessly collaborating the cage-based deformation with neural rendering. Cage-based deformation operates on explicit 3D geometry representations such as polygon mesh, where each vertex is given a mean value coordinate (MVC), which is calculated through barycentric interpolation. MVC encodes the relative position of a given point w.r.t. the vertices on the cage. Deformation on the mesh is carried out by first translating the cage vertices and then using the relative position encoded by MVC to calculate a new position for each mesh vertex. As shown in Figure 2, our framework consists of two modules: a neural rendering module and a cage deformer module. We first optimize the radiance field of a given static object inside canonical space using multiview images with known camera parameters. Then the cage deformer module warps the cage and reconstructs the implicit radiance field by translating the points inside the canonical space, based on the deformed cage vertices.

3.1 Canonical Space Neural Rendering

We first train our neural renderer module to acquire the NeRF volume [30], which acts as a reference canonical space. NeRF models the scene along with its geometry, lighting, and material as a continuous volume and approximates it as a 5D radiance field using an MLP network. Given a 3D

position of a point $\mathbf{p} = (x, y, z)$, the network learns a mapping function of its density σ and color $\mathbf{c} = (r, g, b)$ variations with respect to the view direction $\mathbf{d} = (\theta, \phi)$:

$$\mathbf{F}_{\mathbf{\Theta}} : (\gamma(\mathbf{p}), \gamma(\mathbf{d})) \rightarrow (\mathbf{c}, \sigma), \tag{1}$$

where $\gamma(\cdot)$ is a positional embedding function of sinusoidal basis and Θ is the trainable parameters. Then, volumetric rendering is performed to synthesize images by marching camera rays $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ from viewpoint $\mathbf{o}$, with direction $\mathbf{d}$, through the radiance field. The pixel color $\hat{C}$ is estimated using discrete integration on points sampled along each ray:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} \exp(-\sum_{j=1}^{i-1} \sigma_j \delta_j)(1 - \exp(-\sigma_i \delta_i))c_i, \tag{2}$$

where $\delta_i = t_{i+1} - t_i$ is the interval between adjacent sample points. We then optimize our neural render module inside this canonical space using L_2 loss calculated between the estimated color $\hat{C}(\mathbf{r})$ and ground truth pixel value $C(\mathbf{r})$ of the input images.

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \|\hat{C}(\mathbf{r}) - C(\mathbf{r})\|_2^2, \tag{3}$$

where $\mathcal{R}$ denotes the set of rays that contributes to sampled pixel.

3.2 Cage Optimization

Since it is non-trivial to establish the correspondence between the deformation target (often in explicit form) and the deformation source (i.e., our implicit radiance field), explicit transformation cannot be directly applied to the canonical space. In contrast, cage-based deformation can be applied universally to any point inside the canonical space via barycentric interpolation using the vertices of a given cage. We notice that this space is sparsely occupied by our model, it is impractical and inefficient to densely sample the entire space for deformation. Therefore, we take the explicit geometry of the object as a proxy for our cage deformer module. Specifically, the geometry $\mathcal{G}_{can}$ can be extracted with either marching cubes algorithm [28] or by optimizing a neural SDF representation [43] using the same training dataset as our neural renderer module:

$$\mathcal{G}_{can} = \{x \in \mathbb{R}^3 | f_s(x; \theta) = 0\}, \tag{4}$$

where $f_s(x; \theta)$ is the SDF function with learnable parameters θ, and the geometry of the object is the zero-level set of $f_s(x; \theta)$. We then obtain the geometry proxy $\mathcal{G}_p(\mathbf{v}, \mathbf{f})$ with vertices $\mathbf{v} \in \mathbb{R}^{N \times 3}$ and faces $\mathbf{f}$ by uniformly sampling vertices points $\mathbf{v}$ from $\mathcal{G}_{can}$, and building faces $\mathbf{f}$ using the nearest neighbor algorithm.

Since cage-based deformation in our framework warps the entire neural volume that the cage encloses, the object must reside inside the warping space. Considering that $\mathcal{G}_p(\mathbf{v}, \mathbf{f})$ can be of arbitrary shape, we adopt an optimization process that translates a template cage $\mathcal{C}_t(\mathbf{v}, \mathbf{f})$ into an optimized cage $\mathcal{C}_{can}(\mathbf{v}, \mathbf{f})$, which geometrically aligns and encloses this object. As a starting point, we provide our cage deformer module with the template cage $\mathcal{C}_t(\mathbf{v}, \mathbf{f})$. Then, we calculate the MVC of points uniformly sampled from the geometry proxy $\mathcal{G}_p$. Since negative in MVC occurs when 1) the point lies outside the cage and 2) the cage itself is self-overlapping or highly concave, by penalizing the negative MVC values of the points sampled from the extracted object geometry, our network automatically adapts to the template cage:

$$\max_{\mathbf{v}} \min_{i} \phi_i\{\mathbf{v}\}, \mathbf{v} \in \mathcal{G}_p(\mathbf{v}, \mathbf{f}), i = 1, 2, \cdots, N, \tag{5}$$

where ϕ_i is the calculated MVC weight w.r.t. vertex $\mathbf{v}_i$ of cage $\mathcal{C}_t(\mathbf{v}, \mathbf{f})$.

3.3 Cage-based Deformation

After acquiring a fitting cage $\mathcal{C}_{can}(\mathbf{v}, \mathbf{f})$ from the optimization process, we are able to perform deformation based on the target novel pose/shape and the current cage shape. We take inspiration from Neural Cage [45] and adopt an encoder-decoder structure to calculate the displacement prescribed on each vertex of the optimized cage $\mathcal{C}_{can}$, as shown in Figure 3.

We pretrain our cage deformer in two different datasets: 1) ShapeNet [8] for deformation transfer and geometry editing on general objects; 2) an extended version of Surreal dataset [12] for better generalization ability on novel pose synthesis and animation. Since the optimized cage in the canonical space remains invariant during deformation, we extract the cage feature f_C from $\mathcal{C}_{can}$ using $\mathbf{E}_C$, and concatenate it to the feature extracted from novel pose geometry $\mathbf{f_N}$. The output of our decoder is the displacement $\Delta\mathbf{v}_i$ for each vertex on the optimized cage $\mathcal{C}_{can}(\mathbf{v}, \mathbf{f})$:

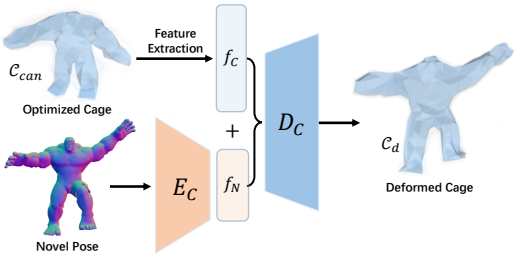

Figure 3: **Structure of the cage deformer.**

$$f_c = \mathbf{E}_C(\mathcal{C}_{can}(\mathbf{v}, \mathbf{f})), \tag{6}$$

$$\Delta\mathbf{v} = \mathbf{D}_C(\mathbf{E}_C(\mathcal{G}(\mathbf{v}, \mathbf{f})) \oplus f_C), \Delta\mathbf{v} \in \mathbb{R}^{N\times 3}, \tag{7}$$

where $\mathcal{G}$ is the geometry representing the target pose and $\oplus$ refers to vector concatenation. Since the cage deformation only changes vertex position while preserving its topology, the deformed cage $\mathcal{C}_d$ can be obtained by adding the displacement $\Delta\mathbf{v}$:

$$\mathcal{C}_d(\mathbf{v}, \mathbf{f}) = \mathcal{C}_{can}(\mathbf{v} + \Delta\mathbf{v}, \mathbf{f}) \tag{8}$$

The deformed cage $\mathcal{C}_d$ is later used to generate an MVC field for editing the radiance field .

3.4 Radiance Field Editing

Once the deformed cage $\mathcal{C}_d$ is obtained by the cage deformer module, we are able to edit the canonical radiance field by building a dense MVC field, which is generated by sampling the deformed space using both $\mathcal{C}_d$ and deformed geometry $\mathcal{G}_n$.

Geometry Proxy as 3D Mask. Since our radiance field is sparsely occupied by the render target. In order to increase the efficiency of our framework while maintaining render quality, we employ the deformed geometry proxy as a 3D mask and only points within the vicinity of this mesh are sampled during volumetric rendering:

$$\mathbf{P}_{render} = \{\mathbf{p} = \mathbf{o} + t\mathbf{d} | \|\mathbf{p} - \mathbf{p}_n\|_2 \leq \epsilon\}, \tag{9}$$

where $\mathbf{P}_{render}$ is the set of points sampled during volumetric rendering, $\mathbf{o}$ is the camera origin $\mathbf{d}$ is the direction of camera ray, $\mathbf{p}_n$ is the closest point on $\mathcal{G}_n$ to $\mathbf{p}$ and ϵ is the vicinity threshold, which is set to 0.03 in our experiments.

To obtain this deformed geometry proxy $\mathcal{G}_n$, given the optimized cage $\mathcal{C}_{can}$, and the geometry proxy $\mathcal{G}_p(\mathbf{v}, \mathbf{f})$, where every point $\mathbf{p} \in \mathcal{G}_p$ has its MVC determined through barycentric interpolation. The MVC coordinate of a point $\mathbf{p}$ with vertices $\mathbf{v}_j \in \mathcal{C}_{can}$ as its basis is denoted as

$$\mathbf{p} = \sum_{j=1}^{N} \phi_j^{\mathcal{C}_{can}}(\mathbf{p})\mathbf{v}_j, \mathbf{v}_j \in \mathcal{C}_{can}, \tag{10}$$

where $\phi_j^{\mathcal{C}}(\cdot)$ is a series of weight function depending on the relative position between the cage vertices $\mathbf{v_c}$ and the given point $\mathbf{p}$. The detailed derivation of the 3D barycentric interpolation is given in the supplementary material. $\mathcal{G}_n$ can be obtained by taking the same MVC calculated under $\mathcal{C}_{can}$ and applying it with vertices $\mathbf{v'}_j \in \mathcal{C}_d$ as its new basis:

$$\mathbf{p}' = \sum_{j=1}^{N} \phi_j^{\mathcal{C}_{can}}(\mathbf{p})\mathbf{v'}_j, \mathbf{v'}_j \in \mathcal{C}_d, \mathbf{p}' \in \mathcal{G}_n. \tag{11}$$

Anti-aliasing MVC Sampling. Traditional cage-based deformation only considers points reside on a mesh surface $\mathcal{S}$, which essentially generates an MVC mesh $\mathcal{M}$ under a given cage $\mathcal{C}_0$ with N vertices, where every point $\mathbf{p} \in \mathcal{M}$ is represented via an N-dimensional MVC coordinate:

$$\mathcal{M} = \{\mathbf{p} = (\phi_1^{\mathcal{C}}, \phi_2^{\mathcal{C}}, \cdots, \phi_N^{\mathcal{C}}) | \mathbf{p} \in \mathcal{S}, \mathcal{C} = \mathcal{C}_0\}. \tag{12}$$

As shown in Equations 11,12, while the coordinate representation of point changed in Euclidean space, the MVC representation remains invariant. We extend this property on meshes into 3D volumes, which yields our editable radiance field. By utilizing this invariant correspondence, we are able to effectively edit the radiance field via volumetric cage-based deformation:

$$\mathcal{F}_d = \{\mathbf{p} = (\phi_1^{\mathcal{C}}, \phi_2^{\mathcal{C}}, \cdots, \phi_N^{\mathcal{C}}) | \mathbf{p} \in \mathcal{F}_{can}, \mathcal{C} = \mathcal{C}_{can}\}, \tag{13}$$

where $\mathcal{F}_d$ is the deformable radiance field represented using MVC coordinate and $\mathcal{F}_{can}$ is the learned canonical radiance field. We can fully express the deformation by keeping track of the point's MVC coordinates inside the canonical space and replacing the original basis using vertices $\mathbf{v}'_d \in \mathcal{C}_d$.

To reduce computation cost, we build a pre-sampled MVC field based on the deformed geometry proxy $\mathcal{G}_n$, as is shown in Figure 4. Specifically, we apply a grid sampling strategy to the deformed space, and the MVC of the grid point $\mathbf{p}_g$ is equal to the coordinates of $\mathbf{p}_n \in \mathcal{G}_n$, which is the nearest point on the deformed geometry proxy $\mathcal{G}_n$ to $\mathbf{p}_g$.

While this sampling strategy is efficient, it causes the aliasing issue as shown in Figure 4. The points $\{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2\}$ are sampled grid points, since they all project to the same closest point $\mathbf{p}_n$ on the deformed object $\mathcal{G}_n$, i.e., they share the same MVC value. Since the point position in the cage space is only determined by its MVC value, when transformed back to canonical space, these points are "squeezed" into the same point on $\mathcal{G}_p$. However, these points are sampled with different Euclidean coordinates during volumetric rendering, this "squeezing" effect in canonical space in turn yields an "expanding" effect in the deformation space. This causes aliased and inflated render results, which cannot correctly reflect the implicit geometry learned by the neural renderer module. To mitigate this issue, we additionally define a shell $\mathcal{G}_s$, which closely bounds the surface of $\mathcal{G}_n$ by a small gap. When assigning MVC value to grid points, we first compare its distance to both $\mathcal{G}_s$ and $\mathcal{G}_d$ and determine the MVC value according to:

$$\Phi_{grid} = \begin{cases} \Phi_g, & \|\mathbf{p}_{grid} - \mathbf{p}_n^g\|_2 \leq \|\mathbf{p}_{grid} - \mathbf{p}_n^s\|_2 \\ \Phi_s, & \|\mathbf{p}_{grid} - \mathbf{p}_n^g\|_2 > \|\mathbf{p}_{grid} - \mathbf{p}_n^s\|_2 \end{cases}, \tag{14}$$

where Φ_{grid} is the MVC of the sampled grid point, Φ_g is the MVC of the closest point $\mathbf{p}_n^g$ on $\mathcal{G}_p$, Φ_s is the MVC of the closest point $\mathbf{p}_n^s$ on $\mathcal{G}_s$.

After acquiring this pre-sampled MVC field, we use the volumetric rendering technique to synthesize images of the deformed radiance field using Equation 1, with a cage deformation function $\mathcal{C}(\cdot)$ in Equation 10 applied:

$$\mathbf{F}_\theta : (\gamma(\mathcal{C}(\mathbf{p})), \gamma(\mathbf{d})) \rightarrow (\mathbf{c}, \sigma). \tag{15}$$

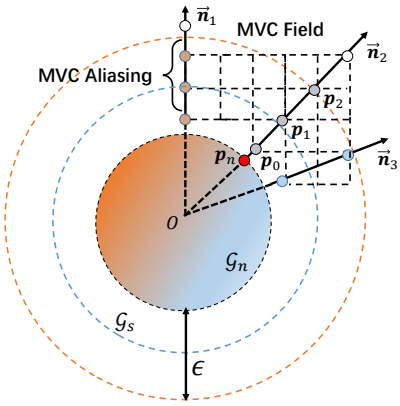

Figure 4: **Illustration of anti-aliasing MVC sample strategy in 2D case.** The grid sampling method projects grid points onto the geometry target w.r.t. the surface normal $\vec{n}_i$. Grid points outside the sample range ϵ will be filtered out. Points $\mathbf{p}_{0,1,2}$ inside the sample range have the same nearest point $\mathbf{p}_n$ and share the same MVC value. This causes aliasing when rendering the deformed radiance field. We devise a shell $\mathcal{G}_s$ (dotted blue circle) which closely bounds the render geometry by a small margin. We assign the MVC value by comparing the distance from the grid point to both the render geometry $\mathcal{G}_n$ and the shell $\mathcal{G}_s$.

4 Experiments

To verify the generalization of cage deformation, we evaluate CageNeRF on three tasks:

Geometry editing via deformation transfer (Section 4.1). Deformation transfer is a task that takes two objects with different geometry properties (e.g., shape, structure and scale), and modifies these properties on the source to make it resemble the target. In this task, CageNeRF is trained on a synthetic benchmark, in which a chair is rendered around 360° using 3D software.

Neural animation (Section 4.2). In this task, we train and evaluate CageNeRF on both synthesized objects and human videos. The synthesized 3D objects contain two characters: 'Hulk' and a blue

Figure 5: **Results of geometry editing via deformation transfer.** Top row shows the source object with texture and deformation target geometries. Row 2-3: deformed model under different views.

Table 1: Animation Performance on Synthetic Datasets

Dataset	Hulk			Whale			
Frame Index	1	27	47	1	35	80	100
PSNR	36.434	35.327	34.078	36.146	35.723	35.755	35.085
SSIM	0.963	0.957	0.948	0.972	0.972	0.971	0.966
LPIPS	0.0126	0.0156	0.0188	0.0117	0.0121	0.0119	0.0126

whale. Furthermore, we evaluate CageNeRF on real humans using Human 3.6M [17]. The animations in the benchmarks are driven by Linear Blend Skinning [24].

Animation reenactment (Section 4.3). In this task, we select a benchmark synthesized by a humanoid robot. This dataset contains the rendered 360° multi-view images with paired camera parameters. We randomly sample posing SMPL models from the Surreal dataset [42] as the target.

We further provide the ablation study w.r.t. the MVC sampling strategy. Following the majority of previous literature [30, 33, 35, 43], we use PSNR/SSIM [46] and LPIPS [51] as evaluation metrics. We refer readers to the supplementary material for more details and results.

4.1 Geometry Editing via Deformation Transfer

In our framework, deformation transfer can be applied to edit the geometric properties of the implicit 3D objects. Our framework is first trained on an office chair in the canonical space, as a deformation source. We randomly select objects from the ShapeNet dataset [8] as the deformation targets, and then apply our cage deformer to edit the implicit geometry stored in the canonical radiance field. As shown in Figure 5, our framework is capable of editing the implicit geometry of object in the canonical space by deforming the radiance field learned by our neural renderer module. The deformed results resemble the selected target in aspects of size, shape and scale. These results demonstrate that our framework can edit the radiance field such that the deformed geometry resembles the target.

4.2 Neural Animation

Synthesized Object Animation. To quantitatively determine the animation precision, we analyze the rendering quality of an animated sequence and compare it to the traditional linear blend skinning (LBS) animation method which operates on the explicit deformable mesh. We train our neural renderer using multiview images of the object rendered at the starting frame. To animate the object, we extract the animated mesh from the animation sequence and use it as the deformed geometry $\mathcal{G}_n$. Our cage deformer then builds the MVC field using Equation 14. We animate our object by applying the extracted meshes $\mathcal{G}_n$ sequentially to our framework.

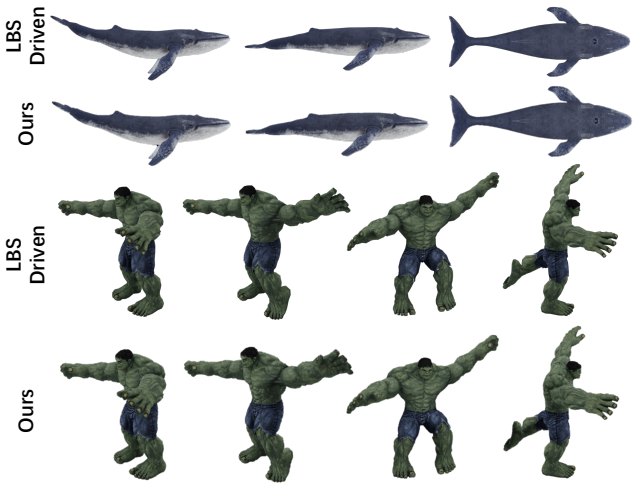

Figure 6: Qualitative results of neural animation on Whale and Hulk datasets. **Row 1, 3**: driven by linear blend skinning. **Row 2, 4**: driven by our framework.

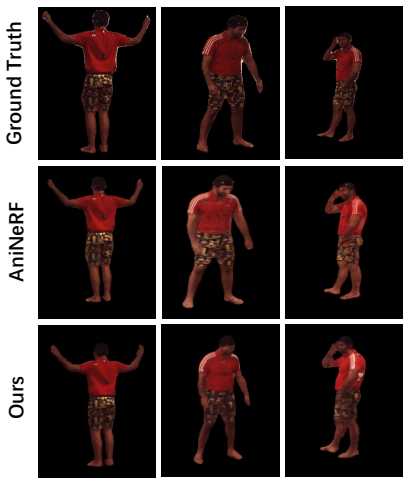

Figure 7: Qualitative comparison of CageNeRF and AniNeRF [33] for human synthesis on Human 3.6M.

As is shown in Figure 6, our framework is capable of deforming the radiance field and generating continuous animation of objects. CageNeRF also achieves nice render quality which is comparable to that driven by explicit animation methods. As shown in Table 1, our framework achieves promising results in animating the neural radiance field when compared to explicit mesh deforming methods.

Human Animation. We also evaluate on a real-world dataset (i.e., Human 3.6M), and compare with the state-of-the-art method AniNeRF [33]. The Human 3.6M dataset is captured by 4 synchronized cameras and has 2D and 3D joint annotations. We first estimate the geometry proxy of the posing human using the SMPL model by optimizing the 3D joint error and 2D re-projection joint error. Then, we utilize the geometry proxy as the "novel pose" and apply the operation in Figure 3 to generate a deformed cage with the matching pose. Then we build the MVC field using this deformed cage and posing geometry proxy. Since our model only stores the information inside the canonical space, CageNeRF can generalize to novel poses without additional training. This is in contrast to AniNeRF, which needs to optimize an additional blend weight network using the estimated geometry in order to synthesize novel pose results.

As shown in Figure 7 and Table 2, the performance of our framework in synthesizing dynamic human body is comparable to AniNeRF. CageNeRF achieves slightly lower PSNR due to the inaccurate estimation of the geometry used in our cage deformer module. Moreover, we employ SMPL model as the geometry proxy during cage deformation, where SMPL only models human body, while the performer in Human 3.6M dataset wears loosely fitted clothes which has complex nonrigid deformations during motion. The inaccuracy in the geometry proxy further induces sampling displacement during volumetric rendering and thus impacts the performance of our framework. Despite the fact that our framework only takes a coarse estimation of the object, it is still able to correctly deform and synthesize complex dynamic objects such as the human body.

Table 2: **Quantitative performance comparison on Human 3.6M between our framework and AniNeRF in human animation.**

Metrics	AniNeRF	Ours
PSNR	24.72	23.58
SSIM	0.908	0.886

Table 3: **Quantitative results of animated hulk using different MVC sampling strategies.** AA Sampling: the anti-aliasing sampling strategy.

Metrics	Grid Sampling	AA Sampling
PSNR	30.695	35.280
SSIM	0.916	0.956
LPIPS	0.0293	0.0156

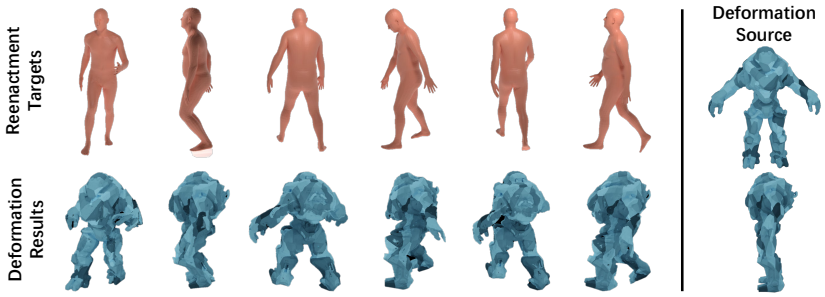

Figure 8: **Qualitative results of animation reenactment using SMPL models as the reenactment target. Top Row** shows SMPL models sampled from Surreal dataset as the reenactment target. **Bottom Row** reenactment results defromed from the souce model shows on the right.

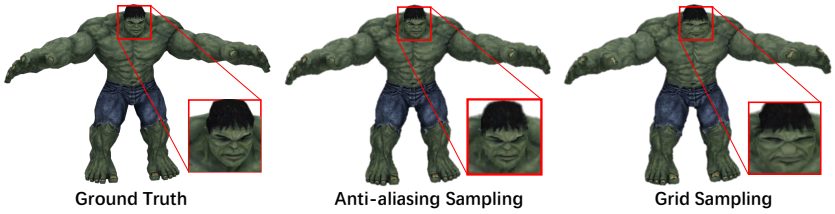

Figure 9: **Comparison between Anti-aliasing Sampling and Grid Sampling.** Our anti-aliasing sampling strategy achieves notably better rendering results w.r.t. geometry and texture.

4.3 Pose Reenactment

To analyze the generalization ability of CageNeRF, we apply our framework to tasks of pose reenactment and motion retargeting from humans to humanoid objects. We keep the same training process as Section 4.1. Afterward, we randomly sample posed SMPL models from the Surreal dataset as the target for deformation transfer. Both the posed SMPL model and the optimized robot cage are fed into our cage deformer to calculate the cage deformation. Finally, the barycentric interpolation is applied to deform the static robot model using Equation 11. As is shown in Figure 8, although the structure of the source robot is notably different from the target human, the deformed pose and geometric structure correspond well to the given target, while keeping the source identity and texture.

4.4 Imapct of the MVC Sampling Strategy

To evaluate the effectiveness of our anti-aliasing MVC sampling strategy, we generate an MVC field using different sampling strategies and qualitatively and quantitatively compare the rendering results. As shown in Figure 9, the rendering result generated with grid-sampled MVC field displays inflated geometry and blurred texture, due to the inaccurate mapping between the canonical and the deformed space. In contrast, the result rendered using our anti-aliasing sampled MVC field has notably sharper textures and a more accurate geometry structure. As also shown in Table 3, the anti-aliasing sampling strategy significantly increases the render quality on all three evaluation metrics.

5 Conclusion

We propose an versatile framework for deforming and animating implicit field. Comparing to previous methods, our framework excels in aspects of generalization, structural flexibility and editing capability. This is achieved by adopting a deformable cage as the universal constraint for our radiance field editing which does not require any data-dependent prior knowledge. We demonstrate the effectiveness of our framework in the tasks of object animation, geometry editing and deformation transfer. Experimental results show that our framework is capable of animating and deforming arbitrary objects represented by implicit field and produces results comparable to other state-of-the-art methods.

References

[1] ShahRukh Athar, Zhixin Shu, and Dimitris Samaras. Flame-in-nerf : Neural control of radiance fields for free view face animation. *CoRR*, abs/2108.04913, 2021.

[2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, pages 5835–5844, 2021.

[3] Matthew Berger, Andrea Tagliasacchi, Lee M. Seversky, Pierre Alliez, Gaël Guennebaud, Joshua A. Levine, Andrei Sharf, and Cláudio T. Silva. A survey of surface reconstruction from point clouds. *Comput. Graph. Forum*, 36(1):301–329, 2017.

[4] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH*, SIGGRAPH '99, page 187–194, 1999.

[5] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH*, pages 187–194, 1999.

[6] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. FAUST: dataset and evaluation for 3d mesh registration. In *CVPR*, pages 3794–3801, 2014.

[7] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. 2010.

[8] 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.

[9] Xu Chen, Yufeng Zheng, Michael J. Black, Otmar Hilliges, and Andreas Geiger. SNARF: differentiable forward skinning for animating non-rigid neural implicit shapes. In *ICCV*, pages 11574–11584, 2021.

[10] Lin Gao, Yu-Kun Lai, Dun Liang, Shu-Yu Chen, and Shihong Xia. Efficient and flexible deformation representation for data-driven surface modeling. *ACM TOG*, 35(5):158:1–158:17, 2016.

[11] Lin Gao, Yu-Kun Lai, Jie Yang, Ling-Xiao Zhang, Shihong Xia, and Leif Kobbelt. Sparse data driven mesh deformation. *IEEE TVCG*, 27(3):2085–2100, 2021.

[12] 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.

[13] Yudong Guo, Keyu Chen, Sen Liang, Yong-Jin Liu, Hujun Bao, and Juyong Zhang. Ad-nerf: Audio driven neural radiance fields for talking head synthesis. In *ICCV*, pages 5784–5794, 2021.

[14] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE TPAMI*, 43(12):4338–4364, 2021.

[15] Yang Hong, Bo Peng, Haiyao Xiao, Ligang Liu, and Juyong Zhang. Headnerf: A real-time nerf-based parametric head model. *CoRR*, abs/2112.05637, 2021.

[16] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[17] 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.

[18] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popovic, and Olga Sorkine. Fast automatic skinning transformations. *ACM TOG*, 31(4):77:1–77:10, 2012.

[19] Wonbong Jang and Lourdes Agapito. Codenerf: Disentangled neural radiance fields for object categories. In *ICCV*, pages 12929–12938, 2021.

[20] Timothy Jeruzalski, David I. W. Levin, Alec Jacobson, Paul Lalonde, Mohammad Norouzi, and Andrea Tagliasacchi. Nilbs: Neural inverse linear blend skinning. *CoRR*, abs/2004.05980, 2020.

[21] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM TOG*, 26(3):71, 2007.

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

[23] Ladislav Kavan, Steven Collins, Jirí Zára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM TOG*, 27(4):105:1–105:23, 2008.

[24] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM TOG*, 27(4):1–23, 2008.

[25] Tianye Li, Timo Bolkart, Michael J. Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression from 4d scans. *ACM TOG*, 36(6):194:1–194:17, 2017.

[26] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM TOG*, 27(3):78, 2008.

[27] 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.

[28] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, pages 163–169, 1987.

[29] Bruce Merry, Patrick C. Marais, and James E. Gain. Animation space: A truly linear framework for character animation. *ACM TOG*, 25(4):1400–1423, 2006.

[30] 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.

[31] Ramakrishnan Mukundan. *3D Mesh Processing and Character Animation - With Examples Using OpenGL, OpenMesh and Assimp*. 2022.

[32] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B. Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *ICCV*, pages 5845–5854, 2021.

[33] Sida Peng, Junting Dong, Qianqian Wang, Shangzhan Zhang, Qing Shuai, Xiaowei Zhou, and Hujun Bao. Animatable neural radiance fields for modeling dynamic human bodies. In *ICCV*, pages 14294–14303, 2021.

[34] Sida Peng, Shangzhan Zhang, Zhen Xu, Chen Geng, Boyi Jiang, Hujun Bao, and Xiaowei Zhou. Animatable neural implicit surfaces for creating avatars from videos. *CoRR*, abs/2203.08133, 2022.

[35] Sida Peng, Yuanqing Zhang, Yinghao Xu, Qianqian Wang, Qing Shuai, Hujun Bao, and Xiaowei Zhou. Neural body: Implicit neural representations with structured latent codes for novel view synthesis of dynamic humans. In *CVPR*, pages 9054–9063, 2021.

[36] Shunsuke Saito, Jinlong Yang, Qianli Ma, and Michael J Black. Scanimate: Weakly supervised learning of skinned clothed avatar networks. In *CVPR*, pages 2886–2897, 2021.

[37] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.

[38] Daeyun Shin, Charless C Fowlkes, and Derek Hoiem. Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction. In *CVPR*, pages 3061–3069, 2018.

[39] Jean-Marc Thiery, Julien Tierny, and Tamy Boubekeur. Jacobians and hessians of mean value coordinates for closed triangular meshes. *CoRR*, abs/1109.1914, 2011.

[40] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *ICCV*, pages 12959–12970, 2021.

[41] Wei-Cheng Tseng, Hung-Ju Liao, Lin Yen-Chen, and Min Sun. Cla-nerf: Category-level articulated neural radiance field. *CoRR*, abs/2202.00181, 2022.

[42] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. In *CVPR*, pages 4627–4635, 2017.

[43] 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 *NeurIPS*, pages 27171–27183, 2021.

[44] Xiaohuan Corina Wang and Cary B. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SIGGRAPH*, pages 129–138, 2002.

[45] 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.

[46] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE TIP*, 13(4):600–612, 2004.

[47] Kai Xu, Vladimir G. Kim, Qixing Huang, Niloy J. Mitra, and Evangelos Kalogerakis. Data-driven shape analysis and processing. In *SIGGRAPH Asia*, pages 4:1–4:38, 2016.

[48] Haotian Yang, Hao Zhu, Yanru Wang, Mingkai Huang, Qiu Shen, Ruigang Yang, and Xun Cao. Facescape: A large-scale high quality 3d face dataset and detailed riggable 3d face prediction. In *CVPR*, pages 598–607, 2020.

[49] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *NeurIPS*, 34, 2021.

[50] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *CoRR*, abs/2010.07492, 2020.

[51] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595, 2018.

[52] Yuzhe Zhang, Jianmin Zheng, and Yiyu Cai. Proxy-driven free-form deformation by topology-adjustable control lattice. *Comput. Graph.*, 89:167–177, 2020.

[53] Yufeng Zheng, Victoria Fernández Abrevaya, Xu Chen, Marcel C Bühler, Michael J Black, and Otmar Hilliges. IM avatar: Implicit morphable head avatars from videos. *CoRR*, abs/2112.07471, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] , our work focuses on providing editing method for implicit 3D representations in a generalized manner, it is possible that someone utilizes our method to produce fake videos.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We do not include theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We describe the implementation details in the supplementary material. The data utilized is publicly available, and we will make our code publicly available.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We discuss the training details in supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary for detailed descriptions.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use existing datasets and cited them in our paper.
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] We discuss this in the supplementary material.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] The data used in our work does not contain such information.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No] Our work does not involve crowdsourcing as such, same as follows.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No]