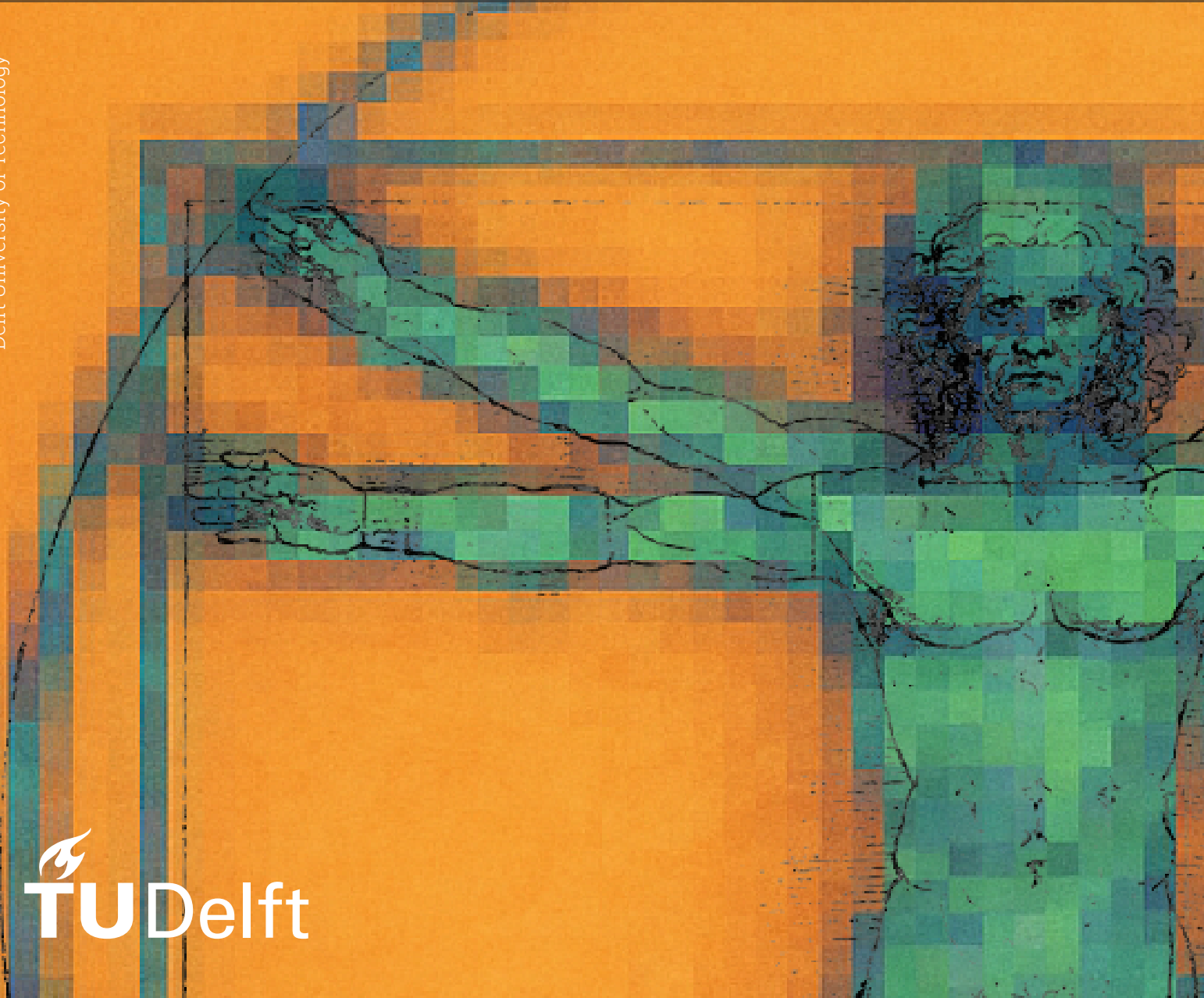
Geometry-Guided Video Generation with Diffusion Feature Textures

MSc Computer Science

Jorge Romeu Huidobro

Delft University of Technology



TU Delft

Geometry-Guided Video Generation with Diffusion Feature Textures

by

Jorge Romeu Huidobro

Advisor:	Dr. Ricardo Marroquim
Supverisors:	Dr. Petr Kellnhofer & Ir. Lukas Uzolas
Project Duration:	August 2024 - July 2025
Faculty:	Electrical Engineering, Mathematics and Computer Science (EEMCS)

TU Delft

Abstract

Recent advances in generative AI have enabled high-quality video generation from text prompts. However, the majority of existing approaches rely exclusively on prompts, making it difficult for an artist to control the generated scene layout and motion. In this thesis, we propose a novel method for geometry-guided Text To Video generation. Our method takes as input an animated mesh sequence and a text prompt and generates a video following both the text prompt and input geometry. Our pipeline consists of two main stages: Firstly, we use an existing text-driven texture generation method to create an initial rough texture for the geometry. Next, a depth-conditioned T2I model is used to generate video frames following the guidance animation, using the generated texture to enforce temporal consistency across frames. By generating video frames rather than directly using the result of the texture generation, our method supports generating deformations from the guidance geometry and variable lighting and by using the texture for feature alignment, we acheive significantly stronger robustness to occlusions and camera motion than existing controllable video-generation approaches. We begin by identifying the failure modes of existing methods through a set of initial experiments, we then use these findings to propose our method and finally evaluate it through a series of comparisons and ablations.

Acknowledgements

Firstly, I thank my supervisors, Dr. Petr Kellnhofer and Lukas Uzolas, for their guidance and support throughout this work. Much like diffusion models, a master student needs sufficient compute and high-quality guidance to produce good results, which is exactly what you provided over the past year. Without your help, the quality of this thesis would likely resemble the leftmost row of Figure 3.4a. I am also grateful to Prof. Ricardo Marroquim and Prof. Elmar Eisemann for their advice on this work, and to the CGV Group for the courses, colloquia, and for introducing me to the field of Computer Graphics. Thanks as well to Xucong Zhang for serving on the committee of this thesis. I would like to thank the friends I've made throughout my years in Delft, as well as friends back home, Gonzalo, Alberto, Chrysanthos, and Violeta for making these past couple of years so enjoyable. And of course, it goes without saying that I am grateful to my parents for their support throughout my university years, and for giving me the opporutnity to study abroad. Finally, this section would be incomplete without acknowedging the contribution of my friends, Pavlos Makridis and Rodrigo Alvarez Lucendo, not only for their advice on this work, but more importantly for their company during our (often too-long) phone calls. If the saying is true, and we really do become the average of the people we spend the most time with, then I'm incredibly lucky to have had it be the two of you.

Jorge Romeu Huidobro
Delft, June 2025

Contents

1

Introduction

Figure 1.1: We present a novel method for Geometry-Guided Text To Video Given a text prompt and an animated mesh sequence, our method generates a video following the guidance geometry and prompt. We employ an existing texture generation method [32] to generate an initial texture for the mesh and then use a depth-conditioned T2I Diffusion Model to generate video frames, leveraging the texture to enforce temporal consistency. Our two-stage approach improves visual quality and temporal consistency compared to Generative Rendering [8] and is capable of capturing deformations from the guidance geometry, unlike techniques based on texturing [32].

Traditional 3D tools for creating videos offer complete artistic control by providing explicit control over scene geometry, appearance, and camera motion. However, this approach is time-consuming, laborious, and requires significant artist expertise. Recent advances in generative AI, such as Text To Image (T2I) [55, 47, 58, 62] and Text To Video (T2V) [7, 31, 30] models, based on Diffusion Models (DMs), offer a promising alternative as they enable the generation of high-quality visuals from text prompts. However, the majority of existing video-generation methods rely *solely* on prompts, making it difficult for artists to control the layout and motion of the generated videos.

Recently, Generative Rendering (GR) [8] proposed geometry-guided T2V. Their method takes as input an artist-provided low-fidelity animated mesh, and a text prompt, and uses a depth-conditioned T2I DM to generate a temporally consistent video that adheres to the guidance geometry. Artists can then rapidly prototype a scene using traditional 3D tools and use the generative model to create the final video. While their approach is promising, our initial experiments reveal that the generated videos exhibit substantial temporal inconsistency and visual artifacts when the input geometry contains occlusion or camera movement. In this thesis, we propose a novel method for geometry-guided T2V which aims to address these limitations.

Our key idea is to build on the approach of GR by incorporating an existing text-driven texture generation method [10, 57, 12], specifically TexGen [32]. This family of methos enable the generation of a globally consistent texture for a mesh given a text prompt. We can immediately apply these techniques to our task by simply generating a texture for the guidance mesh and rendering it to the target animation. However, this introduces a new set of limitations, as a static texture cannot model

various effects, such as lighting or deformations from the guidance geometry.

Our method combines the strengths of both approaches with a two-stage pipeline. Firstly, we use a text-driven texture generation method to generate an initial texture for the guidance geometry. Next, we utilize a depth-conditioned DM to generate the video frames, enforcing temporal consistency by conditioning the generation of each frame on the texture. This conditioning is done via two mechanisms. Firstly, we use the texture to initialize the latent noise used to generate each video frame in a multi-view consistent manner, using a technique inspired by SDEdit [43]. Next, at each denoising step, we extract intermediate features from the reverse diffusion process that generated the texture and inject them into the generation of each video frame through a mechanism similar to the one used by GR.

Figure 1.1 shows an example video generated with our method, and compares it to GR and texture generation and rendering. As can be observed, GR is unable to handle camera zoom, resulting in blurry and corrupted frames for close-up views. On the other hand, using a static texture is robust but cannot model the geometry of the helmet. Our method can model these deformations while offering stronger visual fidelity than GR.

We begin by reproducing and systematically evaluating GR [8] and TexGen [32] and explore the advantages of both approaches in our setting. We use these findings to motivate the design of our method, which we evaluate through a series of comparisons and ablations. In summary, we make the following contributions:

1. Reproduce and systematically evaluate two state-of-the-art methods, namely GR [8] and TexGen [32], on the task of Geometry-guided T2V, and identify and understand their respective failure cases.
2. Propose a novel method for geometry-guided T2V that uses a text-driven texture generation method as a pre-processing step, resulting in stronger robustness to occlusion and camera z-movement than GR.
3. Evaluate our method on a variety of animation sequences and text prompts through a series of comparisons and ablations.

1.1. Outline

We start by giving an overview of related works and theoretical background in chapters 2 and 3. Next, in chapter 4, we formally describe our task, and evaluate two representative methods from the literature to identify and understand their falure cases. Guided by these findings, we present our method in chapter 5, and then evaluate it in chapter 6 through a series of experiments. Finally, we discuss our findings, and give concluding remarks in chapter 7.

2

Related Work

This chapter provides an overview of the relevant literature. We begin by covering existing methods for video generation and controllable video generation. Next, we cover the related task of 3D generation, focusing on texture generation, which we use in our method. Finally, we cover the main techniques for controlling T2I diffusion models by manipulating their intermediate features. Later, in chapter 3, we dive deeper into the theory behind the techniques presented here.

2.1. Video Generation

T2V models extend T2I models to video by training on large-scale video datasets. Most works achieve this by fine-tuning a T2I model on video data [30, 7, 31]. This is typically done by inflating 2D convolutional layers to 3D pseudo-convolutions, incorporating temporal attention layers, and fine-tuning only the newly introduced parameters. Training high-quality T2V models, however, is computationally demanding and challenging due to the limited availability of large captioned video datasets. For this reason, other works extend T2I models to video with no additional training, which is the approach we also follow in this thesis. These methods modify the denoising process to enforce approximate temporal consistency. Text2Video-Zero [35] proposes using a T2I model to generate video frames, ensuring temporal consistency through principled noise initialization and aligning the intermediate features of the U-Net across frames. While these approaches have proven successful at generating video, they lack fine-grained structural control of the video.

2.2. Controllable Video Generation

Various works have attempted to add spatial control to T2V models. Text-guided video stylization approaches [78, 11, 79, 53, 21] take a source video and text prompt as input and generate a new video that follows the motion of the source video but adheres to the new prompt. Tune-A-Video [78] accomplishes this by extending a T2I model to process video and fine-tuning on a single input video. The resulting model can generate videos with motion included in the training video but with a new prompt. Follow-up works [11, 79, 53, 21] obtained comparable results using inversion techniques [66, 46] on the input video frames and re-generate the video with additional modifications to the model to enforce temporal consistency. These techniques are methodologically similar to our method, as we also generate video frames with a T2I model and enforce consistency via feature manipulation; however, they are not directly applicable to our task, as they require a full input video.

Most similar to our work are methods for geometry-guided T2V generation. Gen-1 [19] and CTRL-Adapter [40] fine-tune video diffusion models to support depth-conditioning. However, their reliance on video models makes these methods computationally expensive and limited to very short or low-framerate videos. On the other hand, GR [8] tackles this same task using only a T2I model, enabling efficient generation of arbitrarily long videos. While their method works well, our initial experiments presented in chapter 4 find that it is not robust to camera z-movement or occlusion.

2.3. 3D Generation

A related line of work is Text-to-3D. While distinct from video generation, these methods can, in principle, be adapted to our task by first generating a 3D model from a text prompt and reposing and rendering the generated asset according to the guidance animation. Early works towards 3D generation include DreamFusion [52] and its follow-ups [44, 39, 77, 70]. These methods generate a 3D model from a text prompt by Score Distillation Sampling (SDS), which proposes optimizing a 3D representation such as a NeRF or 3DGS [45, 34] such that its renders have high likelihood under a frozen T2I diffusion model. Follow-up works train diffusion models on 3D data [64, 42], yielding stronger results. While these methods are successful at generating 3D, they are not straightforward to adapt to our setup, as the generated model would have to be skinned and animated to follow the guidance geometry.

More closely related to our task is text-driven texture generation. Given a 3D mesh with UV mapping, these methods aim to synthesize a globally consistent texture for the mesh based on a text prompt [12, 57, 10, 32]. Most of these approaches follow a similar overall pipeline shown in figure 2.1. To generate a texture for a mesh, cameras are placed surrounding the object, and a depth map is rendered for each view. We then use a depth-conditioned T2I to generate a textured image for each view, and assemble the final texture projecting the generated images to the UV space of the mesh.

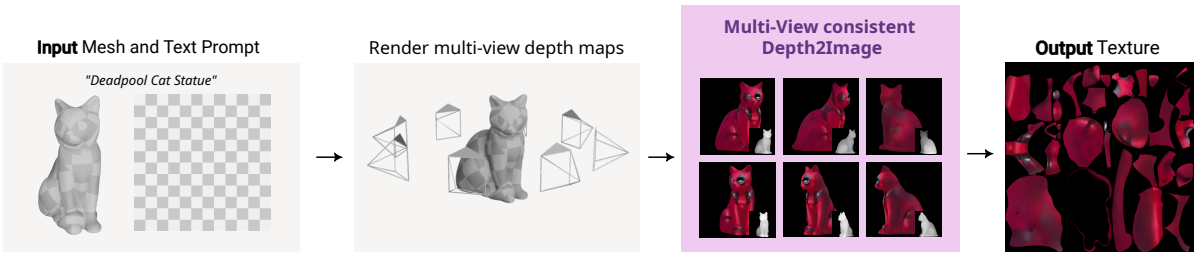


Figure 2.1: Texture generation with depth conditioned diffusion models Given a 3D Mesh and a text prompt, a texture is generated by placing cameras surrounding the object, rendering a depth map for each camera and generating an image conditioned on each depth map with a depth-conditioned T2I model. The resulting images are inverse rendered to the UV space of the mesh to assemble a final texture. Methods differ in how they enforce view-consistent generation.

These methods differ in how they ensure the generated views are multi-view consistent. Early works [12, 57] perform sequential generation, where the first view is fully generated, and each subsequent view is conditioned on all previous views via inpainting. However, this approach suffers from noticeable seams due to irreconcilable artifacts from the early views. TexFusion [10] proposes Sequential Interlaced Multiview Sampling (SIMS), which interleaves the inpainting over multiple denoising steps, leading to significantly stronger results. However, a limitation of SIMS is that when used with a Latent Diffusion Model (LDM) [58], the consistency is only enforced in the low-resolution latent space, resulting in minor inconsistencies when decoded to RGB. Their solution is to fit a neural color field to smooth out the inconsistency, but this leads to over-smoothed textures. Most recently TexGen, [32] employs a strategy similar to SIMS but enforces consistency in the RGB space by adapting inpainting techniques for LDMs [4], resulting in a more straightforward method and better results.

While these methods are successful at generating textures for static meshes, they are still limited when applied to our task, as they produce textures with baked illumination and hence cannot model variable lighting or deformations over the proxy geometry. We instead find that they can be used as a suitable pre-processing step when combined with end-to-end video generations described previously.

Lastly, a concurrent work, Tex4D [6], adapts SIMS to video diffusion models to synthesize multi-view consistent and temporally consistent *video* textures. This resolves some issues with static texturing, as the generated video texture can capture appearance variation; however, it still does not support generating reasonable deformations from the guidance geometry and suffers from the limitations of existing publicly available video diffusion models discussed previously.

2.4. Controlled Generation via Diffusion Feature Manipulation

Recent work has found that intermediate activations extracted from T2I DMs during image generation encode valuable information about the structure and content of the generated image. This has led to a

surge of works that extract these features for various downstream tasks such as establishing image [71, 82, 25, 69] and 3D shape [16, 75] correspondences, depth estimation [73] and more.

Other works instead propose to *manipulate* these features during the image generation process. This has shown to be a practical framework for controlling the generated image. Prompt-to-prompt [26] observed that by manipulating the cross-attention layers, it is possible to control the relation between the spatial layout of the image and each word in the text. Plug-and-Play Diffusion [74] found that manipulating the spatial features enables controlling the overall layout of the generated image. Tune-A-Video [78] finds that by manipulating the keys and values of the self-attention layers, the overall appearance and style of the generated image can be controlled. Many following works employ combinations of these core techniques to implement various control mechanisms such as style-control [27, 13, 2], zero-shot adaptation of T2I models to video [35, 53, 11, 79, 21], text driven image editing [9, 72] and more [50, 18]. While these methods do not address our task, our method employs many of the underlying techniques developed by this family of works. Subsection 3.2.2 provides more in-depth explanation of the particular feature manipulation techniques used in our method.

2.5. Summary of related works

To summarize, geometry-guided T2V can be approached via 3D generation or end-to-end video generation. Techniques based on 3D generation trivially guarantee perfect temporal consistency, but are not straightforward to adapt to our task. Texture Generation can be easily adapted to our task, but they cannot model deformations from the guidance geometry or view-dependent lighting effects. On the other hand, video generation techniques such as GR can handle this, but they suffer from occlusion and z-movement. We summarize the related works in table 2.1. Our method is the only one that exclusively uses a T2I model, supports generating deformations and view-specific lighting, and is robust to occlusion and camera z-movement.

Method	T2I-only	Deformations	Lighting	Occlusion	Cam z-mvmnt
Static Texture [10, 32, 12, 57]	✓			✓	✓
Video Texture [6]			✓	✓	✓
CTRL-Adapter [40], Gen-1 [19]		✓	✓	✓	✓
GR [8]	✓	✓	✓		
Ours	✓	✓	✓	✓	✓

Table 2.1: Summary of methods for Geometry-Guided T2V Our method is the only one that support generating variable lighting, deformations from the guidance geometry, and is robust to occlusion and camera z-movement using only a T2I DM.

3

Background

This chapter covers the background required to understand this work. We provide an overview of diffusion models in general, the Stable Diffusion T2I model specifically, and an overview of techniques for inference time control of T2I diffusion models through feature manipulation, which we employ in our method.

3.1. Diffusion Models

Generative Models are a class of machine learning models that aim to learn the underlying distribution of a dataset in order to generate new samples. The overall pipeline behind generative modeling is shown in figure 3.1. Given a dataset $\mathcal{D}$ of observations $\boldsymbol{x}$ drawn from some unknown probability distribution p_{data}, generative models fit a model p_θ that approximates p_{data} by maximizing the log-likelihood that p_θ assigns to the samples in the dataset, typically via gradient-based optimization. Once trained, such a model can be used to generate plausible samples resembling those in the training data by sampling the learned distribution p_θ.

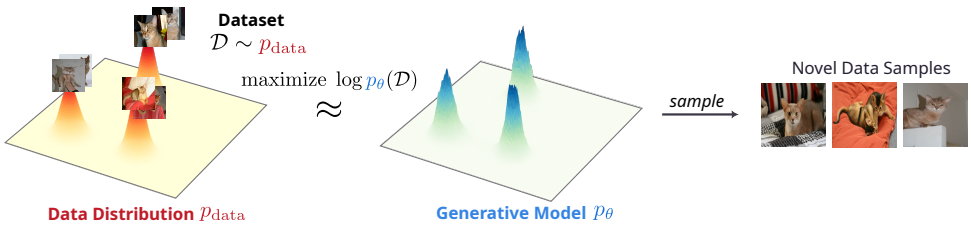


Figure 3.1: Overview of Generative Modeling Generative Models generate novel samples resembling those in the training dataset by learning a model of the probability distribution the training dataset was drawn from.

Various families of generative models have been proposed, which differ in how p_{data} is parameterized, such as Variational Autonencoders (VAEs) [37], Generative Adversarial Networks (GANs) [22], autoregressive models [48], flow-based models [36], and more. Among these, DMs [65, 28, 67, 68] have recently gained significant attention for their ability to produce very high-quality samples, especially for images, video, and audio. DMs approximate the data distribution by learning to reverse a pre-specified corruption process over the data distribution, which gradually transforms it into a standard Gaussian. Once trained, new samples can be generated by sampling from a Gaussian distribution and reversing the corruption process to convert it to a sample from p_θ.

This corruption process is the *forward diffusion process*, defined as a stochastic process which maps a data sample $\boldsymbol{x} \sim p_{\text{data}}$ into a series T of progressively noisier samples $\boldsymbol{x}_{0:T}$, defined by

$$\boldsymbol{x}_t = \sqrt{\alpha_t}\boldsymbol{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{3.1}$$

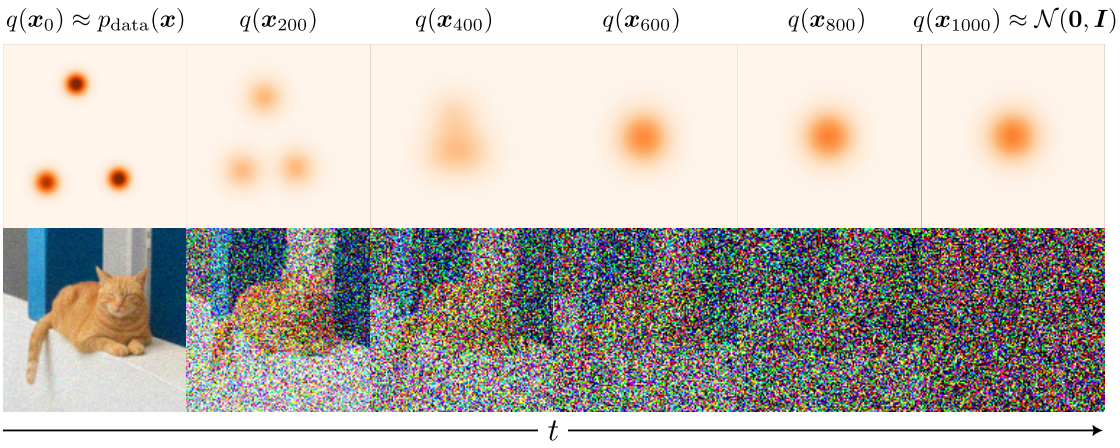


Figure 3.2: Forward Diffusion Process Illustration of the forward diffusion process on a 2D MoG and an RGB image. As the noise level t increases, the marginal distribution gradually approaches a standard Gaussian.

Where $\alpha_{0:T}$ defines the *noise schedule*, determining how much the original sample is maintained at each noise level t. Equivalently we can express the distribution $q(\boldsymbol{x}_t|\boldsymbol{x}_0)$ followed by a noisy sample $\boldsymbol{x}_t$ at noise level t given an initial $\boldsymbol{x}_0$, as a normal distribution centered at $\sqrt{\alpha_t}$ with covariance $(1-\alpha_t)\boldsymbol{I}$

$$q(\boldsymbol{x}_t \mid \boldsymbol{x}_0) = \mathcal{N}\left(\boldsymbol{x}_t; \sqrt{\alpha_t}\boldsymbol{x}_0, (1-\alpha_t)\boldsymbol{I}\right). \tag{3.2}$$

The noise schedule is constructed so that $q(\boldsymbol{x}_0)$ is close, or equal to the data distribution p_{data} and $q(\boldsymbol{x}_T)$ resembles a standard Gaussian distribution. Figure 3.2 shows the effect of the forward diffusion process on the Probability Distribution Function (PDF) of a 2D Mixture of Gaussians (MoG), as well as an RGB image. As the noise level increases, the structure in the data is gradually removed and the distribution approaches a standard Gaussian.

DMs reverse the forward diffusion process, by learning how to map a sample $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t)$ to a corresponding sample $\boldsymbol{x}_{t-1} \sim q(\boldsymbol{x}_{t-1})$ at a lower noise level. Multiple parameterizations for such a model have been proposed [65, 28, 67, 41], one widely adopted approach is *noise matching* where we fit a neural network $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ to predict the noise $\boldsymbol{\epsilon}$ added to a sample $\boldsymbol{x}_t$ at a given noise level t in equation (3.1), the training objective is then

$$\mathcal{L}_{\text{DM}} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}, t\sim U[0,T], \epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[||\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)||_2^2\right]. \tag{3.3}$$

Once trained, ϵ_θ can be used to generate new samples by initializing $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ as Gaussian noise and iteratively applying the denoising model $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ to predict the noise in $\boldsymbol{x}_t$ and removing it until reaching $\boldsymbol{x}_0$, which is a sample from the learned distribution. While there are many ways to perform this sampling process [28, 67, 41] one widely adopted approach is Denoising Diffusion Implicit Models (DDIM) [66]. To denoise a sample from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t-1}$ with DDIM sampling, the following equation is applied

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}}\hat{\boldsymbol{x}}_o + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t), \quad \text{with} \quad \hat{\boldsymbol{x}}_0 = \frac{1-\sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)}{\sqrt{\alpha_t}}. \tag{3.4}$$

In other words, the noise prediction $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ is used to estimate the fully denoised sample $\hat{\boldsymbol{x}}_0$ from the noisy input $\boldsymbol{x}_t$. Then a less noisy sample $\boldsymbol{x}_{t-1}$ is obtained by re-introducing noise to $\hat{\boldsymbol{x}}_0$ to noise level $t-1$, using the noise prediction as the noise. This can be interpreted as taking a deterministic step from $\boldsymbol{x}_t$ in the direction of $\hat{\boldsymbol{x}}_0$. In this work we employ a model trained with noise matching, and use DDIM sampling for generation. The overall procedure for training sampling is illustrated in algorithm 1 and 2.

Figure 3.3: Example Images Generated with Control-Net-Depth By fine-tuning on a paired dataset, ControlNet enables generating images conditioned on both a text prompt and a depth map.

Algorithm 1 Noise Matching (Training)

repeat
  $\boldsymbol{x} \sim p_{\text{data}}$ ▷ Sample data point
  $t \sim U[0, T]$ ▷ Sample Noise Level
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  $\boldsymbol{x}_t \leftarrow \sqrt{\alpha_t}\boldsymbol{x} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ ▷ Add noise (3.1)
  Take gradient step on $\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\|_2^2$
until converged

Algorithm 2 DDIM Sampling (Generation)

$\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
for $t = T, \ldots, 0$ **do**
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$
  $\hat{\boldsymbol{x}}_0^t \leftarrow (\boldsymbol{x}_t - \sqrt{1-\alpha_t}\hat{\boldsymbol{\epsilon}})/\sqrt{\alpha_t}$
  $\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}}\hat{\boldsymbol{x}}_0^t + \sqrt{1-\alpha_{t-1}}\hat{\boldsymbol{\epsilon}}$ ▷ DDIM Step (3.4)
end for
return $\boldsymbol{x}_0$

3.1.1. Conditional Generation

So far, we have discussed modeling a distribution from unlabeled data, however, this approach does not allow for control over the generated samples. Fortunately, diffusion models have also been shown to be successful at modeling *conditional distributions* $p(\boldsymbol{x}|\boldsymbol{c})$ where the learned distribution can be controlled with a conditioning signal $\boldsymbol{c}$ such as a text prompt.

There are various ways to perform conditional generation. The most straightforward is to train the model on labeled dataset $\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{c})\}$ of samples with an accompanying label (e.g., images with text captions). The model $\boldsymbol{\epsilon}_\theta$ is then modified to take an additional conditioning input $\boldsymbol{c}$, and the training objective and sampling procedure are updated accordingly to pass the condition. The updated noise-matching objective for a conditional diffusion model is then

$$\mathcal{L}_{\text{Conditional-DM}} = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{c})\sim\mathcal{D}, t\sim U[0,1], \epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[||\epsilon - \epsilon_\theta(\boldsymbol{x}_t, \boldsymbol{c}, t)||_2^2\right]. \tag{3.5}$$

This approach is used by T2I models for text-conditioning, due to the availability of large captioned image datasets. However, naively training and sampling conditional models this way often leads to suboptimal results, so it is typically combined with *guidance*, discussed in subsection 3.1.2.

An issue with this approach is the reliance on a large labeled dataset, which makes it impractical for various conditioning signals where such datasets are unavailable. Other works instead incorporate an additional condition into a pre-trained model by fine-tuning on a comparatively smaller labeled dataset. This is the approach followed by ControlNet [83] and related adapter methods [81, 40]. They propose a parameter-efficient method for fine-tuning pre-trained T2I models by introducing additional layers in the form of an adapter network which accept the new condition, such as a depth map, canny edges, human poses, etc. The adapter-model is the fine-tuned on a smaller dataset, while keeping the base model frozen. Figure 3.3 shows some example outputs using a ControlNet trained for depth conditioning.

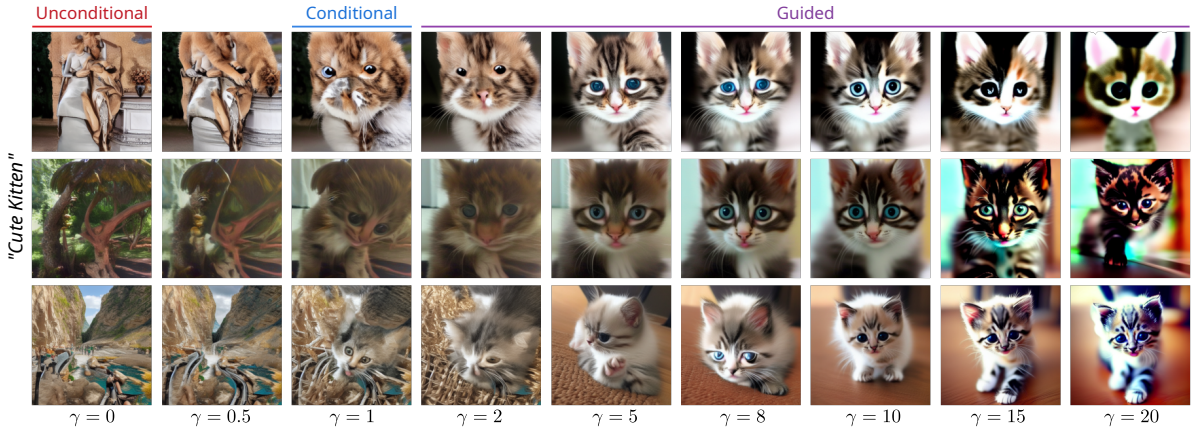
Lastly, *zero-shot control* strategies propose controlling generated samples without any additional training. Instead, these methods explicitly alter the denoising process to control the generated samples. This has been shown to be effective at controlling the style, layout and appearance of generated images with T2I models. Subsection 3.2.2 covers two relevant such techniques. We employ all three kinds of conditioning in our method. We use a pre-trained T2I model trained with text-conditioning, a ControlNet fine-tuned for depth-conditoning, and various zero-shot control techniques to enforce temporal consistency.

3.1.2. Guidance

Early work on conditional diffusion models struggled to generate high-quality samples. Training such models directly using equation (3.5) often results in the ignoring or downplaying the conditioning signal. *Guidance* methods enable explicit control over the strength the model gives to a conditioning signal, and have proven essential for high-quality conditional generation. The most widely used technique is Classifier Free Guidance (CFG) [29]. The main idea behind CFG is to replace the noise prediction at each denoising step with a *guided noise prediction* $\tilde{\epsilon}$, by combining the outputs of both a conditional and unconditional model. More specifically, the guided noise prediction is a barycentric combination with weight γ of both noise predictions

$$\tilde{\epsilon}(x_t, t, c) = \epsilon_\theta(x_t, t) + \gamma(\epsilon_\theta(x, t, c) - \epsilon_\theta(x, t)). \tag{3.6}$$

In practice, a single model $\epsilon(x_t, t, c)$ is used, where during training the conditioning signal c is occasionally dropped and replaced with a null value $\emptyset$, allowing the resulting model to be used in both conditional and unconditional modes. The parameter γ in equation (3.6), is the *guidance scale*. For $\gamma = 0$ the guided noise prediction is equivalent to the unconditional one, and for $\gamma = 1$ the conditional one. CFG takes effect when $\gamma > 1$. Figure 3.4a shows the effect of CFG on samples from a T2I model. When $\gamma > 1$, the generated samples follow the text prompt better, and are higher-quality, at the cost of diversity and oversaturation for very high guidance scales. We additionally show the impact of CFG on the modeled distribution for a toy 2D MoG in Figure 3.4b. As the guidance scale increases, the distribution focuses on the higher-likelihood regions corresponding to the condition. CFG and related techniques [15, 61] have proven crucial for producing high quality samples with DMs, and we employ it in our method.



(a) Impact of CFG on samples from a T2I model. Using $\gamma > 1$ results in higher sample quality at the cost of diversity. When the guidance is too strong, the generated images are often over-saturated.

(b) Impact of guidance scale on a 2D MoG, each mixture component representing data conditioned on a class. The leftmost plot has the non-guided marginal density, and the left-to-right densities have increasing guidance strength. A high guidance scale focuses the distribution on the higher-likelihood areas corresponding to the condition. (Results from [29])

Figure 3.4: Classifier-Free Guidance Impact of guidance scale on samples from a T2I model (a) and the marginal distribution of 2D MoG (b). CFG enables generating much higher-quality samples at the cost of diversity.

3.1.3. Latent Diffusion Models

When applying DMs to high-dimensional data, such as high-resolution images, performing the reverse diffusion process can become computationally intensive since generating a single sample would require multiple forward passes through a large neural network. To mitigate this issue, LDMs [58] propose

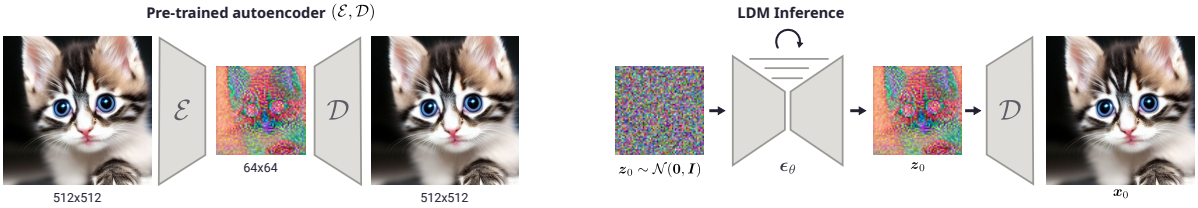


Figure 3.5: Latent Diffusion Models use a pre-trained autoencoder that maps from image space to a lower resolution latent space and back. During smapling, the reverse diffusion process takes place over the lower-resolution latent space, and the denoised latent is decoded to an image.

using a pre-trained autoencoder $(\mathcal{E}, \mathcal{D})$, typically a VAE [37] or VQ-VAE [20, 56], which compresses samples into a perceptually meaningful, lower-dimensional latent space. We can then train the diffusion model over the lower-dimensional latent space, enabling the use of a much smaller model.

Figure 3.5 shows an illustration of how LDMs are used for image generation. During training, the image is encoded into a lower-resolution latent using the encoder $\mathcal{E}$, and the forward diffusion process is applied over the encoded image normally. During sampling, an initial latent is sampled $z_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and iteratively denoised to obtain the fully denoised latent z_0 which is decoded to an image $\boldsymbol{x}_0$ with the decoder $\mathcal{D}$ once fully denoised. We use an LDM in our method which is further described in subsection 3.2.1.

3.2. Stable Diffusion

In this work, we use a T2I diffusion with depth-conditioning, specifically Stable Diffusion (SD) [58], with a ControlNet [83] for depth-conditioning. SD is a widely used, publicly available T2I trained on large-scale image datasets such as LAION-5B [63]. This section discusses the aspects of the model most relevant to our method. We begin with an overview of the model architecture, focusing on the behavior of the self- and cross-attention layers. We then discuss two common strategies for controlling the generated images by manipulating intermediate features of these layers.

3.2.1. Model Architecture

Stable Diffusion is an LDM and parameterizes the denoising network $\boldsymbol{\epsilon}_\theta$ as a U-Net-like model [59] with spatial-self attention layers and time and text conditioning. Figure 3.6 shows an illustration of the model architecture. At each denoising step, SD takes as input the current noisy latent z_t, the time step t, and a text condition $\boldsymbol{c}$. It follows a U-Net architecture composed of a series of *blocks* that first downsample (encoder) and then upsample (decoder) the image features, with skip connections between corresponding encoder-decoder blocks. Each block takes in image features at a given resolution and applies convolutional layers with a ResNet block [24], followed by self- and cross-attention layers. The time-step t is embedded using positional encoding ψ [76] and injected into each ResNet block, and the text condition $\boldsymbol{c}$ is embedded using a text encoder τ, typically CLIP [54], and passed to each block at the cross-attention layer.

Optional depth conditioning is introduced by passing an additional depth map $\boldsymbol{d}$ through an auxiliary ControlNet, whose output is injected into the skip connections of each decoder block. Our method enforces temporal consistency by modifying the features of the self-attention layers, in the next section we cover these layers more in depth.

Self- and Cross-Attention layers in Stable Diffusion

In this section we cover the inner workings of the self- and cross-attention layers in the SD U-Net. We begin by covering attention [5] broadly, and then discuss how it is used by SD. The attention operation receives as input three sequences: queries $\boldsymbol{Q} \in \mathbb{R}^{T, d_{\text{QK}}}$, keys $\boldsymbol{K} \in \mathbb{R}^{T_{\text{ctx}}, d_{\text{QK}}}$ and values $\boldsymbol{V} \in \mathbb{R}^{T_{\text{ctx}}, d}$ where the query and key sequences have matching dimensionality d_{QK}, and the key and value sequences have matching length T_{ctx}. The output of attention is a sequence $\boldsymbol{y} \in \mathbb{R}^{T, d}$, computed like so

$$\boldsymbol{y} = \text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \mathbf{A}\boldsymbol{V}, \qquad \mathbf{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_{\text{QK}}}}\right). \tag{3.7}$$

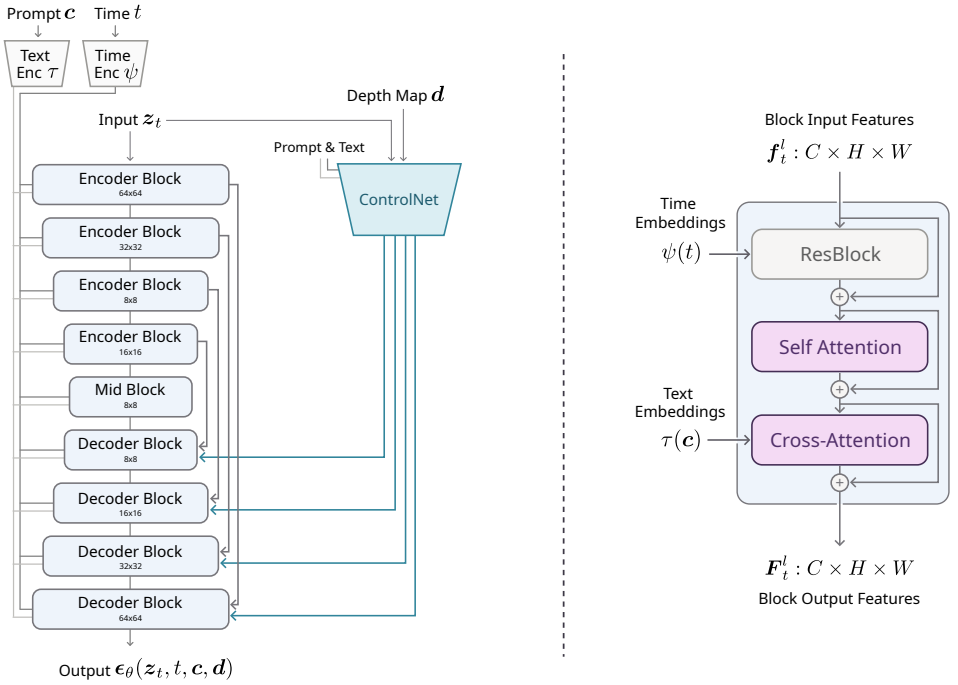


Figure 3.6: Stable Diffusion + ControlNet Model Architecture SD takes as input a noisy latent z_t, a noise level t, and a text condition c. The model follows a U-Net like architecture. Optional depth conditioning is introduced by passing a depth map d through an auxiliary ControlNet, whose output is injected into the SD model. Each U-Net block consists of a ResNet block, and self/cross-attention layers.

The equation is more clearly understood when written explicitly for each output $\boldsymbol{y}[i]$

$$\boldsymbol{y}[i] = \sum_j \boldsymbol{V}[j]\mathbf{A}[i,j], \quad \mathbf{A}[i] = \text{softmax}\left(\left[\frac{\boldsymbol{Q}[i]\boldsymbol{K}[j]}{\sqrt{d_{\text{QK}}}}\right]_j\right). \tag{3.8}$$

Intuitively, each entry in the output sequence $\boldsymbol{y}[i]$ is a weighted sum of all values in $\boldsymbol{V}$ where the weight $\mathbf{A}[i,j]$ assigned to a particular value $\boldsymbol{V}[j]$ is softmax-normalized, and determined by the similarity between query $\boldsymbol{Q}[i]$ and the key $\boldsymbol{K}[j]$ corresponding to the value. We say that a query *attends* to a value when it assigns a high weight to that value. In practice, attention is usually computed over multiple *heads* by splitting all three sequences across their d-dimension into h chunks, each corresponding to a different head, computing the attention of each chunk independently and using a linear layer to unify the heads into a single output.

The SD U-Net employs attention in two kinds of layers: *cross-attention* and *self-attention*, as shown on Figure 3.6. Cross-attention layers receive two sequences as input, an *input sequence* $\boldsymbol{f}$ of image features, and a *conditioning sequence* $\boldsymbol{c}$ of text embeddings. The output is computed by deriving keys/values from learnable linear projections of the conditioning sequence and queries from projections of the input sequence

$$\text{CrossAttention}_{W_Q,W_K,W_V}(\boldsymbol{f},\boldsymbol{c}) = \text{Attention}(W_Q\boldsymbol{f}, W_K\boldsymbol{c}, W_V\boldsymbol{c}). \tag{3.9}$$

In other words, in cross-attention layers, the image features $\boldsymbol{f}$ attend to the conditioning sequence $\boldsymbol{c}$. This is the mechanism through which SD models the influence of the text prompt on the model output. Figure 3.7 shows a visualization of the cross-attention maps extracted from a particular cross-attention layer when generating an image, each map corresponds to *column* in the attention matrix $\mathbf{A}$ corresponding to a particular token in the conditioning sequence, and shows how each image pixel

attends to that token. As can be seen, cross-attention layers learn intuitive representations, assigning higher activation to pixels that correspond to the token.

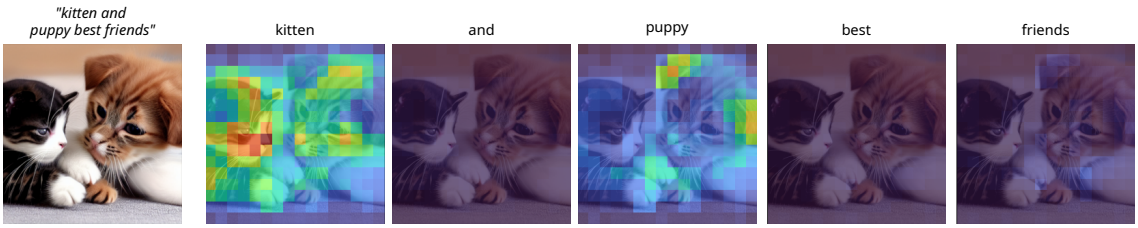


Figure 3.7: Cross-Attention maps Visualization of cross-attention maps at a particular denoising step and cross-attention layer. Each map corresponds to a token in the prompt, and is a column in the attention matrix **A**, reshaped to 2D. The weight assigned at a pixel represents how much the corresponding query attends to the token.

Conversely, self-attention layers, receive only a single input sequence f of image features and derive keys, queries and values from this single sequence. Intuitively, this layer allows the image features to attend to *themselves*, hence the name self-attention

$$\text{SelfAttention}_{W_Q,W_K,W_V}(f) = \text{Attention}(W_Q f, W_K f, W_V f). \tag{3.10}$$

Figure 3.8 shows some example self-attention maps for various query pixels for a particular denoising step, layer, and attention head. Each map corresponds to a *row* in the attention matrix **A** reshaped to 2D and displays which keys/values a query pixel attends to. Like cross-attention layers, self-attention layers learn intuitive representations which attend to pixels with similar semantics and appearance to itself.

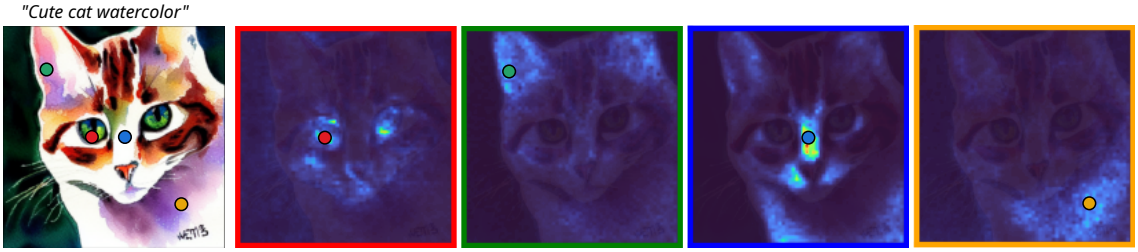


Figure 3.8: Self-Attention Maps Comparison of self-attention maps for a given layer and denoising step at various query pixels. Each map corresponds to a row in the attention matrix **A** reshaped to 2D, showing what the query attends to. Queries assign higher weights to pixels with similar appearance/semantics to itself.

Due to the interpretability of these layers, many works, including ours, manipulate the intermediate queries keys, values or attention maps during image generation, with the goal of controlling the generated image. The subsequent section gives an overview two such techniques we apply in our method, specifically, spatial feature injection and cross-image attention.

3.2.2. Controlling Stable Diffusion by feature manipulation

Recent work, discussed in section 2.4 has found that manipulating the intermediate features of the SD U-Net during image generation is effective at controlling the generated image. In this section we cover two of the main underlying techniques behind this family of works, *spatial feature injection* for controlling the structure and layout of the generated image and *cross-image attention* for controlling the overall appearance of the generated image. Later in chapters 4 and 5 we cover how these techniques are used by existing works, and our own method for the task of geometry-guided video generation.

Spatial Feature Injection

Tumanyan et al. [74] found that features extracted from intermediate layers of the SD U-Net during image generation act as high-dimensional spatial descriptors of the generated image. Figure 3.9 visualizes these features across various generated images. For each image, at a given denoising step t, we extract features F_t^l from the output of several layers l in the decoder block the denoising U-Net. We

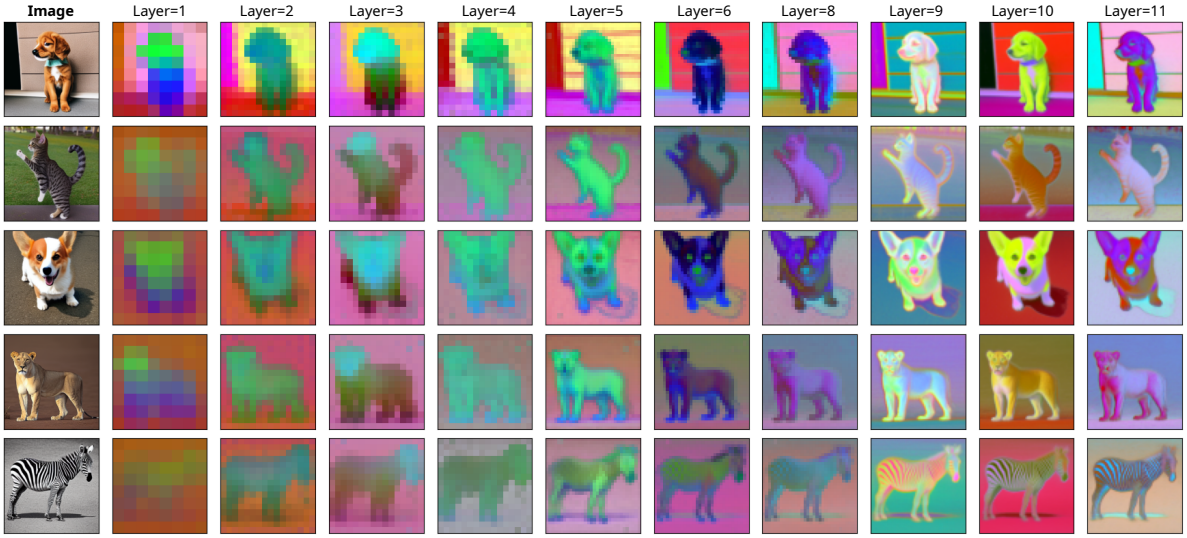


Figure 3.9: Visualization of Diffusion Features We generate multiple images, extracting features $\boldsymbol{F}_t^l$ at a denoising step t, roughly halfway through the generation process, from several decoder layers l of the U-Net. We visualize the top three principal components of these features as RGB channels. Early layers assign similar features to semantically related regions (head, legs, torso), while later layers assign similar features to regions with similar visual characteristics across images.

then apply Principal Component Analysis (PCA) to these features and visualize them as RGB to reveal regions with similar features across images. As shown, features extracted from early layers capture semantics, grouping related parts such as head, legs, and torso, while features obtained from deeper layers encode appearance, coloring regions with similar visual traits across images.

Given that these features encode rich spatial information about the generated image, they can be manipulated during the image generation process to control the generated image's layout and appearance. This is typically achieved by extracting features while generating a source image and injecting them into the model during the generation of a target image, blending the target's native features with those from the source image.

In our setting, spatial-feature injection can be used for generating images of the same geometry under different poses. This is shown in Figure 3.10, the source image is generated normally, using a depth-map of a T-posed character as conditioning, and at each denoising step t we additionally extract the output features $\boldsymbol{F}_t^l$ from various layers l in the U-Net. The target image is then generated with a depth-map at a different pose, where at each denoising step t, we inject the reposed features $\bar{\boldsymbol{F}}_t^l$ into the U-Net by blending them with the naturally generated features according to an alpha parameter

$$(1-\alpha)\boldsymbol{f}_t^{l,i} + \alpha\bar{\boldsymbol{f}}_t^{l,i}. \tag{3.11}$$

As can be seen this technique is effective at ensuring fine-grained spatial control, but since it directly overwrites the intermediate features in the target image, it can result in considerable visual artifacts, especially when using a high α or performing this feature injection at late layers and denoising steps.

Cross-Image Attention

Another widely used feature manipulation technique is Cross-Image Attention. Cross-image attention proposes replacing the keys and values in self-attention layers with those extracted from one or more source images. This was initially proposed by [78] but has since been generalized by several subsequent works that have found it to be effective at controlling the overall appearance and style of generated images [27, 13, 2, 35].

Recall from equation (3.10) that a self-attention layer l in the SD U-Net receives input features $\boldsymbol{f}^l$, projects them into queries, keys, and values, and computes attention over these sequences. Cross-Image Attention proposes extracting the input features $\boldsymbol{f}^{l,\text{src}}$, which we call *pre-attention features*, when

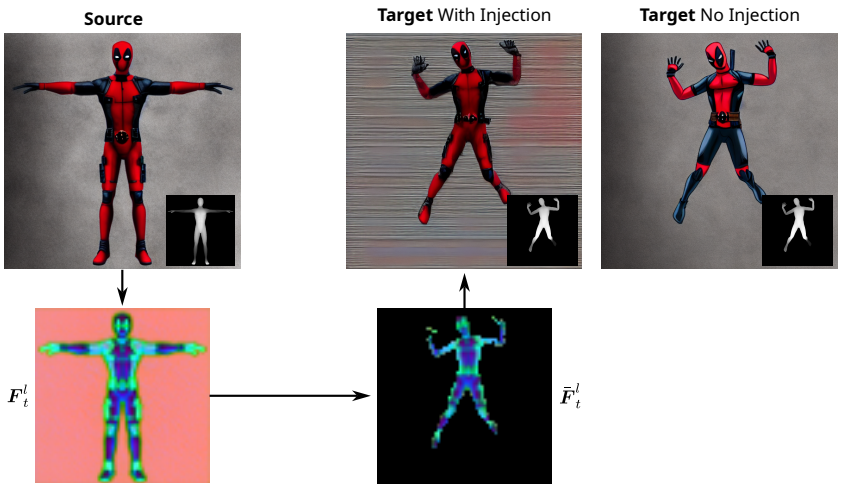


Figure 3.10: Spatial Feature Injection Spatial Feature Injection can be used with a depth-conditioned T2I model to generate spatially consistent images. At each denoising step features are extracted from the source image, and rendered to the target pose. The target image is then spatially consistent

denoising a source image, and then, generating a target image where the self-attention layers are altered to compute keys and values from the extracted pre-attention features. The modified self-attention layer for the target image becomes

$$\text{CrossImageAttention}_{W_Q,W_k,W_v}(\boldsymbol{f}^l, \boldsymbol{f}^l_{\text{src}}) = \text{Attention}(W_Q \boldsymbol{f}^l, W_K \boldsymbol{f}^l_{\text{src}}, W_V \boldsymbol{f}^l_{\text{src}}). \tag{3.12}$$

Intuitively, this is equivalent to replacing each self-attention layer with a *cross*-attention layer that attends to features from the source image. As a result, the queries in the modified self-attention layer attend to the injected values and return a weighted sum of the source features. Additionally, since attention does not require the key and value sequences to match the length of the query sequence, this can be further generalized to attend to *multiple* source images by simply injecting the concatenation of the source features extracted from multiple source images.

Figure 3.11 shows examples of images generated with Cross-Image Attention. We generate three images, each with a unique prompt and initial noise latent under various cross-image attention schemes: 1) attending to itself i.e. standard self-attention, 2) attending to a reference image, 3) attending to a reference image and itself, and lastly 4) attending to all three images. For each scenario, we also visualize the self-attention maps, discussed in Figure 3.8, for a particular layer, denoising step, and query pixel p_{qry}. When each image attends only to its own features (standard self-attention), the resulting images display distinct content and appearance. However, when attending to other source frames, the generated image adopts a combination of the appearances of the source images. By examining the self-attention weights, we can see that when cross-image attention is used, the query pixel attends to semantically similar features across the source images it attends to.

Unlike spatial feature injection, which resulted in noticeable artifacts, cross-image attention is considerably less destructive. This is because instead of overwriting the output of a given layer, we rely on the attention mechanism to determine which injected features to use based on their similarity to the image queries. Generally this works well, however we note that in order for this to work effectively, the injected source features be semantically similar to the image features. When this is not the case, the attention weights will be very sparse, and after softmax-normalization will become uniform, resulting in chaotic results in the generated image. Figure 3.12 shows various images generated with self-attention, and attending to a source image. When the source image is semantically similar to the target we obtain good results, but when they are very different the generated images are incoherent.

We employ cross-image attention in our method to generate video frames displaying the consistent appearance, and we further incorporate spatial feature injection to enforce fine-grained consistency

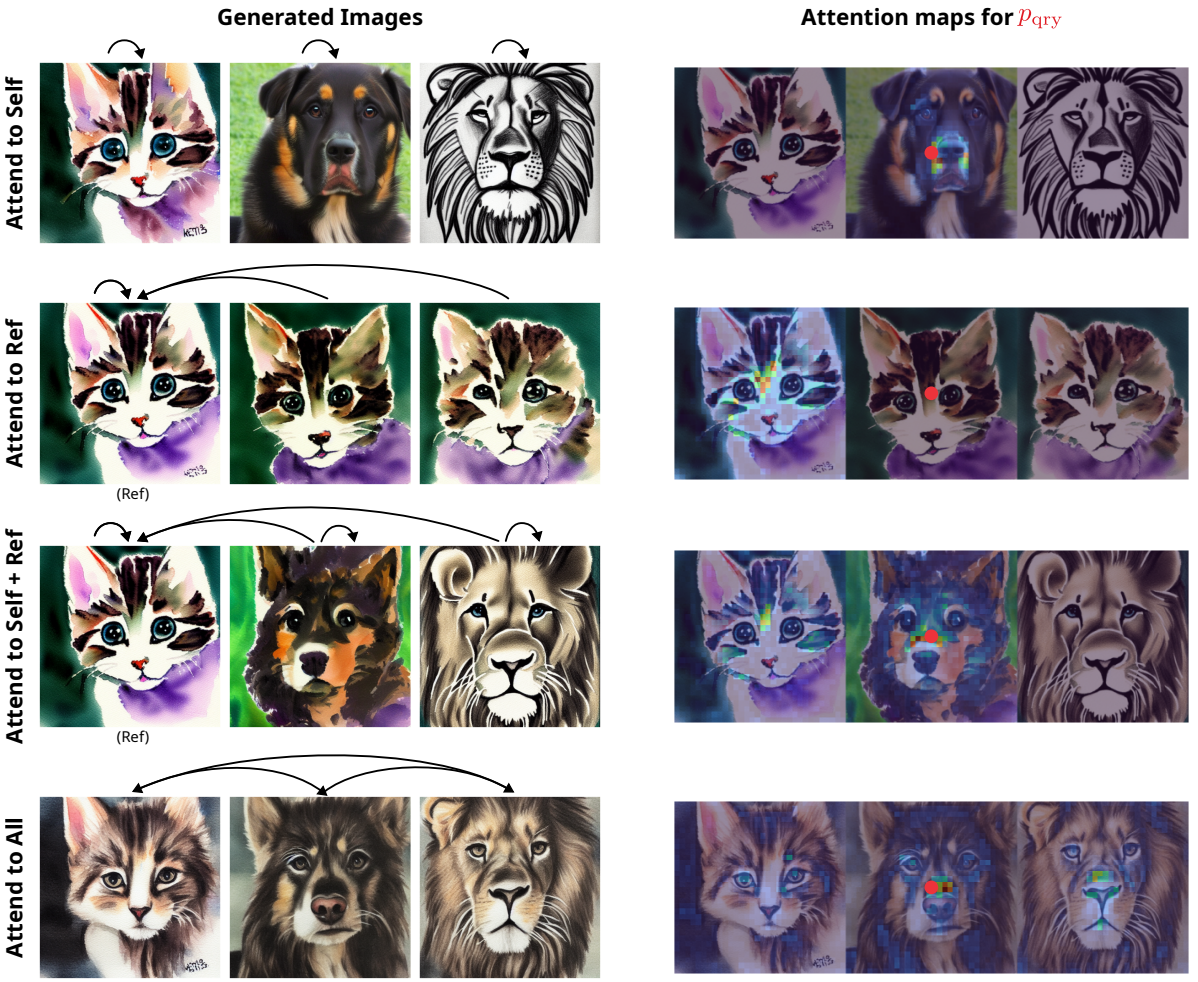


Figure 3.11: Cross-Image Attention we generate three images under four different cross-image attention schemes, where an arrow indicates an image attends to the keys and values of another. The Left column shows the generated images, and right column shows the attention weights extracted from a particular denoising timestep, layer and query pixel p_{qry} marked in red. When an image is generated attending to the features of one or more source images, the overall appearance of the source image is transfered to the target image, as it allows the query pixel to attend to its features.

Figure 3.12: Cross-image attention across diverse images We show generated images for various prompts, as well as the resulting image when cross-attending to a reference image. Cross-image attention relies on the image queries establishing sensible correspondences to the injected key and value features. This fails when the injected keys and values are taken from semantically very different images, giving incoherent results.

3.2.3. Summary

In summary, we have discussed the architecture of the SD U-Net as well as two common feature-manipulation techniques for controlling the generated image. Spatial feature injection is conceptually simple and enables spatially controlling the overall layout and appearance of the generated image by simply blending the image features with desired image features, but can lead to significant visual artifacts. Cross-Image attention instead manipulates the keys and values in self-attention layers, which enables controlling the overall appearance of the generated image. These feature manipulations are used by prior work, as discussed in chapter 4, as well as in our proposed method discussed in chapter 5.

4

Exploration of Existing Methods

Before presenting our method, we explore two existing approaches to our task. From the related works, we identify that GR [8] is the only prior work that directly addresses our task using only a T2I model. We additionally note that techniques for texture generation, discussed in section 2.3, can be readily adapted to our task by generating a texture for a single static pose and rendering it to the target geometry.

We begin by formally describing our problem formulation in section 4.1; we then implement and evaluate both of these methods through a series of targeted experiments. Our main finding is that the two techniques are complementary: GR has several desirable properties but is not robust to all geometries, specifically occlusions and camera z-movement. Texture Generation, on the other hand, is straightforward and robust, but cannot model deformations or lighting. Our method, presented in chapter 5, builds on these findings by combining both approaches into a single unified framework that addresses these failure cases.

4.1. System Setup

Our overall system setup is shown in figure 4.1. Given a text prompt $\boldsymbol{c}$ and guidance geometry $\mathcal{A}$, our goal is to generate a sequence of frames $I^{1\ldots N}$ forming a video aligned with both the text prompt and guidance geometry. The guidance geometry $\mathcal{A}$ is represented as an animated mesh sequence $\mathcal{A} = \{(\mathcal{C}_i, \mathcal{M}_i)\}_{i=1}^{N}$ consisting of N camera-mesh pairs, where each mesh $\mathcal{M}_i = (V_i, F)$ shares a common topology F, and UV mapping $\Phi : \mathcal{V} \rightarrow [0,1]^2$.

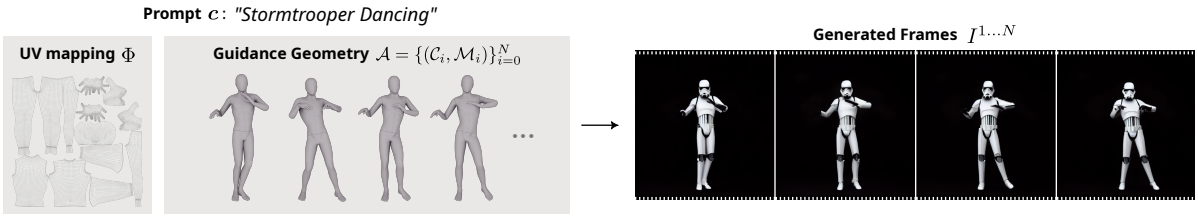


Figure 4.1: System Setup Given a text prompt $\boldsymbol{c}$, and guidance geometry $\mathcal{A}$ represented as an animated mesh sequence with UV mapping, our goal is to generate video frames $I^{1\ldots N}$ which adhere to both the text prompt $\boldsymbol{c}$ and the input geometry $\mathcal{A}$.

Additionally, we define texture rendering and inverse rendering functions $\mathcal{R}$ and $\mathcal{R}^{-1}$ for the guidance geometry. Rendering a texture $\mathcal{U}$ from UV space to frame i of $\mathcal{A}$ is denoted as $\mathcal{R}_i(\mathcal{U})$, while inverse rendering an image I from camera space at frame i to UV space is denoted as $\mathcal{R}_i^{-1}(I)$.

4.2. Generative Rendering

Generative Rendering [8] is a recent work which addreses our task directly. They propose using a depth-conditioned T2I model to generate video frames following an animated mesh sequence, and enforce consistency between frames by aligning the features in the DM across frames, using the techniques from subsection 3.2.2. We describe their feature alignment strategy in subsection 4.2.1, and based on their method pose hypothesized failure cases, which validate empirically.

4.2.1. Generative Rendering Method

Given an animated mesh sequence $\mathcal{A} = \{(\mathcal{M}_i, C_i)\}_{i=1}^N$, and a text prompt $\boldsymbol{c}$, GR generates video frames following the animation with a depth-conditioned DM. This is done by initializing a Gaussian noise latent for each frame $\boldsymbol{z}_T^i$, and repeatedly denoising them to obtain the final video frames $\boldsymbol{z}_0^{1..N}$. GR enforces consistency among frames at each denoising step from $\boldsymbol{z}_t^{1...N}$ to $\boldsymbol{z}_{t-1}^{1...N}$. This is done by, at each denoising step, randomly selecting a set frames from $\mathcal{A}$ as *keyframes*, which are used to condition the denoising of all other frames. Specifically, the keyframe-latents $\boldsymbol{z}_t^{\text{KF}}$ are passed through the U-Net to extract features, which are subsequently used to guide the denoising of the remaining frames in the animation.

Keyframe Feature Extraction Firstly we describe the keyframe-feature extraction. The latents corresponding to the selected keyframes $\boldsymbol{z}_t^{\text{KF}}$ are passed to the U-Net, and denoised with *extended attention*, discussed in subsection 3.2.2, where the self-attention layers are modified to attend to all other keyframes. The modified self-attention layer with extended attention is then:

$$\begin{aligned}\text{SelfAttn}^{\text{extr}}_{W_Q,W_K,W_V}(\boldsymbol{f}_t^{l,i}) = \textbf{do } & \boldsymbol{f}_t^{l,\text{KF}} \leftarrow [\boldsymbol{f}_t^{l,i}, \dots, \boldsymbol{f}_t^{m}] \\ & \boldsymbol{F}_t^{l,i} \leftarrow \text{Attention}(W_Q \boldsymbol{f}_t^{i}, W_K \boldsymbol{f}_t^{l,\text{KF}}, W_V \boldsymbol{f}_t^{l,\text{KF}}) \\ & \textbf{return } \boldsymbol{F}_t^{l,i}.\end{aligned} \tag{4.1}$$

We additionally save the attention output $\boldsymbol{F}_t^{l,i}$ for each keyframe, and the concatenated attention input features $\boldsymbol{f}_t^{l,\text{KF}}$ at each self-attention layer l and denoising timestep t. We denote a full forward pass through the denosing U-Net with extended attention and feature extraction like so

$$\boldsymbol{f}_t^{l_1,\dots l_N}, \boldsymbol{F}_t^{(l_1 \dots l_N, i \dots N)} \leftarrow \boldsymbol{\epsilon}_\theta^{\text{extr}}(\boldsymbol{z}_t^{\text{KF}}, t, \boldsymbol{c}, \boldsymbol{d}_{\text{KF}}). \tag{4.2}$$

Denoising with Keyframe Feature Injection The extracted keyframe features are now used to condition the denoising of each animation frame. Firstly, for each layer the extracted post-attention features are projected to the UV space to assemble a *feature texture* $\boldsymbol{F}_t^{l,\text{UV}}$. This is done by inverse-rendering the extracted keyframe post-attention features $\boldsymbol{F}_t^{(l,i)}$ onto the mesh and averaging, like so

$$\boldsymbol{F}_t^{l,\text{UV}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{R}_i^{-1} \left(\boldsymbol{F}^{(l,i)} \right). \tag{4.3}$$

Given the assembled feature textures, we now denoise each frame latent $\boldsymbol{z}_t^i$. We do this by rendering the feature textures to the frames pose, $\bar{\boldsymbol{F}}_t^{l,i} = \mathcal{R}_i(\boldsymbol{F}_t^l)$ and use these rendered features as, well as the extracted keyframe pre-attention features $\boldsymbol{f}_t^{\text{KF}}$ to compute a noise prediction for the frame by denoising its latent with feature injection, denoted like so:

$$\hat{\boldsymbol{\epsilon}}_i \leftarrow \boldsymbol{\epsilon}_\theta^{\text{inj}}(\boldsymbol{z}_t^i, t, \boldsymbol{c}, \boldsymbol{d}_i, \{\boldsymbol{f}_t^l, \bar{\boldsymbol{F}}_t^{l,i}\}). \tag{4.4}$$

Internally, each self-attention layer in the U-Net is modified to compute keys and values from the concatenation of the image features and the injected keyframe pre-attention features $\boldsymbol{f}_t^{l,\text{KF}}$. Additionally,

the output of each attention layer is alpha blended with the rendered post attention features. The modified self-attention layer with feature injection is

$$\text{SelfAttn}^{\text{inj}}_{W_Q,W_K,W_V}(\boldsymbol{f}_t^{l,i}, \{\boldsymbol{f}_t^{l,\text{KF}}, \bar{\boldsymbol{F}}_t^{l,i}\}) = (1-\alpha)\cdot\text{Attention}(W_Q\boldsymbol{f}_t^{l,i}, W_K[\boldsymbol{f}_t^{l,i}, \boldsymbol{f}_t^{l,\text{KF}}], W_V[\boldsymbol{f}_t^{l,i}, \boldsymbol{f}_t^{l,\text{KF}}]) + \alpha\bar{\boldsymbol{F}}_t^{l,i}. \quad (4.5)$$

This combines both cross-image attention and spatial feature injection, using the keyframes as the source frames. By having each animation frame cross-attend to the keyframe features $\boldsymbol{f}_t^{l,\text{KF}}$, consistent global appearance is maintained, and by alpha-blending the self-attention output features with the rendered post-attention features $\bar{\boldsymbol{F}}_t^{(l,i)}$ spatial consistency is also maintained. Finally, we obtain the less noisy video frames by bringing them to a lower noise level with the computed noise prediction with DDIM sampling.

4.2.2. Generative Rendering Experiments

We now perform a series of experiments with our implementation of GR. Since their method relies on rendering features from one set of frames to another, we hypothesize that this will cause issues in scenes where features at some frames cannot be rendered to all other frames. This can occur when the guidance geometry contains *occlusion*, as not all frames cover the same UV space, or when the mesh appearas at different scales throughout the animation. In the following sections we validate each of these hypothesees.

Inconsistency due to Occlusion We hypothesize that GR will struggle to generate temporally consistent frames in scenes with occlusion, i.e., when different frames observe disjoint regions in the UV space. Since only a subset of keyframes is used for feature alignment at each denoising step, it is possible that the randomly selected keyframes do not cover the entire UV space. In such cases, the rendered features will be incomplete in the unseen regions, potentially leading to temporal inconsistencies. Our findings support this hypothesis. Figure 4.2 shows a video generated with GR for a scene with occlusion, as well as a visualization of the extracted keyframe features (purple border) and the rendered features for all animation frames. Due to the randomness of keyframe selection, the assembled feature texture only covers the front of the character, and the resulting rendered features contain holes in the frames showing the back of the character, resulting in temporal inconsistencies in these frames, as highlighted by the zoom windows.

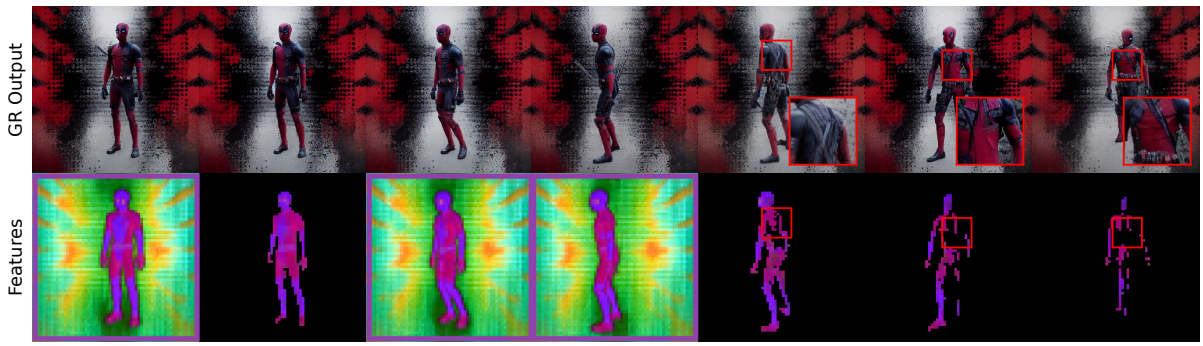


Figure 4.2: Inconsistency due to Occlusion An example GR output for a scene displaying occlusion, along with a PCA visualization of the extracted keyframe post-attention features (purple border) and rendered post-attention features for a particular time step and layer. If the randomly selected keyframes do not span the whole UV space, then the uncovered regions do not receive any spatial feature injection. In this particular scene this leads to visible inconsistency in the back of the character, highlighted by the zoom windows.

We further validate this hypothesis by generating multiple videos altering *only* the seed that determines which keyframes are selected at each denoising step. The results of this experiment are presented in figure 4.3. We observe that the choice of keyframes influences where inconsistencies appear on the mesh. For each generated video, we display the frames and highlight specific regions using zoom windows, marking them in red when inconsistencies are present and in green when they are not. As shown, the first row exhibits inconsistencies on the front of the character, the second on

the back, and the third shows no inconsistency. Since the only change across these generations is keyframe-selection seed, we conclude that the randomness of keyframe selection directly causes the observed inconsistencies.

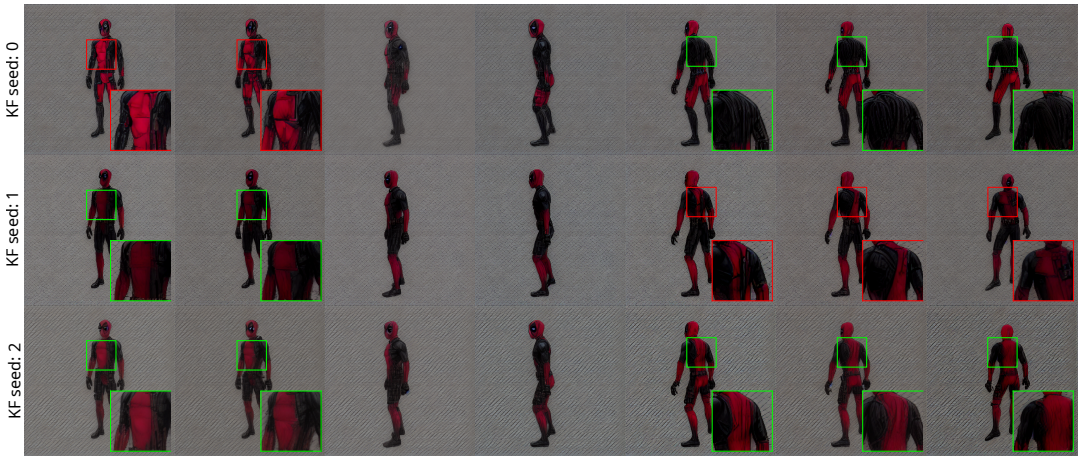


Figure 4.3: Altering Keyframe-Selection Randomness GR outputs generated with different seeds for keyframe selection. The resulting videos display inconsistency in different areas of the mesh highlighted with zoom windows, with a red border when inconsistent and purple border when consistent. The first row shows inconsistency in the front of the character, the second in the back, and the third displays no inconsistency.

We note that the severity of this issue can be partially mitigated by selecting a large number of keyframes, increasing the likelihood of covering the entire mesh. However, due to the use of extended attention for feature extraction in equation (4.1), using many keyframes becomes memory intensive and impractical, particularly for long animation sequences or scenarios where only a small subset of frames covers a specific region of the UV space. Our method avoids this limitation entirely by instead extracting features from the images used to generate a texture that is guaranteed to cover the relevant parts of the UV space.

Blurring due to Camera z-movement We further hypothesize that the GR feature-alignment strategy will struggle in scenes where the mesh appears at various scales. If, at a given denoising step, the randomly selected keyframes show the subject from far-away, then when these features are rendered to close-up frames, they will be heavily blurred. Figure 4.4 shows an example output for such a scene as well as a visualization of the extracted and rendered features. As can be seen, the extracted features depict the subject from far away, so the resulting rendered features are blurred in the close-up frames resulting in visible corruption.

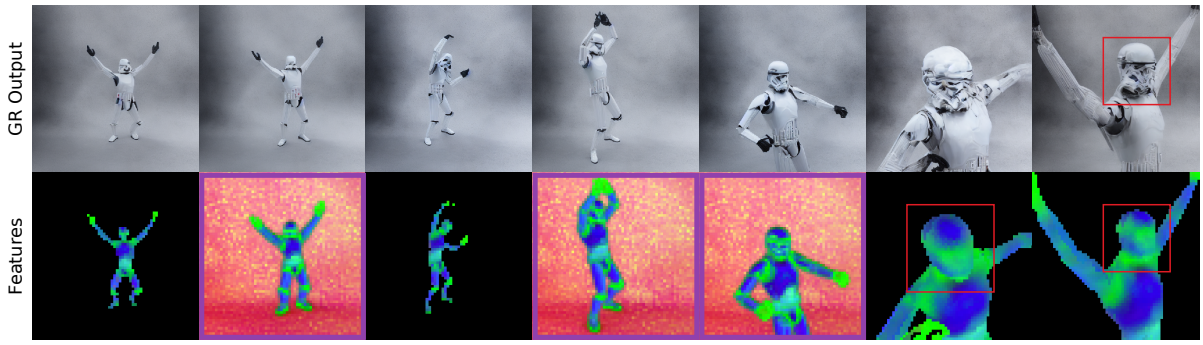


Figure 4.4: Blurring due to Camera z-movement We show an example GR output for a scene with camera z-movement, along with a PCA visualization of the keyframe post-attention features (purple border) and the rendered post-attention features at a particular time step and layer. When the randomly chosen keyframes show the mesh from far away, the rendered features in the close-up frrames are blurred, resulting in a corrupted output frame.

Unlike the occlusion issue, this failure case cannot be mitigated by increasing the amount of keyframes used, as it will manifest even if all frames are used for feature extraction, since the feature texture

is constructed by averaging all frames. Our method resolves this issue by extracting features from latents used to generate a texture, and using a heuristic to ensure we only inject features from views at a similar scale as the target frame.

4.3. TexGen applied to Video Generation

While not originally developed for our task, we note that recent work on text-driven texture generation, described in section 2.3 can be readily adapted to our setting, by simply generating a texture for the guidance mesh and rendering it to all other frames in $\mathcal{A}$ to obtain a video. In this section we explore the viability of this approach. We implement a recent method, namely TexGen [32] and evaluate its suitability video-generation through a series of experiments. We find that while effective at generating textures, the results obtained when adapted to video generation are limited by the capabilities of what a static texture can model.

4.3.1. TexGen Method

TexGen receives as input a text prompt c and a mesh $\mathcal{M} = (F, V)$ with UV mapping $\Phi : \mathcal{V} \rightarrow [0, 1]^2$, and generates a texture for the mesh according to the text prompt. TexGen follows the general pipeline of most text-driven texture generation methods, shown in figure 2.1, by placing N cameras surrounding the object and using a depth-conditioned T2I model to generate an image for each camera. To enforce consistency between the generated views, TexGen uses sequential inpainting at each denoising step. Specifically, at each denoising step t we denoise each view i sequentially, and when denoising camera i we condition on the result of denoising all previous cameras by with Blended Latent Diffusion [3]. In this section we focus on the viability of a static texture, in general, for video generation, so we don't cover the details of the method in depth. For more details, see the paper [32] and supplementary material for detailed pseudocode. Our implementation of TexGen additionally incorporates a few minor modifications over their method which we find beneficial described in subsection 5.2.1.

4.3.2. TexGen Experiments

We run experiments with our implementation of TexGen to show the limitations of texture-generation for geometry-guided video generation. We generate a video by creating a texture for the first frame and rendering it to all target frames. We additionally compare our results to GR, to highlight the limitations of this approach. Specifically, we show that a static texture cannot model deformations from the guidance geometry, or variable lighting.

Deformations From the Guidance Geometry Figure 4.5 shows an example video generated with GR and with a static texture. As can be observed, for this particular choice of prompt and geometry, GR is abele to generate video frames depicting deformations from the reference geometry around the helmet of the character. On the other hand the resulting video using a static texture depicts the head with the geometry from the mannequin.

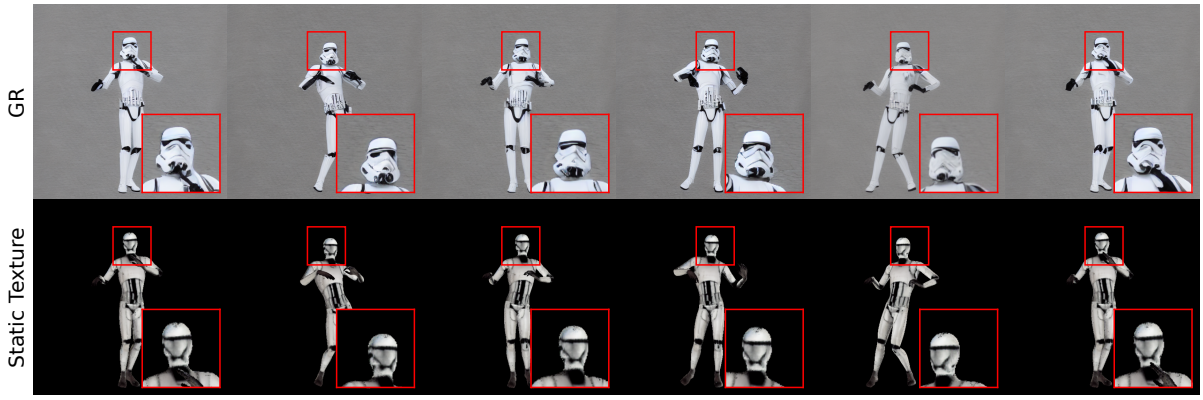


Figure 4.5: A static texture cannot model deformations For this particular choice of prompt and geometry, GR is able to generate suitable deformations from the reference geometry, but a static texture cannot model these effects.

View-Specific Appearance Variations An additional issue with using a static texture, is that it cannot model appearance variation over time. Figure 4.6 shows an example scene where this is prblematic. The scene depicts an object with view-dependent materials. GR is able to generate suitable view-variation, but a static texture cannot model these effects.

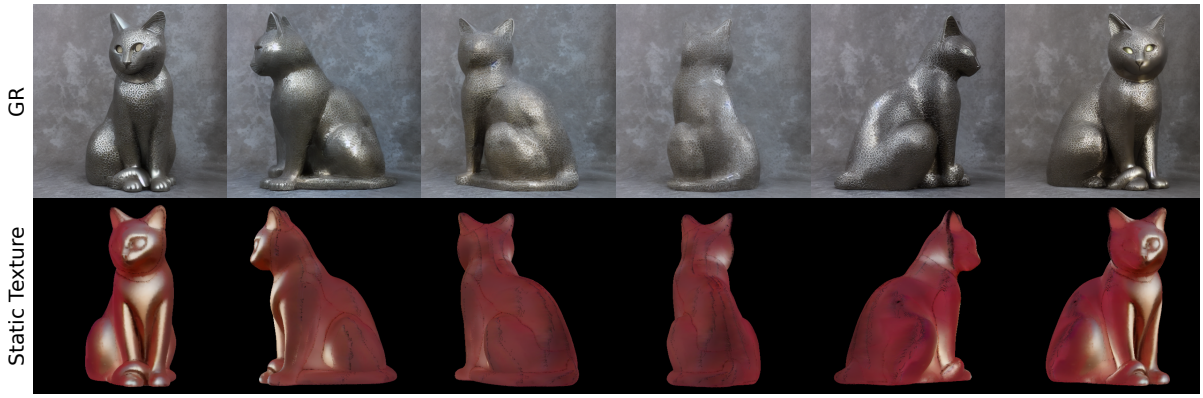


Figure 4.6: A static texture cannot model view-specific apperance variations In this scene GR is able to generate suitable appearance variation between frames, placing the specularities at different places on the mesh. A static texture on the other hand cannot model asigning a different color to the same texel across views.

4.4. Summary of Findings

In summary, we implement two existing methods and evaluate them in our setting. Our main finding is that the advantages and limitations of these two methods are complementary. GR generates frames directly with a depth-conditioned T2I model and enforces consistency via feature alignment, which has several desireable properties, such as generating reasonable deformations from the guidance geometry, and generating desireable per-view appearance variation. However, their approach suffers in scenes with occlusion or camera z-movement. On the other hand, using a static texture is simple and robust to all geometries, but when used for video generation cannot model appearance variation or deformations.

5 Method

From our experiments in chapter 4, we identified that GR is effective at generating temporally consistent videos, but is not robust to occlusion or camera z-movement. We also noted that generating and rendering a texture with TexGen is simple and robust, but when directly applied to our setting, cannot model various desireable effects. In this section, we present our method, which aims to resolve these issues by bridging both approaches into a single framework. We begin by giving an overview of our method, and then describe each of its components in depth.

5.1. Overview

Our method pipeline is shown in figure 5.1. We receive as input a text prompt $\boldsymbol{c}$, and two animated mesh-sequences $\mathcal{A}_{\text{src}}$, and $\mathcal{A}_{\text{tgt}}$ parameterized as sequences of camera-mesh pairs $\{(C_i, \mathcal{M}_i)\}_N^{i=1}$ both depicting the same mesh, with shared UV-mapping. The *target sequence* $\mathcal{A}_{\text{tgt}}$ is the animation we aim to generate a video for, and the *source sequence* $\mathcal{A}_{\text{src}}$ consists of a set of cameras facing the subject from various viewpoints and scales, which can be provided explicitly, or automatically generated (e.g. by placing cameras in a 360° around the first frame of the animation sequence at various distances/FoVs). Given these inputs our method generates a video, in two main stages.

Firstly, the source sequence is used to generate a full RGB texture for the mesh by applying a modified version of TexGen to generate a set of multi-view consistent images depicting the subject mesh, which are used to assemble an RGB texture. We additionally save the intermediate latent trajectory $z_{T\ldots0}^{\text{src}}$ of noisy latents used to generate these source views, as it will be used later for feature extraction.

The second stage uses the generated texture to generate a video following the target sequence. This is done by generating an image for each frame in the target sequence using a depth-conditioned DM where we additionally condition the generation of each target frame on the generated texture, to ensure consistent appearance across frames. This is done via two mechanisms. Firstly, the generated texture is used to initialize the noisy latents z_T^{tgt}, used to generate each target frame, which we call *Texture Driven Noise Initialization*. Next, at each denoising step from z_t^{tgt} to z_{t-1}^{tgt} we extract features from the source frames used to generate the texture and inject them into the denoising model when generating each target frame, using a feature injection approach similar to the one used by GR. Once the target frames are fully denoised, the final video is obtained by decoding each target latent into an RGB image.

In the following sections we describe each of these method components in depth. We begin by discussing the texture generation stage in section 5.2. Next, we describe our texture-driven noise initialization in section 5.3, and finally, we cover the feature extraction and injection in section 5.4.

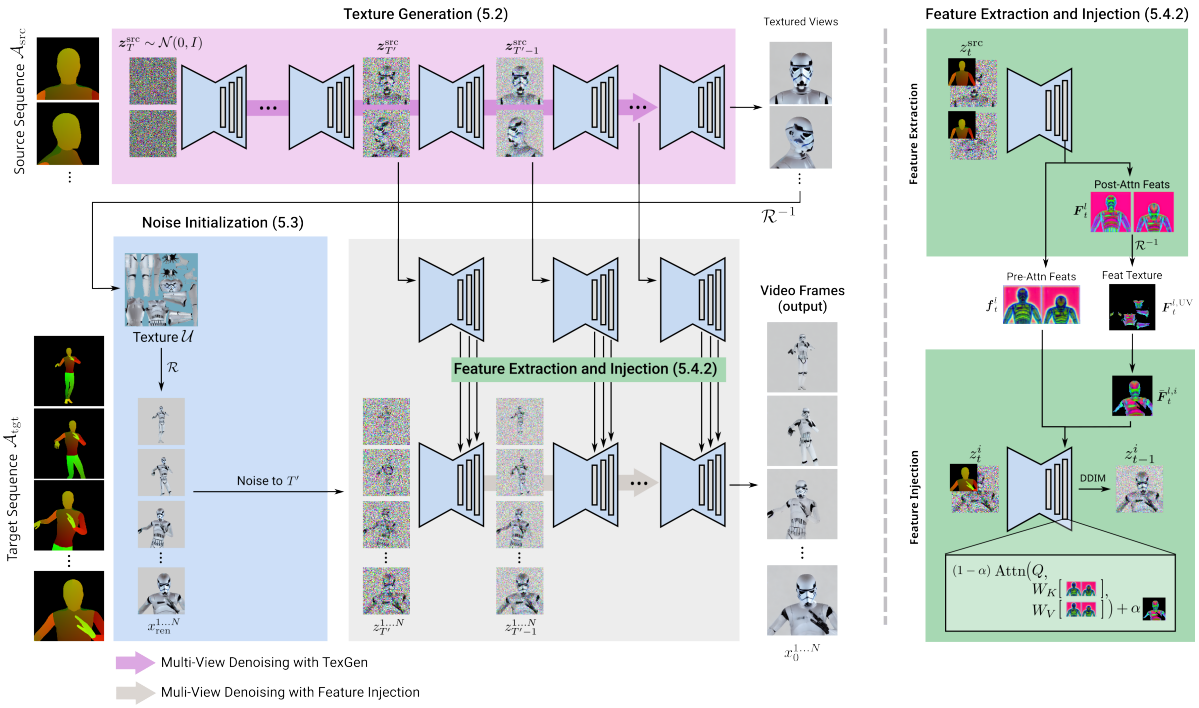


Figure 5.1: Overview of our pipeline: Our method receives as input a text prompt c and two animated mesh sequences $\mathcal{A}_{\text{src}}, \mathcal{A}_{\text{tgt}}$ depicting the same subject. Firstly we use TexGen to generate a set of multi-view consistent images of the subject according to the source sequence. Next, we use a depth-conditioned DM to generate video frames following the target sequence, and enforce temporal consistency among the generated frames using the generated texture from the previous stage.

5.2. Texture Generation

In the first stage of our method the goal is to generate a series of multi-view consistent images of the subject according to the source sequence $\mathcal{A}_{\text{src}}$. We then use the generated views to assemble an RGB texture for the mesh, and additionally save the intermediate diffusion trajectory used to generate them. We accomplish this by using a modified implementation of TexGen [32], however in principle, any text-driven texture generation method should be applicable. In the next section we briefly cover the modifications we made to TexGen to improve the quality of the generated textures.

5.2.1. Quality-based multi-view sampling

Our initial implementation of TexGen, following the method described in the paper resulted in significant seams in the generated textures. We observe that this issue is caused when texels filled at early views from a camera with poor view of the texel are enforced in subsequent views. Inspired by TexFusion [10] we incorporate a heuristic based on image space UV gradients to determine which camera to use to fill each texel. We measure camera quality using the negative Jacobian determinant magnitude of the image space UV gradients, i.e. the change in uv-coordinate per infinitesimal change in image space. This quantity is often used to determine mip-mapping scale. More specifically, for a given camera view we calculate its view quality $Q(p, q)$ at pixel (p, q) like so

$$Q(p,q) = -\left|\frac{\partial u}{\partial p} \cdot \frac{\partial v}{\partial q} - \frac{\partial u}{\partial q} \cdot \frac{\partial v}{\partial p}\right|. \tag{5.1}$$

Figure 5.2 shows a visualization of this view-quality measure for cameras arranged in a 360° ring around an object. As can be seen, areas on the mesh viewed directly and up-close have a high view-quality (blue), wheras areas on the mesh viewed at a high grazing angle are assigned a low quality (red).

We then modify the multi-view sampling algorithm described in section 3.2 of the TexGen paper [32] to only fill a texel when inverse rendering a view to the texture space, if the new view is the highest-quality view seen so far for at that texel. Later in figure 6.7, we ablate the impact of this modification.

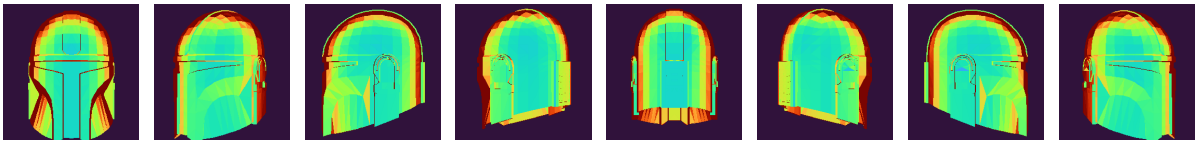


Figure 5.2: UV-Quality Maps We show the UV-quality maps computed with equation (5.1) for cameras placed in a 360° around an object. We modify the multi-view sampling method proposed in TexGen [32] to fill each texel with the camera which has the best quality at each texel according to this metric.

5.3. Texture Driven Noise Initialization

We now focus on the second stage of our method, which uses the result of the generated texture to generate the final video frames. Before generating each frame, we initialize the latent noise $z_{T'}^{\text{tgt}}$ used to generate it. We do this by rendering the generated texture $\mathcal{U}$ to the frames of the target animation

$$x_{\text{ren}}^{i} = \mathcal{R}_i(\mathcal{U}). \tag{5.2}$$

Each render is then used to obtain an initial latent for the sequence. Inspired by SDEdit [43], for each render we encode it into the latent space with the VAE encoder $\mathcal{E}$ and add noise up to an initial noise level T' according to equation (3.1). To further boost initial consistency, instead of adding independent Gaussian noise to each frame, we instead sample a UV noise texture ϵ_{UV} and render it to each target frame to have multi-view consistent initial noise. The initial noise latent $z_{T'}^{i}$ for each target frame is then:

$$z_{T'}^{i} = \sqrt{\alpha_{T'}}\,\mathcal{E}(x_{\text{ren}}^{i}) + \sqrt{1-\alpha_{T'}}\,\mathcal{R}_i(\epsilon_{\text{UV}}), \quad \epsilon_{\text{UV}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{5.3}$$

From here on, these latents are only denoised from T' to 0. The parameter T' determines the strength of the noise initialization. If $T' = T$, then $z_{T'}^{i}$ will resemble Gaussian noise, and as T' decreases the initialization becomes stronger. We find a suitable value of T' in most cases to be roughly 40% of the denoising process, which is high-enough to provide strong initialization, but low-enough to enable generating deformations and view-dependent effects for each frame. We ablate the impact of this parameter in figure 6.5.

5.4. Consistent Denoising with Feature Injection

While noise initialization boosts similarity between frames, it is not sufficient for full temporal consistency. Therefore, we use a feature manipulation approach similar to the one used by GR, but instead of sourcing features from randomly selected keyframes, we obtain them from the reverse diffusion process that generated the texturing views in the previous stage. Firstly, we automatically select a representative set of source frames for each target frame using heuristics to assess frame similarity described in subsection 5.4.1. Once a good set of source frames has been selected, we use them to perform feature injection.

5.4.1. Source Frame Selection

To perform feature injection from source views to a target frame, we first select a set of source frames that best represent the target. For each target frame, we aim to select 1-3 source frames that sufficiently cover its UV space, depict the subject at a similar scale, and show similar overall content. We select these source frames using a simple greedy algorithm that repeatedly chooses the best source frame, stopping when either the maximum number of frames is selected or the target UV space is adequately covered. To assess the similarity between a source frame $(\mathcal{M}_{\text{src}}, C_{\text{src}})$ and a target frame $(\mathcal{M}_{\text{tgt}}, C_{\text{tgt}})$, we rely on two heuristics:

Scale Distance Given a source and a target frame, we measure whether they depict the subject at a similar scale using UV gradients. For each frame, we estimate its scale by computing the mean value of its UV-quality map, as defined in equation (5.1). The scale distance between two frames is then defined as their absolute difference raised to a power

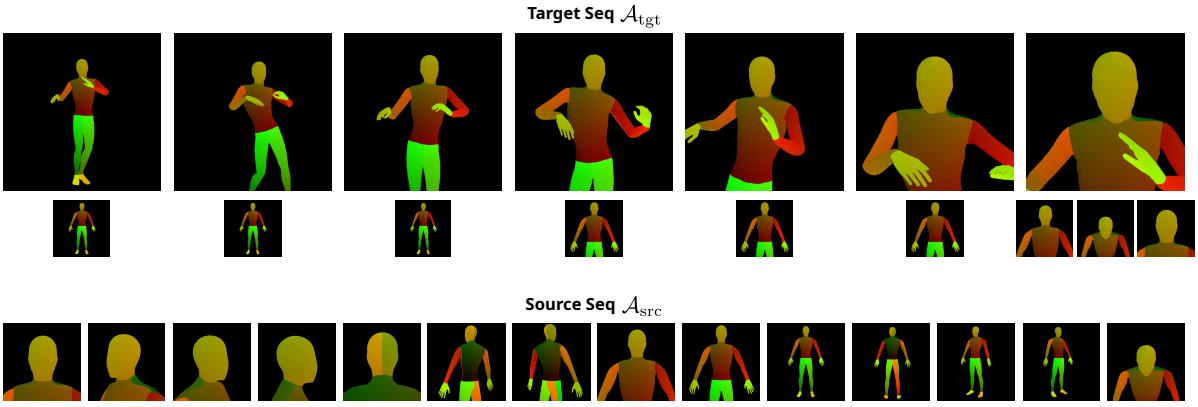


Figure 5.3: Source Camera Selection For each frame in the target sequence, we select a representative set of cameras in the source sequence, according to our camera selection heuristic. We show the selected cameras for an example source and target sequence.

$$d_{\text{scale}} = \|\text{scale}_{\text{src}} - \text{scale}_{\text{tgt}}\|^{\gamma_{\text{scale}}}. \tag{5.4}$$

UV-IoU Distance We also employ a heuristic to assess whether two views depict similar content by measuring their overlap in UV space. Specifically, we compute the Intersection Over Union (IoU) of the regions each frame covers in the UV map, denoted as $\mathcal{U}_{\text{src}}$ and $\mathcal{U}_{\text{tgt}}$. The UV-IoU distance is then defined as

$$d_{\text{UV-IoU}} = 1 - \frac{\mathcal{U}_{\text{src}} \cap \mathcal{U}_{\text{tgt}}}{\mathcal{U}_{\text{src}} \cup \mathcal{U}_{\text{tgt}}}. \tag{5.5}$$

Figure 5.3 illustrates an example source and target sequence, along with the selected source frames for each target frame in the animation. Our heuristics ensure that the selected frames depict similar content to the target frame, at a similar scale.

5.4.2. Feature Extraction and Injection

Once a suitable set of source frames has been selected for a particular target frame i we use them to condition the denoising of z_t^i. Firstly, we extract features from the selected latents by passing them through the denoising U-Net, and at each self-attention layer l, we extract the input and output features $f_t^{l,j}$ and $F_t^{l,j}$ for each selected source frame j. We denote this feature extraction like so:

$$f_t^{l_1,\dots l_n,i\dots N}, F_t^{l_1\dots l_n,i\dots N} \leftarrow \epsilon_\theta^{\text{extr}}(x_t^{\text{src}}, t, c, d). \tag{5.6}$$

Note, that unlike GR we do not perform extended attention, and instead simply perform standard self-attention, while extracting the input and output features. The extracted features are now used to condition the generation of the target frame. Like in GR, for each layer l we assemble a feature texture from the extracted post-attention by inverse rendering them to the texture space

$$F_t^{l,\text{UV}} = \frac{1}{N} \sum_{i=1}^{N} \mathcal{R}_i^{-1}\left(F^{l,i}\right). \tag{5.7}$$

We now use the assembled feature texture and the extracted pre-attention features to denoise the target frame. We do this by rendering the feature textures to the target pose $\bar{F}_t^{l,i} = \mathcal{R}(F_t^{l,\text{UV}})$, and by providing the concatenated pre-attention features for the layer, like so

$$\hat{\epsilon}_i \leftarrow \epsilon_\theta^{\text{inj}}(z_t^i, t, c, d, \{[f_t^{l,1..n}], \bar{F}_t^{l,i}\}). \tag{5.8}$$

Where $\epsilon_\theta^{\text{inj}}$ is the denoising model with feature injection. At each self-attention layer, we perform cross-image attention and spatial feature injection with some α , according to equation (4.5). Later we ablate the inpact α in figure 6.6.

5.5. Texture Rendering and Inverse Rendering

Our method makes use of the texture rendering and inverse rendering functions $\mathcal{R}_i$ and $\mathcal{R}_i^{-1}$ we defined in section 4.1. In principle our method can be used with any geometry representation for which these operations can be defined, but in practice we use a triangle mesh $\mathcal{M} = (V, F)$ with UV mapping. In this section we give additional details about how these operations are implemented.

Texture Rendering To render a texture to from UV space to some camera, camera we employ standard rasterization techniques. Firstly, the mesh is projected to the camera plane and rasterized. The color at each pixel is then obtained by sampling the texture at its interpolated UV coordinate. Unlike most setups we do-not model any shading effects and instead directly output the sampled texture color. When rendering RGB textures, as is done during texture generation (section 5.2) and texture-driven noise initialization, (section 5.3) we employ mip-mapping and antialiasing, however for rendering feature images, as is done in section 5.4, or Gaussian noise, as is done in equation (5.3), we find it better to not use these techniques and instead use nearest neighbor filtering. We implement texture-rendering with NVDIFFRAST [38].

Texture Inverse Rendering To inverse render an image I from camera space to the UV space of a mesh we employ *texture baking*. Specifically, we compute the 2D coordinate each texel projects to in camera space, and fill each texel by sampling $\mathcal{I}$ at this coordinate. To determine the projected 2D coordinate of each texel we first rasterize the UV triangulation with a resolution matching the texture resolution to efficiently obtain, for each texel, which face it is in and its barycentric coordinates. We then obtain the 3D coordinate of each texel on the surface of the mesh by interpolating the world-space coordinates of the vertices of the texels triangle. These 3D coordinates are then mapped to the 2D camera space with the camera matrix, and we additionally perform a visibility test to only fill visible texels. By performing texture-baking instead of naively filling the texture map based on the rendered UV coordinates, we ensure every visible texel is filled regardless of image and texture resolution.

Precomputing and reusing inverse and forward mappings A naive implementation of our method can be computationally costly, as we perform rendering and inverse rendering for each frame, layer and timestep throughout the generation of a video. We note that even though the textures and images we are rendering change over time, the geometry remains fixed throughout the generation process. Thus, we can precompute rasterization fragments and texel-camera mappings once, and then to render or inverse render a texture we only need to perform sampling. We find this optimization to be critical for efficient generation, especially when using many denoising steps.

6

Experiments

In this chapter, we evaluate our method through a series of experiments. In section 6.1 we compare our method to those described in chapter 4, and show that it does not suffer from the failure cases we identified. Next, in section 6.2 we ablate several of the key components of our method, and show the impact of various hyperparameters on the generated videos.

6.1. Comparisons

We begin by comparing our method to the existing approaches described in chapter 4 with the aim of validating if our two-stage generation approach addresses the identified failure cases of GR while maintaining its desireable properties.

Baselines We use the methods described in chapter 4 as baselines, namely TexGen [32] and GR [8]. Both baselines are our best-effort reproductions of the original papers, as none of them have code available. For completeness, we additionally compare to full per-frame generation by simply using the depth-conditioned model to generate each frame independently. All methods are implemented with SD v1.5 [58] with ControlNet-Depth [83] as the backbone model. We use 15 DDIM denoising steps, with a CFG guidance scale of 7.5 in all videos, and use the same initial noise across methods, to ensure fairness in comparison.

Datasets We construct a variety of inputs depicting diverse animations, camera motions and prompts which we use to test our method. We construct these scenes using a variety of animated humans performing dance sequences obtained from Mixamo [1], as well as various static meshes sourced from PolyHaven [51] and Objaverse [14]. For meshes which do not have a UV mapping we automatically generate one using xatlas [33]. To validate our hypothesees, we group our test scenes into three categories:

1. **GR Failure Cases:** These scenes contain the failure modes discussed in subsection 4.2.2, such as occlusion or camera z-motion. In these cases, we aim to show that our method performs **on par with TexGen**.
2. **TexGen Failure Cases:** These scenes involve low-fidelity geometry and prompts where deformations or lighting variations are desirable. Here, we aim to show that our method performs **on par with GR**.
3. **Combined Failure Cases:** These scenes combine both types of failure: they benefit from deformations or variable lighting and also involve occlusion or camera z-motion. In these cases, we aim to show that our method performs **best**.

For each category, we create approximately 10 test inputs and generate three videos per input using different seeds. We qualitatively inspect selected outputs to support our hypotheses and further validate them by computing quantitative metrics over the generated videos.

6.1.1. Qualitative Comparison

For each scene category, we qualitatively compare selected example results across methods, as shown in figures 6.1 to 6.3. We use green bounding boxes and zoom windows to highlight desirable effects, and red ones to indicate undesirable artifacts. We discuss the results for each category separately:

GR Failure Cases Figure 6.1 shows results for scenes with the failure cases of GR. As seen in Figure 6.1a, in scenes with occlusion, GR produces temporally inconsistent results, as highlighted by the zoom windows. It is important to note that these specific outputs are cherry-picked for GR, as the occurrence of this artifact is stochastic, as demonstrated in figure 4.3. The remaining results are not cherry-picked. On the other hand, Figure 6.1b shows results for scenes with camera z-movement. In these cases, GR consistently produces overly blurred and distorted results in close-up views, as highlighted by the bounding boxes. Neither TexGen, nor our method exhibit any of these artifacts.

TexGen Failure Cases We now focus our attention on the failure cases of TexGen. Figure 6.2a shows results in scenes where the prompt and geometry benefit from having deformations from the guidance geometry. In these scenes, GR effectively generates these effects as highlighted by the zoom windows. TexGen on the other hand cannot model them, as it is restricted by the guidance geometry. In these scenes, our method performs on-par with GR, generating very similar deformations. Figure 6.2b instead shows scenes that benefit from view-specific lighting, due to the presence of view-dependent materials. In these cases, TexGen produces static results, while GR is able to generate minor view-dependent variations. Our method also produces reasonable view variations and further demonstrates significantly higher visual fidelity.

Combined Failure Cases Finally, figure 6.3 shows results in scenes constructed to suffer from both sets of failure cases. In these scenes our method is the only one that consistently gives good results. Videos generated with GR either suffer from inconsistency due to occlusion, or heavy blurring, and results generated with TexGen do not model deformations or lighting. Our method is able to address both of these issues simultaneously.

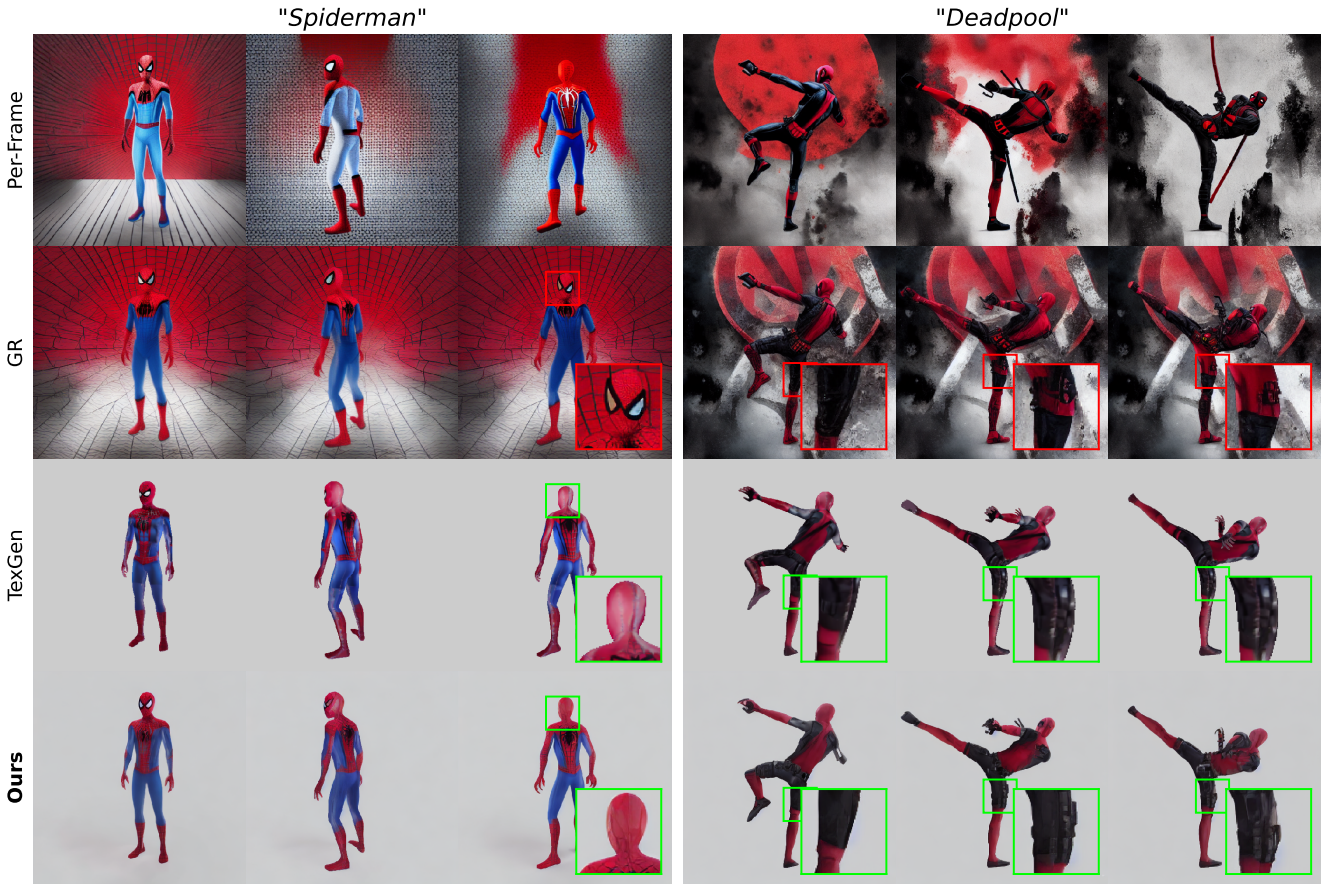
6.1.2. Quantitative Comparison

We additionally report various metrics for the generated videos. We report a collection of video metrics used in prior works [8, 11, 19, 21, 7] based on CLIP. These are Frame Consistency (FC), which is computed as the mean cosine-similarity between the CLIP-image embeddings of all frames, and Prompt Fidelity (PF), which is measured by computing the mean similarity between frame CLIP embeddings and the prompt CLIP embedding. Additionally, we report our own UV Mean Squared Error (UV-MSE) metric, computed by inverse rendering each generated frame to UV space and computing the mean squared error between consecutive frames. We compute these metrics for all generated videos and report the average value for method

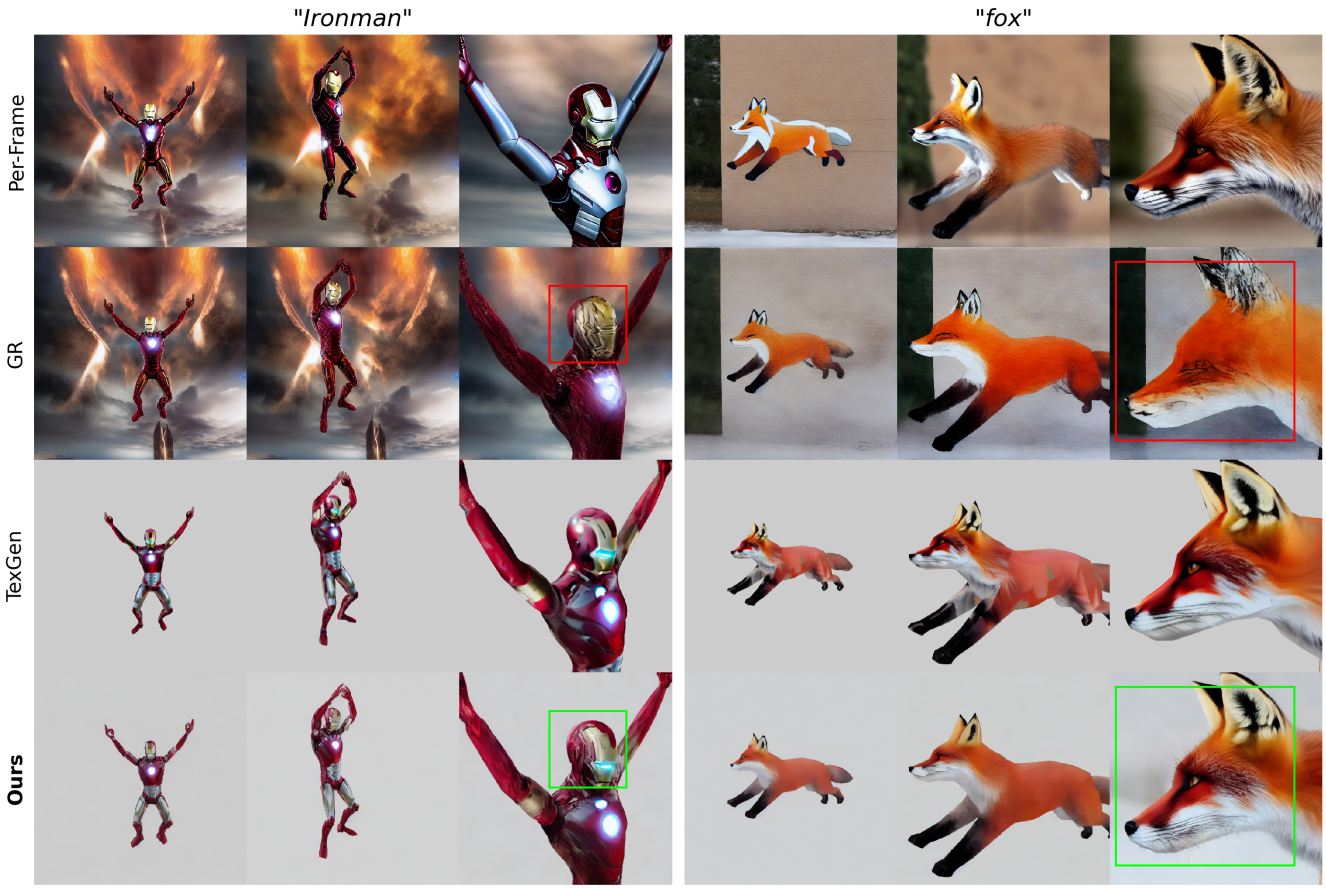
Method	Prompt Fidelity (↑)	Frame Consistency (↑)	UV-MSE (↓)
Per-Frame	0.3155	0.9090	0.0763
GR	0.3122	**0.9666**	0.0138
TexGen	0.3152	0.9678	**0.0001**
Ours	**0.3194**	0.9653	0.0134

Table 6.1: Quantiative Comparison Following prior works we compute prompt fidelity and frame consistency with CLIP. We additionally use our UV-MSE metric. Our method obtains the highest prompt-fidelity, and competetive frame-consistency.

Our method obtains the highest prompt fidelity across all baselines and frame-consistency and UV-MSE competetive with GR. Trivially, TexGen obtains the lowest UV-MSE, our method obtains lower UV-MSE than GR.

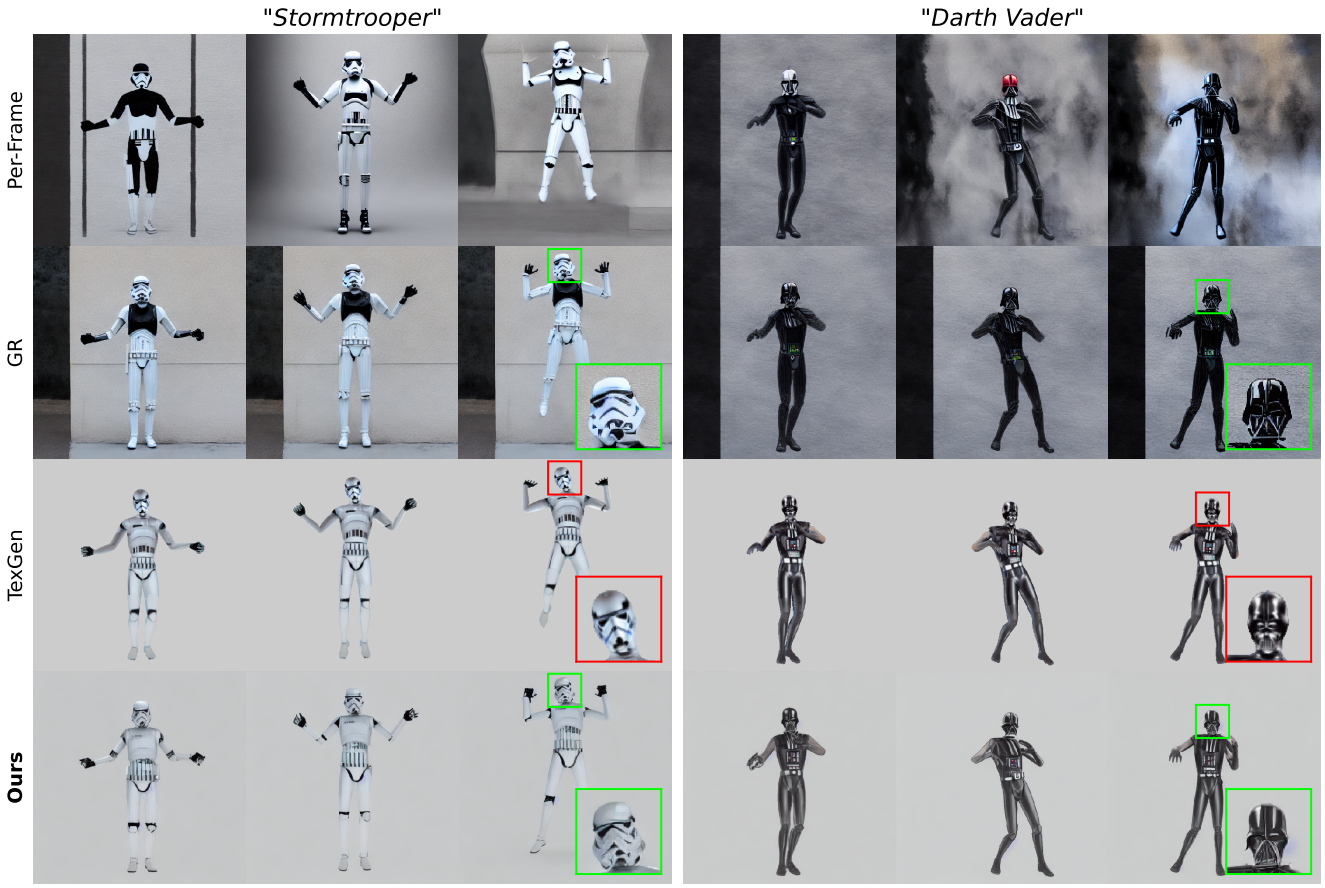


(a) Qualitative Comparisons in scenes with occlusion. Highlighted content is temporally inconsistent with GR

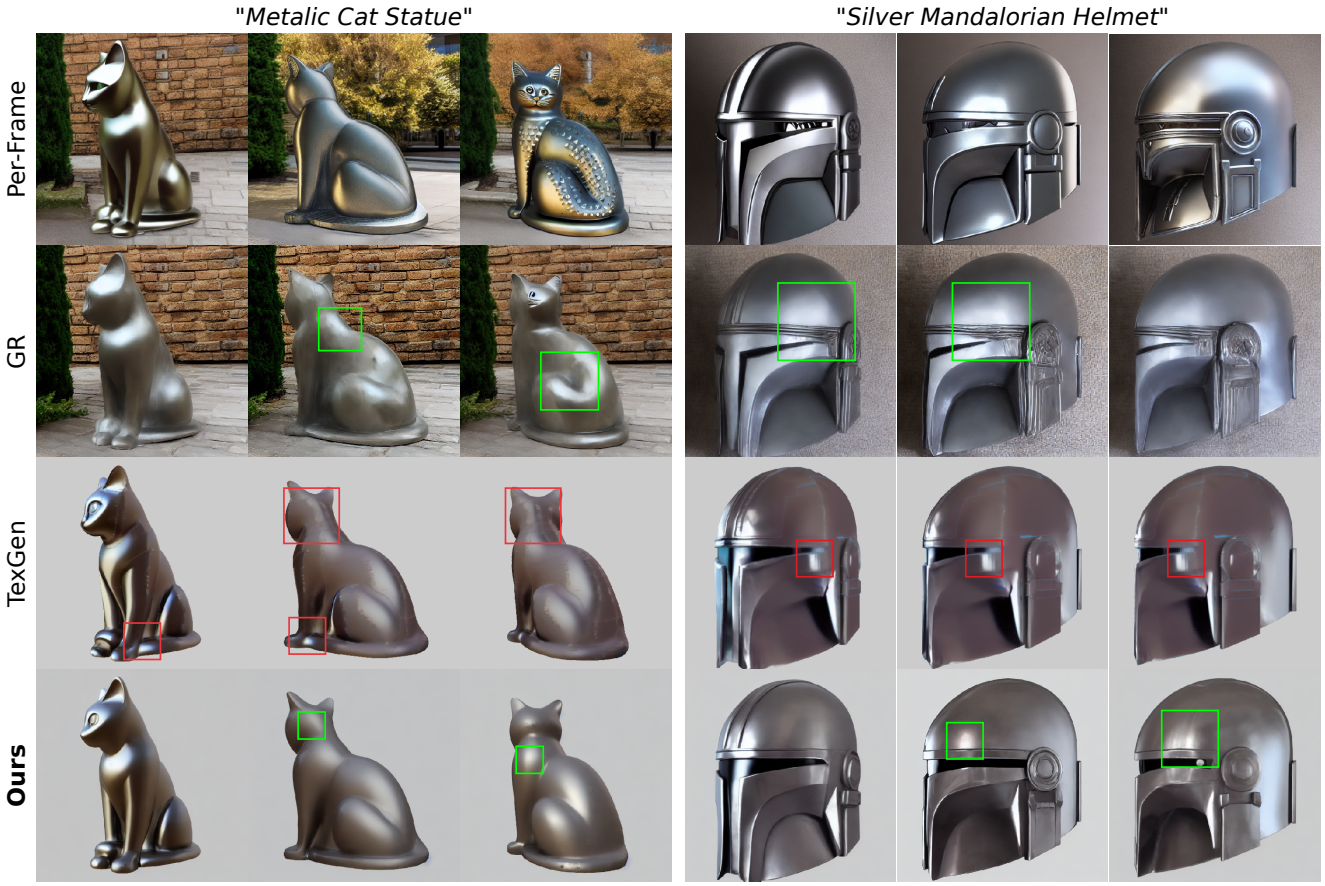


(b) Qualitative comparisons in scenes with camera z-movement. Frames displaying up-close content apperar blurry with GR.

Figure 6.1: Qualitative Comparison - GR Failure Cases Comparison across methods for scenes containing the failure cases of GR, described in subsection 4.2.2. In these scenes GR consistently produces blurry or temporally inconsistent results. Static texturing, and ours do not suffer from these limitations.



(a) Qualitative comparison in scenes with desireable defomrations. Our method generates similar defomrations to GR.



(b) Qualitative comparisons for scenes view dependent materials. Our method is able to generate plausible view-specific lighting.

Figure 6.2: Qualitative Comparisons - TexGen Failure Cases Comparison across methods for scenes which suffer from the failure cases of TexGen, identified in subsection 4.3.2. In these scenes a static-texture cannot capture desireable effects. Our method generates these effects on-par with GR.

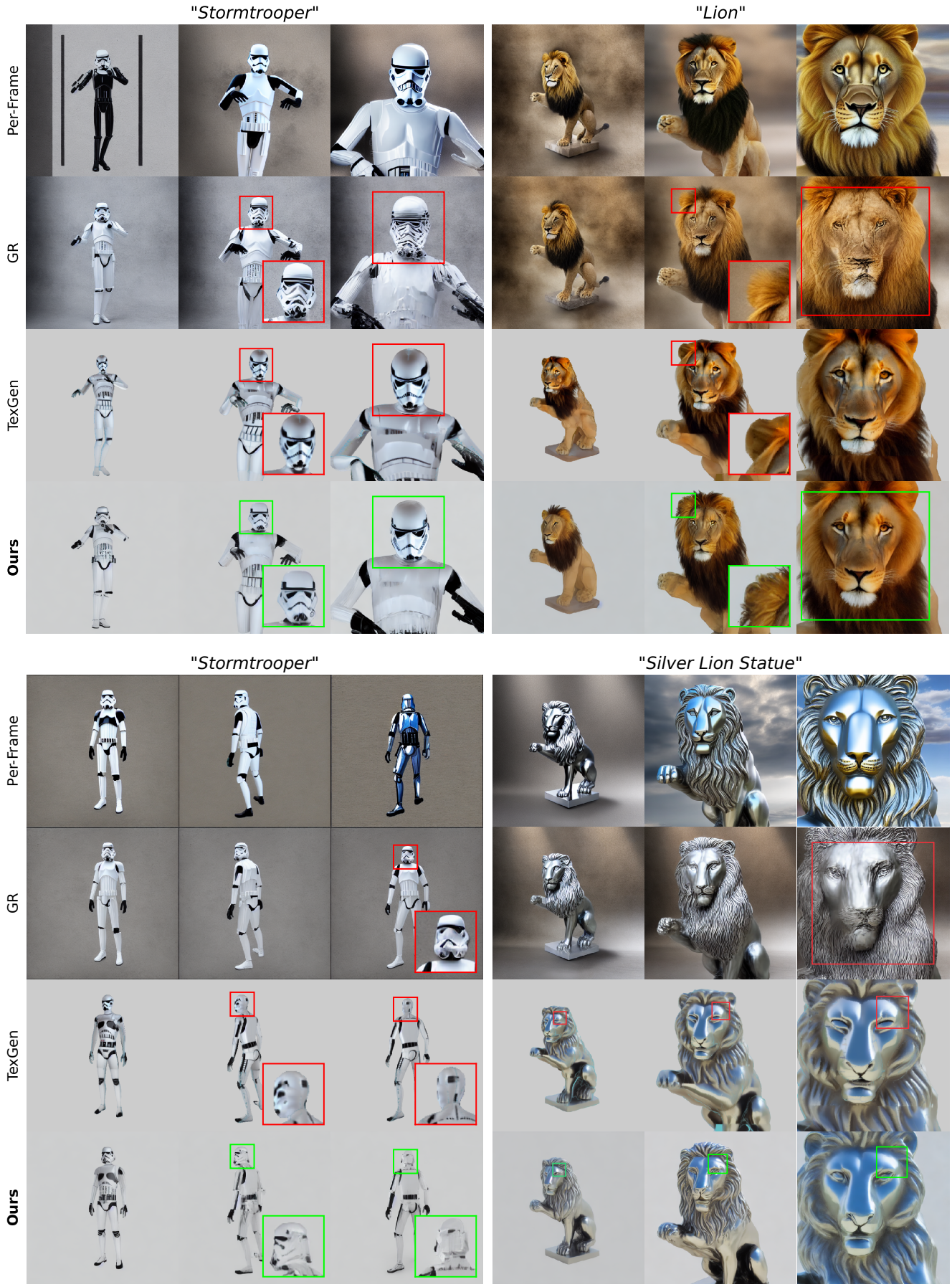


Figure 6.3: Qualitative Comparisons - Combined Failure Cases Comparison across methods for scenes displaying both the failure cases of GR and TexGen. In these scenes a static-texture cannot capture desireable effects and GR displays temporal inconsistency and blurring. Our method is able to address both of these issues simultaneously.

6.2. Ablations

To further justify the design of our method we perform a series of qualitative ablations, showing the impact of each of our individual method components. We additionally show the impact of various hyperparameters on the generated videos.

Impact of Method Components To better understand the role of each of our method components, we generate an example video with several key components disabled. Specifically, we show the generated frames under four configurations: 1) Full per-frame generation, so with all method components disabled, 2) Using texture-driven noise initialization only, 3) Using noise initialization, and additionally performing cross-image attention in equation (4.5), and finally, 4) Using our full method, with noise initialization, and full feature injection with both cross-image attention and spatial feature injection.

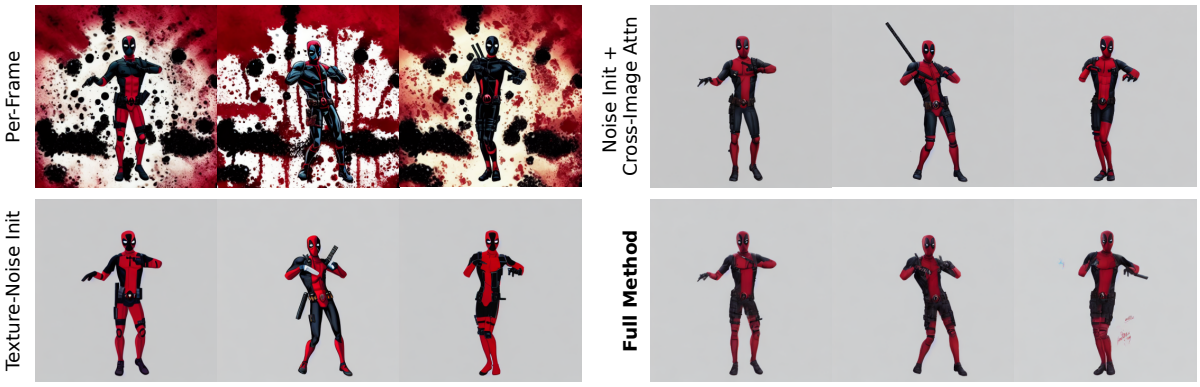


Figure 6.4: Main Ablation We generate a video disabling various method components. Texture-driven noise-initialization significantly boosts appearance similarity between frames but is insufficient for full consistency. Cross-Image attention further boosts overall appearance similarity, and finally, incorporating spatial-feature injection gives full fine-grained spatial consistency.

Our results for this experiment are summarized in figure 6.4. Texture-driven noise initialization results in significantly stronger overall appearance similarity than full per-frame generation but by itself is not sufficient. Incorporating cross-image attention, further boosts consistency, but still does not provide full fine-grained spatial consistency. Finally, incorporating spatial feature injection with features extracted from the texturing views gives fine-grained spatial consistency, at the cost of introducing minor visual artifacts in the generated frames.

Importance of Noise Initialization From figure 6.4, it is clear that feature injection is the primary component enforcing temporal consistency. This raises the question: is texture-driven noise initialization necessary, or is feature injection sufficient? To answer this, we show two example frames for a video generated under three configurations: 1) using feature injection only, with no noise, 2) directly rendering the generated texture, and 3) our full method combining feature injection with noise initialization at varying strengths T'.

The results of this experiment are presented in figure 6.5. Without noise initialization, the generated frames exhibit strong visual artifacts. We hypothesize that these artifacts arise from a strong mismatch between the injected features, derived from the texturing views, and the native features of the target frame. Incorporating noise initialization mitigates this issue, as it encourages stronger initial alignment between injected and native features. We further note the importance of selecting an appropriate initial noise level T'. While a stronger initialization reduces artifacts and improves temporal consistency, it also diminishes view-dependent effects and deformations, as shown by the zoom windows on the fur of the Lion. We find $T' = 0.4$ to provide a good trade-off in most cases, and we adopt this value in our comparisons.

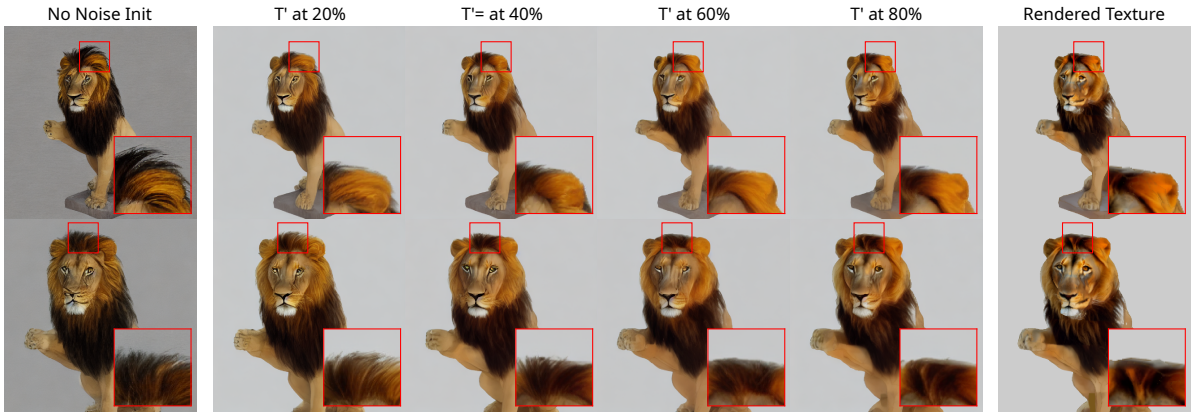


Figure 6.5: Impact of Texture-Guided Noise Initialization When we disable texture-driven noise initialization, the resulting video (left col) displays considerable artifacts due to the mismatch between the native and injected features. When noise initialization is incorporated, these artifacts are reduced significantly. The strength of the initialization is also relevant. When using a stronger initialization, the artifacts are weaker, but the desireable deformations and view-specific lighting are weakened as well.

Impact of Feature Injection Alpha We further ablate the α parameter used to blend rendered features from the texturing views to the animation views in equation (4.5). Figure 6.6 shows two example videos generated with various α values, with the appropriate alpha value for each video shown in green. Higher α values promote stronger temporal consistency, at the cost of weakening the generated lighting and deformations. On the other hand, a smaller alpha enables desireable view-variation, but may introduce undesireable temporal inconsistency. The right choice of α is scene-dependent, and we leave it as an open parameter for the artist to adjust.

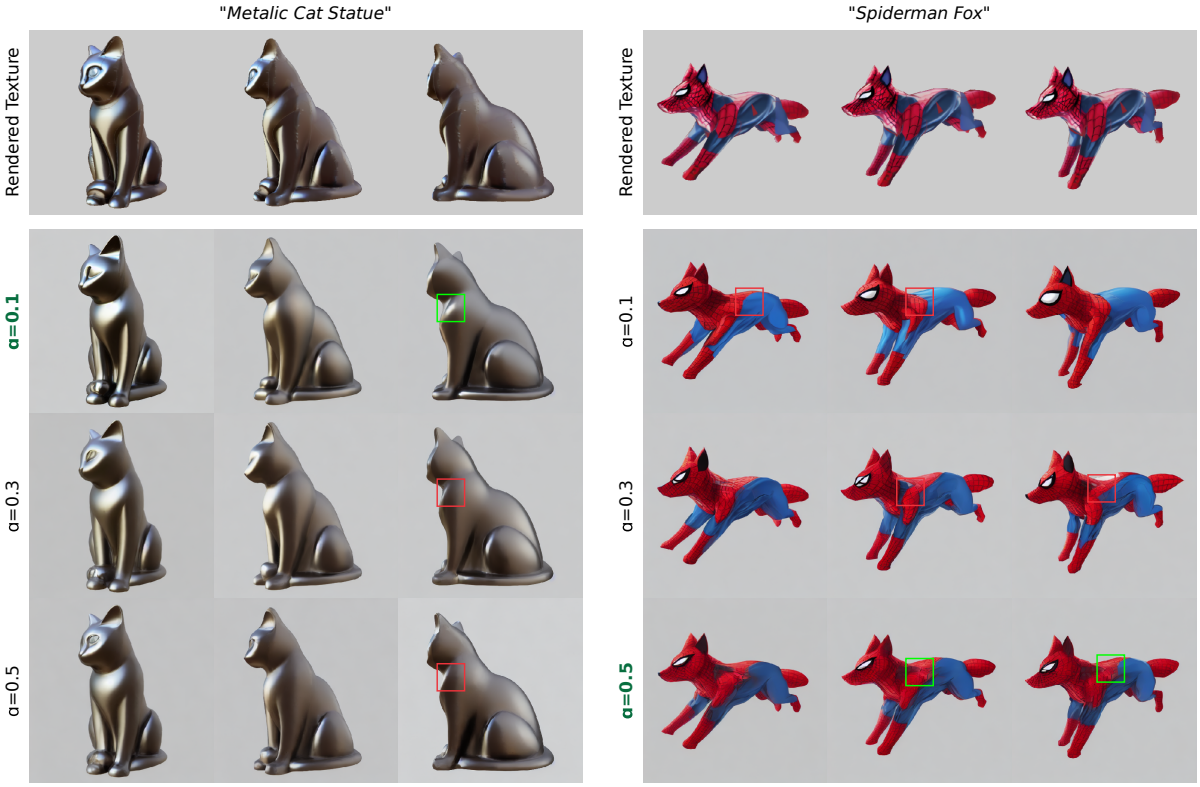


Figure 6.6: Impact of Feature Blend Alpha We show two example generated videos with varying α values. A higher alpha results in stronger consistency, at the cost of weakening the generated deformations and lighting. The right choice of alpha depends on the scene.

Importance of Quality-Based Texture Generation Lastly, we ablate the importance of our quality-based texturing described in subsection 5.2.1. Our original implementation of TexGen, following the methodology in the paper, produced noticeable seams in the generated textures. We addressed this issue by using a heuristic based on image space UV gradients to select the optimal camera for each texel. As shown in figure 6.7, this modification substantially improves the robustness of texture generation, significantly reducing visible artifacts.

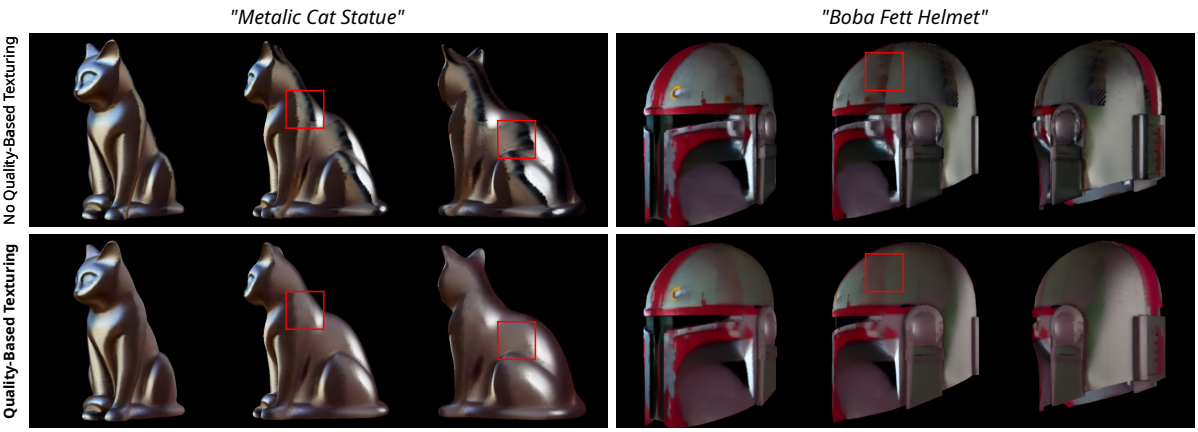


Figure 6.7: Importance of Quality-Based Texturing Our initial implementation of TexGen exhibited notable seams in the generated textures. By using our quality-based texturing approach described in subsection 5.2.1, our generated textures are considerably stronger.

7

Discussion And Conclusion

We now discuss the results from the limitations of our method and give recomendations towards future work. We finish this section with concluding remarks about our work.

7.1. Limitations and Future Work

Our method successfully addresses the failure cases of GR [8] and TexGen [32] we identified in chapter 4, however it still suffers from several limitations, some of which are inherited from GR, and others intrinsic to our approach. In this section we explain these limitations, and indicate directions for future research.

Lack of Meaningful Quantitative Evaluation In subsection 6.1.2 we use the FC and PF metrics based on CLIP, following convention set by prior work [8, 7, 11, 21, 19] as well as our UV-MSE metric to quantiatively evaluate our method. However, due to the subjective nature of our work, these metrics do not accurately capture the goals of our method, hence the majority of our evaluation and ablations are done qualitatively. To adequately assess our method a user study would be neccesary.

Pixel-Level Inconsistencies Our method is implemented using an LDM [58] but enforces temporal consistency only in the low-resolution latent space. Since each frame's latent is decoded to RGB independently, minor inconsistencies in the latent space are amplified, appearing as noticeable flicker in the decoded RGB videos. This effect is illustrated in figure 7.1, where the zoomed-in head shows slight variations across frames. This limitation is inherited from GR, but it could potentially be mitigated with techniques such as Pixel-Wise Guidance [17] or the "guided latent update" in [11], which optimize the latents during generation to encourage similarity in RGB space.

Figure 7.1: Pixel-Level Inconsistencies Our method only operates on the low-dimensional latent space of the LDM, so minor inconsistencies are inflated into noteicable inconsistencies in the RGB space.

Physically Inaccurate Lighting Our method distinguishes itself from static texturing by its ability to generate plausible lighting for each frame, as shown in Figure 6.2b. However, similar to other image and video generation techniques based on diffusion models, the generated lighting is not physically accurate. Figure 7.2 shows an example video where, although the lighting appears visually plausible, it does not adhere to physical correctness.

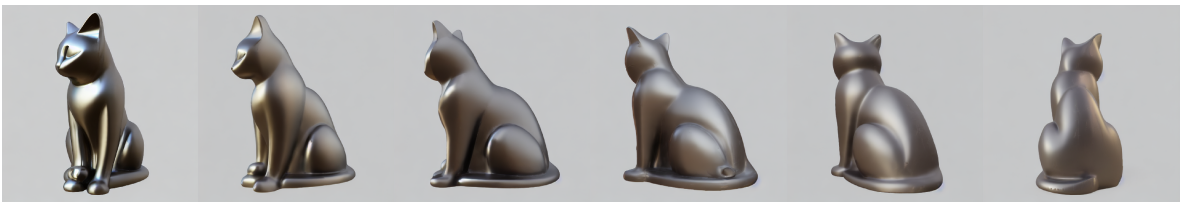


Figure 7.2: Physically Unrealistic Lighting Our method can generate plausible lighting for view-dependent materials, but is not physically realistic.

Background Inconsistency We observe that in some scenes, our method produces inconsistent artifacts in the background, as highlighted in figure 7.3. This limitation is also inherited from GR, and occurs because our mehtod only explicitly enforces temporal consistency in the regions of the image covered by the guidance geometry. Background consistency is only enforced via noise initialization and cross-image attention, which do not enforce fine-grained consistency. This, of course, could be resolved by simply using guidance geometry with multiple meshes, and ensuring that every pixel is covered by at least one mesh, by placing a floor, or background plane.

Figure 7.3: Background inconsistency Our method occasionally produces inconsistent artifacts in the background. This is because we only explicitly enforce temporal consistency in the areas covered by the mesh. This could be addressed by using guidance geometry which fully covers the image plane, by placing a plane/ground in the background of the scene.

Reliance on generated texture Since our method employs TexGen as a pre-processing step to generate a rough texture for the mesh, the quality of our generated video depends on the generated texture. In most cases, the generated texture is sound, however TexGen can occasionally produce unnatural results, which are then present in the generated video as shown in figure 7.4.

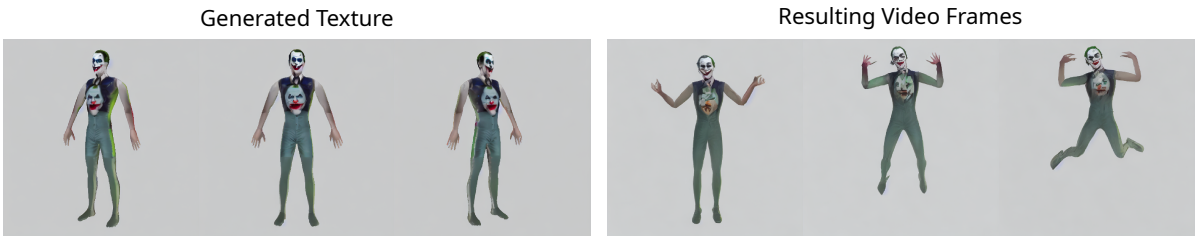


Figure 7.4: Degenerate Texture Our method generates a texture as a pre-processing step. If this generated texture is incoherent, the resulting video inherits these issues.

Reliance on a T2I model Our method relies exclusively on a depth-conditioned T2I model, both to generate the initial texture and to produce the final video frames. This is advantageous, as T2I models and datasets are widely available and computationally efficient. However, during the course of this thesis, both 3D generation and video generation models, trained natively on 3D and video data, have advanced significantly [49, 60, 23, 80]. Leveraging more recent models would likely improve visual quality and address several of the limitations discussed above. Specifically, video-models do not suffer from per-pixel inconsistencies [7]. While expanding our method to these models is nontrivial, our core idea of first generating a textured 3D representation and then using it as guidance for controllable video generation could, in principle, be adapted to more recent models, potentially enabling stronger

geometric coherence and temporal consistency than existing methods based on video models [40, 19]. We believe this represents an exciting direction for future research.

7.2. Conclusion

In conclusion, we introduce a novel method for reometry-guided Text To Video (T2V), which enables generating videos controlled by a guidance animated mesh sequence. Our approach first generates a rough texture for the input geometry using an existing texture-generation method, and then leverages a depth-conditioned T2I model to synthesize video frames, enforcing temporal consistency betweeen frames by conditioning each generated frame on the previously generated texture, using a feature-manipulation strategy similar to the one proposed by GR. Our experiments show that our method can produce videos with desireable deformations and lighting effects comparable to results obtained with GR, while at the same time, being robust to several of the failure cases of GR. We further justify the importance of each of our method components through a series of ablations. This work represents a step forward in the emerging field of geometry-guided video generation, and we hope it will motivate further research in this direction.

References

[1] Adobe Inc. *Mixamo: Web-based character auto-rigging and animation*. `https://www.mixamo.com`. 2025.

[2] Yuval Alaluf, Daniel Garibi, Or Patashnik, Hadar Averbuch-Elor, and Daniel Cohen-Or. *Cross-Image Attention for Zero-Shot Appearance Transfer*. Nov. 6, 2023. (Visited on 10/19/2024). Pre-published.

[3] Omri Avrahami, Ohad Fried, and Dani Lischinski. "Blended Latent Diffusion". In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–11. (Visited on 12/05/2024).

[4] Omri Avrahami, Dani Lischinski, and Ohad Fried. "Blended Diffusion for Text-driven Editing of Natural Images". In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA: IEEE, June 2022, pp. 18187–18197. (Visited on 03/30/2025).

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. May 19, 2016. (Visited on 10/23/2024). Pre-published.

[6] Jingzhi Bao, Xueting Li, and Ming-Hsuan Yang. *Tex4D: Zero-shot 4D Scene Texturing with Video Diffusion Models*. Oct. 14, 2024. (Visited on 10/18/2024). Pre-published.

[7] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. *Align Your Latents: High-Resolution Video Synthesis with Latent Diffusion Models*. Dec. 28, 2023. (Visited on 10/28/2024). Pre-published.

[8] Shengqu Cai, Duygu Ceylan, Matheus Gadelha, Chun-Hao Paul Huang, Tuanfeng Yang Wang, and Gordon Wetzstein. *Generative Rendering: Controllable 4D-Guided Video Generation with 2D Diffusion Models*. Dec. 3, 2023. (Visited on 12/04/2024). Pre-published.

[9] Mingdeng Cao, Xintao Wang, Zhongang Qi, Ying Shan, Xiaohu Qie, and Yinqiang Zheng. "MasaCtrl: Tuning-Free Mutual Self-Attention Control for Consistent Image Synthesis and Editing". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023 IEEE/CVF International Conference on Computer Vision (ICCV). Paris, France: IEEE, Oct. 1, 2023, pp. 22503–22513. (Visited on 03/27/2025).

[10] Tianshi Cao, Karsten Kreis, Sanja Fidler, Nicholas Sharp, and Kangxue Yin. *TexFusion: Synthesizing 3D Textures with Text-Guided Image Diffusion Models*. Oct. 20, 2023. (Visited on 12/04/2024). Pre-published.

[11] Duygu Ceylan, Chun-Hao Paul Huang, and Niloy J. Mitra. *Pix2Video: Video Editing Using Image Diffusion*. Mar. 22, 2023. (Visited on 06/25/2024). Pre-published.

[12] Dave Zhenyu Chen, Yawar Siddiqui, Hsin-Ying Lee, Sergey Tulyakov, and Matthias Nießner. "Text2Tex: Text-driven Texture Synthesis via Diffusion Models". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023 IEEE/CVF International Conference on Computer Vision (ICCV). Paris, France: IEEE, Oct. 1, 2023, pp. 18512–18522. (Visited on 08/16/2024).

[13] Jiwoo Chung, Sangeek Hyun, and Jae-Pil Heo. *Style Injection in Diffusion: A Training-free Approach for Adapting Large-scale Diffusion Models for Style Transfer*. Mar. 20, 2024. (Visited on 11/12/2024). Pre-published.

[14] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. *Objaverse: A Universe of Annotated 3D Objects*. Dec. 15, 2022. (Visited on 05/12/2025). Pre-published.

[15] Prafulla Dhariwal and Alex Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: ().

[16] Niladri Shekhar Dutt, Sanjeev Muralikrishnan, and Niloy J Mitra. "Diffusion 3D Features (Diff3F): Decorating Untextured Shapes with Distilled Semantic Features". In: (Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) June 2024), pp. 4494–4504.

[17] Abdelrahman Eldesokey and Peter Wonka. *LatentMan: Generating Consistent Animated Characters Using Image Diffusion Models*. June 2, 2024. (Visited on 08/24/2024). Pre-published.

[18] Dave Epstein, Allan Jabri, Ben Poole, Alexei A. Efros, and Aleksander Holynski. *Diffusion Self-Guidance for Controllable Image Generation*. June 11, 2023. (Visited on 11/24/2024). Pre-published.

[19] Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. *Structure and Content-Guided Video Synthesis with Diffusion Models*. Feb. 6, 2023. (Visited on 10/20/2024). Pre-published.

[20] Patrick Esser, Robin Rombach, and Björn Ommer. *Taming Transformers for High-Resolution Image Synthesis*. June 23, 2021. (Visited on 10/18/2024). Pre-published.

[21] Michal Geyer, Omer Bar-Tal, Shai Bagon, and Tali Dekel. *TokenFlow: Consistent Diffusion Features for Consistent Video Editing*. Nov. 20, 2023. (Visited on 09/10/2024). Pre-published.

[22] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. *Generative Adversarial Networks*. June 10, 2014. (Visited on 05/06/2025). Pre-published.

[23] Google Cloud / DeepMind. *Veo 3 is now available for everyone in public preview on Vertex AI*. `https://cloud.google.com/blog/products/ai-machine-learning/veo-3-available-for-everyone-in-public-preview-on-vertex-ai`. June 2025.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. Dec. 10, 2015. (Visited on 04/20/2024). Pre-published.

[25] Eric Hedlin, Gopal Sharma, Shweta Mahajan, Xingzhe He, Hossam Isack, Abhishek Kar Helge Rhodin, Andrea Tagliasacchi, and Kwang Moo Yi. *Unsupervised Keypoints from Pretrained Diffusion Models*. May 21, 2024. (Visited on 08/27/2024). Pre-published.

[26] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. *Prompt-to-Prompt Image Editing with Cross Attention Control*. Aug. 2, 2022. (Visited on 10/22/2024). Pre-published.

[27] Amir Hertz, Andrey Voynov, Shlomi Fruchter, and Daniel Cohen-Or. "Style Aligned Image Generation via Shared Attention". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Seattle, WA, USA: IEEE, June 16, 2024, pp. 4775–4785. (Visited on 10/22/2024).

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. Dec. 16, 2020. (Visited on 06/09/2024). Pre-published.

[29] Jonathan Ho and Tim Salimans. *Classifier-Free Diffusion Guidance*. July 26, 2022. (Visited on 10/12/2024). Pre-published.

[30] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. *Video Diffusion Models*. June 22, 2022. (Visited on 10/19/2024). Pre-published.

[31] Jonathan Ho et al. *Imagen Video: High Definition Video Generation with Diffusion Models*. Oct. 5, 2022. (Visited on 01/15/2025). Pre-published.

[32] Dong Huo, Zixin Guo, Xinxin Zuo, Zhihao Shi, Juwei Lu, Peng Dai, Songcen Xu, Li Cheng, and Yee-Hong Yang. "TexGen: Text-Guided 3D Texture Generation with Multi-view Sampling and Resampling". In: arXiv, Aug. 2, 2024. (Visited on 08/16/2024).

[33] jpcy. *xAtlas: Mesh parameterization / UV unwrapping library*. `https://github.com/jpcy/xatlas`. 2025.

[34] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics* 42.4 (Aug. 2023), pp. 1–14. (Visited on 05/06/2024).

[35] Levon Khachatryan, Andranik Movsisyan, Vahram Tadevosyan, Roberto Henschel, Zhangyang Wang, Shant Navasardyan, and Humphrey Shi. *Text2Video-Zero: Text-to-Image Diffusion Models Are Zero-Shot Video Generators*. Mar. 23, 2023. (Visited on 08/24/2024). Pre-published.

[36] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. July 10, 2018. (Visited on 06/27/2025). Pre-published.

[37] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. Dec. 10, 2022. (Visited on 03/30/2025). Pre-published.

[38] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. *Modular Primitives for High-Performance Differentiable Rendering*. Nov. 6, 2020. (Visited on 06/16/2025). Pre-published.

[39] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. *Magic3D: High-Resolution Text-to-3D Content Creation*. Mar. 25, 2023. (Visited on 08/14/2024). Pre-published.

[40] Han Lin, Jaemin Cho, Abhay Zala, and Mohit Bansal. *Ctrl-Adapter: An Efficient and Versatile Framework for Adapting Diverse Controls to Any Diffusion Model*. May 24, 2024. (Visited on 11/11/2024). Pre-published.

[41] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. *Flow Matching for Generative Modeling*. Feb. 8, 2023. (Visited on 01/21/2025). Pre-published.

[42] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. *Zero-1-to-3: Zero-shot One Image to 3D Object*. Mar. 20, 2023. (Visited on 06/29/2025). Pre-published.

[43] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. *SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations*. Jan. 5, 2022. (Visited on 11/23/2024). Pre-published.

[44] Gal Metzer, Elad Richardson, Or Patashnik, Raja Giryes, and Daniel Cohen-Or. *Latent-NeRF for Shape-Guided Generation of 3D Shapes and Textures*. Nov. 14, 2022. (Visited on 06/16/2024). Pre-published.

[45] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. Aug. 3, 2020. (Visited on 04/04/2024). Pre-published.

[46] Ron Mokady, Amir Hertz, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. *Null-Text Inversion for Editing Real Images Using Guided Diffusion Models*. Nov. 17, 2022. (Visited on 04/25/2025). Pre-published.

[47] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. Mar. 8, 2022. (Visited on 06/29/2025). Pre-published.

[48] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. *Pixel Recurrent Neural Networks*. Aug. 19, 2016. (Visited on 06/27/2025). Pre-published.

[49] OpenAI. *Sora is here: Introducing OpenAI's Text-to-Video Model*. `https://openai.com/index/sora-is-here/`. Dec. 2024.

[50] Or Patashnik, Rinon Gal, Daniel Cohen-Or, Jun-Yan Zhu, and Fernando De la Torre. *Consolidating Attention Features for Multi-view Image Editing*. Feb. 22, 2024. (Visited on 11/23/2024). Pre-published.

[51] *Poly Haven*. `https://polyhaven.com`. 2025.

[52] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. *DreamFusion: Text-to-3D Using 2D Diffusion*. Sept. 29, 2022. (Visited on 06/18/2024). Pre-published.

[53] Chenyang Qi, Xiaodong Cun, Yong Zhang, Chenyang Lei, Xintao Wang, Ying Shan, and Qifeng Chen. *FateZero: Fusing Attentions for Zero-shot Text-based Video Editing*. Oct. 11, 2023. (Visited on 10/17/2024). Pre-published.

[54] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. Feb. 26, 2021. (Visited on 05/23/2024). Pre-published.

[55] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. *Zero-Shot Text-to-Image Generation*. Feb. 26, 2021. (Visited on 06/29/2025). Pre-published.

[56] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. *Generating Diverse High-Fidelity Images with VQ-VAE-2*. June 2, 2019. (Visited on 05/02/2025). Pre-published.

[57] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. *TEXTure: Text-Guided Texturing of 3D Shapes*. Feb. 3, 2023. (Visited on 06/16/2024). Pre-published.

[58] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. Apr. 13, 2022. (Visited on 06/16/2024). Pre-published.

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. May 18, 2015. (Visited on 04/18/2024). Pre-published.

[60] Runway AI. *Introducing Runway Gen-4*. `https://runwayml.com/research/introducing-runway-gen-4`. Mar. 2025.

[61] Seyedmorteza Sadat, Otmar Hilliges, and Romann M. Weber. *Eliminating Oversaturation and Artifacts of High Guidance Scales in Diffusion Models*. Oct. 3, 2024. (Visited on 02/17/2025). Pre-published.

[62] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. May 23, 2022. (Visited on 06/29/2025). Pre-published.

[63] Christoph Schuhmann et al. *LAION-5B: An Open Large-Scale Dataset for Training next Generation Image-Text Models*. Oct. 16, 2022. (Visited on 10/18/2024). Pre-published.

[64] Yichun Shi, Peng Wang, Jianglong Ye, Mai Long, Kejie Li, and Xiao Yang. *MVDream: Multi-view Diffusion for 3D Generation*. Apr. 18, 2024. (Visited on 11/14/2024). Pre-published.

[65] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. *Deep Unsupervised Learning Using Nonequilibrium Thermodynamics*. Nov. 18, 2015. (Visited on 06/16/2024). Pre-published.

[66] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models*. Oct. 5, 2022. (Visited on 06/16/2024). Pre-published.

[67] Yang Song and Stefano Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. Oct. 10, 2020. (Visited on 07/26/2024). Pre-published.

[68] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. Feb. 10, 2021. (Visited on 02/18/2025). Pre-published.

[69] Nick Stracke, Stefan Andreas Baumann, Kolja Bauer, Frank Fundel, and Björn Ommer. *CleanDIFT: Diffusion Features without Noise*. Dec. 4, 2024. (Visited on 12/06/2024). Pre-published.

[70] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. *DreamGaussian: Generative Gaussian Splatting for Efficient 3D Content Creation*. Mar. 29, 2024. (Visited on 08/14/2024). Pre-published.

[71] Luming Tang, Menglin Jia, Qianqian Wang, Cheng Perng Phoo, and Bharath Hariharan. *Emergent Correspondence from Image Diffusion*. Dec. 6, 2023. (Visited on 09/04/2024). Pre-published.

[72] Yoad Tewel, Omri Kaduri, Rinon Gal, Yoni Kasten, Lior Wolf, Gal Chechik, and Yuval Atzmon. *Training-Free Consistent Text-to-Image Generation*. May 30, 2024. (Visited on 01/10/2025). Pre-published.

[73] Fabio Tosi, Pierluigi Zama Ramirez, and Matteo Poggi. *Diffusion Models for Monocular Depth Estimation: Overcoming Challenging Conditions*. July 23, 2024. (Visited on 06/29/2025). Pre-published.

[74] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. *Plug-and-Play Diffusion Features for Text-Driven Image-to-Image Translation*. Nov. 22, 2022. (Visited on 10/17/2024). Pre-published.

[75] Lukas Uzolas, Elmar Eisemann, and Petr Kellnhofer. *Surface-Aware Distilled 3D Semantic Features*. Mar. 24, 2025. (Visited on 03/25/2025). Pre-published.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. Aug. 1, 2023. (Visited on 03/23/2024). Pre-published.

[77] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. *ProlificDreamer: High-Fidelity and Diverse Text-to-3D Generation with Variational Score Distillation*. Nov. 22, 2023. (Visited on 10/22/2024). Pre-published.

[78] Jay Zhangjie Wu, Yixiao Ge, Xintao Wang, Weixian Lei, Yuchao Gu, Yufei Shi, Wynne Hsu, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. *Tune-A-Video: One-Shot Tuning of Image Diffusion Models for Text-to-Video Generation*. Mar. 17, 2023. (Visited on 06/26/2024). Pre-published.

[79] Shuai Yang, Yifan Zhou, Ziwei Liu, and Chen Change Loy. *Rerender A Video: Zero-Shot Text-Guided Video-to-Video Translation*. Sept. 17, 2023. (Visited on 08/24/2024). Pre-published.

[80] Yuanbo Yang, Jiahao Shao, Xinyang Li, Yujun Shen, Andreas Geiger, and Yiyi Liao. "Prometheus: 3D-Aware Latent Diffusion Models for Feed-Forward Text-to-3D Scene Generation". In: *CoRR* abs/2412.21117 (2024).

[81] Denis Zavadski, Johann-Friedrich Feiden, and Carsten Rother. *ControlNet-XS: Rethinking the Control of Text-to-Image Diffusion Models as Feedback-Control Systems*. Aug. 12, 2024. (Visited on 02/10/2025). Pre-published.

[82] Junyi Zhang, Charles Herrmann, Junhwa Hur, Eric Chen, Varun Jampani, Deqing Sun, and Ming-Hsuan Yang. "Telling Left from Right: Identifying Geometry-Aware Semantic Correspondence". In: ().

[83] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. *Adding Conditional Control to Text-to-Image Diffusion Models*. Nov. 26, 2023. (Visited on 05/11/2024). Pre-published.