ActionMesh: Animated 3D Mesh Generation with Temporal 3D Diffusion
Supplementary Material

In this supplementary document, we present additional results (Section A), more ablation studies (Section B) and details of implementations (Section C). We encourage readers to consult the video results provided on our webpage: **https://remysabathier.github.io/actionmesh/**.

A. Additional results

Consistent4D comparison. We include additional qualitative comparisons on the standard Consistent4D [15] evaluation set in Fig. 9. Our method produces reconstructions with sharper geometry, fewer temporal artifacts, and more faithful motion than competing approaches. We release videos for all Consistent4D scenes and all baselines, rendered from three viewpoints each, on the supplementary website under the section *Application 1: video-to-4D*.

DAVIS results. We add more qualitative results on real-world videos from DAVIS [29] in Fig. 10. Despite being trained exclusively on synthetic videos, our approach generalizes well to in-the-wild footage, recovering sharp geometry and plausible motion. Videos for all DAVIS examples are provided on the supplementary website under the section *Application 1: video-to-4D*.

Long animation generation. In the standard setting, our model is trained to generate animated sequences of 16 frames. Thanks to its autoregressive modeling, we can apply our model to longer videos by recursively rolling out predictions: the last 3D output of one inference is fed back as conditioning for the next one. In the supplementary website, we showcase video-to-4D reconstructions of sequences with 61 frames, obtained from a single standard 16-frame inference pass followed by three additional autoregressive passes. These examples illustrate that our model can maintain coherent geometry and motion over significantly extended time horizons beyond its training sequence length. Videos are listed under the section *Application 1: video-to-4D*.

Video results. We provide on our supplementary website a comprehensive set of videos illustrating the five main applications of our model. *(i) Video-to-4D:* we show reconstructions on the Consistent4D [15] evaluation set and on real-world DAVIS [29] videos (see paragraphs above). *(ii) {3D+text}-to-4D:* given a static 3D textured mesh and a motion description, we animate well-known benchmark meshes (the armadillo and the cow) as well as additional textured meshes produced by an external 3D generative model. Since our method outputs animated meshes with fixed topology, textures remain coherent and are consistently propagated throughout the entire sequence. *(iii) {Image+text}-to-4D:* starting from single images drawn from standard 3D evaluation sets used in TripoSG [18], Dora [5], and Trellis [48], we manually attach a short motion description to each image (*e.g.*, the astronaut "dancing", the mushroom "singing opera") and generate an animated 3D mesh out of them. *(iv) Text-to-4D:* we generate animated meshes directly from textual prompts describing both the object and its motion, such as "an octopus playing maracas". *(v) Motion transfer:* we include motion transfer videos with two input motions, each applied on two different meshes.

Mesh Animation. As discussed in Section 3.4, ActionMesh allows the integration of known 3D shapes into the generation process, enabling solutions to both the {3D+video}-to-animation and {3D+text}-to-animation problems. Several recent works have focused specifically on this task [14, 35, 47]. Among these, AnimateAnyMesh [47] stands out as a text-driven, feed-forward mesh animation model capable of generating animations in approximately 6 seconds on a single NVIDIA H100 GPU. We present a qualitative comparison between ActionMesh and AnimateAnyMesh in Figure 8. While ActionMesh is notably slower, it produces animations with significantly greater motion expressiveness. For example, in the "panda doing martial art" scenario, ActionMesh reconstructs a coordinated motion involving both the upper and lower body, whereas AnimateAnyMesh primarily activates only the upper body. Similarly, for the "armadillo casually walking" prompt, AnimateAnyMesh fails to animate the subject's legs, while ActionMesh generates a more natural, full-body motion. The superior expressiveness of ActionMesh stems from its reliance on the strong motion priors of video diffusion models, rather than being constrained by a limited dataset of animated objects. As described in Section 3.4, our approach animates a mesh from a text prompt by first querying an off-the-shelf video generator, then applying ActionMesh to the known 3D shape and the generated video. In contrast, AnimateAnyMesh learns its motion prior from a dataset of approximately 66k animations—primarily from animated objects in Objaverse [7, 8]—which inherently limits the diversity and expressiveness of the generated motions.

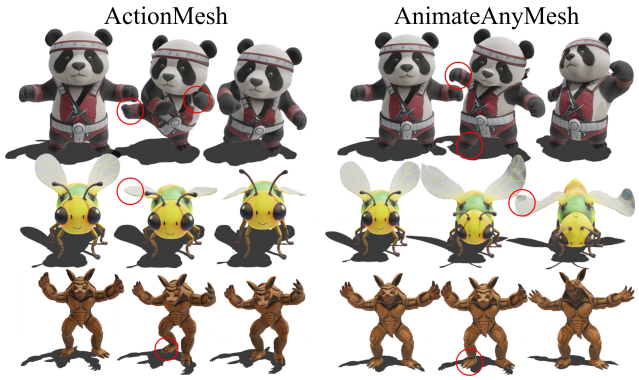

Figure 8. **Comparison to AnimateAnyMesh [47].** Panda "doing martial art", Firefly "flapping its wings", Armadillo "casually walking". The Armadillo mesh is sourced from the Stanford Computer Graphics Laboratory.

Table 3. **Ablation study - Temporal 3D denoiser.**

Ablation type	CD-3D↓	CD-4D↓
Full model	**0.050**	**0.069**
w/o rotary embedding	0.054	0.084
w/o masked modeling	0.062	0.116

Table 4. **Ablation study - Temporal 3D autoencoder.**

Ablation type	CD-M↓
Full model	**0.137**
w/o normals	0.148
w/o $\{t_{\text{src}}, t_{\text{tgt}}\}$ in self-attentions	0.151

B. Additional ablation studies

Temporal 3D diffusion (stage I). We report additional ablations of the temporal 3D diffusion model in Tab. 3. First, we study the importance of injecting relative frame information inside the inflated self-attention layers. In our default setting, we add rotary positional embedding [39] of the relative frame index in the inflated self-attention layers. We train a variant where inflated attention is kept but all temporal rotary embeddings are removed. On our Objaverse evaluation set, this leads to a clear degradation of both CD-3D and CD-4D, confirming that explicit temporal encoding is critical for stable, temporally coherent 4D generation. Second, we ablate the masked generation mechanism. In the default configuration, for video-to-4D we compute a reference mesh latent $\mathbf{z}_1^*$ from an off-the-shelf image-to-3D model and inject it as a clean source latent while generating the remaining (masked) latents. We train a model where this mechanism is disabled and the diffusion model is conditioned only on video frames, with no noise-free latents. This variant cannot support several applications enabled by our masking strategy, including {3D+text}-to-4D, {image+text}-to-4D and autoregressive long-horizon generation. Beyond this loss of functionality, it also underperforms our full model on video-to-4D reconstruction metrics, indicating that leveraging a strong image-to-3D prior through masked modeling both broadens the application scope and improves geometric and motion fidelity.

Temporal 3D autoencoder (stage II). We report ablations of the temporal 3D autoencoder in Tab. 4. First, we study the impact of augmenting query 3D points with surface normals. In our default configuration, each query point is represented by its position and normal, a choice motivated by the need to disambiguate deformations of points that are spatially close but topologically distant on the surface. Removing normals and using positions yields a drop in performance. Second, we ablate how the source and target timesteps $(t_{\text{src}}, t_{\text{tgt}})$ that parameterize the deformation field are injected into the model. By default, we treat t_{src} and t_{tgt} as an additional token concatenated to the shape latents, so that they are jointly processed in the self-attention layers of the temporal 3D autoencoder. We compare this with a variant where $(t_{\text{src}}, t_{\text{tgt}})$ are instead appended as extra features to the query points (alongside position and normals). While this latter design has a practical advantage—enabling substantial speed-ups when predicting multiple deformation fields for the same shape tokens, since the post–self-attention context can be cached and reused—we observe a degradation in performance.

Autoregressive generation. We ablate our autoregressive design choices in Tabs. 5 and 6. In Table 5, we vary the number of keyframes N used to train the temporal 3D denoiser (stage I) and the temporal 3D autoencoder (stage II). In the default configuration, both stages operate on sequences of 16 keyframes. We additionally train variants of each stage with 4 or 8 keyframes, and evaluate some pairwise combinations of (stage I, stage II) configurations on our evaluation set of 16-frame sequences. For models trained with fewer than 16 keyframes, we reconstruct the sequence by splitting it into shorter chunks and we process them autoregressively. We observe that having a larger sequence length is particularly important in stage I: increasing the number of keyframes in the denoiser (autoencoder fixed) leads to a significant improvement across all metrics. In contrast, stage II is less sensitive to this choice. In Table 6, we study the design choices of our autoregressive regime. In the standard setting, we process long videos by splitting them into consecutive chunks; the first chunk is reconstructed as usual, and for each subsequent chunk we feed the last reconstructed 3D output as additional input (reference) along with the current video segment. In practice, both the temporal 3D denoiser and the temporal 3D autoencoder can accept one or more reference meshes as input, thus defining a *context window* c_{w} for our model. Increasing the number of context frames, however,

Figure 9. **Additional qualitative comparison on Consistent4D [15].**

Figure 10. **Additional qualitative results on real videos from DAVIS [29].**

implicitly increases the total number of inference steps, since fewer new frames are generated per pass. On sequences of 31 timesteps, we find that enlarging the context window has limited effect on reconstruction metrics, while incurring additional computational cost. We therefore set $c_w = 1$ in both stages in our final model to maximize efficiency.

Reference frame. We ablate the impact of the reference frame in Tab. 7. Specifically, we compare using the first (default), middle, or last frame as the reference for stage I and stage II. We observe that selecting either the middle or last frame as the reference leads to slightly worse quantitative results. We attribute this to the fact that the mesh is typically in a more favorable position for 3D reconstruction in the first frame, with fewer topological ambiguities and less motion blur in the reference image used by the image-to-3D model. Regardless of which reference frame is chosen, we note that our method consistently outperforms all baselines reported in the main paper.

Table 5. **Ablation study - Number of frames.** We train the temporal 3D diffusion model and the temporal 3D autoencoder with various number of frames N and compare reconstructions on 16 keyframes.

Stage I -N	Stage II -N	CD-3D↓	CD-4D↓	CD-M↓
4	16	0.057	0.091	0.187
8		0.055	0.086	0.176
16	4	0.051	0.073	0.144
	8	0.050	0.073	0.144
4	4	0.056	0.091	0.187
8	8	0.055	0.085	0.175
16	16	**0.050**	**0.069**	**0.137**

C. Implementation details

Temporal 3D diffusion. The temporal 3D diffusion model follows the TripoSG [18] backbone: it is composed of 21 self-attention DiT blocks (16 heads, 2048 width, 64 latent dim). We train with AdamW (learning rate 1×10^{-4}, weight decay 1×10^{-2}) in bfloat16 mixed precision, using a global

Table 6. **Ablation study - Autoregressive context window.** We evaluate our model trained with $N = 16$ keyframes on sequences of 31 timesteps for different context window c_{w}.

Stage I - c_{w}	Stage II - c_{w}	CD-3D↓	CD-4D↓	CD-M↓
1		0.051	0.098	0.195
4	1	0.051	0.094	0.190
8		0.051	0.090	0.185
	1	0.051	0.098	0.195
1	4	0.050	0.097	0.196
	8	0.051	0.098	0.196
1	1	0.051	0.098	0.195
4	4	0.050	0.094	0.191
8	8	0.051	0.091	0.187

Table 7. **Ablation study - Reference frame.** We evaluate our model by taking either first (default), middle or last frame as the reference frame, for stage I and stage II.

Reference frame	CD-3D↓	CD-4D↓	CD-M↓
First	**0.050**	**0.069**	**0.137**
Middle	0.054	0.074	0.148
Last	0.055	0.079	0.161

batch size of 96 for 170,000 steps. The dataset comprises 13,200 animated object sequences drawn from Objaverse [8], Objaverse-XL [7], and an internal corpus. **Inputs.** (i) *Input frames:* for each armature-driven sequence (at least 16 and up to 128 keyframes), we render 16 viewpoints per keyframe with uniformly spaced azimuths and elevations in $[40°, 85°]$. (ii) *Input point-clouds:* for each sequence we construct a canonical point cloud $\mathbf{P} \in \mathbb{R}^{N_{\text{p}} \times 6}$ with $N_{\text{p}} = 500{,}000$ points (XYZ and normals) together with the armature configurations over keyframes; we then deterministically deform $\mathbf{P}$ to each keyframe k to obtain $\mathbf{P}_k$, aligned with the rendered image $\mathbf{I}_k$. Training uses 16-frame sequences: each batch contains a video clip $(16 \times H \times W \times 3)$ and the aligned sequence of deformed point clouds with normals $(16 \times N_{\text{p}} \times 6)$. We extend TripoSG image pre-processing to video by computing the union object bounding box over the whole sequence and resizing every frame so that this box occupies 90% of the height or width; applying a consistent crop across frames avoids spurious scale/translation cues. Consistent with [53], we find that deforming a single canonical point cloud over time—rather than re-sampling points at each timestep—is critical for stable training of our temporal 3D diffusion model. We encode point clouds with a frozen $\mathcal{E}_{\text{3D}}$ to obtain latents; during training, we randomly select $N_{\text{S}} \in \{1, 2, 3\}$ latents as ground-truth conditioning and train on the remaining latents using the standard flow-matching objective. Due to memory constraint, we train with $T = 1024$ tokens, however, at inference, $T = 2048$.

Temporal 3D autoencoder. The temporal 3D autoencoder model follows the TripoSG [18] backbone. It has 16 self-attention and 1 cross-attention DiT layers (8 heads, 1024 width). We train the temporal 3D autoencoder with the same optimization hyperparameters and bfloat16 mixed-precision setup as temporal 3D diffusion, using the identical corpus of 13,200 animated object sequences. Training again operates on 16-frame clips: for each sequence, we provide the model with the deformed point cloud trajectory, where each point cloud contains XYZ coordinates and normals at keyframe k. In addition, we sample a source and a target timestep $(t_{\text{src}}, t_{\text{tgt}})$ uniformly between the first and last keyframe and extract a set of surface points by deforming randomly sampled mesh points from t_{src} to t_{tgt}. The temporal 3D autoencoder is trained to regress these deformations from the input 3D latents on the keyframes, minimizing an ℓ_2 loss between predicted and ground-truth positions. Unlike the temporal 3D diffusion stage, this training setup does not rely on latents from a temporally synchronized canonical point cloud: surface points can be re-sampled independently at each timestep, simplifying data preparation and decoupling the autoencoder from the canonical-deformation assumption.

Quantitative evaluation. Our quantitative benchmark ActionBench is composed of 128 animated scenes from Objaverse [7, 8], each comprising a textured, rigged mesh with a predefined animation sequence. For every scene we select the first $N = 16$ keyframes and render from a fixed camera with azimuth $70°$. Each mesh is converted to a watertight representation and the animation is rigidly normalized to a canonical cube $[-1, 1]^3$, yielding synchronized sequences $\{\mathbf{V}_k, \mathbf{F}\}_{k=1}^{N}$ and $\{\mathbf{I}_k\}_{k=1}^{N}$.

Given watertight ground-truth meshes $\{\mathcal{M}_k\}_{k=1}^{N}$ and predicted meshes $\{\widehat{\mathcal{M}}_k\}_{k=1}^{N}$, we uniformly sample $P = 100{,}000$ surface points from each mesh (area-weighted). Let $\mathbf{S}_k = \{\mathbf{x}_{k,i}\}_{i=1}^{P} \subset \mathbb{R}^3$ and $\hat{\mathbf{S}}_k = \{\hat{\mathbf{x}}_{k,i}\}_{i=1}^{P}$ denote the sampled point sets. Prior to distance computation, we rigidly align predictions to the ground truth using Iterative Closest Point (ICP) [1]. We report two alignment protocols: *ICP3D* (frame-wise), where for each k, we estimate $(\mathbf{r}_k, \mathbf{t}_k) \in SO(3) \times \mathbb{R}^3$ that aligns $\hat{\mathbf{S}}_k$ to $\mathbf{S}_k$; and *ICP4D* (sequence-wise), where we estimate $(\mathbf{r}_1, \mathbf{t}_1)$ on $k = 1$ and apply it to all frames giving $\{\mathbf{r}_1\hat{\mathbf{x}}_{k,i} + \mathbf{t}_1\}_i$. We denote by $\bar{\mathbf{S}}_k^{\text{3D}}$ and $\bar{\mathbf{S}}_k^{\text{4D}}$ the aligned predictions under the chosen protocol. In practice, we use the gradient-based implementation of [25] to align shapes with ICP.

Chamfer-based shape metrics. The symmetric Chamfer distance (CD) between the ground-truth and predicted point sets at frame k is defined as

$$\text{CD}(\mathbf{S}_k, \hat{\mathbf{S}}_k) = \frac{1}{P} \sum_{i=1}^{P} \Big[\min_{\hat{\mathbf{x}} \in \hat{\mathbf{S}}_k} \|\mathbf{x}_{k,i} - \hat{\mathbf{x}}\|_2^2 + \min_{\mathbf{x} \in \mathbf{S}_k} \|\mathbf{x} - \hat{\mathbf{x}}_{k,i}\|_2^2 \Big]. \tag{2}$$

We define CD-3D as the temporal average of the symmetric Chamfer distance computed *per frame* under ICP3D alignment.

$$\text{CD-3D} = \frac{1}{K}\sum_{k=1}^{N} \text{CD}(\mathbf{S}_k, \bar{\mathbf{S}}_k^{\text{3D}}). \tag{3}$$

When the alignment is ICP4D (single transform from $k = 1$), we use $\bar{\mathbf{S}}_k^{\text{4D}}$ in the equation above and we refer to the resulting average as CD-4D.

Motion Chamfer distance. Analogous to Chamfer distance, after aligning sequence-wise using ICP4D, we establish two directed nearest-neighbor maps (GT $\rightarrow$ PRED and PRED $\rightarrow$ GT) at the first frame and keep them fixed for the whole sequence:

$$\sigma_i = \underset{j\in\{1,\dots,P\}}{\text{argmin}} \left\|\mathbf{x}_{1,i} - \hat{\mathbf{x}}_{1,j}\right\|_2^2, \tag{4}$$

$$\tau_i = \underset{j\in\{1,\dots,P\}}{\text{argmin}} \left\|\mathbf{x}_{1,j} - \hat{\mathbf{x}}_{1,i}\right\|_2^2. \tag{5}$$

We then propagate these correspondences to every frame k without recomputing nearest neighbors, and refer to the resulting average as CD-M:

$$\text{CD-M} = \frac{1}{NP}\sum_{k=1}^{N}\sum_{i=1}^{P} \left\|\mathbf{x}_{k,i} - \hat{\mathbf{x}}_{k,\sigma_i}\right\|_2^2 + \left\|\mathbf{x}_{k,\tau_i} - \hat{\mathbf{x}}_{k,i}\right\|_2^2. \tag{6}$$