EditableNeRF: Editing Topologically Varying Neural Radiance Fields by Key Points

Chengwei Zheng, Wenbin Lin, and Feng Xu

Abstract—Neural radiance fields (NeRF) achieve highly photo-realistic novel-view synthesis, but it's a challenging problem to edit the scenes modeled by NeRF-based methods, especially for dynamic scenes. We propose editable neural radiance fields that enable end-users to easily edit dynamic scenes and support topological changes. Input with an image sequence from a single camera, our network is trained automatically and models topologically varying dynamics using our picked-out surface key points. Then end-users can edit the scene by easily dragging the key points to desired new positions. To achieve this, we propose a scene analysis method to detect and initialize key points by considering the dynamics in the scene, and a weighted key points strategy to model topologically varying dynamics by joint key points and weights optimization. Our method supports intuitive multi-dimensional (up to 3D) editing and can generate novel scenes that are unseen in the input sequence. Experiments demonstrate that our method achieves high-quality editing on various dynamic scenes and outperforms the state-of-the-art. Our code and captured data are available at https://chengwei-zheng.github.io/EditableNeRF/.

Index Terms—neural radiance fields, novel-view synthesis, scene editing, topological changes.

1 Introduction

Neural radiance fields (NeRF) [1] have shown great power in novel-view synthesis and enable many applications as this method achieves photo-realistic rendering [2]. Recent techniques have further improved NeRF by extending it to handle dynamic scenes [3], [4], [5] and even topologically varying scenes [6]. However, while these works mainly focus on reconstruction itself, they do not consider scene editing. As a result, although camera views can be changed for rendering, scene editing according to user preferences remains challenging.

Recently, some frameworks have been proposed to make neural radiance fields editable in different aspects. Some of them aim to edit the reconstructed appearance and enable relighting [7], [8], [9]; some allow controlling the shapes and colors of objects from a specific category [10], [11], [12], [13]; and some divide the scene into different parts and the location of each part can be modified [14], [15], [16]. However, the dynamics of moving objects cannot be edited by the previous methods, and this task becomes particularly challenging when the dynamics involve topological changes. Such topological changes can lead to motion discontinuities (e.g., between the hammer and the piano keys, between the cups and the table in Fig. 1) and further cause noticeable artifacts if not modeled well. A state-of-the-art framework CoNeRF [17] tries to resolve this problem by using manual supervision. Nonetheless, it only supports limited and one-dimensional editing for each part, requiring user annotations as supervision.

We propose EditableNeRF, editable topologically varying neural radiance fields that are trained without manual supervision and support intuitive multi-dimensional (from 1D to 3D) editing. The key of our method is to represent motions and topological changes by the movements of some sparse surface key points. Each key point is able to control the topologically varying dynamics of a moving part, as well as other effects like shadow and reflection changes through the neural radiance fields. This key-point-based method enables end-users to edit the scene by easily dragging the key points to desired new positions.

To achieve this, we first apply a scene analysis method to detect key points in the canonical space and track them in the full sequence for key point initialization. We introduce a network to estimate spatially-varying weights for all scene points and use the weighted key points to model the dynamics in the scene, including topological changes. In the training stage, our network is trained to reconstruct the scene using the supervision from the input image sequence, and the key point positions are also optimized by incorporating motion (optical flow) and geometry (depth maps) constraints as additional supervision. After training, the scene can be edited by manipulating the positions of these key points, and novel scenes that are unseen during training can also be generated.

The contribution of this paper lies in the following aspects:

- Key-point-driven neural radiance fields achieving intuitive multi-dimensional editing even with topological changes, without requiring annotated training data.
- A weighted key points strategy modeling topologically varying dynamics by joint key points and weights optimization.
- A scene analysis method to detect and initialize key points by considering the dynamics in the scene.

- *Chengwei Zheng is with the Department of Computer Science, ETH Zürich, Switzerland, and also with the School of Software and BNRist, Tsinghua University, Beijing, China (e-mail: zhengcw18@gmail.com).*
- *Wenbin Lin and Feng Xu are with the School of Software and BNRist, Tsinghua University, Beijing, China (e-mail: lwb20@mails.tsinghua.edu.cn; xufeng2003@gmail.com).*

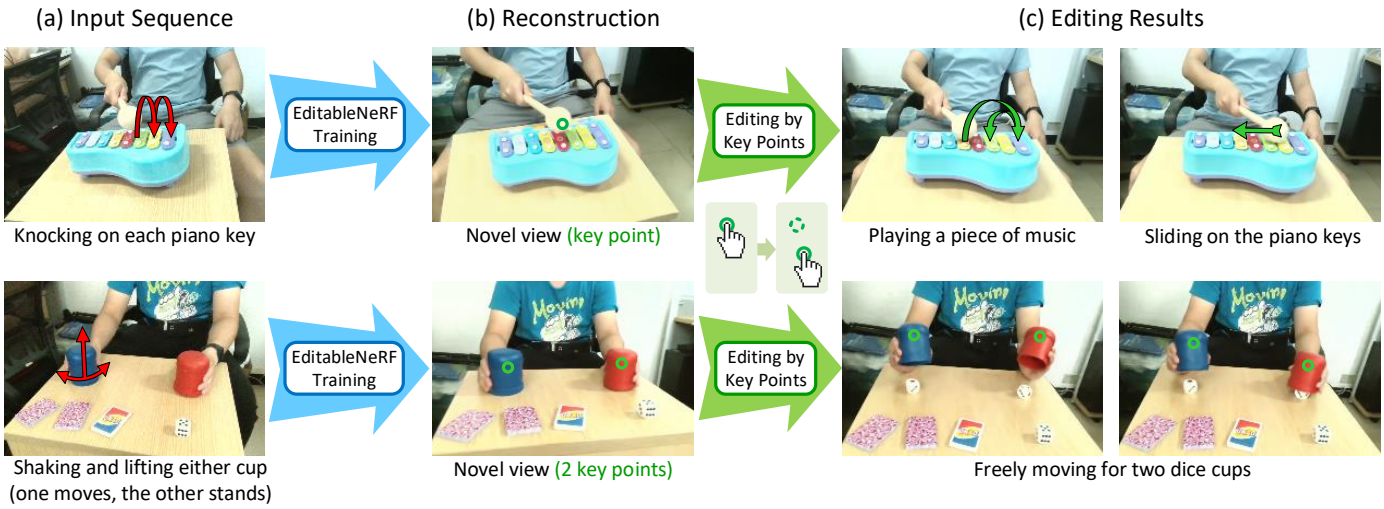

Fig. 1. Taking an image sequence (a) as input, EditableNeRF is trained fully automatically to reconstruct the captured scene (b) and can handle topological changes. After training, end-users are able to edit the scene (c) by controlling the automatically picked-out key points (circled in green). Our method enables up to three-dimensional editing and can generate novel scenes that are unseen during training.

An earlier version of this manuscript appeared in [18], and the current version makes several novel contributions and significant enhancements. Firstly, we improve the weight estimating network in our weighted key points strategy to model *dynamic* key point weights instead of the *static* key point weights in the earlier version. The dynamic key point weights allow our method to represent more complex scenes, wherein different objects may move to the same location across different frames. Secondly, we introduce a virtual key point to model the background. This addition can eliminate some artifacts in the background, particularly when utilizing dynamic key point weights. Thirdly, we present a new loss function in our training stage, leading to a more stable optimization and enabling our method to tackle more challenging scenes. Lastly, comparisons with a point-based image editing method, comparisons with a baseline method, new applications, additional results, and expanded datasets are provided to evaluate the effectiveness of our proposed techniques and the overall system.

2 Related Work

2.1 Novel-View Synthesis

Many methods achieve rendering novel-view images by reconstructing scenes and objects into meshes [19], [20], [21], [22], neural voxels [23], [24], and multi-plane images [25], [26]. Besides these methods based on discrete representations, some methods also achieve novel-view synthesis by using continuous representations [27], [28] and have shown great potential in this task.

Neural radiance fields (NeRF) [1] achieve photo-realistic rendering in novel-view synthesis by leveraging continuous implicit functions of density and view-dependent color to represent static scenes. To facilitate dynamic reconstructions, time-variant codes could be used to encode dynamic components within NeRF, but requiring multi-view video inputs [29], [30]. Additionally, deformation fields implemented by multilayer perceptron (MLP) are applied to warp objects in each frame into a canonical space [3], [4], [5], enabling monocular dynamic reconstructions. Some methods also leverage estimated depth maps [31], ToF depth images [32], or optical scene flow [33] to improve the performance of dynamic neural radiance fields. HyperNeRF [6] advances dynamic NeRF to reconstruct topologically varying scenes by extending 3D canonical spaces to a hyperspace. However, unlike traditional explicit representations such as triangular meshes, NeRF-based methods represent the scenes by implicit functions, which poses significant challenges for scene editing.

2.2 NeRF Editing

As our method focuses on NeRF editing, we mainly discuss NeRF-based methods that support user editing in this section. For editing on explicit 3D representations or other implicit 3D representations, please refer to [34], [35], [36].

One approach for editing neural radiance fields is to segment the scene into different components and construct MLP for each component [14], [15], [16]. Assuming that different components are individual, this representation allows control of the placements and the relative positions of these components, as well as deleting or reduplicating a component. However, these methods do not support editing the dynamics inside a component and their application remains constrained.

In addition, some methods achieve relighting and material editing on neural fields [7], [8], [9] by decomposing the scene into surface normals, lights, albedo, and material. Texture editing can also be accomplished by a 3D-to-2D texture mapping [37].

Some methods focus on modeling a specific category of objects [10], [12], [13] instead of general objects. A commonly employed solution involves employing conditional NeRF to model the category of objects while using latent codes as conditions to encode the variations among different objects within that category. Consequently, the shape and appearance can be edited by modifying the latent codes or through network fine-tuning [10], and even controlled via text [38] with the assistance of a multi-modal model. Many methods also focus on modeling editable human bodies

and faces based on NeRF representation. By incorporating human body parametric models and skinning techniques like SMPL [39], neural radiance fields have been extended to model the human body and can be animated through the manipulation of skeleton poses [40], [41], [42]. Similarly, human face parametric models contribute to extending NeRF for human face modeling and control [43], [44], [45], and even driven by audio [46]. However, general objects cannot be handled by these methods.

Recently, NeRF-editing [47] proposes to deform NeRF on static objects by extracting explicit meshes, deforming the meshes, and transferring the deformations back into the implicit representations. However, this method cannot handle dynamic scenes. CoNeRF [17] proposes an attribute re-rendering method based on dynamic NeRF. This method requires user annotations in several frames for network training, including masks for every dynamic part and their corresponding attribute values. Then these parts could be edited through control over the 1D attribute values. We will show our advantages over CoNeRF in Sec. 4.2.

2.3 Point-based Image Editing

Point-based editing methods have emerged as user-friendly and intuitive tools for image manipulation, allowing users to directly interact with images by dragging points. To enable point-based editing on StyleGAN, Endo [48] uses a latent transformer to compute the latent codes after dragging to achieve drag-style editing. DragGAN [49] allows users to drag the content of GAN-generated images, accomplishing fine-grained editing through an iterative optimization process involving latent code optimization and point tracking. FreeDrag [50] introduces adaptive template features and line search with backtracking to enhance the reliability of point dragging. Furthermore, leveraging the capabilities of the diffusion model [51], DragonDiffusion [52] enables drag-style manipulation on diffusion-generated images and extends its applicability to tasks such as appearance replacing and object pasting. AnyDoor [53] achieves diffusion-based image editing by placing objects at user-specified locations. DragDiffusion [54] further proposes a point-based editing framework applicable to both real and diffusion-generated images. Seamlessly looping videos can also be generated from a single image through the dragging and releasing of points [55]. However, these image editing methods do not possess the capacity for 3D modeling, thereby limiting their ability to generate novel-view images.

2.4 Preliminaries: NeRF and HyperNeRF

Before introducing our approach, we first provide a brief overview of NeRF [1]. NeRF is a technique that represents a particular 3D scene by encoding it within an MLP. This MLP is trained under supervision from multi-view color images. The input of the MLP H is a position x and a view direction d, and the output is the density σ at x and the appearance color c from the view direction d.

$$(c, \sigma) = H(\mathbf{x}, \mathbf{d}). \tag{1}$$

To render an image, NeRF method traces the camera rays of all pixels, samples points along these rays, obtains points' colors and densities from the MLP, and runs volumetric rendering. To be specific, when rendering a pixel, a sample point on the traced ray $\mathbf{r}$ is defined as $\mathbf{r}_j = \mathbf{o} + j\mathbf{d}$, where $\mathbf{o}$ is the ray origin and $\mathbf{d}$ is the ray direction. And the pixel color C is computed by integrating all the sample point colors c, weighted by their densities σ and the accumulated transmittance V:

$$C(\mathbf{r}) = \int_{j_n}^{j_f} V(j)\sigma(\mathbf{r}_j)c(\mathbf{r}_j, \mathbf{d})dj, \tag{2}$$

$$V(j) = \exp\left(-\int_{jn}^{j} \sigma(\mathbf{r}_s)ds\right). \tag{3}$$

where j_n and j_f are the near and far planes of the rendering volume.

HyperNeRF [6] further extends NeRF to reconstruct topologically varying dynamic scenes. For motion modeling, HyperNeRF first employs a warp field T to deform query point $\mathbf{x}$ into its canonical position $\mathbf{x}'$.

$$\mathbf{x}' = T(\mathbf{x}, \beta_t), \tag{4}$$

where β_t is the warp latent code in frame t and is optimized during training. However, this warp field can only model spatial-smoothing deformation but not topologically varying motions. If an object in two frames is in different topology states, this warp field cannot deform them into a shared 3D canonical space. To figure this out, HyperNeRF introduces ambient dimensions, denoted as $\mathbf{a}$, in addition to 3D canonical space. This combination forms hyperspace, a higher-dimensional canonical space. Then, various 3D canonical spaces with different topology states can be modeled into a unified continuous hyperspace. The discontinuous deformations in 3D space caused by topological changes can be modeled by continuous functions (such as MLP) within the hyperspace.

$$\mathbf{a} = A(\mathbf{x}, \beta_t), \tag{5}$$

$$(c, \sigma) = H(\mathbf{x}' \oplus \mathbf{a}, \mathbf{d}, \alpha_t). \tag{6}$$

Here the mapping function A is implemented as an MLP, and α_t is an appearance latent code. The symbol $\oplus$ represents the concatenation operation.

3 EDITABLENERF

Input with color image sequence of a dynamic scene, our method can reconstruct the scene automatically based on neural radiance fields. Several surface key points are selected to encode topologically varying dynamics. Key point positions in every frame are optimized during training and different key point positions encode different motion states. After reconstruction, end-users can edit the scene by manipulating key point positions.

Our pipeline is illustrated in Fig. 2. First, we utilize two methods, HyperNeRF [6] and RAFT [56], to compute the depth maps of input frames and optical flow between consecutive input images, respectively. Then we apply a scene analysis method to detect and initialize key point positions (Sec. 3.3). After that, our NeRF-based network (Sec. 3.1) is trained automatically (Sec. 3.2) to model the captured scene based on our weighted key points strategy. Finally, the reconstructed scene can be edited by dragging the key points to their desired positions.

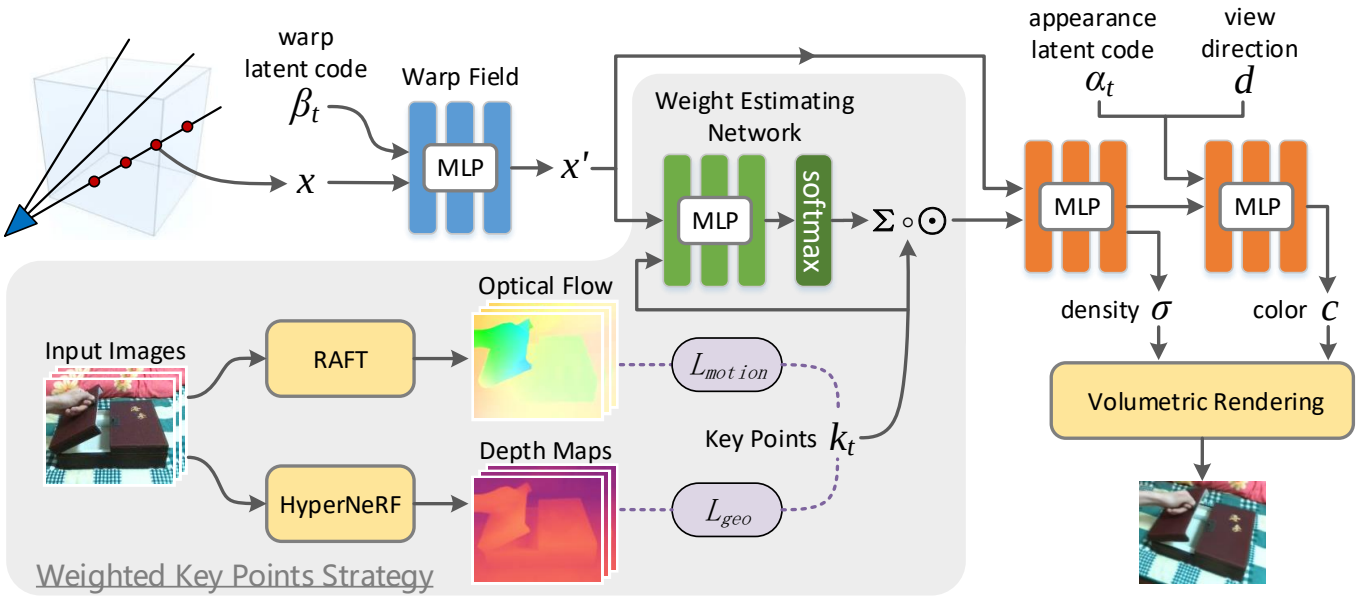

Fig. 2. EditableNeRF pipeline. The query point $\mathbf{x}$ is first warped into the canonical space by a warp field with a latent code β_t in frame t. Next, we compute the *key point weights* for this canonical point $\mathbf{x}'$ and use it to calculate a linear combination of all key point positions $\mathbf{k}_t$, called *weighted key points*. After that, we feed $\mathbf{k}_t$ concatenated with $\mathbf{x}'$ to the NeRF MLP, and the output density and color can be used for volumetric rendering. In the training stage, optical flow and depth maps are utilized to supervise key point positions. Editing can be achieved by manipulating key points.

3.1 Network

We first introduce our network architecture, which is shown in Fig. 2. Our network represents the scene as a field of density and radiance [1]. Given a query point, similarly to HyperNeRF [6] as detailed in Sec. 2.4, we first use a warp field to model slight movements:

$$\mathbf{x}' = T(\mathbf{x}, \beta_t). \tag{7}$$

Here the warp field T maps a query 3D point $\mathbf{x}$ to its canonical location $\mathbf{x}'$, and β_t is the warp latent code in frame t. This warp field models slight movements and ensures alignment of the scene across different frames despite some errors in the input camera parameters. However, as discussed in [6], it's hard for this continuous warp field to model discontinuous movements caused by topological changes.

Then it becomes necessary to represent various topology and motion states in the canonical space. Recognizing that motions and topological changes are always related to surface point movements, we address this challenge by making use of sparse surface key points. These 3D key points are attached to the objects' surfaces and also move with the objects. Each key point controls the topologically varying dynamics of a moving part and also some effects caused by this part like shadow and reflection changes. An example of key points is presented in (a) of Fig. 3. For each moving part in the scene, we automatically select one corresponding key point, which will be introduced in Sec. 3.3, and the number of key points is denoted as N. The key points' positions in each input frame will be optimized automatically in our training stage to achieve this modeling. Besides, we also use a virtual key point to model the background, which will be detailed later.

We assume that different locations in the canonical space are affected by different key points. So for a query point $\mathbf{x}'$, an MLP followed by softmax is used to decide which key point should control its dynamics. This network, termed *weight estimating network*, takes the canonical coordinate $\mathbf{x}'$ and a concatenated vector containing all key point positions $\mathbf{k}_t \in \mathbb{R}^{3N}$ as input, and outputs a weight vector $w \in \mathbb{R}^N$, indicating the impact of each key point on the query point $\mathbf{x}'$. And an example is shown in Fig. 3.

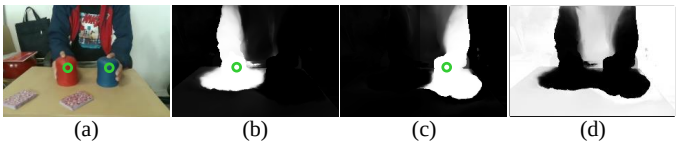

Fig. 3. Examples of key points and key point weights. (a) shows a color image and its key points (circled in green). (b) and (c) demonstrate the weights of the two key points, respectively. (d) shows the weights of the virtual key point for background modeling. These weights are obtained by employing the surface points corresponding to pixels as query points.

$$w = W(\mathbf{x}', \mathbf{k}_t). \tag{8}$$

In the earlier version [18], only the query point $\mathbf{x}'$ is inputted into the weight estimating network to model the *static* key point weights. In contrast, the current version incorporates both the query point $\mathbf{x}'$ and the key point positions $\mathbf{k}_t$ as inputs to model *dynamic* key point weights, allowing the key point weights to dynamically adjust according to key point positions. This is because, in some challenging scenes, different objects that are controlled by distinct key points may move to the same region in different frames. In such cases, a point within this region should be controlled by different key points under different motion states. Thus, attempting to model these scenes using the *static* key point weights will lead to artifacts. Different from the earlier version [18], additionally inputting key point positions $\mathbf{k}_t$ makes it possible for the network to determine which key point should affect the position $\mathbf{x}'$ in frame t.

Furthermore, we use a virtual key point for background modeling, to eliminate the artifacts caused by the introduc-

tion of dynamic key point weights. This virtual key point is actually a 3D latent code and doesn't correspond to any physical point within the scene. Its values are optimized during training. Without this virtual key point, the background may be controlled by different key points across the sequence when utilizing dynamic key point weights. This can lead to artifacts when the key point that controls the background suddenly changes from one to another. The inclusion of the virtual key point ensures that the background is consistently modeled by this virtual key point, avoiding sudden changes in key point weights. The weights of this virtual key point are shown in (d) of Fig. 3.

Subsequently, we formulate a *weighted key points* vector $\mathbf{p}$ by computing a linear combination of all key point positions $\mathbf{k}$ to model the topologically varying dynamics at $\mathbf{x}'$.

$$p_t(\mathbf{x}') = \sum_{i=1}^{N} w^i(\mathbf{x}') \cdot \mathbf{k}_t^i. \tag{9}$$

The superscript i is the index of key points, and the subscript t is the frame index. The virtual background key point is included in these N key points.

Next, the 3D canonical coordinate $\mathbf{x}'$ and the weighted key points $\mathbf{p}$ are concatenated to form a 6D coordinate in hyperspace for topologically varying scene modeling. This hyperspace is proposed in HyperNeRF [6] and is introduced in Sec. 2.4. In addition to 3D space, HyperNeRF incorporates ambient dimensions to model canonical objects in hyperspace and encodes different topology and motion states with different ambient coordinates. In our approach, we utilize the weighted key points $\mathbf{p}$ as ambient coordinates to effectively model topologically varying dynamics.

Finally, the following NeRF MLP is fed with the 6D coordinate in hyperspace:

$$(c, \sigma) = H(\mathbf{x}' \oplus \mathbf{p}_t(\mathbf{x}'), \mathbf{d}, \alpha_t). \tag{10}$$

Here $\mathbf{d}$ is the view direction, and α_t is the appearance latent code. This NeRF MLP outputs the color c and the density σ that can be used for NeRF volumetric rendering, as introduced in Sec. 2.4.

After training, users can easily edit the modeled scene by feeding the network with desired key point positions. As the key points are in the 3D space, our method supports up to three-dimensional editing for each part. We also provide a graphical user interface (GUI) in Sec. 4.4.

3.2 Loss Functions and Training

In the training stage, all the latent codes and MLP parameters are optimized to effectively model the scene. As we employ key points to encode topologically varying dynamics, we need to additionally optimize key point positions in each input frame. To keep our key points on the object surfaces and maintain temporal consistency, novel loss functions are incorporated into our training stage.

First, we propose a motion loss, which constrains that the key point positions in two consecutive frames should be consistent with the optical flow from pre-trained RAFT [56].

$$L_{motion}(t,i) = \left\| \Pi_{t+1}(\mathbf{k}_{t+1}^i) - \Pi_t(\mathbf{k}_t^i) - F_t^{t+1}(\Pi_t(\mathbf{k}_t^i)) \right\|^2, \tag{11}$$

where Π_t is the projection function employing the camera pose of frame t, and F_t^{t+1} is the optical flow from frame t to frame $t+1$. This loss ensures that the 2D key point positions in different frames correspond to the same surface point.

The motion loss provides sufficient supervision within the 2D image space, whereas the key points are in the 3D space. Therefore, the inclusion of a geometry loss becomes crucial in preserving the key points on the surfaces.

$$L_{geo}(t,i) = \left\| \Phi_t(\mathbf{k}_t^i) - D_t(\Pi_t(\mathbf{k}_t^i)) \right\|^2. \tag{12}$$

Here the function $\Phi_t(\mathbf{k}_t^i)$ calculates the distance from the key point $\mathbf{k}_t^i$ to the camera position in frame t, and D_t denotes the depth map rendered from HyperNeRF [6] in the original camera view. This HyperNeRF is pre-trained before our training stage and takes the same input as ours. Note that neither the motion loss nor the geometry loss is applied to the virtual background key point.

Different from the previous version [18], we propose a novel weight regularization loss to improve the robustness. This loss function constraints that each key point should fully control the position where it is located.

$$L_{weight}(t,i) = \left\| W(\mathbf{k}_t^i, \mathbf{k}_t) - \varepsilon(i) \right\|^2. \tag{13}$$

W and $\mathbf{k}_t$ are the weight estimating network and key point vector in (8). $\varepsilon(i)$ is a one-hot vector, with its ith element set to 1 while others set to 0. Also, this loss will not be applied to the virtual key point.

Besides, we use a reconstruction loss between the rendered RGB images C and the input images $\widetilde{C}$, as well as a warp regularization loss similar to HyperNeRF [6].

$$L_{rec}(t) = \left\| C_t(\mathbf{k}_t, \alpha_t, \beta_t) - \widetilde{C}_t \right\|^2, \tag{14}$$

$$L_{reg}(t) = \frac{1}{\|S_t\|} \sum_{\mathbf{x} \in S_t} \left\| \mathbf{x} - T(\mathbf{x}, \beta_t) \right\|^2, \tag{15}$$

where S_t is the set of surface points in frame t. This warp regularization loss makes sure that the warp field only models slight movements to compensate for the errors in the input camera parameters and distortions in the input images. Consequently, this loss helps avoid the ambiguity between the warp field and our weighted key points model.

3.3 Key Point Detection and Initialization

To initialize the network training, it is necessary to determine the key point number N and acquire the initial 3D locations of key points. To achieve this, we propose a scene analysis method, which first detects reference key points in canonical space and corresponding reference frames, then initializes key points' positions in each frame. The virtual background key point is initialized randomly.

Our key points are used to model topologically varying dynamics, so we need to identify 3D points with significant dynamics in the canonical space as our reference key points. As detailed in Sec. 2.4, HyperNeRF [6] utilizes different ambient coordinates to encode varying topology and motion states at a position. We have already trained a HyperNeRF for depth maps in (12). By leveraging this pre-trained HyperNeRF, we can detect key points based on the ambient dimensions $\mathbf{a}$. For a point $\mathbf{x}$, substantial variations of its $\mathbf{a}$

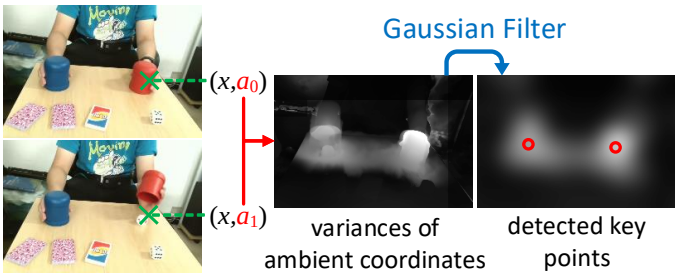

Fig. 4. A 2D visualization of our key point detection method. For each point $\mathbf{x}$, we compute the variance of its ambient coordinates $\mathbf{a}$ across the entire sequence. Then all the points with local maximum variances after a 2D Gaussian filter are selected as our reference key points.

across different frames indicate significant dynamics at $\mathbf{x}$, so we use all the positions with locally maximal variations in ambient coordinates as our reference key points.

To be specific, for each input frame, we trace the original camera rays of all pixels and find the corresponding surface points. Subsequently, a 3D voxel volume in the canonical space is constructed to record the ambient coordinates of these surface points. After traversing the entire input sequence, we calculate the variance of ambient coordinates for every voxel, followed by a 3D Gaussian filter. All the center points of the voxels with local maximum variances after the Gaussian blur are then selected as our reference key points $\mathbf{k}_{ref}$. Consequently, the number of key points N is equal to the number of local maximum points. As our 3D detection is difficult to visualize, we present a 2D version in Fig. 4 by rendering all frames in a fixed camera view, computing the variance for each pixel, and applying a 2D Gaussian filter.

Then for each reference key point, we need to select a reference frame wherein the reference key point lies on the object surfaces. To achieve this, we employ a formulation akin to the geometry loss described in (12). For simplicity, we omit the key point indices here, as each key point is handled independently in this process.

$$\|\Phi_t(\mathbf{k}_{ref}) - D_t(\Pi_t(\mathbf{k}_{ref}))\|^2 < \delta. \tag{16}$$

Here δ is a pre-defined threshold. The first frame t that satisfies (16) will be selected as the reference frame t_{ref}.

Now for each key point, we have both a reference key point position and a corresponding reference frame. To initialize key point positions in the whole sequence, we propagate this reference key point to other frames by optical flow from pre-trained RAFT [56]. The reference key point is first projected into the input image of the reference frame to get its 2D position, and the 2D position is propagated frame-by-frame using optical flow as illustrated in Fig. 5. Then these 2D positions are projected back into 3D space using the depth maps from HyperNeRF. Note that there are accumulative errors in this initialization due to frame-by-frame propagation, while these errors will be eliminated during the subsequent training stage.

For particularly long input sequences, the aforementioned initialization method may not yield satisfactory results. This is because the accumulative errors over time may become too large, and the key points in the images may be incorrectly propagated into other objects such as the background. To address this issue, we propose a skipping propagation method as shown in Fig. 5, which additionally

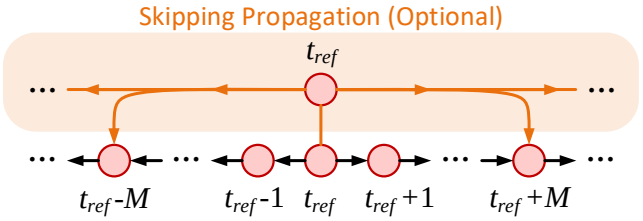

Fig. 5. Propagating the reference key point in the reference frame to other frames for initialization. Skipping propagation is only used for particularly long input sequences.

propagates the reference key point in the reference frame every M frames (i.e., t_{ref} to $t_{ref}+M$, t_{ref} to $t_{ref}+2M$, and so on). Then we use them to replace the frame-by-frame positions before further frame-by-frame propagation if the skipping confidences are greater than a threshold. This skipping confidence is calculated based on the consistency between the forward and backward optical flow:

$$Conf(t) = \left\| F_t^{t_{ref}}(F_{t_{ref}}^{t}(\widehat{\mathbf{k}}_{ref})) - \widehat{\mathbf{k}}_{ref} \right\|^{-1}, \tag{17}$$

where $t = t_{ref} + iM$, $i \in \mathbb{Z}$, and the hat of $\widehat{\mathbf{k}}_{ref}$ indicates that it is a 2D position in the reference frame.

4 Experiments

We present some results after editing in Fig. 6. Some effects, including shadow and reflection changes, can also be accurately edited. Please refer to our accompanying video for more results and experiments.

4.1 Implementation Details

We set the weight of motion loss to 10^{-4}, the weight of geometry loss to 0.5, and the weight of warp regularization loss to 0.1. The real data is captured in a resolution of 1280×720 and down-sampled to 320×180 for training. Our network is trained on 4 NVIDIA GeForce RTX 3090 graphics cards, requiring around 5 hours for completion over 250k iterations. The camera poses of input frames are solved by COLMAP [57].

Local coordinate systems for key points. The key point positions are transferred into local coordinate systems for normalization. For each key point, excluding the virtual background key point, this local coordinate system sets the average key point position as the origin and scales the coordinates to ensure that the largest range among the three dimensions is normalized to 1, both based on the initialized positions. This normalization step is crucial, otherwise, the positional encoding functions [1] for key points become nearly linear when positions vary in a small range. Besides, in scenes involving objects with complex geometries, our method may select duplicate key points on the same object. We can remove duplicate key points if the initialized local coordinates of two key points are always very similar throughout the full sequence.

4.2 Comparisons

We compare our method with HyperNeRF [6], CoNeRF [17], DragDiffusion [54] and a baseline method HyperNeRF+Opt on both real and synthetic data.

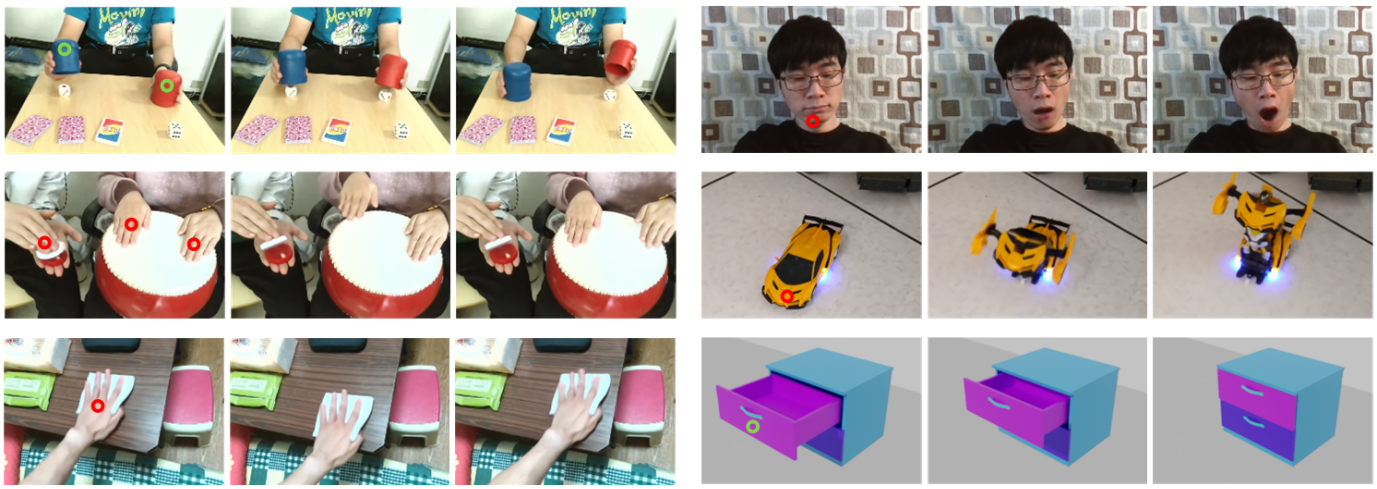

Fig. 6. Our editing results on various scenes. The first image of each scene also shows the key points (circled in red or green). The input data of the *car transformer* scene on the right side is provided by CoNeRF [17]. The last row on the right side is from a synthetic sequence.

TABLE 1
Quantitative comparisons of *reconstruction* and *editing* qualities on synthetic data. The reconstruction qualities are measured by the errors in novel-view synthesis. We report PSNR, MS-SSIM [58], and LPIPS [59]. Our method performs the best.

Method	Reconstruction			Editing		
	PSNR ↑	MS-SSIM ↑	LPIPS ↓	PSNR ↑	MS-SSIM ↑	LPIPS ↓
HyperNeRF [6]	40.97	0.9963	0.0524	-	-	-
HyperNeRF+Opt	40.97	0.9963	0.0524	31.97	0.9763	0.0691
CoNeRF [17]	39.29	0.9916	0.0610	37.10	0.9879	0.0637
Ours (pre) [18]	42.10	0.9967	0.0522	37.29	0.9912	0.0577
Ours	**42.55**	**0.9973**	**0.0515**	**38.18**	**0.9935**	**0.0553**

Dataset. For synthetic data, we generate five sequences using Kubric [60], each containing 400 frames. We evaluate novel view synthesis and editing abilities by manipulating the viewpoints and object positions in these sequences and then rendering ground truth. Two of these sequences are newly introduced in the current version, featuring challenging scenes where different objects move to the same location in different frames. Some results on these synthetic sequences are shown in Fig. 6 and Fig. 10. For real data, we use both our captured dataset and CoNeRF dataset [17]. As it is difficult to obtain real data ground truth for novel view synthesis and editing, quantitative results on real data focus solely on a video interpolation task.

Comparison with HyperNeRF [6] and baseline. HyperNeRF is capable of scene reconstruction but does not enable scene editing. We extend HyperNeRF by integrating RAFT to establish a baseline method HyperNeRF+Opt, aimed at achieving scene editing. This baseline is newly introduced upon the previous version [18]. HyperNeRF+Opt leverages HyperNeRF for reconstruction while can be edited through the optimization of the latent code β in (4) and (5). Optical flow serves as supervision for this optimization process. To be specific, given an original rendered image C_0 from HyperNeRF, a source pixel point x_0 on this image, and a target pixel point x_1, HyperNeRF+Opt edits the scene by finding a proper latent code β so that the surface point corresponding to x_0 in the original image C_0 is relocated to the target point x_1. This optimization is guided by the following loss function:

$$L_{opt}(\beta) = \left\| x_1 - x_0 - F_{C_0}^{C_\beta}(x_0) \right\|^2. \tag{18}$$

TABLE 2
Quantitative comparisons of *interpolation* qualities on real data. We use 15 FPS videos as training sets and interpolate for 30 FPS videos. All the compared methods get similar results, as this task is less challenging compared to novel view synthesis and editing.

Method	PSNR ↑	MS-SSIM ↑	LPIPS ↓
HyperNeRF [6]	30.56	0.9864	0.1281
CoNeRF [17]	30.65	0.9869	0.1307
Ours	30.68	0.9870	0.1311

Here $F_{C_0}^{C_\beta}$ is the optical flow from the original image C_0 to the rendered image C_β generated using latent code β. We compare the reconstruction and editing qualities on synthetic data, presented in Tab. 1, and interpolation qualities on real data, shown in Tab. 2. For editing, the baseline method doesn't perform as well as our method due to the inaccuracies of the optical flow, particularly in cases involving long-distance editing.

Comparison with CoNeRF [17]. CoNeRF allows topologically varying editing but only supports one-dimensional editing for each dynamic part. Additionally, users are required to select frames and provide annotations for each frame, including masks for every editable part and their attribute values. We show some editing results of ours and CoNeRF in Fig. 7. Firstly, as illustrated in the (a) results, our method enables editing 3D dynamics, which cannot be encoded by 1D attribute values in CoNeRF. Secondly, our training stage is fully automatic while CoNeRF needs user annotations. Especially when different parts are close to each other, it becomes quite difficult for users to provide precise masks at the boundaries, which leads to artifacts

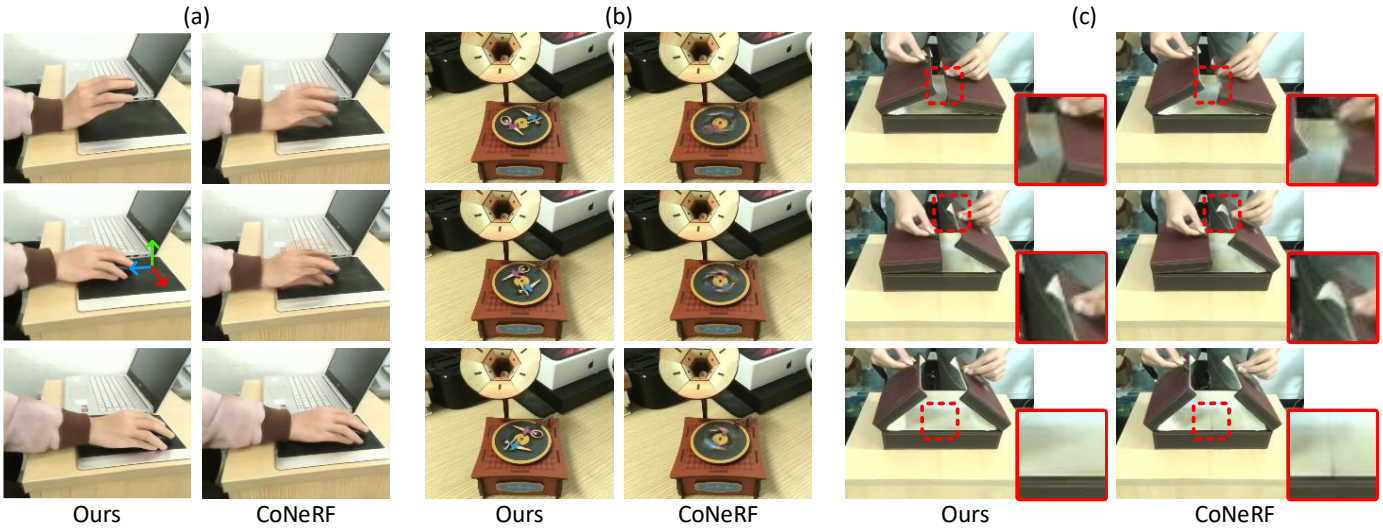

Fig. 7. Comparisons with CoNeRF [17]. Our method does not require user annotations for training and supports up to three-dimensional editing. Note that the rotations in (b) also cannot be represented by the one-dimensional attribute values in CoNeRF.

TABLE 3
Ablation studies on editing ability. We evaluate the motion loss, the geometry loss, and the initialization stage. Our final method in the last row performs the best.

Method	PSNR ↑	MS-SSIM ↑	LPIPS ↓
Base (w/o supervision)	24.17	0.8529	0.1156
+ L_{motion}	27.02	0.9426	0.0775
+ L_{motion} + L_{geo}	30.61	0.9651	0.0671
+ L_{motion} + L_{geo} + init	**38.18**	**0.9935**	**0.0533**

in CoNeRF as shown in (c) of Fig. 7. Thirdly, our drag-style editing is more intuitive than inputting attribute values as in CoNeRF. For CoNeRF training, we utilize ground truth masks and ground truth attribute values, whereas our method still leverages the optical flow from RAFT and the depth maps from HyperNeRF. Quantitative comparisons are also shown in Tab. 1 and Tab. 2.

Comparison with DragDiffusion [54]. We compare our method with a 2D image editing method DragDiffusion [54] in Fig. 8. This method enables drag-style editing on both real and diffusion-generated images. To edit a single image, users need to input a mask for the editable region, an original point, and a target point. Compared to our method, DragDiffusion only takes a single image as input and cannot integrate information from an entire sequence, thus it doesn't reach the same performance as ours and often results in artifacts. In addition, DragDiffusion is not able to generate novel-view images.

Besides, for rendering a single frame, CoNeRF takes 1.77s, HyperNeRF takes 0.95s, and ours takes 0.90s. Comparisons with our previous version [18] are also provided in the supplementary video.

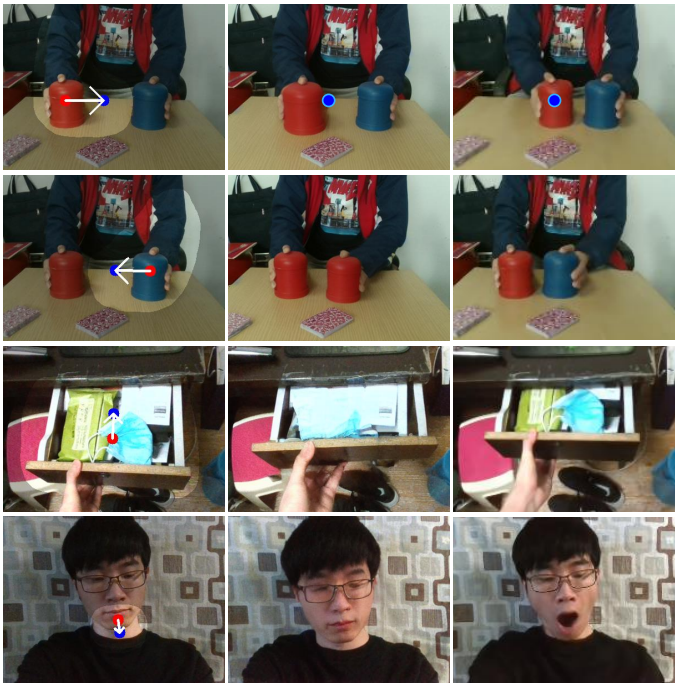

Fig. 8. Comparisons with DragDiffusion [54]. Input images show the editable region (brighter area), source points (red), and target points (blue). The editing results in the first row also show the target points.

4.3 Evaluations

Ablation studies on editing ability. Our method makes use of 3D key points based on the supervision of 2D optical flow and 1D depth maps. They are first utilized to initialize key point positions, then to formulate the motion loss and the geometry loss. We evaluate the two losses and the initialization stage in Tab. 3 by measuring the editing qualities on our synthetic dataset. The base method does not use any key point supervision and the modeled scene changes randomly according to key point movements. Our final method performs the best.

Ablation studies on reconstruction quality. As presented in Tab. 1, our method slightly outperforms HyperNeRF on reconstruction quality. For a more detailed analysis, we show ablation studies on reconstruction in Tab. 4. We evaluate the two losses (motion loss and geometry loss) and the initialization stage. These results indicate that this improvement is mainly due to the key point initialization. In

TABLE 4
Ablation studies on reconstruction quality. We evaluate the two losses (motion loss and geometry loss) and the initialization stage.

Method	PSNR ↑	MS-SSIM ↑	LPIPS ↓
HyperNeRF [6]	40.97	0.9963	0.0524
Ours w/o init	40.42	0.9959	0.0527
Ours w/o 2 losses	42.87	0.9974	0.0515
Ours	42.55	0.9973	0.0515

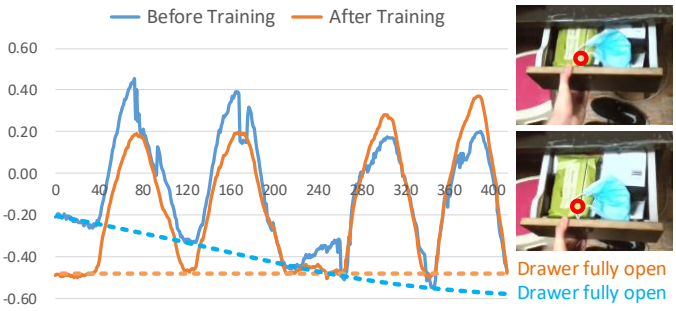

Fig. 9. Evaluation of key points optimization. This data sequence contains pushing and pulling a drawer four times. The vertical coordinates show the key point coordinates along the moving direction, and the horizontal coordinates are the frame indices. Dot lines indicate the key point coordinates when the drawer is fully open. After training, different frames in which the drawer is fully open have the same coordinate, and the accumulative errors in the initialization are eliminated. The two images on the right demonstrate the key point positions after training.

our initialization, the frames that shared similar motions are initialized with similar key point positions, which leads to better results than the random initialization in HyperNeRF. The two losses have little effect on reconstruction quality, but note that the modeled scene can not be edited without these losses, and our method focuses on editing rather than improving the reconstruction quality.

Evaluation of weighted key points strategy. In our method, we use the weighted key points vector to model topologically varying dynamics. A naive alternative method is directly using the concatenated vector containing all key point positions $\mathbf{k}_t \in \mathbb{R}^{3N}$ to replace our weighted key points. However, this naive method results in artifacts as shown in (a) of Fig. 10. This is because our weighted key points vectors perform as ambient coordinates in hyper-space [6]. In our method, the number of ambient dimensions is always 3, while the concatenated vector introduces a higher number of dimensions ($3N$), resulting in a high-dimensional hyperspace. Many areas in this high-dimensional hyperspace are far away from the training space, which leads to artifacts when the key points are edited into these areas for novel scene generation.

Evaluation of key points optimization. Our key point positions are optimized in the training stage after initialization. We show the key point positions both before and after training in Fig. 9. These results demonstrate that there exist accumulative errors in the initialization, and these errors are eliminated through our training stage. In addition, directly using the initialized key point positions without optimization leads to artifacts in the results, as shown in (b) of Fig. 10.

Evaluation of dynamic key point weights. Instead of using the static key point weights in the earlier version [18], we use dynamic key point weights to model more challenging scenes. Results in our accompanying video and (c) of Fig. 10 show that this method successfully models the scenes where two objects move to the same location across different frames, which our earlier version fails to do.

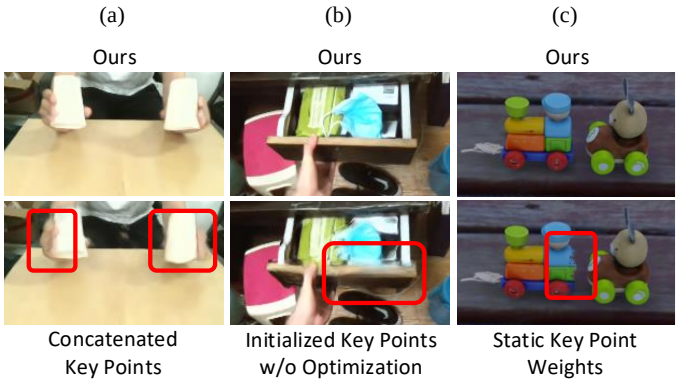

Fig. 10. Evaluations of novel techniques in our method. (a) shows replacing our weighted key points vector with the concatenated key points vector leads to artifacts in novel scene generation. (b) demonstrates that directly using the initialized key point positions without optimization will cause artifacts in the results. (c) shows the static key point weights used in our previous version [18] fail to model overlap regions that different objects can move into. (c) is from a synthetic sequence.

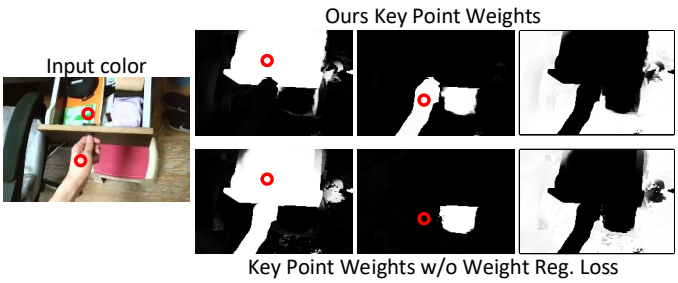

Fig. 11. Evaluation of weight regularization loss. In this sequence, both the hand and the drawer can move to the middle region and are close to each other in some frames. Without the weight regularization loss, the hand may be incorrectly controlled by the key point on the drawer. The last column presents the weights of the virtual background key point.

Evaluation of background key point. Different from the previous version [18], a virtual key point is further introduced to model the background in the scene. This is because, in real data, the captured backgrounds in different frames are not perfectly aligned. A warp field could help to handle the misalignment by warping the background in each frame into the canonical space. But when we use the dynamic key point weights, the canonical background can be modeled by different key points in different states, leading to potential artifacts when the key point that controls the canonical background suddenly switches from one to another. In our accompanying video, we showcase these artifacts, and using a virtual key point for background modeling successfully addresses this problem.

Evaluation of weight regularization loss. In addition to the losses used in the previous version [18], we further add a weight regularization loss in (13) to improve the robustness of our system and enable it to effectively model more challenging scenes. The results in Fig. 11 demonstrate that without this loss function, our method may fail in modeling some complex scenes, where different objects move to the same location in different frames and are close to each other.

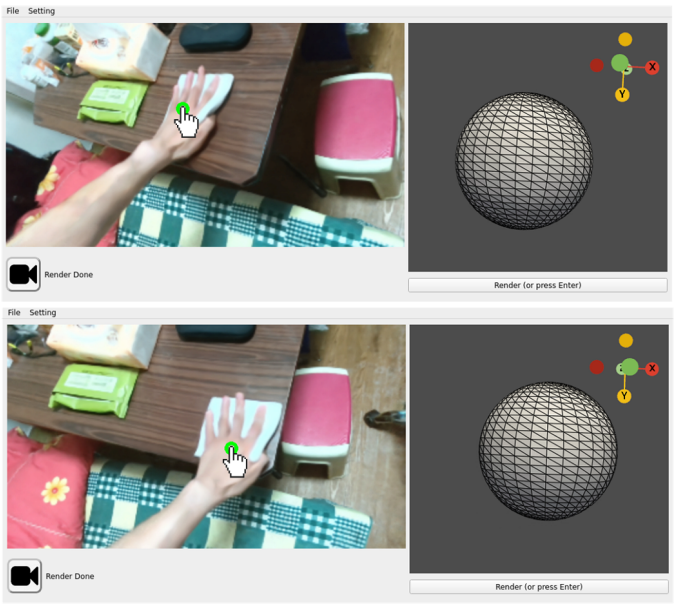

Fig. 12. Graphical user interface. The left widget shows the rendered image and the corresponding draggable key points. The right widget allows the user to adjust the viewpoint.

4.4 Applications

Graphical user interface (GUI). We implement a user-friendly GUI for editing and novel-view synthesis, which is shown in Fig. 12 and in our accompanying video. Our GUI also supports drawing a trail of key points and then generating a video. Note that end-users actually manipulate the key points in the 2D interface, so we provide 1D default depth values for key points to form their corresponding 3D positions, and we also allow end-users to further edit these depth values as needed. To compute the default depth value for a key point located at pixel q, we first project all the key point positions in the input sequence into the current view, then find K closest key point positions to the pixel q in 2D image space, and compute the average depth value of these K key point positions.

Novel scenes generation. Novel scenes that are unseen in the training sequence can also be generated by our method. For example, in the piano toy sequence of Fig. 1, the input data comprises only instances of knocking on individual piano keys, while our method can generate sliding on the piano keys by interpolation. Our method can also combine various dynamics of different parts to create novel scenes, such as the dice cup results shown in Fig. 1 and in our accompanying video.

Motion transfer. After the reconstruction, our modeled scenes can be driven by motions from other sequences. In our accompanying video, we show a phonograph toy driven by a rotating disk from another source video. To achieve this, optical flow is used to track a manually selected point within the source video and an affine transformation then maps the tracked point into our key point space.

Segmentation improvement. Our method can be used to improve video object segmentation results. We show the segmentation results from Track-Anything [61] and our improved results, both in Fig. 13 and in our supplementary video. This improvement is applicable when the tracked

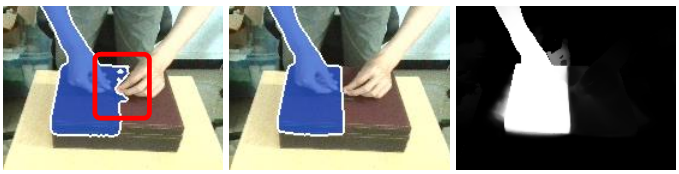

Fig. 13. Segmentation improvement. We use key point weight to refine the video object segmentation results from Track-Anything [61].

Fig. 14. Editing results on a challenging scene where the selected key point is not always visible in the full sequence.

object is a moving part corresponding to a key point. To achieve this, we render the key point weights in the camera view and then use them to clip the masks from Track-Anything [61]. Our key point weights cannot be directly used as segmentation masks because some background regions are also affected by key points due to shadow and reflectance changes.

4.5 Discussions

We build our framework based on surface key points. However, in certain challenging sequences, there may not exist a proper surface point that is visible in all frames to serve as a key point. Our method can still produce plausible results for these sequences, but the consistency of key points is not as good as in other sequences, as shown in Fig. 14.

Limitations. We assume that the dynamics of a canonical location primarily depend on a single key point. If the scene becomes very complex and the dynamics of a position are affected by numerous parts, such as dancing humans, our method may fail. Our method cannot work well when RAFT or HyperNeRF fails and it's challenging for our method to pick out surface key points for semi-transparent objects like smoke. Our generated dynamics are learned from input videos. Therefore, extrapolation editing where key points are far away from their training space cannot be performed well. Also, our method can only generate one-dimensional dynamics when the objects only move in one direction, such as drawers. Our drag-style editing doesn't support other types of editing such as relighting and object insertion.

Future works. Our method supports 3D editing, while a freely moving object has a total of 6 DOFs, which cannot be modeled by only a key point position. To address this problem, incorporating the orientation information in Canonical Capsules [62] can be a valuable solution. Moreover, the integration of techniques like Instant-NGP [63] or Gaussian splatting [64] has the potential to further enhance our method's performance.

5 Conclusions

We propose EditableNeRF, editable topologically varying neural radiance fields that enable end-users to easily edit dynamic scenes. The key to achieving this is leveraging

weighted key points to model topologically varying dynamics, which further achieves intuitive multi-dimensional editing. A scene analysis method that can measure the dynamics in the scene is also proposed to detect and further initialize these key points. Our method is designed for user-friendliness, with fully automatic training using a single-view input sequence, introducing novel applications for editable photo-realistic novel-view synthesis.

ACKNOWLEDGMENTS

This work was supported by the National Key R&D Program of China (2018YFA0704000), Beijing Natural Science Foundation (M22024), the NSFC (No.62021002), and the Key Research and Development Project of Tibet Autonomous Region (XZ202101ZY0019G). This work was also supported by THUIBCS, Tsinghua University, and BLBCI, Beijing Municipal Education Commission. Feng Xu is the corresponding author.

REFERENCES

[1] B. Mildenhall, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *European Conference on Computer Vision*. Springer, 2020, pp. 405–421.

[2] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li, "Nerf: Neural radiance field in 3d vision, a comprehensive review," *arXiv preprint arXiv:2210.00379*, 2022.

[3] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5865–5874.

[4] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 318–10 327.

[5] E. Tretschk, A. Tewari, V. Golyanik, M. Zollhöfer, C. Lassner, and C. Theobalt, "Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 959–12 970.

[6] K. Park, U. Sinha, P. Hedman, J. T. Barron, S. Bouaziz, D. B. Goldman, R. Martin-Brualla, and S. M. Seitz, "Hypernerf: a higher-dimensional representation for topologically varying neural radiance fields," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 6, pp. 1–12, 2021.

[7] X. Zhang, P. P. Srinivasan, B. Deng, P. Debevec, W. T. Freeman, and J. T. Barron, "Nerfactor: Neural factorization of shape and reflectance under an unknown illumination," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 6, pp. 1–18, 2021.

[8] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. Lensch, "Nerd: Neural reflectance decomposition from image collections," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 684–12 694.

[9] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron, "Nerv: Neural reflectance and visibility fields for relighting and view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7495–7504.

[10] S. Liu, X. Zhang, Z. Zhang, R. Zhang, J.-Y. Zhu, and B. Russell, "Editing conditional radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5773–5783.

[11] C. Xie, K. Park, R. Martin-Brualla, and M. Brown, "Fig-nerf: Figure-ground neural radiance fields for 3d object category modelling," in *2021 International Conference on 3D Vision (3DV)*. IEEE, 2021, pp. 962–971.

[12] W. Jang and L. Agapito, "Codenerf: Disentangled neural radiance fields for object categories," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12 949–12 958.

[13] F. Wei, R. Chabra, L. Ma, C. Lassner, M. Zollhöfer, S. Rusinkiewicz, C. Sweeney, R. Newcombe, and M. Slavcheva, "Self-supervised neural articulated shape and appearance models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 15 816–15 826.

[14] B. Yang, Y. Zhang, Y. Xu, Y. Li, H. Zhou, H. Bao, G. Zhang, and Z. Cui, "Learning object-compositional neural radiance field for editable scene rendering," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 13 779–13 788.

[15] J. Zhang, X. Liu, X. Ye, F. Zhao, Y. Zhang, M. Wu, Y. Zhang, L. Xu, and J. Yu, "Editable free-viewpoint video using a layered neural representation," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 4, pp. 1–18, 2021.

[16] H.-X. Yu, L. Guibas, and J. Wu, "Unsupervised discovery of object radiance fields," in *International Conference on Learning Representations*, 2021.

[17] K. Kania, K. M. Yi, M. Kowalski, T. Trzciński, and A. Tagliasacchi, "Conerf: Controllable neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18 623–18 632.

[18] C. Zheng, W. Lin, and F. Xu, "Editablenerf: Editing topologically varying neural radiance fields by key points," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 8317–8327.

[19] K. Guo, F. Xu, T. Yu, X. Liu, Q. Dai, and Y. Liu, "Real-time geometry, albedo, and motion reconstruction using a single rgb-d camera," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 3, pp. 1–13, 2017.

[20] M. Dou, P. Davidson, S. R. Fanello, S. Khamis, A. Kowdle, C. Rhemann, V. Tankovich, and S. Izadi, "Motion2fusion: Real-time volumetric performance capture," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–16, 2017.

[21] J. Thies, M. Zollhöfer, and M. Nießner, "Deferred neural rendering: Image synthesis using neural textures," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.

[22] C. Zheng and F. Xu, "Dtexfusion: Dynamic texture fusion using a consumer rgbd sensor," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 10, pp. 3365–3375, 2021.

[23] S. Lombardi, T. Simon, J. Saragih, G. Schwartz, A. Lehrmann, and Y. Sheikh, "Neural volumes: learning dynamic renderable volumes from images," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.

[24] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhofer, "Deepvoxels: Learning persistent 3d feature embeddings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2437–2446.

[25] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, "Stereo magnification: learning view synthesis using multiplane images," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–12, 2018.

[26] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon, N. K. Kalantari, R. Ramamoorthi, R. Ng, and A. Kar, "Local light field fusion: Practical view synthesis with prescriptive sampling guidelines," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–14, 2019.

[27] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," *Advances in Neural Information Processing Systems*, vol. 32, pp. 1121–1132, 2019.

[28] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.

[29] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, R. Newcombe *et al.*, "Neural 3d video synthesis from multi-view video," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 5521–5531.

[30] M. Işık, M. Rünz, M. Georgopoulos, T. Khakhulin, J. Starck, L. Agapito, and M. Nießner, "Humanrf: High-fidelity neural radiance fields for humans in motion," *arXiv preprint arXiv:2305.06356*, 2023.

[31] W. Xian, J.-B. Huang, J. Kopf, and C. Kim, "Space-time neural irradiance fields for free-viewpoint video," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9421–9431.

[32] B. Attal, E. Laidlaw, A. Gokaslan, C. Kim, C. Richardt, J. Tompkin, and M. O'Toole, "Törf: Time-of-flight radiance fields for dynamic scene view synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 289–26 301, 2021.

[33] Z. Li, S. Niklaus, N. Snavely, and O. Wang, "Neural scene flow fields for space-time view synthesis of dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6498–6508.

[34] Y.-J. Yuan, Y.-K. Lai, T. Wu, L. Gao, and L. Liu, "A revisit of shape editing techniques: From the geometric to the neural viewpoint," *Journal of Computer Science and Technology*, vol. 36, no. 3, pp. 520–554, 2021.
[35] Z. Zheng, T. Yu, Q. Dai, and Y. Liu, "Deep implicit templates for 3d shape representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1429–1439.
[36] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7462–7473, 2020.
[37] F. Xiang, Z. Xu, M. Hasan, Y. Hold-Geoffroy, K. Sunkavalli, and H. Su, "Neutex: Neural texture mapping for volumetric neural rendering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7119–7128.
[38] C. Wang, M. Chai, M. He, D. Chen, and J. Liao, "Clip-nerf: Text-and-image driven manipulation of neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3835–3844.
[39] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "Smpl: A skinned multi-person linear model," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–16, 2015.
[40] S. Peng, J. Dong, Q. Wang, S. Zhang, Q. Shuai, X. Zhou, and H. Bao, "Animatable neural radiance fields for modeling dynamic human bodies," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 14 314–14 323.
[41] J. Chen, Y. Zhang, D. Kang, X. Zhe, L. Bao, X. Jia, and H. Lu, "Animatable neural radiance fields from monocular rgb videos," *arXiv preprint arXiv:2106.13629*, 2021.
[42] L. Liu, M. Habermann, V. Rudnev, K. Sarkar, J. Gu, and C. Theobalt, "Neural actor: Neural free-view synthesis of human actors with pose control," *ACM Transactions on Graphics (TOG)*, vol. 40, no. 6, pp. 1–16, 2021.
[43] G. Gafni, J. Thies, M. Zollhofer, and M. Nießner, "Dynamic neural radiance fields for monocular 4d facial avatar reconstruction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8649–8658.
[44] J. Sun, X. Wang, Y. Zhang, X. Li, Q. Zhang, Y. Liu, and J. Wang, "Fenerf: Face editing in neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7672–7682.
[45] Y. Hong, B. Peng, H. Xiao, L. Liu, and J. Zhang, "Headnerf: A real-time nerf-based parametric head model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 20 374–20 384.
[46] Y. Guo, K. Chen, S. Liang, Y.-J. Liu, H. Bao, and J. Zhang, "Ad-nerf: Audio driven neural radiance fields for talking head synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5784–5794.
[47] Y.-J. Yuan, Y.-T. Sun, Y.-K. Lai, Y. Ma, R. Jia, and L. Gao, "Nerf-editing: geometry editing of neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18 353–18 364.
[48] Y. Endo, "User-controllable latent transformer for stylegan image layout editing," in *Computer Graphics Forum*, vol. 41, no. 7. Wiley Online Library, 2022, pp. 395–406.
[49] X. Pan, A. Tewari, T. Leimkühler, L. Liu, A. Meka, and C. Theobalt, "Drag your gan: Interactive point-based manipulation on the generative image manifold," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 1–11.
[50] P. Ling, L. Chen, P. Zhang, H. Chen, and Y. Jin, "Freedrag: Point tracking is not you need for interactive point-based image editing," *arXiv preprint arXiv:2307.04684*, 2023.
[51] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.
[52] C. Mou, X. Wang, J. Song, Y. Shan, and J. Zhang, "Dragondiffusion: Enabling drag-style manipulation on diffusion models," *arXiv preprint arXiv:2307.02421*, 2023.
[53] X. Chen, L. Huang, Y. Liu, Y. Shen, D. Zhao, and H. Zhao, "Anydoor: Zero-shot object-level image customization," *arXiv preprint arXiv:2307.09481*, 2023.
[54] Y. Shi, C. Xue, J. Pan, W. Zhang, V. Y. Tan, and S. Bai, "Dragdiffusion: Harnessing diffusion models for interactive point-based image editing," *arXiv preprint arXiv:2306.14435*, 2023.
[55] Z. Li, R. Tucker, N. Snavely, and A. Holynski, "Generative image dynamics," *arXiv preprint arXiv:2309.07906*, 2023.
[56] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*. Springer, 2020, pp. 402–419.
[57] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4104–4113.
[58] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, vol. 2. Ieee, 2003, pp. 1398–1402.
[59] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.
[60] K. Greff, F. Belletti, L. Beyer, C. Doersch, Y. Du, D. Duckworth, D. J. Fleet, D. Gnanapragasam, F. Golemo, C. Herrmann *et al.*, "Kubric: A scalable dataset generator," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 3749–3761.
[61] J. Yang, M. Gao, Z. Li, S. Gao, F. Wang, and F. Zheng, "Track anything: Segment anything meets videos," *arXiv preprint arXiv:2304.11968*, 2023.
[62] W. Sun, A. Tagliasacchi, B. Deng, S. Sabour, S. Yazdani, G. E. Hinton, and K. M. Yi, "Canonical capsules: Self-supervised capsules in canonical pose," *Advances in Neural information processing systems*, vol. 34, pp. 24 993–25 005, 2021.
[63] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics (ToG)*, vol. 41, no. 4, pp. 1–15, 2022.
[64] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics (ToG)*, vol. 42, no. 4, pp. 1–14, 2023.

Chengwei Zheng received both his B.S. degree and Ph.D. degree in software engineering from Tsinghua University, Beijing, China, in 2018 and 2023, respectively. He is currently a postdoc at the Advanced Interactive Technology lab at ETH Zürich. His research interests include dynamic reconstruction and 3D animation.

Wenbin Lin received a B.S. degree in Department of Automation, Tsinghua University, Beijing, China, in 2020. He is currently working toward a Ph.D. degree in the School of Software, Tsinghua University. His research interests include 3D human digitalization, dynamic reconstruction, and 3D animation.

Feng Xu received a B.S. degree in physics from Tsinghua University, Beijing, China, in 2007 and Ph.D. in automation from Tsinghua University, Beijing, China, in 2012. He is currently an associate professor in the School of Software, Tsinghua University. His research interests include face animation, performance capture, and 3D reconstruction.