SIGGRAPH 2025

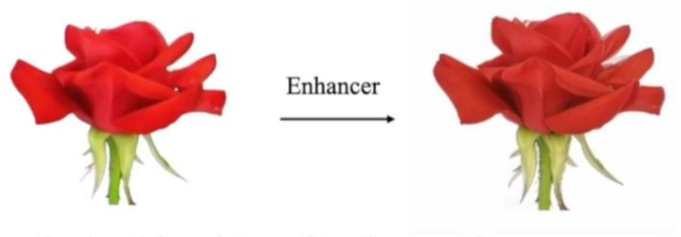
Splat4D: Diffusion-Enhanced 4D Gaussian Splatting for Temporally and Spatially Consistent Content Creation

MINGHAO YIN, The University of Hong Kong, Hong Kong
YUKANG CAO, Nanyang Technological University, Singapore
SONGYOU PENG, Google DeepMind, USA
KAI HAN*, The University of Hong Kong, Hong Kong

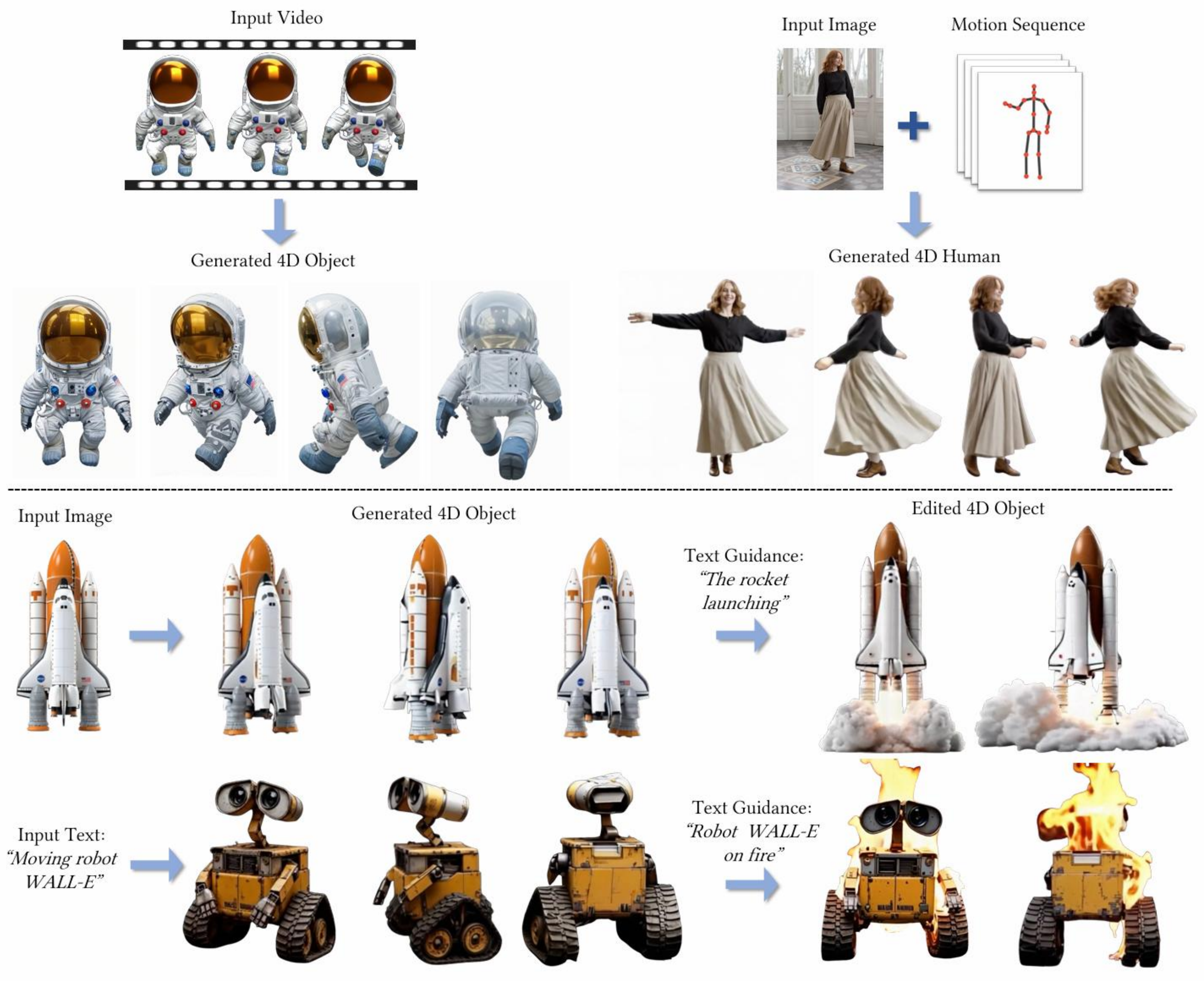
Splat4D - a novel framework enabling high-fidelity 4D content generation from a monocular video.

HOW TO ENSURE VISUAL QUALITY?

- Introduce enhancer to boost visual fidelity and fine-grained details:



- How to ensure temporal and spatial consistency after enhancement?
- → Leverage a video diffusion model for 4D consistency refinement.



Our method demonstrates strong generalization capabilities, enabling the generation of temporally stable and high-fidelity 4D content from monocular videos, images, and text prompts.

Pipeline

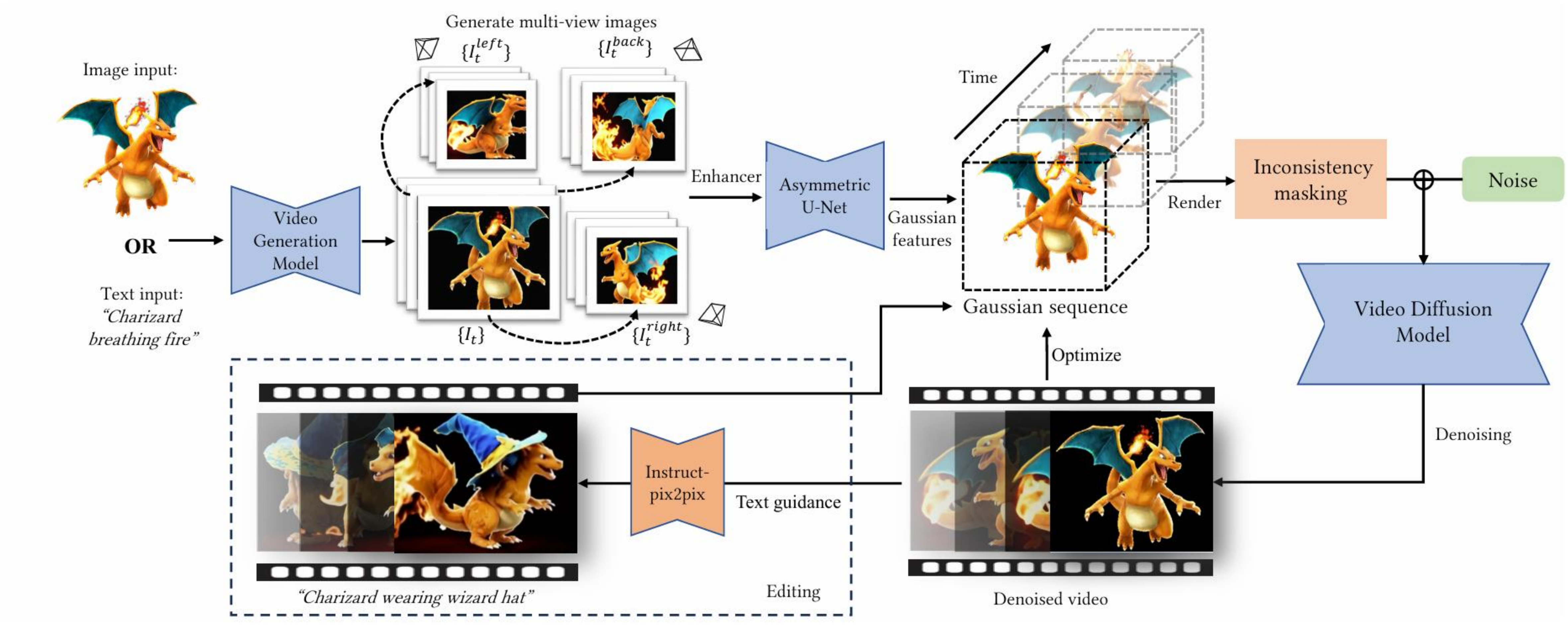


Fig. 2. **Overview of Splat4D**. Our method for 4D content generation begins with processing input data (text, image, or monocular video) to produce high-quality multi-view image sequences. These sequences are used to initialize a 4D Gaussian representation via an asymmetry U-Net and image splattering. Refinement steps include leveraging uncertainty masking and video denoising diffusion to ensure high fidelity and spatial-temporal consistency, culminating in versatile 4D content creation. The pipeline supports optional text-guided content editing, enabling dynamic modifications of the 4D output for enhanced flexibility and creative control.

Pipeline - Coarse 4D Gaussian Generation

- Utilize the **video diffusion model** to generate the image sequence.
- Use **MV-Adapter** to generate additional views.
- Apply **image enhancer** to improve quality and details.
- Utilize an asymmetric U-Net and Splatter Image head to transform multi-view image sequence into a Gaussian sequence

3.2.1 Multi-view Video Generation. We utilize the video diffusion model [Blattmann et al. 2023a] to generate the image sequence. However, relying solely on a single-view video does not provide enough information for robust 4D modeling. This limitation stems from issues like depth ambiguity and the lack of side and back view information. To address this, we enhance the single-view video by using MV-Adapter [Huang et al. 2024] to generate additional views, including the front, back, and sides, thereby enriching the model with more comprehensive rotational perspectives:

$$\texttt{MV-Adapter}(I_t) \rightarrow \{I_t, I_t^{\text{left}}, I_t^{\text{right}}, I_t^{\text{back}}\}, \quad (2)$$

See Supplementary Material for evaluation for the choice of MV-Adapter over SV4D.

3.2.2 Multi-view Image Enhancer. Although MV-Adapter can robustly provide multi-view perspectives, their generated videos often lack fine-grained details and the high resolution required for realistic 4D content (see the figure in the supplementary material). This issue is expected as the input samples would always fall outside the training distribution of MV-Adapter. To address this, we propose to apply an image enhancer model [Wang et al. 2018] IE to refine textures, edges, and details for each frame and each view.

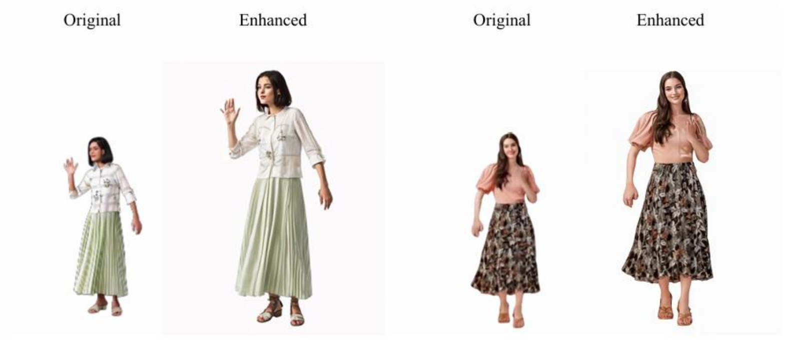
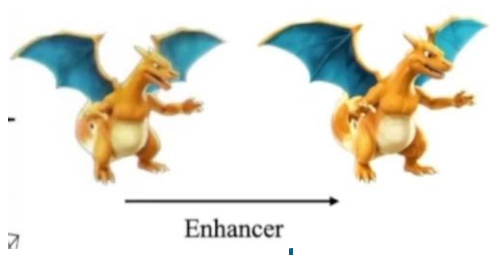
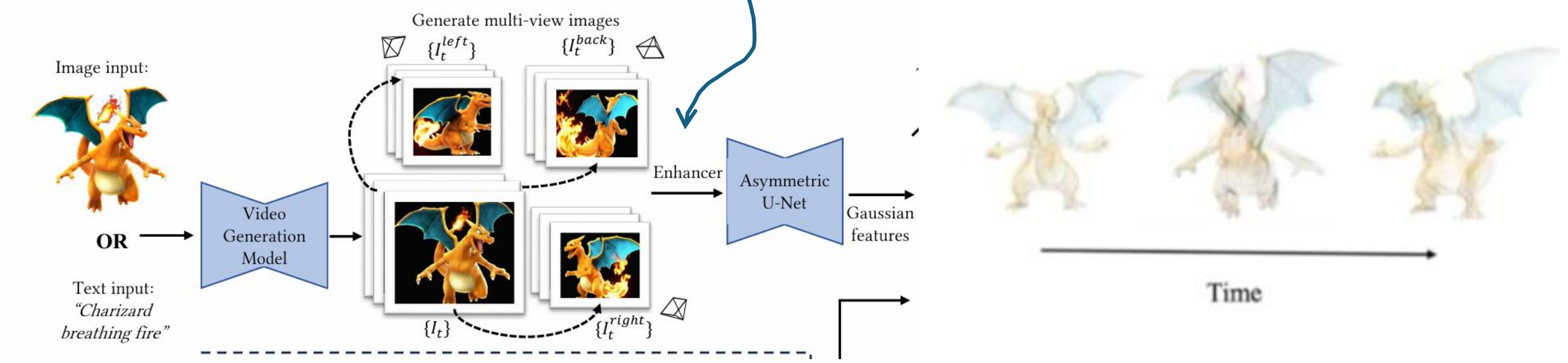


Fig. 7. **Effect of Image Enhancer.** We show the difference between human images before and after enhancement with the image enhancer. The enhanced images contain more fine details.



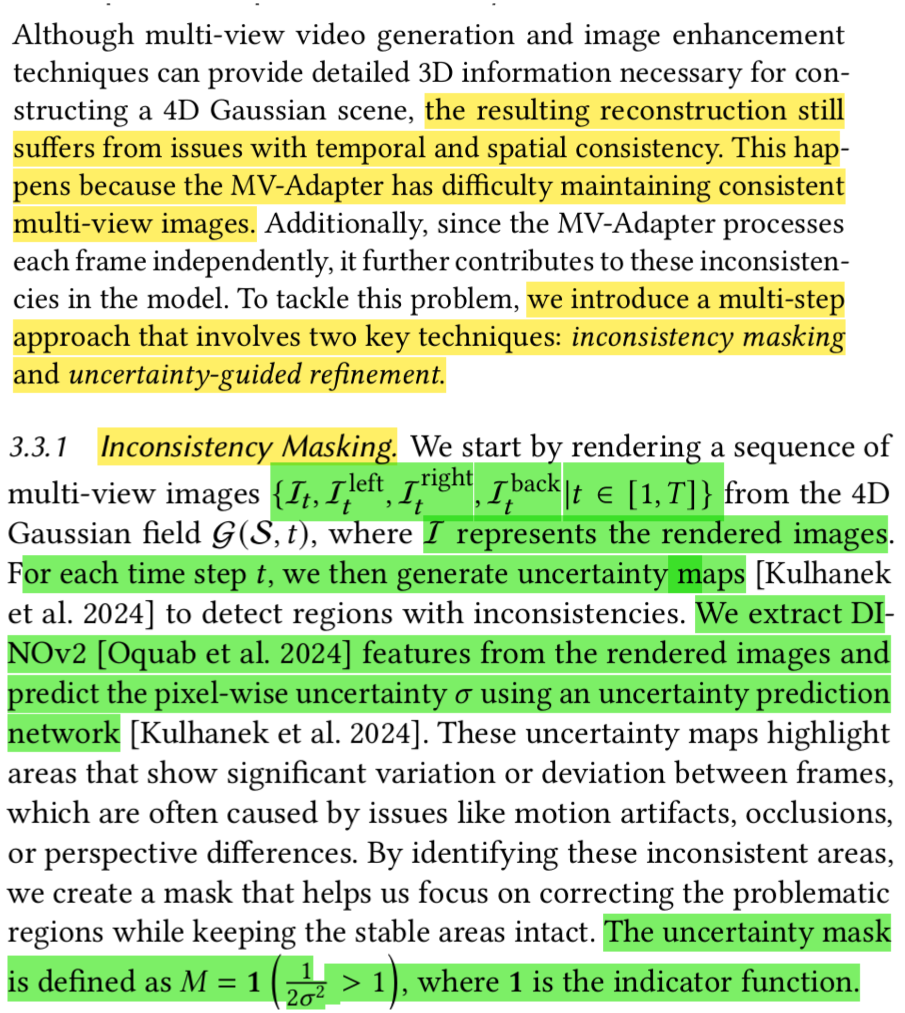


3.2.3 4D Gaussian Reconstruction. After generating a high-quality, multi-view image sequence, we proceed to construct a 4D Gaussian field. Following LGM [Tang et al. 2024a], we first input the multi-view image sequence into U-Net to encode key spatial and depth features across the views. The U-Net architecture is well-suited for this task because it can capture detailed structures at multiple resolutions through its encoder-decoder structure. The encoder captures feature maps at different scales, identifying essential textures and depth cues, while the decoder reconstructs these features into a cohesive representation.

Once the U-Net has processed the multi-view sequence, we apply the Splatter Image [Szymanowicz et al. 2024] method to project these learned features into a continuous 4D Gaussian field. Specifically, Splatter Image maps each pixel from the feature maps into a series of localized Gaussian distributions in 3D space, with each Gaussian representing a small spatial region from the scene. To form the final temporally consistent Gaussian sequences, we design our network to separately reconstruct a 3D Gaussian field for each frame (time step). Specifically, we construct a stacked representation of multiple 3D Gaussians, represented as $\mathcal{G}(\mathcal{S}, t) = [\mathcal{X}_t, s_t, r_t, \sigma_t, \zeta_t]$, with position, scale, rotation, opacity and Spherical Harmonics (SH) at time t. This Gaussian field serves as a foundational structure, providing a spatially continuous and temporally stable representation that can be rendered from any angle.

Pipeline - Spatial-Temporal Consistency Refinement

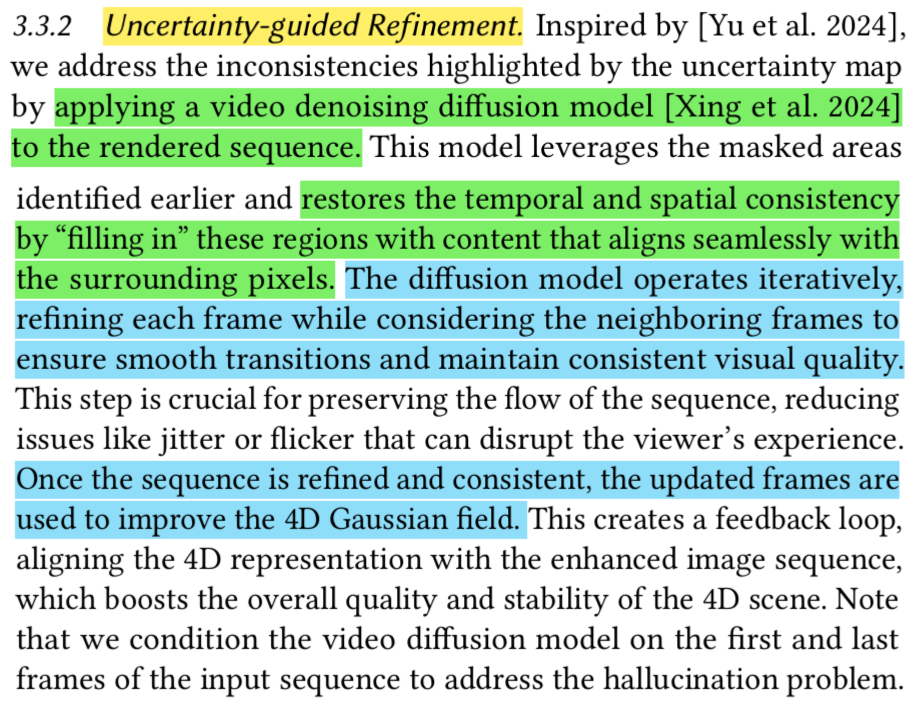
- Mask inconsistent areas by **uncertainty prediction**.
- Inpaint masked areas by **video diffusion model**.
- Optimize gaussian sequence by denoised video.



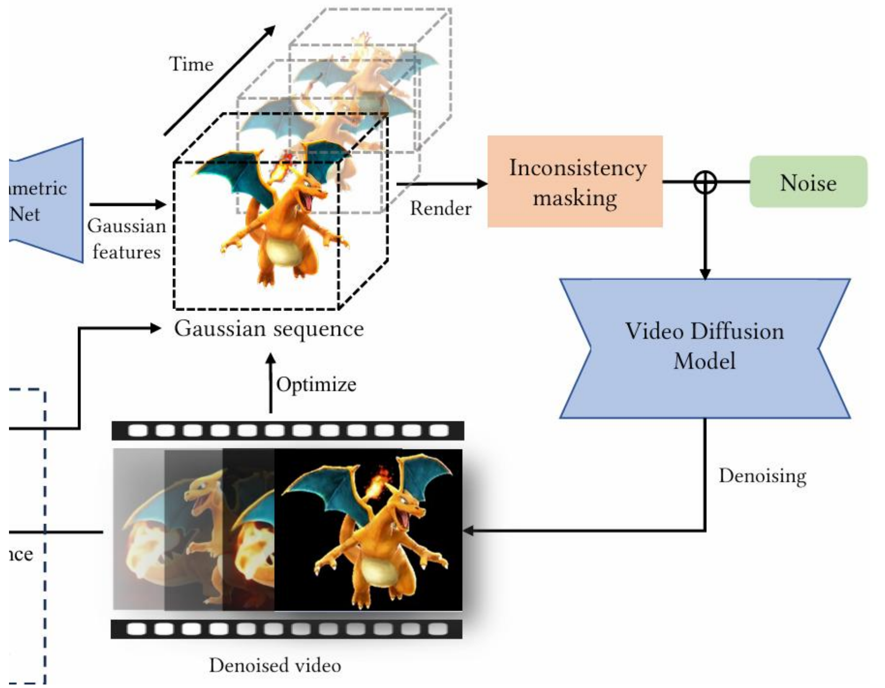
Although multi-view video generation and image enhancement techniques can provide detailed 3D information necessary for constructing a 4D Gaussian scene, the resulting reconstruction still suffers from issues with temporal and spatial consistency. This happens because the MV-Adapter has difficulty maintaining consistent multi-view images. Additionally, since the MV-Adapter processes each frame independently, it further contributes to these inconsistencies in the model. To tackle this problem, we introduce a multi-step approach that involves two key techniques: *inconsistency masking* and *uncertainty-guided refinement*.

3.3.1 Inconsistency Masking. We start by rendering a sequence of multi-view images $\{\mathcal{I}_t, \mathcal{I}_t^{\text{left}}, \mathcal{I}_t^{\text{right}}, \mathcal{I}_t^{\text{back}} | t \in [1, T]\}$ from the 4D Gaussian field $\mathcal{G}(\mathcal{S}, t)$, where $\mathcal{I}$ represents the rendered images. For each time step t, we then generate uncertainty maps [Kulhanek et al. 2024] to detect regions with inconsistencies. We extract DINOv2 [Oquab et al. 2024] features from the rendered images and predict the pixel-wise uncertainty σ using an uncertainty prediction network [Kulhanek et al. 2024]. These uncertainty maps highlight areas that show significant variation or deviation between frames, which are often caused by issues like motion artifacts, occlusions, or perspective differences. By identifying these inconsistent areas, we create a mask that helps us focus on correcting the problematic regions while keeping the stable areas intact. The uncertainty mask is defined as $M = \mathbf{1}\left(\frac{1}{2\sigma^2} > 1\right)$, where $\mathbf{1}$ is the indicator function.

DINOv2 + Pixel-wise uncertainty prediction network



3.3.2 Uncertainty-guided Refinement. Inspired by [Yu et al. 2024], we address the inconsistencies highlighted by the uncertainty map by applying a video denoising diffusion model [Xing et al. 2024] to the rendered sequence. This model leverages the masked areas identified earlier and restores the temporal and spatial consistency by "filling in" these regions with content that aligns seamlessly with the surrounding pixels. The diffusion model operates iteratively, refining each frame while considering the neighboring frames to ensure smooth transitions and maintain consistent visual quality. This step is crucial for preserving the flow of the sequence, reducing issues like jitter or flicker that can disrupt the viewer's experience. Once the sequence is refined and consistent, the updated frames are used to improve the 4D Gaussian field. This creates a feedback loop, aligning the 4D representation with the enhanced image sequence, which boosts the overall quality and stability of the 4D scene. Note that we condition the video diffusion model on the first and last frames of the input sequence to address the hallucination problem.



Pipeline - Generalizable 3D Gaussian Field Predictor Learning

Aside from the inconsistency issues we've already improved, we also find that the quality of the 4D Gaussian fields doesn't always match the improvements made by the image enhancer. This is expected as there is a notable domain gap between the pre-trained distribution of the image enhancer model [Wang et al. 2018], which is trained on DIV2K dataset [Agustsson and Timofte 2017], and LGM [Tang et al. 2024a], which is trained on Objaverse [Deitke et al. 2023]. To address this issue, we propose to fine-tune the U-Net model derived from LGM with the pre-processed Objaverse dataset. Specifically, we first follow LGM [Tang et al. 2024a] to filter low-quality 3D models. In each training step, we randomly choose an input image with an elevation angle between -30 and 30 degrees. The MV-Adapter [Huang et al. 2024] is then used to generate four orthogonal views, including the original image. These views are processed through the image enhancer and consistency refinement steps, and are subsequently passed into the U-Net model to produce the 4D Gaussian field. Finally, we render images from the Gaussian field based on the angles of the four orthogonal views for supervision. This training process allows the fine-tuned U-Net to reduce the domain gap between the pre-trained U-Net from LGM and the image enhancer model, resulting in improved quality of the 4D Gaussian fields.

Table 4. **Ablation Study.** The experiments are conducted on Consistent4D dataset [Jiang et al. 2023].

Model	LPIPS↓	CLIP-S↑	FVD-F↓	FVD-V↓
w/o mask	0.114	0.93	507.15	413.79
w/o train	0.107	0.96	445.33	364.81
Ours	**0.090**	**0.98**	**390.85**	**282.79**

Table 5. **Additional Ablation Study.** The first row illustrates the results without image enhancer. The second row shows the results using SV4D over MV-Adapter for multi-view generation.

Model	LPIPS↓	CLIP-S↑	FVD-F↓	FVD-V↓
w/o loop	0.120	0.90	831.84	473.91
w/o enhancer	0.108	0.96	463.39	384.26
*SV4D**	0.105	0.94	441.72	356.25
w/o cond	0.101	0.94	425.72	339.05
Ours	**0.090**	**0.98**	**390.85**	**282.79**

Implementation Details

For the evaluation of video-to-4D generation, we utilize the video dataset provided by Consistent4D [Jiang et al. 2023]. We employ the Segment Anything Model (SAM) [Kirillov et al. 2023] to preprocess the input image sequences to extract the foreground objects. To evaluate image-to-4D generation, we curate a dataset by collecting images from the internet. These images are converted to RGBA format and resized to a resolution of 512×512 to ensure compatibility with our pipeline. For fine-tuning, we utilize the 80K 3D object subset[Tang et al. 2024a] of the Objaverse dataset [Deitke et al. 2023] after filtering out low-quality models. Each 3D model is rendered into RGB images from 100 camera views at a resolution of 512 × 512.

The training process is being conducted using the asymmetric U-Net model on 4 NVIDIA V100 GPUs, with each GPU processing a batch size of 4 under bfloat16 precision. For each batch, a single camera view is being randomly sampled, while 4 orthogonal views are being generated using the MV-adapter [Huang et al. 2024] based on the input view. The asymmetric U-Net model is generating the 3D Gaussian field, which is then being rendered into images for the orthogonal views. Original Objaverse 3D object rendered images are being used as supervision signals. The rendered 3D Gaussians are being compared to the original at a resolution of 512×512 using the mean squared error (MSE) loss. To optimize memory usage, images are being resized to 256×256 for LPIPS loss calculation. The AdamW optimizer is being employed with a learning rate of 4×10^{-4}, a weight decay of 0.05, and momentum parameters of 0.9. The learning rate is following a cosine annealing schedule to gradually decay to zero during training. Gradients are being clipped to a maximum norm of 1.0 to enhance stability. Additionally, grid distortion and camera jitter are being applied with a probability of 50% to improve generalization.

Comparison

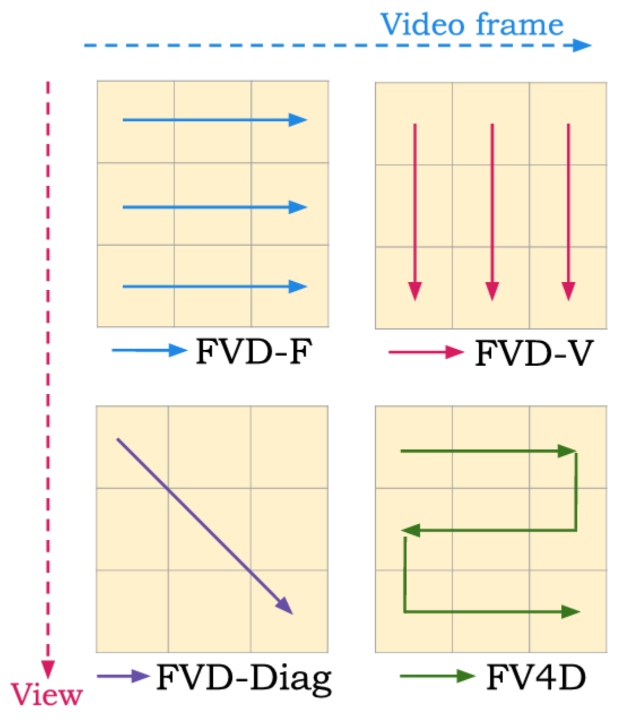


Figure 7: **Illustrations of video and 4D metrics.** FVD-F evaluates coherence between video frames from a fixed view. FVD-V captures multi-view consistency. We also design FVD-Diag and FV4D to evaluate 4D consistency by traversing the image matrix through different paths.

Table 1. ***Video-to-4D* quantitative Comparison on Consistent4D Dataset [Jiang et al. 2023].**

Model	LPIPS↓	CLIP-S↑	FVD-F↓	FVD-V↓
Consistent4D	0.134	0.87	1133.93	735.79
STAG4D	0.126	0.91	992.21	685.23
SV4D	0.118	0.92	732.40	503.51
4Diffusion	0.13	0.94	489.2	405.5
Ours	**0.090**	**0.97**	**390.85**	**282.79**

Table 2. ***Video-to-4D* Quantitative Comparison on ObjaverseDy Test Set [Deitke et al. 2023; Xie et al. 2024].**

Model	LPIPS↓	CLIP-S↑	FVD-F↓	FVD-V↓
Consistent4D	0.165	0.896	880.54	488.38
STAG4D	0.158	0.860	929.10	453.62
SV4D	0.131	0.905	659.66	368.53
Ours	**0.112**	**0.939**	**383.71**	**267.94**

Table 3. **Quantitative Comparison on *Image-to-4D* Generation.**

Model	LPIPS↓	CLIP-S↑	PSNR↓	FVD↓
4DGen	0.28	0.84	14.4	736.6
STAG4D	0.24	0.86	15.2	675.4
Diffusion4D	0.18	0.89	16.8	490.2
Ours	**0.12**	**0.94**	**19.2**	**395.0**

Comparison

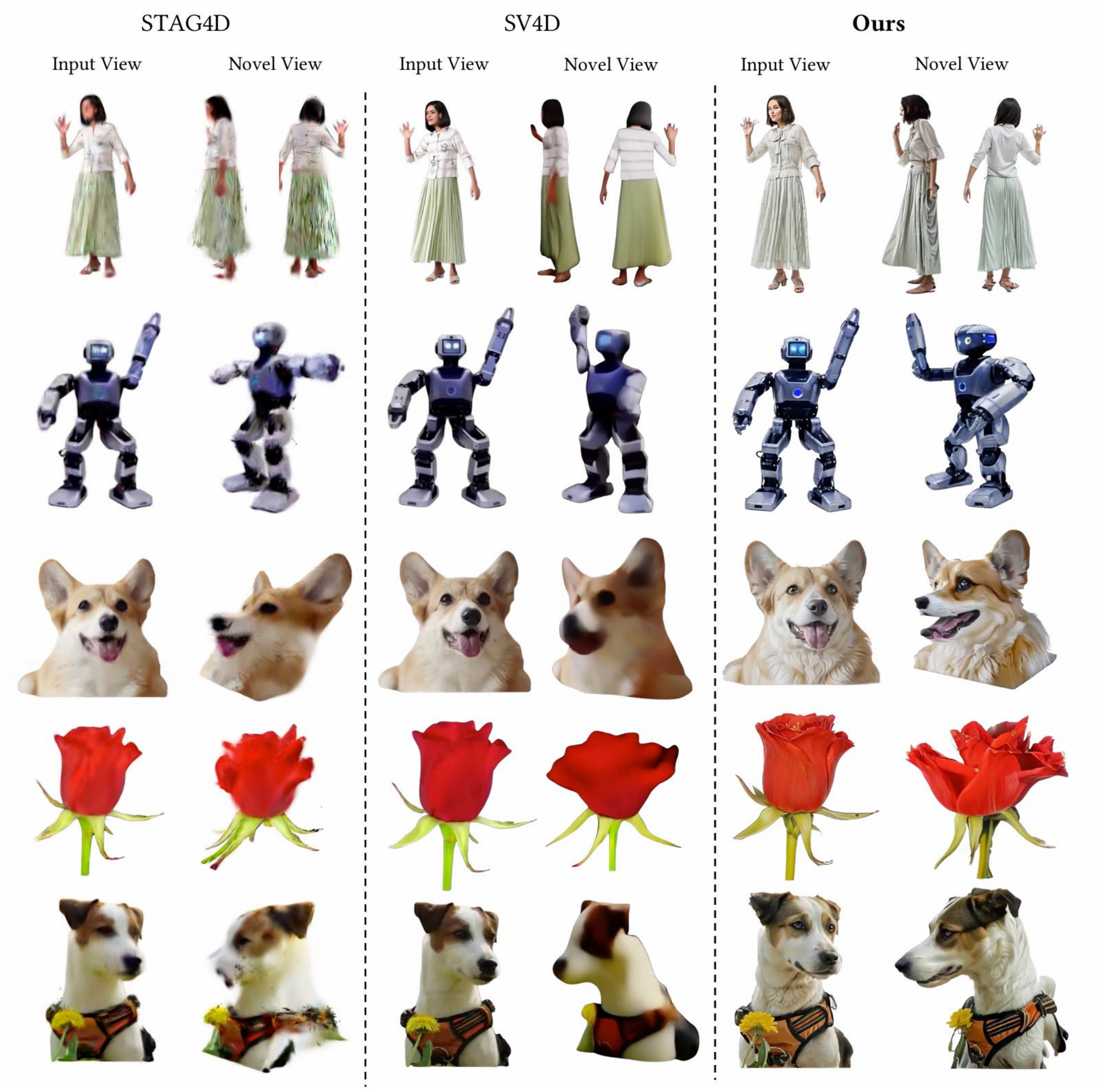


Fig. 3. **Comparison on *video-to-4D* generation**. The rendered image of the input view from the 4D object is on the left column, and the rendered images of the novel view are illustrated on the two right columns.

Application - Text/Image Conditioned 4D Generation.

+ text-to-image diffusion model
+ stable video diffusion model

4.4.1 Text/Image Conditioned 4D Generation. To demonstrate our method's capability for *text-to-4D* and *image-to-4D* generation, we show our generation results in Fig. 4. For *text-to-4D generation*, we first employ a text-to-image diffusion model to convert the input textual prompt into a high-quality image and then combine that image with the original text in a stable video diffusion model to produce a coherent short video. In *image-to-4D generation*, the pipeline simply begins with an input image, bypassing the text-to-image step. From the resulting videos, we use a multi-view diffusion model to generate four orthogonal view sequences, which are then fed into our reconstruction pipeline to construct a 4D Gaussian field.

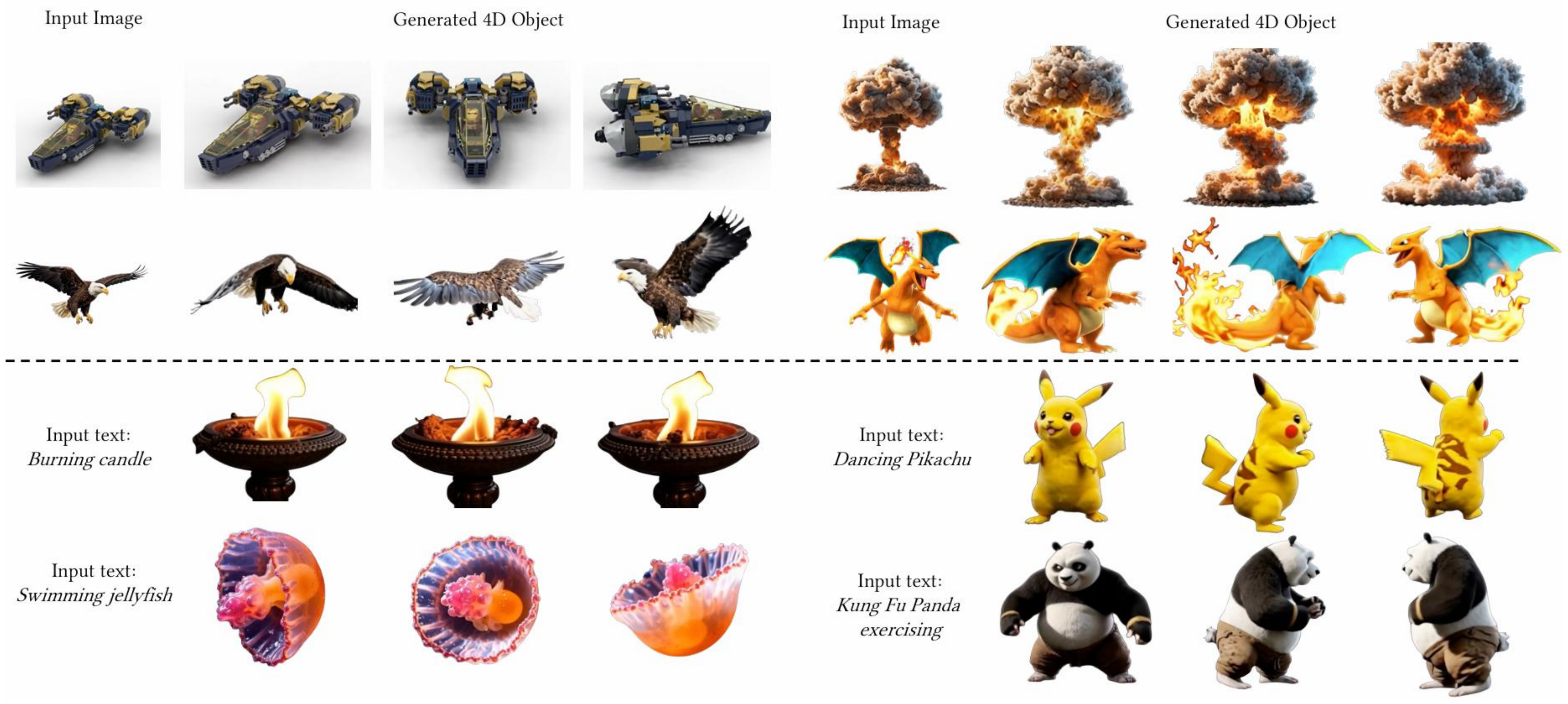

Input Image
Generated 4D Object
Input Image
Generated 4D Object
Input text: *Burning candle*
Input text: *Swimming jellyfish*
Input text: *Dancing Pikachu*
Input text: *Kung Fu Panda exercising*

Fig. 4. **4D Content Creation with Text/Image as Input**. The first two rows are results with image inputs, and bottom two rows are results with text inputs.

Application - 4D Human Generation.

+ human pose extraction model
+ 2D human motion transfer model

4.4.2 4D Human Generation. Given a source video with the desired motion to be transferred and an image of the human subject to be animated, we first use a pose extraction model [Kocabas et al. 2020] to detect key body landmarks, skeletal poses, and motion trajectories. Next, we apply Champ [Zhu et al. 2024], a 2D motion transfer model, to animate the input image, making it move according to the extracted motion. Our method then uses the resulting animated image sequence $\{I_t | t \in [1, T]\}$, where T is the total number of frames, to generate the corresponding 4D Gaussian scenes. Fig. 5 showcases the results of our *4D human generation* pipeline, which combines a single input image with a motion sequence to produce high-fidelity, dynamic human representations. First, the input image and motion sequence are processed through a 2D motion transfer model to create a video of the subject performing the specified action (see details in Supplementary Material). Next, we follow our pipeline and apply a multi-view diffusion model and construct 4D Gaussians of the human.

Fig. 5 illustrates the effectiveness of our approach. These results highlight our ability to preserve intricate human details, including complex structures like facial details and loose clothing, across varying views and motions.

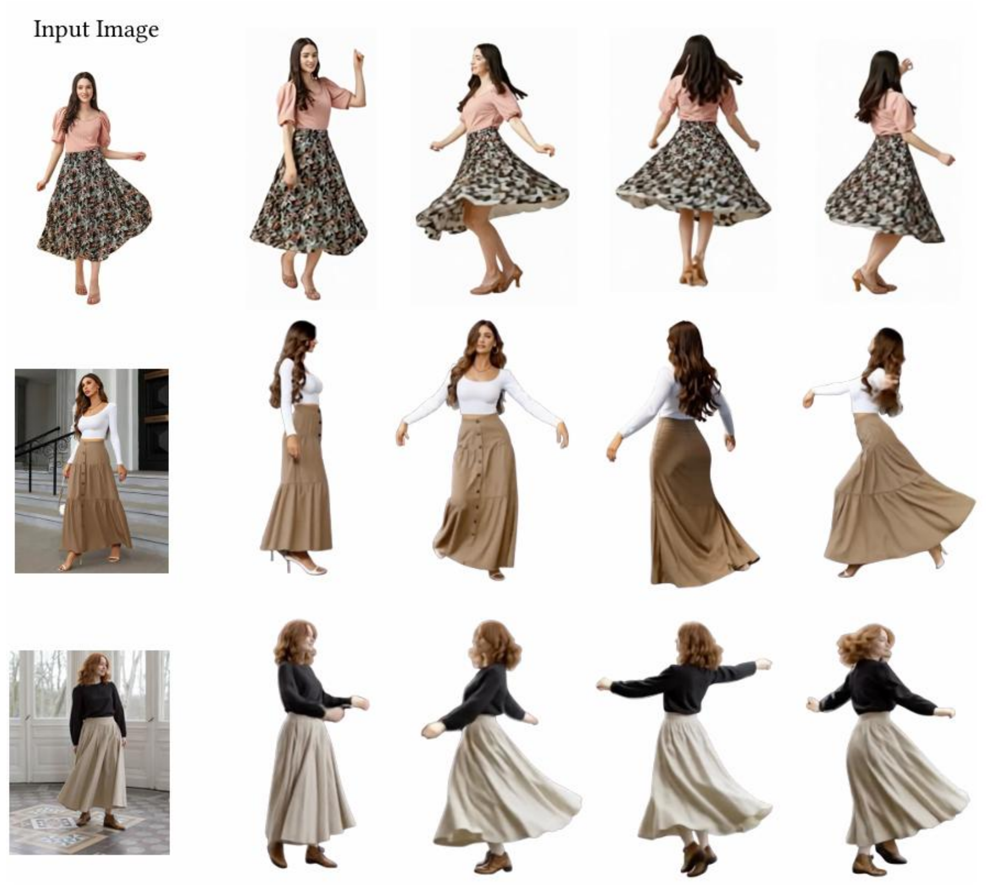


Fig. 5. **4D Human Generation with an Input Image as Guidance**. The first row shows the input image, while the subsequent rows depict rendered novel views under various poses.

Application - Text-guided Editing

+ text-to-image diffusion model
+ Instruct-Pix2Pix (text guided image-to-image translation)

4.4.3 *Text-guided Editing.* For this application, we use the Instruct-Pix2Pix [Brooks et al. 2023] network to modify the output video generated by the video diffusion model, guided by a text prompt (see Fig. 2). For instance, starting with a video of a house, the pix2pix network is applied to transform the video based on a prompt like "house on fire". Specifically, the pix2pix network performs image-to-image translation, adjusting each frame of the video to match the specified scene changes. Once the transformation is complete, the modified video sequence is used to refine the corresponding 4D Gaussian sequence, resulting in a final 4D content that accurately reflects the updated dynamics of the "house on fire" scenario. In Fig. 6, we showcase our method's *text-guided 4D editing* capability that transforms 4D Gaussian representations based on user-specified textual or visual prompts. Starting from our 4D Gaussian field, we employ a pix2pix network to edit the rendered video according to the guidance text, producing an updated video sequence. This sequence is further optimized using the 4D Gaussian representation, ensuring coherence and alignment with the guidance.

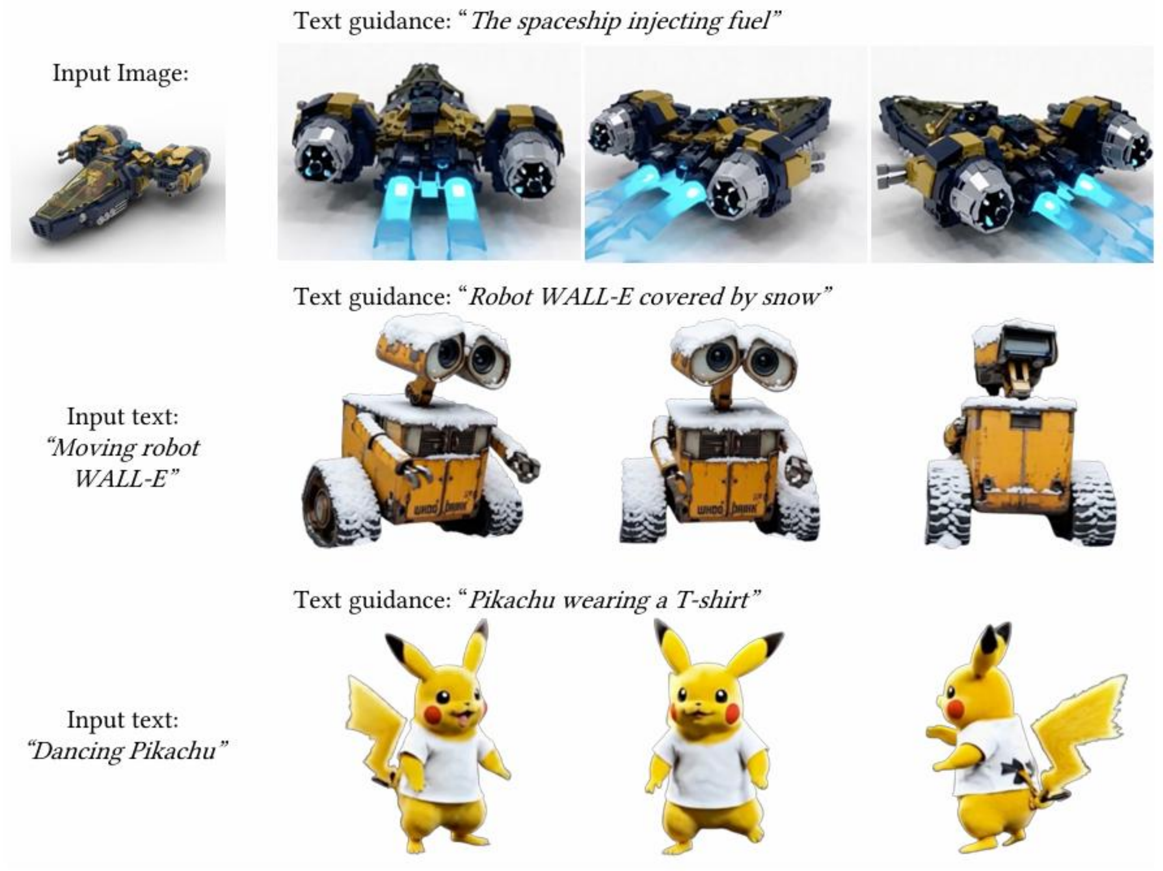

Text guidance: "*The spaceship injecting fuel*"
Input Image:
Text guidance: "*Robot WALL-E covered by snow*"
Input text: "*Moving robot WALL-E*"
Text guidance: "*Pikachu wearing a T-shirt*"
Input text: "*Dancing Pikachu*"

Fig. 6. **4D Content Editing with Text Guidance**. The first column showcases the original input (text or image), while the subsequent three columns present the edited outputs. Each edited 4D object is displayed beneath the corresponding text.

Spline Deformation Field

MINGYANG SONG, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
YANG ZHANG, DisneyResearch|Studios, Switzerland
MARKO MIHAJLOVIC, ETH Zürich, Switzerland
SIYU TANG, ETH Zürich, Switzerland
MARKUS GROSS, DisneyResearch|Studios, Switzerland and ETH Zürich, Switzerland
TUNÇ OZAN AYDIN, DisneyResearch|Studios, Switzerland

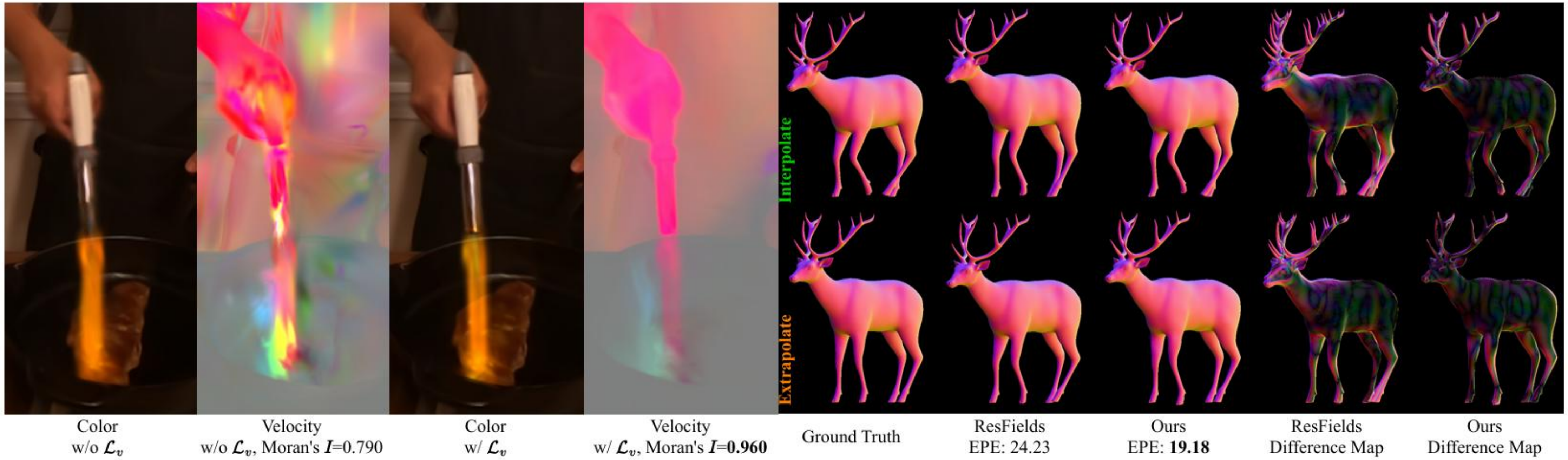
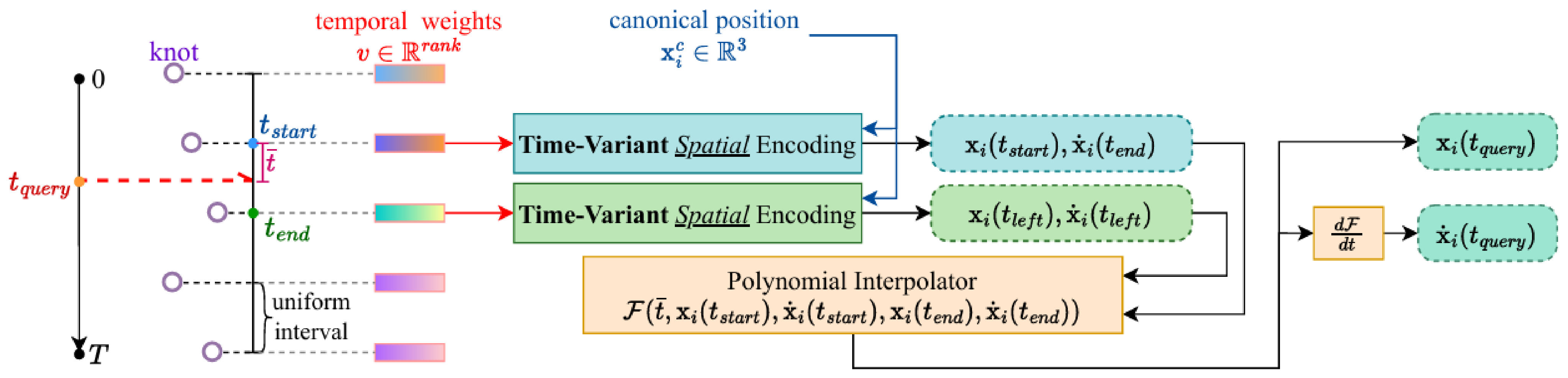


Fig. 1. We represent trajectories with splines, in which analytically derived velocity v enables effective spatial coherency preservation through our novel loss term $\mathcal{L}_v$. Furthermore, we propose a metric concepting on Moran's I to quantify it. Spline interpolation can generate fair interpolated and extended motions, compared to purely relying on the smoothness inductive bias of the implicit deformation field, such as ResFields [Mihajlovic et al. 2024].

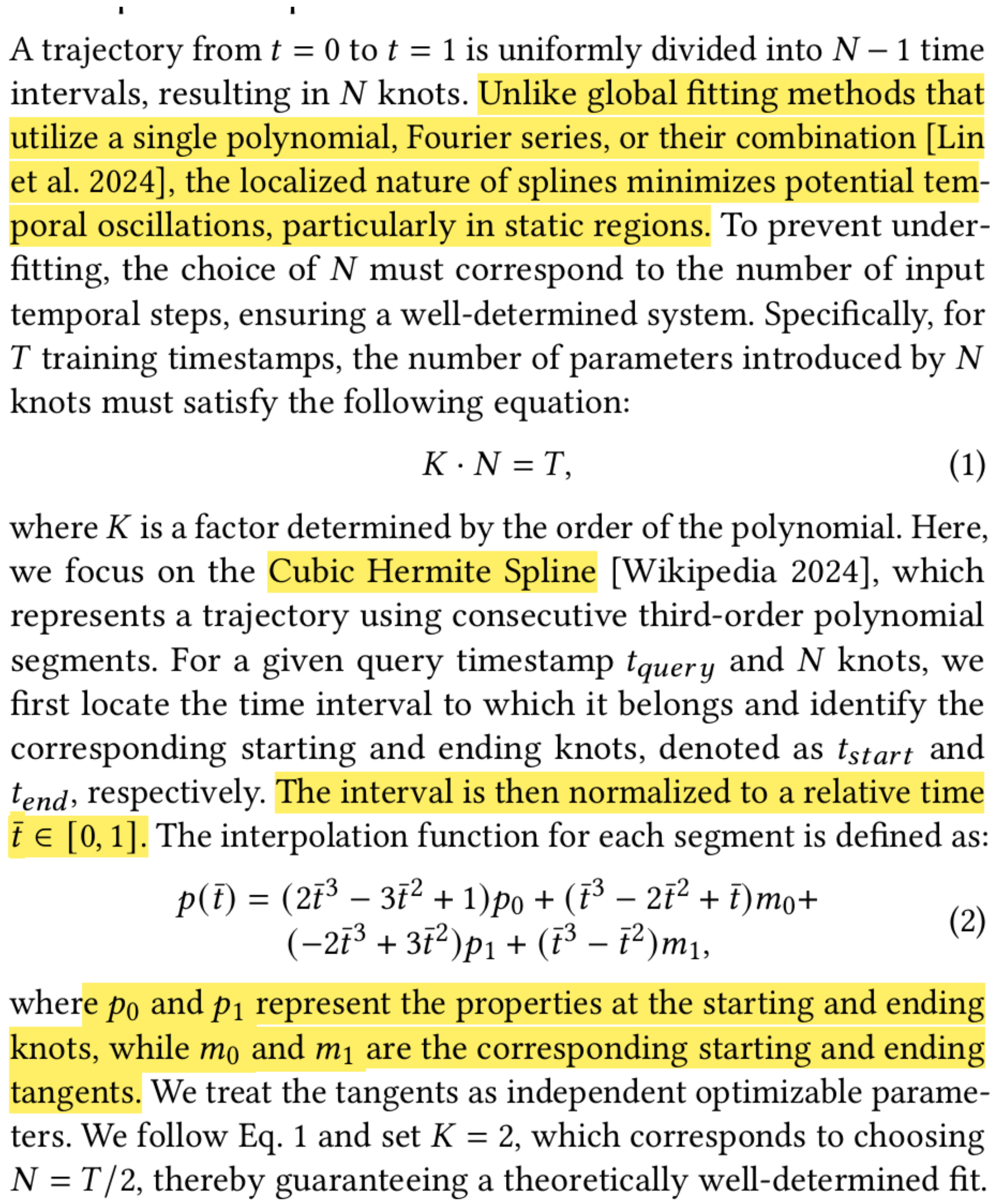
Pipeline



knot
temporal weights
$v \in \mathbb{R}^{rank}$
canonical position
$\mathbf{x}_i^c \in \mathbb{R}^3$
0
T
t_{start}
$\bar{t}$
t_{query}
t_{end}
uniform interval
Time-Variant *Spatial* Encoding
Time-Variant *Spatial* Encoding
$\mathbf{x}_i(t_{start}), \dot{\mathbf{x}}_i(t_{end})$
$\mathbf{x}_i(t_{left}), \dot{\mathbf{x}}_i(t_{left})$
Polynomial Interpolator
$\mathcal{F}(\bar{t}, \mathbf{x}_i(t_{start}), \dot{\mathbf{x}}_i(t_{start}), \mathbf{x}_i(t_{end}), \dot{\mathbf{x}}_i(t_{end}))$
$\frac{d\mathcal{F}}{dt}$
$\mathbf{x}_i(t_{query})$
$\dot{\mathbf{x}}_i(t_{query})$

Fig. 2. The pipeline of our method. We explain how to query value at arbitrary timestamps through polynomial interpolation in Sec. 2.1, corresponding to the timelines in the left part and orange block. We describe our design of spatiotemporal conditioning in Sec. 2.2, which corresponds to the red (temporal signal) and blue (spatial signal) arrows.

Pipeline - Spline Interpolation



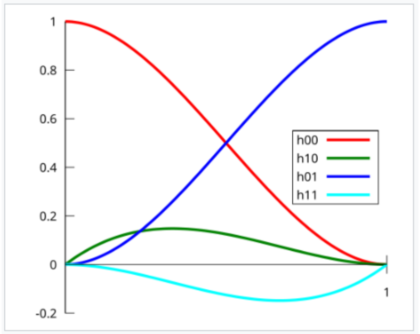
A trajectory from $t = 0$ to $t = 1$ is uniformly divided into $N - 1$ time intervals, resulting in N knots. Unlike global fitting methods that utilize a single polynomial, Fourier series, or their combination [Lin et al. 2024], the localized nature of splines minimizes potential temporal oscillations, particularly in static regions. To prevent underfitting, the choice of N must correspond to the number of input temporal steps, ensuring a well-determined system. Specifically, for T training timestamps, the number of parameters introduced by N knots must satisfy the following equation:

$$K \cdot N = T, \tag{1}$$

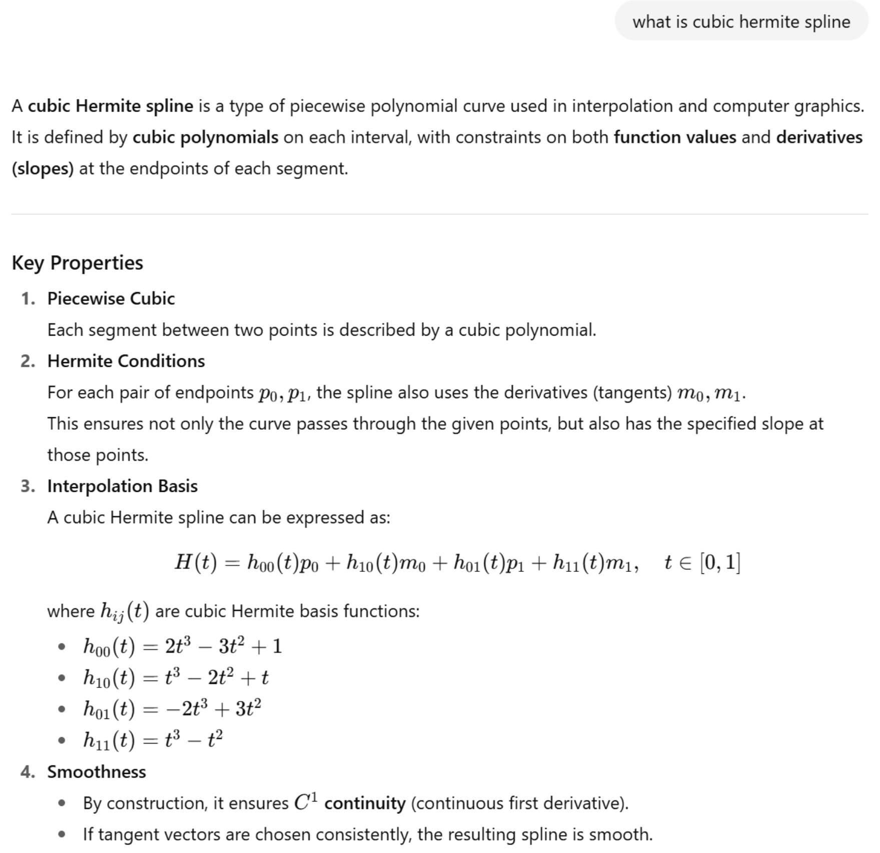
where K is a factor determined by the order of the polynomial. Here, we focus on the Cubic Hermite Spline [Wikipedia 2024], which represents a trajectory using consecutive third-order polynomial segments. For a given query timestamp t_{query} and N knots, we first locate the time interval to which it belongs and identify the corresponding starting and ending knots, denoted as t_{start} and t_{end}, respectively. The interval is then normalized to a relative time $\bar{t} \in [0, 1]$. The interpolation function for each segment is defined as:

$$\begin{aligned} p(\bar{t}) = (2\bar{t}^3 - 3\bar{t}^2 + 1)p_0 + (\bar{t}^3 - 2\bar{t}^2 + \bar{t})m_0 + \\ (-2\bar{t}^3 + 3\bar{t}^2)p_1 + (\bar{t}^3 - \bar{t}^2)m_1, \end{aligned} \tag{2}$$

where p_0 and p_1 represent the properties at the starting and ending knots, while m_0 and m_1 are the corresponding starting and ending tangents. We treat the tangents as independent optimizable parameters. We follow Eq. 1 and set $K = 2$, which corresponds to choosing $N = T/2$, thereby guaranteeing a theoretically well-determined fit.



The four Hermite basis functions. The interpolant in each subinterval is a linear combination of these four functions.



what is cubic hermite spline

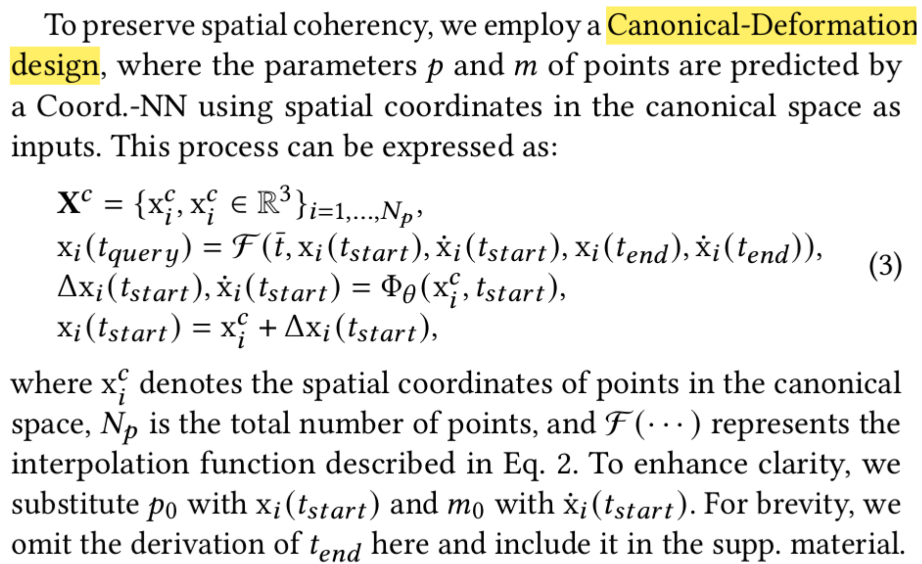
A **cubic Hermite spline** is a type of piecewise polynomial curve used in interpolation and computer graphics. It is defined by **cubic polynomials** on each interval, with constraints on both **function values** and **derivatives (slopes)** at the endpoints of each segment.

Key Properties

1. **Piecewise Cubic**
   Each segment between two points is described by a cubic polynomial.
2. **Hermite Conditions**
   For each pair of endpoints p_0, p_1, the spline also uses the derivatives (tangents) m_0, m_1.
   This ensures not only the curve passes through the given points, but also has the specified slope at those points.
3. **Interpolation Basis**
   A cubic Hermite spline can be expressed as:

$$H(t) = h_{00}(t)p_0 + h_{10}(t)m_0 + h_{01}(t)p_1 + h_{11}(t)m_1, \quad t \in [0,1]$$

   where $h_{ij}(t)$ are cubic Hermite basis functions:
   - $h_{00}(t) = 2t^3 - 3t^2 + 1$
   - $h_{10}(t) = t^3 - 2t^2 + t$
   - $h_{01}(t) = -2t^3 + 3t^2$
   - $h_{11}(t) = t^3 - t^2$
4. **Smoothness**
   - By construction, it ensures C^1 **continuity** (continuous first derivative).
   - If tangent vectors are chosen consistently, the resulting spline is smooth.



To preserve spatial coherency, we employ a Canonical-Deformation design, where the parameters p and m of points are predicted by a Coord.-NN using spatial coordinates in the canonical space as inputs. This process can be expressed as:

$$\begin{aligned} &\mathbf{X}^c = \{\mathrm{x}_i^c, \mathrm{x}_i^c \in \mathbb{R}^3\}_{i=1,\dots,N_p}, \\ &\mathrm{x}_i(t_{query}) = \mathcal{F}(\bar{t}, \mathrm{x}_i(t_{start}), \dot{\mathrm{x}}_i(t_{start}), \mathrm{x}_i(t_{end}), \dot{\mathrm{x}}_i(t_{end})), \\ &\Delta\mathrm{x}_i(t_{start}), \dot{\mathrm{x}}_i(t_{start}) = \Phi_\theta(\mathrm{x}_i^c, t_{start}), \\ &\mathrm{x}_i(t_{start}) = \mathrm{x}_i^c + \Delta\mathrm{x}_i(t_{start}), \end{aligned} \tag{3}$$

where x_i^c denotes the spatial coordinates of points in the canonical space, N_p is the total number of points, and $\mathcal{F}(\cdots)$ represents the interpolation function described in Eq. 2. To enhance clarity, we substitute p_0 with $\mathrm{x}_i(t_{start})$ and m_0 with $\dot{\mathrm{x}}_i(t_{start})$. For brevity, we omit the derivation of t_{end} here and include it in the supp. material.

Pipeline - Time-variant Spatial Encoding (TVSE)

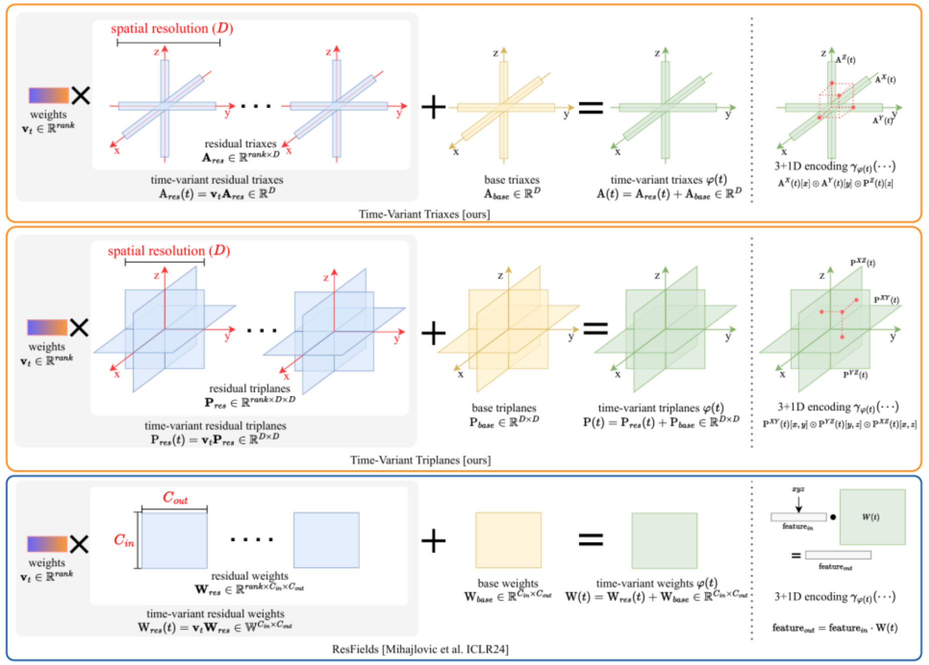

spatial resolution (D)
weights $\mathbf{v}_t \in \mathbb{R}^{rank}$
residual triaxes $\mathbf{A}_{res} \in \mathbb{R}^{rank \times D}$
time-variant residual triaxes $\mathbf{A}_{res}(t) = \mathbf{v}_t \mathbf{A}_{res} \in \mathbb{R}^D$
base triaxes $\mathbf{A}_{base} \in \mathbb{R}^D$
time-variant triaxes $\varphi(t)$ $\mathbf{A}(t) = \mathbf{A}_{res}(t) + \mathbf{A}_{base} \in \mathbb{R}^D$
3+1D encoding $\gamma_{\varphi(t)}(\cdots)$
$\mathbf{A}^X(t)[x] \odot \mathbf{A}^Y(t)[y] \odot \mathbf{P}^Z(t)[z]$
$\mathbf{A}^Z(t)$ $\mathbf{A}^X(t)$ $\mathbf{A}^Y(t)$
Time-Variant Triaxes [ours]
spatial resolution (D)
weights $\mathbf{v}_t \in \mathbb{R}^{rank}$
residual triplanes $\mathbf{P}_{res} \in \mathbb{R}^{rank \times D \times D}$
time-variant residual triplanes $\mathbf{P}_{res}(t) = \mathbf{v}_t \mathbf{P}_{res} \in \mathbb{R}^{D \times D}$
base triplanes $\mathbf{P}_{base} \in \mathbb{R}^{D \times D}$
time-variant triplanes $\varphi(t)$ $\mathbf{P}(t) = \mathbf{P}_{res}(t) + \mathbf{P}_{base} \in \mathbb{R}^{D \times D}$
$\mathbf{P}^{XZ}(t)$ $\mathbf{P}^{XY}(t)$ $\mathbf{P}^{YZ}(t)$
3+1D encoding $\gamma_{\varphi(t)}(\cdots)$
$\mathbf{P}^{XY}(t)[x,y] \odot \mathbf{P}^{YZ}(t)[y,z] \odot \mathbf{P}^{XZ}(t)[x,z]$
Time-Variant Triplanes [ours]
C_{out}
C_{in}
weights $\mathbf{v}_t \in \mathbb{R}^{rank}$
residual weights $\mathbf{W}_{res} \in \mathbb{R}^{rank \times C_{in} \times C_{out}}$
time-variant residual weights $\mathbf{W}_{res}(t) = \mathbf{v}_t \mathbf{W}_{res} \in \mathbb{W}^{C_{in} \times C_{out}}$
base weights $\mathbf{W}_{base} \in \mathbb{R}^{C_{in} \times C_{out}}$
time-variant weights $\varphi(t)$ $\mathbf{W}(t) = \mathbf{W}_{res}(t) + \mathbf{W}_{base} \in \mathbb{R}^{C_{in} \times C_{out}}$
xyz
feature$_{in}$
$W(t)$
feature$_{out}$
3+1D encoding $\gamma_{\varphi(t)}(\cdots)$
feature$_{out}$ = feature$_{in} \cdot \mathbf{W}(t)$
ResFields [Mihajlovic et al. ICLR24]

Fig. 3. An intuitive diagram of our TVSE. Rounded boxes with different edge colors mark grid-based and MLP and methods. $A^{\cdot}(t)[\cdot]$ and $P^{\cdot\cdot}(t)[\cdot,\cdot]$ denote sampling values at specific position from an axis and plane, respectively. Following [Fridovich-Keil et al. 2023], we aggregate features sampled from the three axes and planes using element-wise multiplication $\odot$.

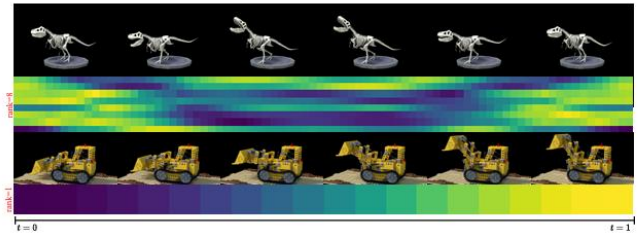


Fig. 11. Visualization of $\mathbf{v} \in \mathbb{R}^{rank}$ in Eq. 4. We concatenate the temporal weights of SF-Triplanes along the temporal dimension, i.e., each **column** in the heat map is a vector representing a deformed state. Not surprisingly, scene *lego* can be reconstructed with $rank = 1$.

Previous works [Fridovich-Keil et al. 2023; Huang et al. 2023; Pumarola et al. 2021; Wu et al. 2023; Xu et al. 2024; Yang et al. 2023] treat four-dimensional inputs (three spatial coordinates and one timestamp) equally. Recently, ResFields [Mihajlovic et al. 2024] introduced a novel approach by injecting temporal information into the weights of pure MLP-based spatial encodings through time-variant residual and base weights, enhancing the performance of various previous methods. Its architecture-agnostic design naturally extends to other encoding schemes. The key conceptual difference can expressed as:

$$\begin{aligned} &\text{Previous work: } \Phi_\theta(\gamma(x), \gamma(y), \gamma(z), \gamma(t)), \\ &\text{Ours: } \Phi_\theta(\gamma_{\varphi(t)}(x), \gamma_{\varphi(t)}(y), \gamma_{\varphi(t)}(z)), \end{aligned} \tag{4}$$

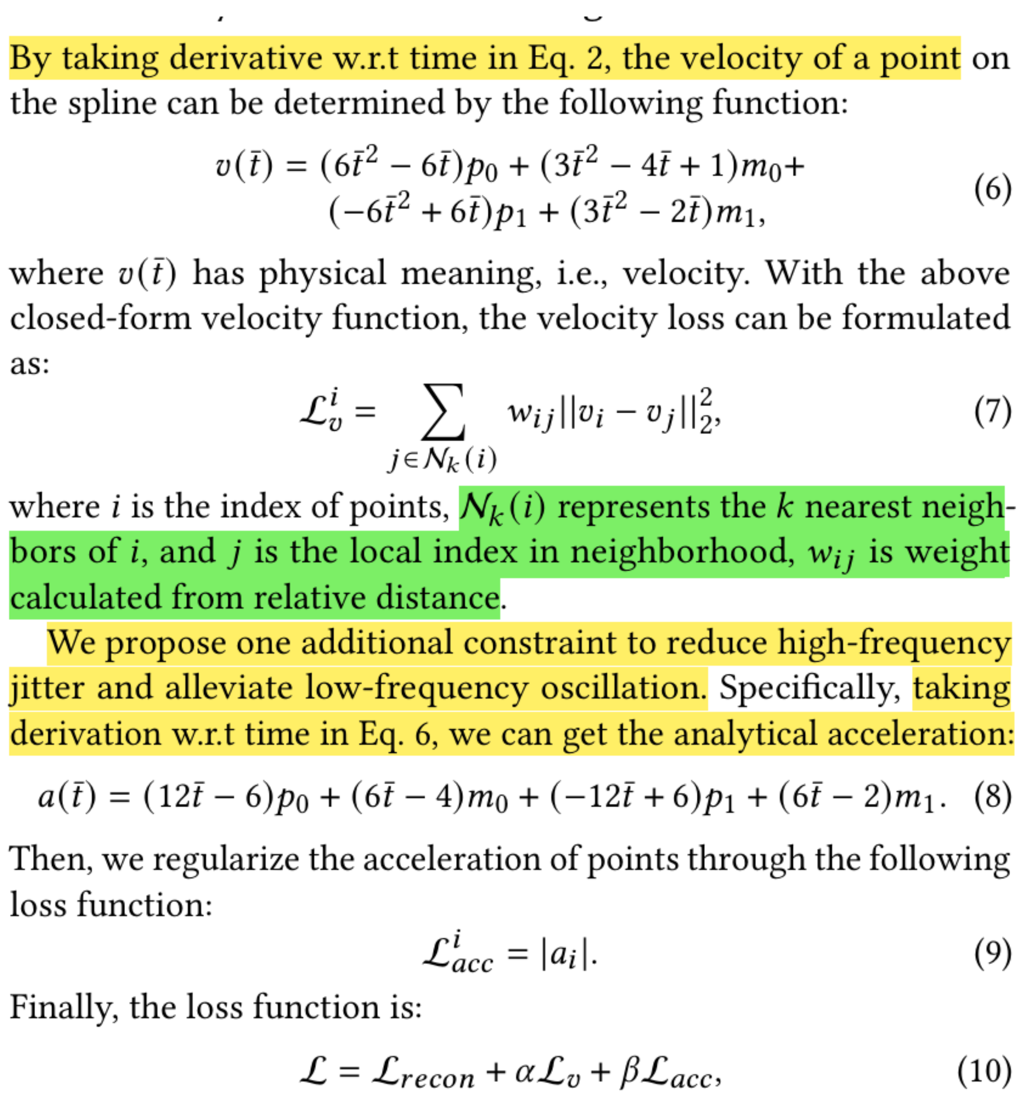
where $\gamma(\cdot)$ represents the encoding (e.g., Sinusoidal [Mildenhall et al. 2021] or grids [Cao and Johnson 2023; Müller et al. 2022]), $\gamma_{\varphi(t)}(\cdot)$ denotes time-variant encoding, and $\varphi(t)$ is the temporal signal injection function, which varies based on the specific design of $\gamma(\cdot)$.

Building on the concept of ResFields [Mihajlovic et al. 2024], we incorporate low-rank decomposition in the temporal domain to achieve both compactness and implicit regularization. This approach can be expressed as:

$$\phi(t) = \mathrm{b}_{base} + \sum_{r=1}^{rank} \mathrm{v}_t[r] \cdot \mathrm{B}_{res}[r], \tag{5}$$

where $[r]$ denotes element indexing, $\mathbf{v}_t \in \mathbb{R}^{rank}$ represents trainable weights associated with each timestep, $\mathbf{B}_{res} \in \mathbb{R}^{rank \times \cdots}$ are the residual bases, and b_{base} corresponds to the time-invariant encoding. These notations carry different meanings depending on the chosen encoding scheme. In this work, we primarily focus on cosine functions and multi-resolution dense grids. Fig. 3 provides an intuitive visualization of the design of $\gamma_{\varphi(t)}$. We refer to the supplementary material for detailed implementations.

Pipeline – Velocity and Acceleration Regularization



By taking derivative w.r.t time in Eq. 2, the velocity of a point on the spline can be determined by the following function:

$$v(\bar{t}) = (6\bar{t}^2 - 6\bar{t})p_0 + (3\bar{t}^2 - 4\bar{t} + 1)m_0 + (-6\bar{t}^2 + 6\bar{t})p_1 + (3\bar{t}^2 - 2\bar{t})m_1, \quad (6)$$

where $v(\bar{t})$ has physical meaning, i.e., velocity. With the above closed-form velocity function, the velocity loss can be formulated as:

$$\mathcal{L}_v^i = \sum_{j \in \mathcal{N}_k(i)} w_{ij} ||v_i - v_j||_2^2, \quad (7)$$

where i is the index of points, $\mathcal{N}_k(i)$ represents the k nearest neighbors of i, and j is the local index in neighborhood, w_{ij} is weight calculated from relative distance.

We propose one additional constraint to reduce high-frequency jitter and alleviate low-frequency oscillation. Specifically, taking derivation w.r.t time in Eq. 6, we can get the analytical acceleration:

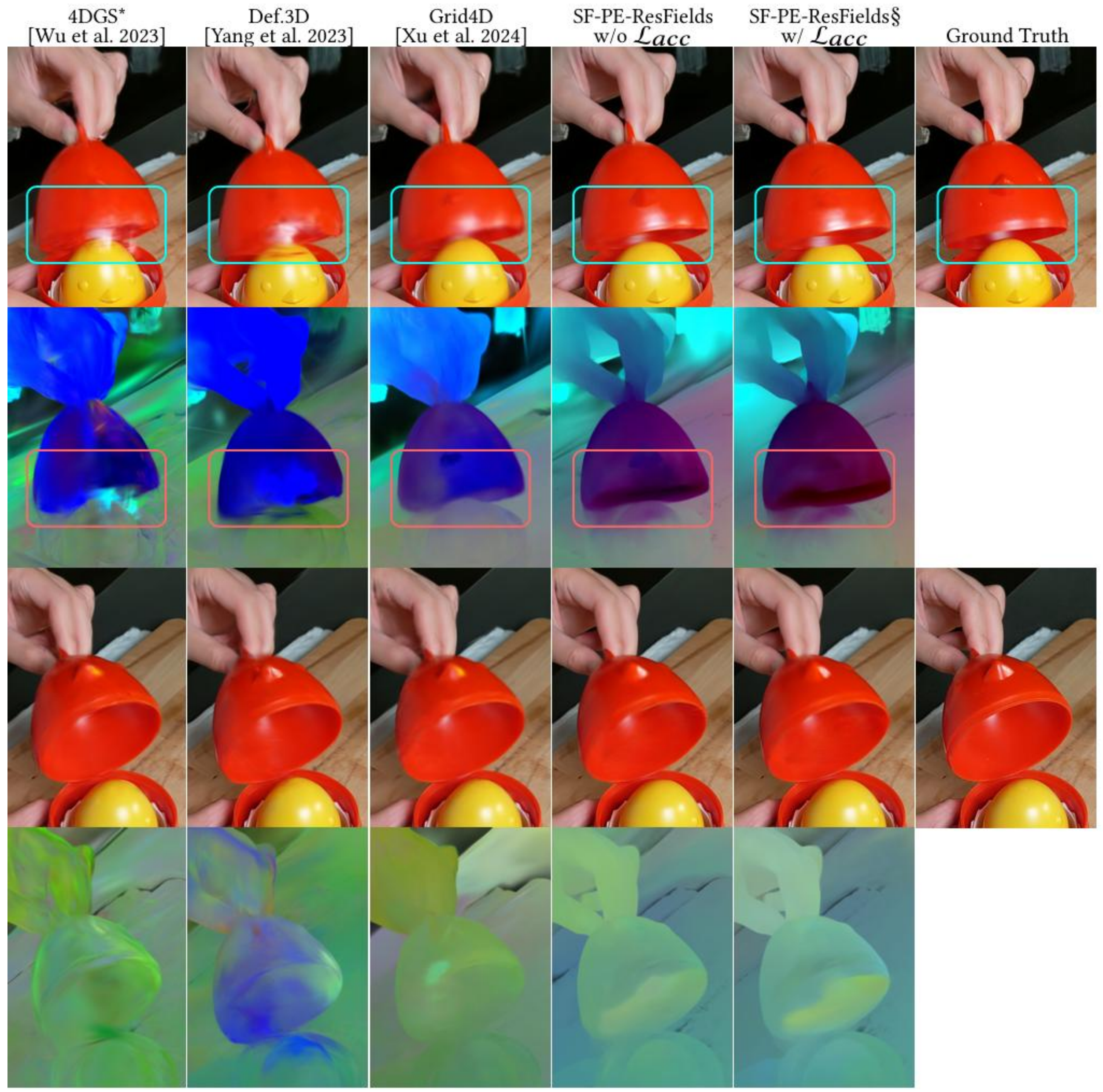
$$a(\bar{t}) = (12\bar{t} - 6)p_0 + (6\bar{t} - 4)m_0 + (-12\bar{t} + 6)p_1 + (6\bar{t} - 2)m_1. \quad (8)$$

Then, we regularize the acceleration of points through the following loss function:

$$\mathcal{L}_{acc}^i = |a_i|. \quad (9)$$

Finally, the loss function is:

$$\mathcal{L} = \mathcal{L}_{recon} + \alpha\mathcal{L}_v + \beta\mathcal{L}_{acc}, \quad (10)$$


4DGS* [Wu et al. 2023]
Def.3D [Yang et al. 2023]
Grid4D [Xu et al. 2024]
SF-PE-ResFields w/o $\mathcal{L}_{acc}$
SF-PE-ResFields§ w/ $\mathcal{L}_{acc}$
Ground Truth

3.1 Learning Continuous Deformation Field

As a proof of concept, we start with a more constrained problem. Given sparsely sampled point trajectories, the goal is to fit an implicit continuous deformation field that can infer the trajectories of unseen points, ensuring alignment with observed data. We focus on a more challenging yet practical scenario where available temporal signals are scarce to demonstrate the strength of explicit spline interpolation. Specifically, we choose nine long sequences (each exceeding 500 frames) from DeformingThings4D (humanoids) [Li et al. 2021] and train models with every 4th/6th timestep, leaving 75%/83.3% timesteps of timesteps for evaluation. Following DOMA [Zhang et al. 2024], 25% vertices from the starting mesh are sampled to calculate L1 distance for supervision during training. The determination of N follows Eq. 1. In addition to end point error (EPE), we propose a new metric for evaluating spatial coherence using Moran's I [Mihajlovic et al. 2025; Moran 1950]. We provide its detailed implementation in the supplementary material.

Table 1. Interpolation of scene flow on DeformaingTings4D's long sequences. EPE results are multiplied by $\times 10^4$ for better readability. The highlighted rows denote 1st, 2nd, and 3rd best models. Note that when c=90%, our methods still produce plausible interpolations (shown in Fig. 10).

Method	Settings		x4		x6	
			EPE↓	M.'s I↑	EPE↓	M.'s I↑
DOMA [Zhang et al. 2024]		#p=0.03M	138.80	0.912	146.51	0.911
		#p=0.1M	92.59	0.876	105.28	0.877
		#p=1.6M	473.17	0.249	452.23	0.254
		#p=2.1M	927.26	0.115	906.67	0.118
ResFields [Mihajlovic et al. 2024]	c=100%	r=30	52.09	0.899	80.74	0.905
		r=60	54.23	0.891	80.63	0.898
		r=90	55.01	0.882	84.19	0.885
	c=90%	r=60	52.19	0.894	81.60	0.898
	c=75%		50.75	0.896	80.56	0.898
	c=50% †		45.79	0.896	74.35	0.894
	c=25%		48.60	0.894	74.92	0.889
	† w/ AIAP		45.72	0.927	73.00	0.923
SF-Siren-ResFields	c=100%	r=30	42.98	0.917	68.73	0.927
		r=60‡	40.74	0.919	68.28	0.926
		r=90	41.67	0.917	69.38	0.926
	‡ w/o $\mathcal{L}_v$		42.15	0.916	68.40	0.924
	‡ w/o $\mathcal{L}_{acc}$		42.66	0.917	70.12	0.929
	‡ w/ AIAP		41.64	0.938	68.93	0.943
	c=90%	r=60	45.97	0.921	80.22	0.927

on setting †). For example, c=50% indicates that the number of temporal weights, $v \in \mathbb{R}^{rank}$, is reduced to half the number of training timesteps, thereby decreasing the temporal DoF. However, motions

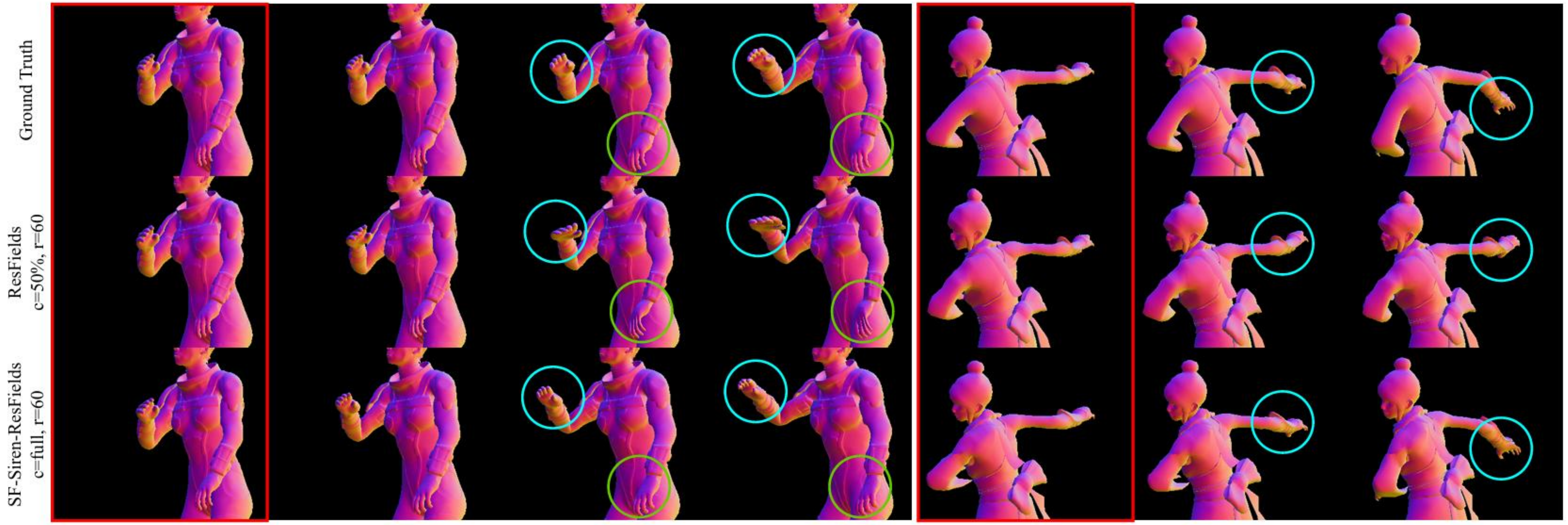


Fig. 4. Qualitative comparison of ×6 (left) and ×4 (right) scene flow interpolation. Frames boxed in red are training timesteps, and the following ones interpolated. Our method achieves fair interpolated motions without skinning. We provide more visual results in the extra pages.

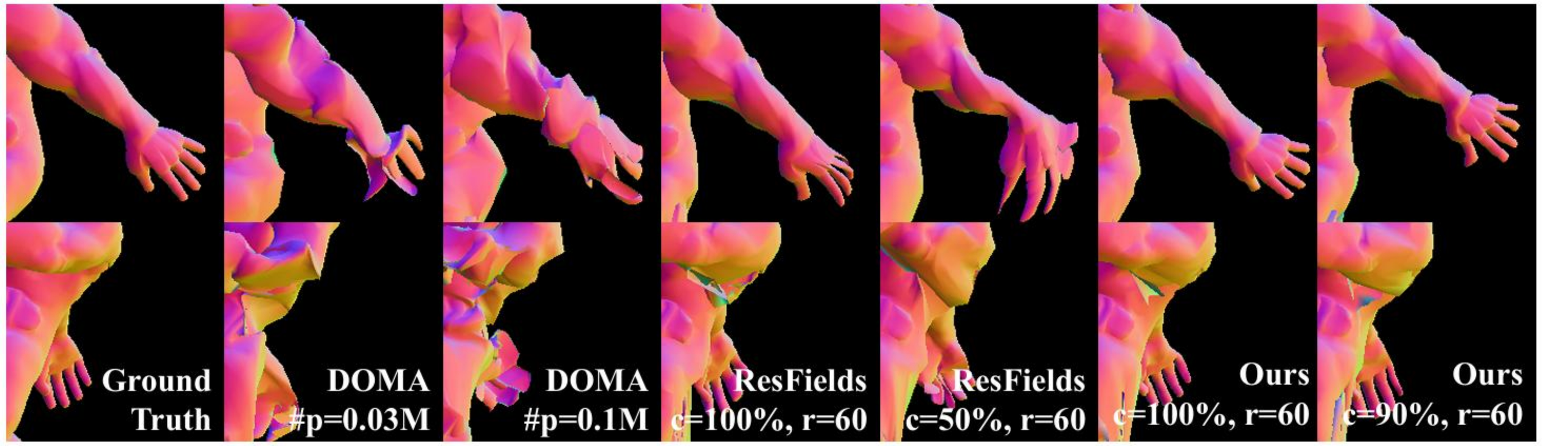


Fig. 10. We present additional scene flow interpolation results, a task with significant practical value yet remains underexplored. Our model demonstrates strong performance and can serve as a baseline for future research.

Table 2. Quantitative results on NeRF-DS [Yan et al. 2023] dataset. Notably, traditional metrics (PSNR and SSIM) penalize sharper but misaligned results over blurry results. We introduce perturbation in Grid4D [Xu et al. 2024] to 4DGS [Wu et al. 2023] since it consistently improves grid-based methods, which we denote with *. SC-GS's [Huang et al. 2023] metrics except for M.'s I are adopted from the official script.

Method	As				Basin				Bell				Cup			
	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑
Def.3D [Yang et al. 2023]	26.01	87.61	12.33	0.685	19.51	77.97	13.43	0.735	25.23	83.94	12.07	0.676	24.53	88.11	11.47	0.719
SC-GS [Huang et al. 2023]	26.00	-	11.40	0.924	19.60	-	15.40	0.919	25.10	-	11.70	0.946	24.50	-	11.50	0.955
4DGS* [Wu et al. 2023]	25.60	86.27	13.36	0.780	18.90	76.97	15.43	0.775	24.60	82.66	14.05	0.758	24.42	88.23	11.55	0.787
4DGS [Wu et al. 2023]	25.69	86.43	13.60	0.845	18.88	75.92	17.59	0.723	24.46	82.30	13.87	0.732	24.44	88.04	12.02	0.739
SF-Triaxes	26.89	88.46	13.40	0.865	19.45	79.29	16.04	0.936	26.05	86.06	12.58	0.920	24.24	88.53	11.90	0.925
SF-Triplanes	26.34	88.75	12.43	0.891	19.61	79.85	14.67	0.937	25.68	85.27	13.01	0.928	24.40	88.60	11.33	0.932
SF-PE-ResFields	26.82	88.90	12.01	0.932	19.70	79.18	13.67	0.959	25.55	84.50	12.41	0.972	24.85	88.92	11.21	0.962

Method	Plate				Press				Sieve				Average			
	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑	PSNR↑	SSIM↑	LPIPS↓	M.'s I↑
Def.3D [Yang et al. 2023]	20.33	80.26	19.12	0.702	25.40	85.61	13.66	0.720	25.24	86.72	10.85	0.703	23.75	84.32	13.28	0.706
SC-GS [Huang et al. 2023]	20.20	-	20.20	0.938	26.60	-	13.50	0.945	26.00	-	11.40	0.927	24.10	-	14.00	0.939
4DGS* [Wu et al. 2023]	20.02	79.81	20.19	0.869	25.61	85.03	14.41	0.744	25.76	87.47	11.51	0.801	23.56	83.78	14.36	0.788
4DGS [Wu et al. 2023]	19.32	78.37	21.29	0.846	25.04	84.35	15.82	0.682	24.94	87.01	11.83	0.765	23.25	83.20	15.15	0.762
SF-Triaxes	20.92	81.74	18.63	0.986	26.22	88.21	14.09	0.908	26.71	87.68	11.47	0.943	24.36	85.71	14.01	0.926
SF-Triplanes	21.17	82.57	18.14	0.982	26.49	88.29	13.10	0.917	26.57	87.78	11.24	0.927	24.32	85.87	13.42	0.931
SF-PE-ResFields	21.02	82.51	17.41	0.993	27.09	88.33	12.29	0.967	25.80	88.32	10.96	0.965	24.40	85.81	12.85	0.964

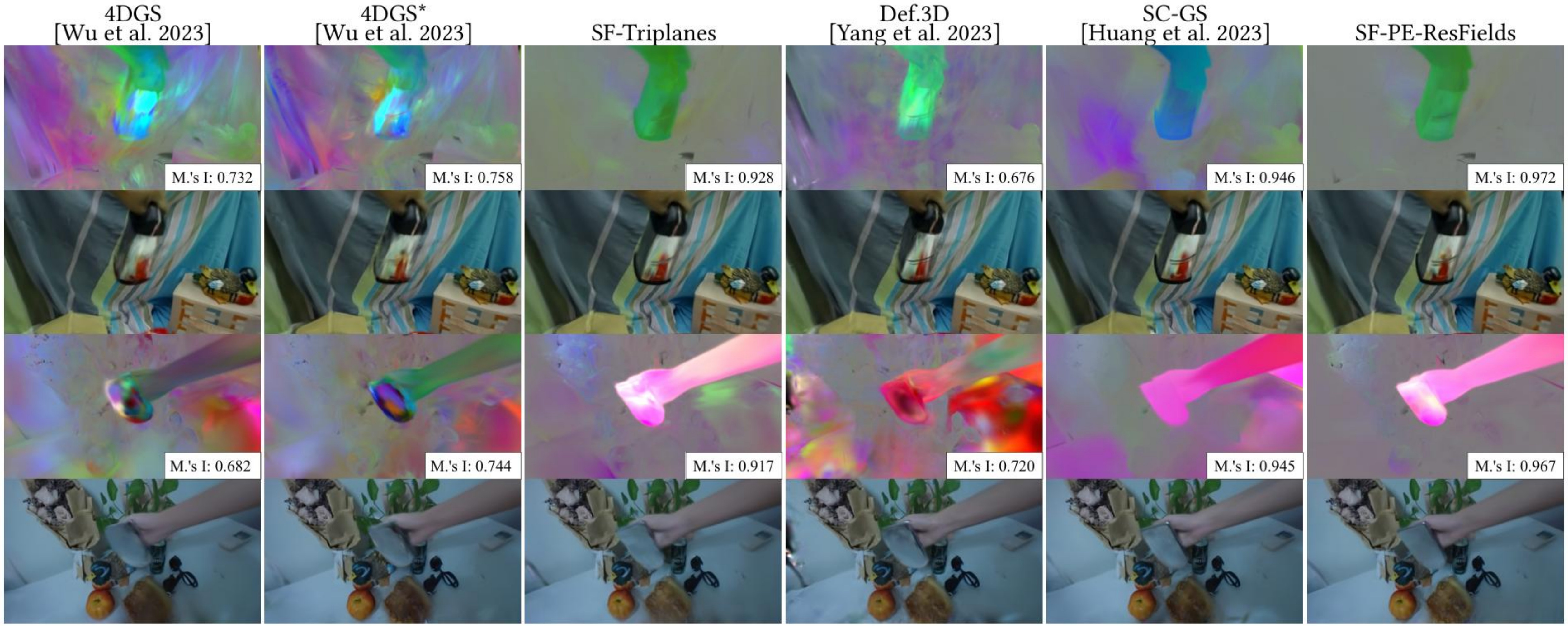


Fig. 5. Qualitative comparison of scene flows on NeRF-DS [Yan et al. 2023] dataset. With the derived regularizations in Sec. 2.3, our representation effectively alleviates the high frequency caused by imprecise camera positions. We normalize all frames' motion vectors or velocities, so the gray color indicates static regions. We encourage readers to focus on the smoothness of motions.

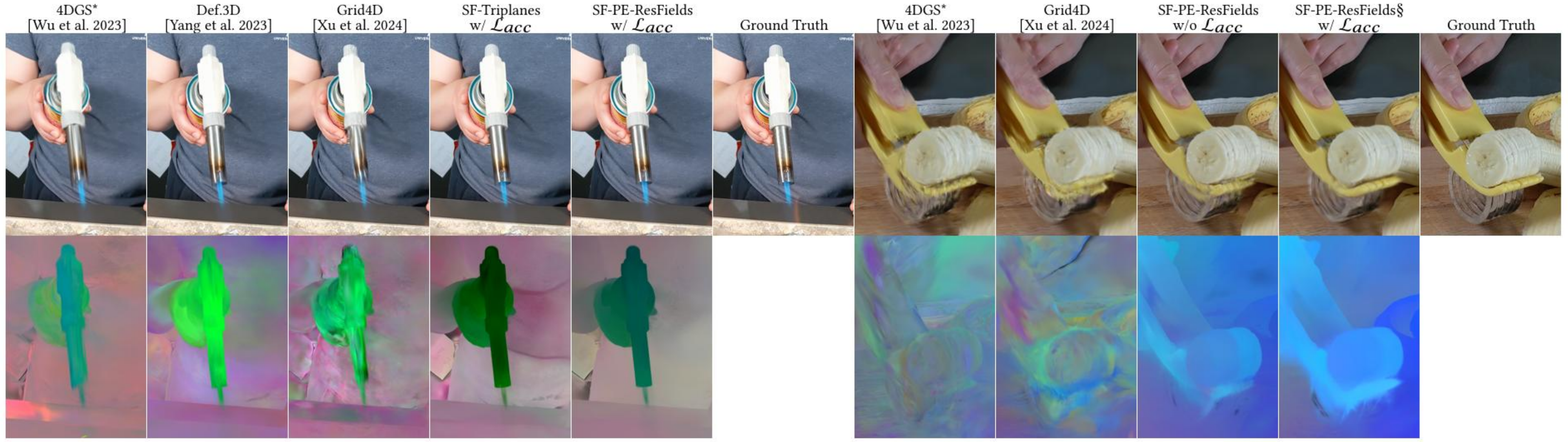

4DGS* [Wu et al. 2023]
Def.3D [Yang et al. 2023]
Grid4D [Xu et al. 2024]
SF-Triplanes w/ $\mathcal{L}_{acc}$
SF-PE-ResFields w/ $\mathcal{L}_{acc}$
Ground Truth
4DGS* [Wu et al. 2023]
Grid4D [Xu et al. 2024]
SF-PE-ResFields w/o $\mathcal{L}_{acc}$
SF-PE-ResFields§ w/ $\mathcal{L}_{acc}$
Ground Truth

Fig. 6. Qualitative comparison of rendered images and scene flows on Hyper-NeRF [Park et al. 2021] dataset. Since the whole scene is jittery due to inaccurate cameras, static regions are also colored, compared with the NeRF-DS[Yan et al. 2023] dataset. Additional results are included in Fig. 9.

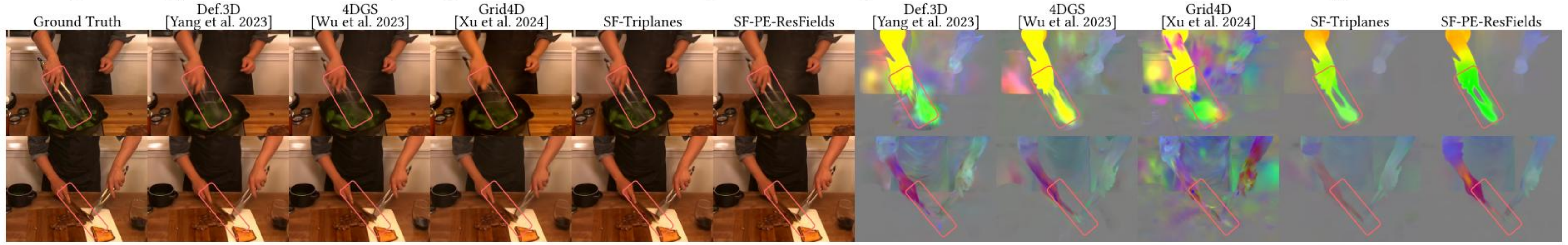


Fig. 7. Qualitative comparison of rendered images and scene flows on Neu3D [Li et al. 2022] dataset. Our velocity regularizations preserve the handle with reflection batter than the baselines. The differences are more evident in rendered motion vectors when photometric reconstructions are mixed.

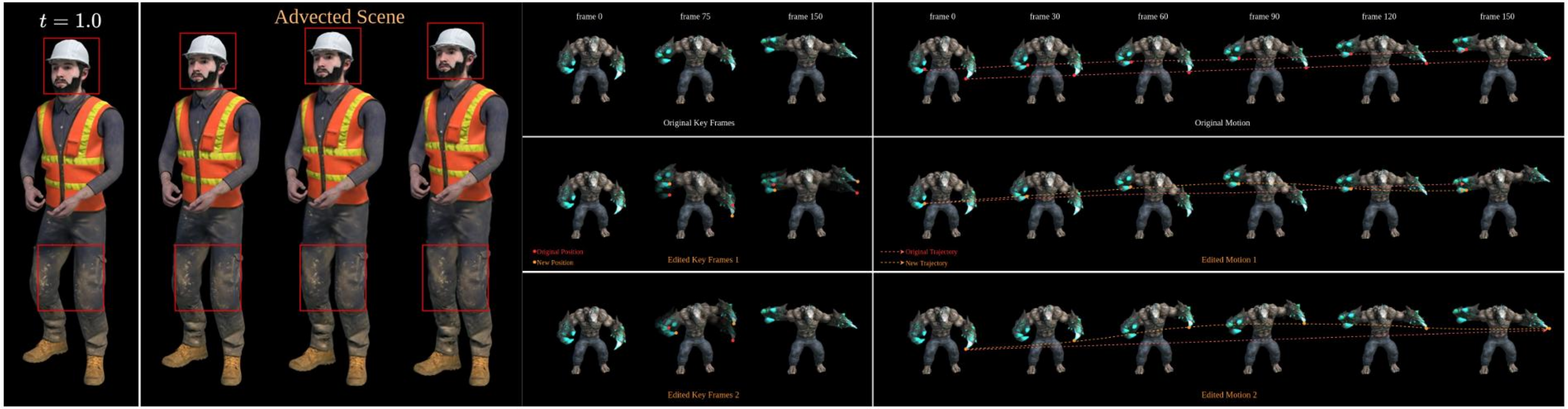


Fig. 8. Examples of motion advection and editing. **Left**: We propagate points in the last frame with the derived velocity. The plausible visual results showcase the effectiveness of $\mathcal{L}_v$. **Right**: We adjust motions by only editing keyframes. In this work, we can only neatly achieve trifling modifications since the definition of '*motion* editing' is still ambiguous, and LBS-based deformation is tedious for users. We leave a clearer description of such a task and a streamlined editing pipeline as future work (e.g., cage-based deformation).