




# Learning a Style Space for Interactive Line Drawing Synthesis from Animated 3D Models

Zeyu Wang<sup>1</sup> , Tuanfeng Y. Wang<sup>2</sup>  and Julie Dorsey<sup>1</sup> 

<sup>1</sup>Yale University   <sup>2</sup>Adobe Research

## Abstract

Most non-photorealistic rendering (NPR) methods for line drawing synthesis operate on a static shape. They are not tailored to process animated 3D models due to extensive per-frame parameter tuning needed to achieve the intended look and natural transition. This paper introduces a framework for interactive line drawing synthesis from animated 3D models based on a learned style space for drawing representation and interpolation. We refer to style as the relationship between stroke placement in a line drawing and its corresponding geometric properties. Starting from a given sequence of an animated 3D character, a user creates drawings for a set of keyframes. Our system embeds the raster drawings into a latent style space after they are disentangled from the underlying geometry. By traversing the latent space, our system enables a smooth transition between the input keyframes. The user may also edit, add, or remove the keyframes interactively, similar to a typical keyframe-based workflow. We implement our system with deep neural networks trained on synthetic line drawings produced by a combination of NPR methods. Our drawing-specific supervision and optimization-based embedding mechanism allow generalization from NPR line drawings to user-created drawings during run time. Experiments show that our approach generates high-quality line drawing animations while allowing interactive control of the drawing style across frames.

## CCS Concepts

• **Computing methodologies** → **Non-photorealistic rendering; Animation; Learning latent representations;**

## 1. Introduction

The gap between what humans and computers draw is one of the main reasons why artists are reluctant to employ NPR in their workflow. Such generative techniques could have served as a collaborative and efficient tool if they allowed for intuitive user control rather than dictating the output. For example, creating line drawings for animated shapes is tedious and hard to control, which traditionally relies on the heavy manual labor of tweeners, i.e., artists who create intermediate frames between keyframes. In this paper, we investigate how to efficiently produce a line drawing animation given a 3D sequence while allowing interactive editing of the drawings inspired by the modern 3D-based workflow. Our key observation in enabling this interactive editing pipeline is to interpret a drawing as a *style* for depicting the underlying geometry. Here a style captures the relationship between stroke placement and geometric properties of the underlying surface. By adopting a keyframe-based framework, we seek to develop a system that facilitates the synthesis of line drawing animations with interactive editing by propagating the style of drawings at several keyframes to all other frames.

There are numerous ways to create a drawing. Even for the same shape, people with different levels of experience for various purposes can employ distinct styles. In this work, our goal is to build an efficient animation authoring tool for line drawings with fixed

stroke color and width. This is a commonly studied type of drawing in graphics literature [CGL\*08, LLM\*19, LNHK20] and what most people draw in practice. We leave the handling of other artistic choices such as hatching and stippling to future work.

Extracting line drawings from 3D geometry has been discussed under the scope of non-photorealistic rendering (NPR) [DFRS03, OBS04, JDA07]. However, common parameter-controlled NPR approaches fall short in a creative workflow. There may not exist a set of parameters that can generate a desired line drawing. Also, smooth transition between keyframes may not be possible since the parameters often change at target keyframes and discrete parameters are difficult to interpolate. Most NPR methods only operate on local features, while artists usually have context-dependent and more sophisticated treatment in their drawing.

In our system, we directly allow the user to create line drawings at a few keyframes. With known 3D geometry at each frame, we extract a latent style code that faithfully represents the user's drawings at the keyframes. Our system can then synthesize line drawings at the inbetween frames from the underlying geometry by interpolating the latent style codes between the keyframes. Technically, with a given camera, we represent the geometry at each frame as a set of 2D geometric signals, e.g., normal maps, depth maps, and surface curvatures. We adopt a StyleGAN-based gener-

ator [KLA\*20] to produce a line drawing from a 2D feature map learned from the geometric signals and modulated by a latent style code at each frame. We train the network with synthetic line drawings generated by different NPR methods. At run time, we employ an optimization-based GAN inversion technique to map the input drawings into the latent style space.

To summarize, this paper makes the following contributions:

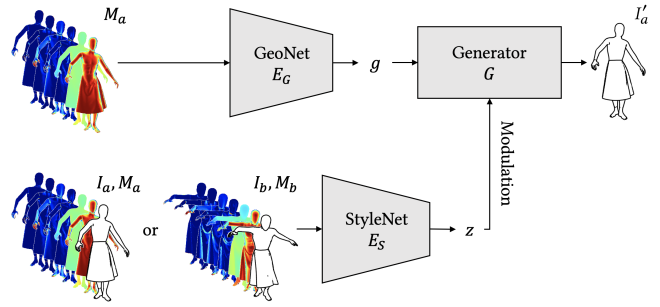
- A pipeline for interactive line drawing synthesis from animated 3D models, which allows users to control the synthesized style using their drawings at keyframes.
- Novel loss terms specifically designed to facilitate interpolation and style/geometry disentanglement in for line drawings.
- An optimization-based embedding strategy that makes the network trained on synthetic NPR output generalize to user-created line drawings at run time.

## 2. Related Work

**Non-photorealistic rendering.** NPR techniques attempt to create artistic styles in contrast to photorealism. These methods usually take a 3D model as input and extract lines by examining differential properties such as surface curvatures and their derivatives. Widely used ones include suggestive contours [DFRS03], ridges and valleys [OBS04], and apparent ridges [JDA07]. Recent deep learning-based methods can produce more stylistic results [LFHK21]. These results are believed to resemble drawing, but studies have shown that there is still a significant gap between what people and machines draw [WQF\*21]. In particular, it is difficult to use NPR methods to produce an intended output due to the lack of intuitive control and interpolation space. This becomes more challenging for animation, which involves varying geometry over time. Animators and tweeners have to draw every single frame to create a drawing animation, with little help from digital tools. Our work aims to reduce manual labor in line drawing animation synthesis while allowing intuitive control of the generator by drawing at keyframes.

**Sketch animation.** Sketch animation can date back to rotoscoping [Fle05], where animators trace over movie images projected onto a glass panel frame by frame. Digital technologies have provided sketch animation with many new possibilities, such as systems for animating vector graphics [WNS\*10, CAC22], propagating artistic styles [BCK\*13, XWSY15], and deforming segments driven by video and physical simulation [WLP\*17, SBF\*18]. In particular, it is beneficial to bring the 2D and 3D workflows together for more expressiveness and versatility, such as augmenting 2D sketches with 3D motion capture data and converting 2D sketches to 3D proxies [JSH09, JSMH12]. Inspired by this line of work, our system attempts to facilitate sketch animation by synthesizing sketches from a 3D animation sequence in a user-specified style at keyframes. Users can interactively edit the predicted sketches and propagate the changes to other frames similar to a previous workflow for garment animation [WSFM19].

**Style-driven image generation and embedding.** Generating rasterized drawings in 2D image space can be categorized as a sub-topic of image synthesis with control signals. A notable technique is StyleGAN [KLA19], which maps a high dimensional noise into a latent space and apply the learned latent feature as *style*



**Figure 1:** Our network architecture. *GeoNet*  $E_G$  learns a 2D geometric feature  $g$  for the underlying geometry  $M_a$  captured at frame  $a$ . *StyleNet*  $E_S$  encodes a drawing  $I_a$  at frame  $a$  or one with the same style  $I_b$  at any frame  $b$  into a 1D latent style code  $z$ . A StyleGAN-based generator  $G$  [KLA\*20] is initialized by  $g$  and modulated by  $z$  at each layer. This pipeline outputs a drawing  $I'_a$  generated by  $G$ .

to modulate the weights of a convolutional neural network-based generator. StyleGAN and its successors have achieved state-of-the-art performance in high-quality image synthesis. Based on the powerful generator, using a learned 2D feature map instead of a constant tensor allows explicit control over the spatial structure of a generated image [SGLT21]. Inspired by these previous efforts, we interpret a line drawing as a 3D shape presented with a specific style. We adopt a StyleGAN-based generator and initialize it with a 2D geometric feature map for geometry/style disentanglement. The inversion of generation is also useful to embed an input image into the latent space. This can be done by projecting the image into latent space via a neural network encoder [TEB\*20, TAN\*21, RAP\*21] or optimizing the latent code to reconstruct the target image [UVL18, AQW19, AQW20]. During training, our style encoder learns a style code, while during testing, we adopt optimization-based embedding for better generalization.

## 3. Methodology

With the given 3D animation sequence, our method addresses two core issues: 1) represent an line drawing as a latent style code at a keyframe, and 2) generate high-quality line drawing images with an interpolated style code for all other frames. This requires us to learn a latent space to capture the style difference between drawings. Here *style* refers to stroke placement on a given 3D geometry, not stroke appearance like width, color, and texture. Since the underlying geometry is known, we adopt an autoencoder-like framework to learn the latent space disentangled from geometry (Section 3.1). We then train our network on a synthetic dataset with customized loss terms that encourage the network to generate smooth transitions (Section 3.2). At run time, we adopt an optimization-based GAN inversion process on a user-created line drawing to obtain its corresponding latent representation. The learned latent space, the generator, and the keyframe embedding enable an interactive workflow for line drawing animation synthesis (Section 5).

### 3.1. Learning a Latent Style Space

We adopt an autoencoder-like framework to learn a latent style code for an input line drawing, as shown in Figure 1. Specifically, we en-

code the underlying geometry into a 2D feature map via the geometry encoder,  $E_G$ , and encode the 2D line drawing into a 1D style code via the encoder,  $E_S$ . We utilize a StyleGAN-based generator,  $G$ , as our decoder which takes the 2D geometry feature map as input and is modulated by the 1D style code.

**Encoding geometry with GeoNet  $E_G$ .** Given a 3D animation sequence, we first set camera parameters to put the character in the center of the viewport. We then represent the underlying geometry at each frame with a set of geometric properties. Specifically, we render 1) shading,  $N \cdot V$ , where  $N$  is the normal direction in the world space and  $V$  is the view direction pointing to the camera; 2) depth map,  $dep$ , distance from the surface to the camera; 3) derivative of the radial curvature,  $D_{\kappa}$ , used in suggestive contours [DFRS03]; 4) positive first principal curvature  $\kappa_1$  and 5) negative first principal curvature  $\kappa_2$ , used in ridges and valleys [OBS04]; and 6) view-dependent curvature  $\kappa_t$ , used in apparent ridges [JDA07], with a resolution of  $H \times W$ . Each curvature is normalized using its 90<sup>th</sup> percentile. The geometry encoder  $E_G$  encodes the multi-channel map  $M_a \in \mathbb{R}^{H \times W \times 6}$  containing geometric properties at frame  $a$  as a tensor  $g \in \mathbb{R}^{H_s \times W_s \times 512}$ :

$$g = E_G(M_a), \quad (1)$$

where  $H_s, W_s = H/16, W/16$ . We implement  $E_G(\cdot)$  with a set of convolutional neural networks for efficient 2D feature extraction.

**Encoding drawing with StyleNet  $E_S$ .** We represent a line drawing as a single-channel black-and-white image at a resolution of  $H \times W$ . We adopt a convolutional neural network to extract a 1D vector as style code from the input drawing. We use 1D feature vectors to represent style instead of using 2D feature maps since we do not expect to capture any spatial information. Specifically, starting from a line drawing  $I_a \in \mathbb{R}^{H \times W \times 1}$  at frame  $a$ , we concatenate it with the corresponding multi-channel geometric map  $M_a \in \mathbb{R}^{H \times W \times 6}$ . The combined input is then passed through our style encoder  $E_S$  to produce a latent style vector  $z \in \mathbb{R}^{1 \times 1 \times 2048}$ :

$$z = E_S(I_a, M_a). \quad (2)$$

Note that our training dataset is generated by NPR methods. Therefore,  $I_a$  can be labelled as  $I_{a,x}$  where  $x$  refers to a set of NPR parameters with which the line drawing is generated. We assume the drawings generated with same NPR parameters should be the same style, so one property of our StyleNet  $E_S$  should be:

$$E_S(I_{a,x}, M_a) = E_S(I_{b,x}, M_b), \quad (3)$$

where  $a$  and  $b$  are randomly sampled frames.

**StyleGAN-based image generator  $G$ .** We choose a StyleGAN-based network as our generator. The original StyleGAN architecture starts from a constant spatial tensor and utilizes a latent noise vector passed through a mapping network to demodulate intermediate layers to control the details of the generated image. Inspired by Sarkar et al. [SGLT21], our generator  $G$  starts from the learned geometry feature  $g$  instead of a constant input. The intermediate layers are then demodulated by the learned style code  $z$  in forward passing.  $G$  outputs a line drawing  $I'_a \in \mathbb{R}^{H \times W \times 1}$  at frame  $a$ :

$$I'_a = G(g; z) = G(E_G(M_a); z). \quad (4)$$

	$I_1$	$I_t$	$I_2$	$ I_t - I_1 $	$ I_2 - I_t $	$L_{\text{interp}}$	$L_{\text{stroke}}$
Case 1						0.3121	0.2507
Case 2						0.7313	0.2952
Case 3						0.6566	0.2594

**Figure 2:** A toy example explaining our interpolation and stroke-ness losses. Given two line drawings  $I_1$  and  $I_2$ , Case 1 shows natural inbetweening, i.e., gradually growing the stroke from  $I_1$  to  $I_2$  in a certain direction. Case 2 and Case 3 present two counterintuitive examples. In Case 2, the stroke does not follow the minimum path principle and goes beyond the range of  $I_2$ . In Case 3, the stroke grows in fragments instead of growing smoothly in a stable direction.  $|I_t - I_1|$  and  $|I_2 - I_t|$  further highlight the differences.

### 3.2. Learning for Drawing Generation

A typical image-based generator is usually trained to minimize per-pixel and perceptual differences. In our setup, the generator should also produce images that 1) look like line drawings rather than natural images, and 2) behave like real inbetweening across frames. Specifically, in addition to widely used terms like reconstruction, perceptual, and adversarial losses, we supervise our network training with 1) sparsity loss, to remove gray pixels; 2) interpolation loss, to encourage the generated image to transition smoothly between sampled positions in the latent space; 3) stroke-ness loss, to encourage strokes at the in-between frames to grow or vanish naturally. We visually explain the loss terms in Figure 2 with an intuitive example. The details of our loss terms are as follows.

**Reconstruction loss and perceptual loss.** For a frame  $a$  in the training sequence, we have:

$$L_{\text{recon}} = \|I_a - I'_a\|_1 \quad (5)$$

$$L_{\text{percep}} = \sum_k \|\text{VGG}_k(I_a) - \text{VGG}_k(I'_a)\|_1. \quad (6)$$

where  $\text{VGG}_k(\cdot)$  is the  $k^{\text{th}}$  layer of a VGG network pre-trained on ImageNet. We expect line drawings rendered with same NPR parameters to have the same latent style code. We implement this by generating  $I'_a$  with the style code learned from another frame  $b$  rendered with the same NPR parameters  $x$ , i.e.,  $I'_a = G(E_G(M_a); E_S(I_{b,x}, M_b))$ . We also explicitly supervise this property with an intrinsic loss:

$$L_{\text{intrinsic}} = \|z_{a,x} - z_{b,x}\|_2^2, \quad (7)$$

where  $a, b$  are randomly sampled frames,  $x$  is a set of parameters used for NPR rendering,  $z_{a,x} = E_S(I_{a,x}, M_a)$  and  $z_{b,x} = E_S(I_{b,x}, M_b)$ .

**Sparsity loss.** Since our goal is to generate a line drawing with binary strokes, therefore, we adopt a sparsity loss to penalize gray pixels in the generated image.

$$L_{\text{sparsity}} = \|I'_a\|_1. \quad (8)$$

**Interpolation loss.** It is not obvious to generate in-between

styles for any pair of synthetic NPR line drawings as they may use different methods. Here we propose a self-supervised loss to encourage a smooth transition between sample pairs in training:

$$L_{\text{interp}} = \sum_k \|\text{VGG}_k(I'_2) - \text{VGG}_k(I'_t)\|_1 \quad (9)$$

$$+ \|\text{VGG}_k(I'_t) - \text{VGG}_k(I'_1)\|_1, \quad (10)$$

where  $I'_1 = G(E_G(M_a); z_1)$  and  $I'_2 = G(E_G(M_a); z_2)$  are generated images with two different styles  $z_1$  and  $z_2$  for the same frame.  $I'_t = G(E_G(M_a); z_t)$  is generated with  $z_t = (1-t)z_1 + tz_2$ , a randomly sampled style code between  $z_1$  and  $z_2$ .

**Strokeness loss.** Here we take a closer look at the growing or vanishing of a stroke during style change. A natural way should treat the stroke as a whole and elongate or shorten it in a certain direction instead of placing fragmented segments and connecting them. Therefore, we adopt a strokeness term here to encourage a continuous stroke during interpolation. Specifically,

$$L_{\text{stroke}} = \|\text{GauSm}_k(|I'_2 - I'_t|)\|_{0.5} \quad (11)$$

$$+ \|\text{GauSm}_k(|I'_t - I'_1|)\|_{0.5}, \quad (12)$$

where  $\text{GauSm}_k(\cdot)$  is the 2D Gaussian smoothing operator with a kernel size of  $k = 7$ . This is based on our observation that Gaussian smoothing produces more non-zero pixels when the pixels from the input image is more separate from each other.

**Adversarial loss.** Finally, we apply a standard adversarial loss with a discriminator  $D$  [KLA\*20] when training our network.

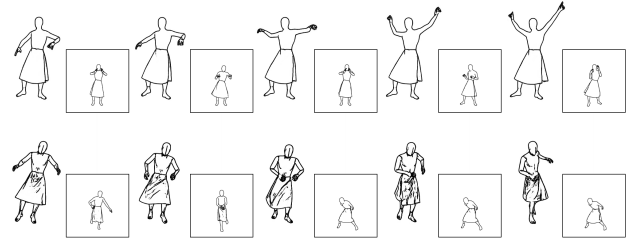
We train our network with a combination of all the loss terms listed above with the same weight. In the supplementary material, we discuss the details of our dataset and implementation and show that the loss terms derived from prior knowledge of strokes leads to better drawing generation and interpolation.

#### 4. Generalization

A systematic evaluation in terms of latent style space embedding, disentanglement between style and geometry, and style interpolation can be found in the supplementary material. Here we show how our network can generalize to unseen frames and scenarios.

**Unseen frames.** Once a network is trained, we test how well it generalizes to the unseen frames in the same animation. This happens when an animator edits the 3D animation, e.g., inserts new frames to the given sequence or updates the 3D shape at some existing frames. In Figure 3, we train a model on the first 2000 frames and apply it on the following unseen subsequence from the Human dataset. The style code is obtained from the first frame which is an edited NPR line drawing, and the style code remains the same for the entire unseen sequence. The nearest neighbors retrieved from the training dataset indicate that although the target pose is very different from the training dataset, our method generalizes well to the unseen sequence. This is particularly useful as the line drawing animation can update automatically without further manual effort.

**Unseen animation.** Training our network from scratch is time consuming due to the complexity of the architecture, especially our StyleGAN-based generator  $G$ . Alternatively, once a new animation sequence is given, we can make the most use of existing models



**Figure 3: Generalization to unseen frames.** We evaluate our trained model on an unseen subsequence from the same animation. In each row, the style is properly propagated to unseen frames, compared to the nearest neighbor from the training set (black box).

trained on previous cases. In Figure 4, we see that directly applying the model trained on another dataset performs reasonable on style embedding and line drawing reconstruction, while style interpolation generates artifacts at in-between frames. This is because although our training NPR line drawings are generated based on local geometric features, our self-supervision losses break such locality for better line drawing generation during style interpolation. Fortunately, the pre-trained model provides a good starting point for fine-tuning. In both test cases, after 500 iterations of fine-tuning ( $\sim 7$  minutes), the produced in-between frames are getting reasonable. After 1k iterations ( $\sim 15$  minutes), the in-between frames are already very similar to the results generated by the network trained from scratch, which is supported by the decreasing error.

#### 5. Applications

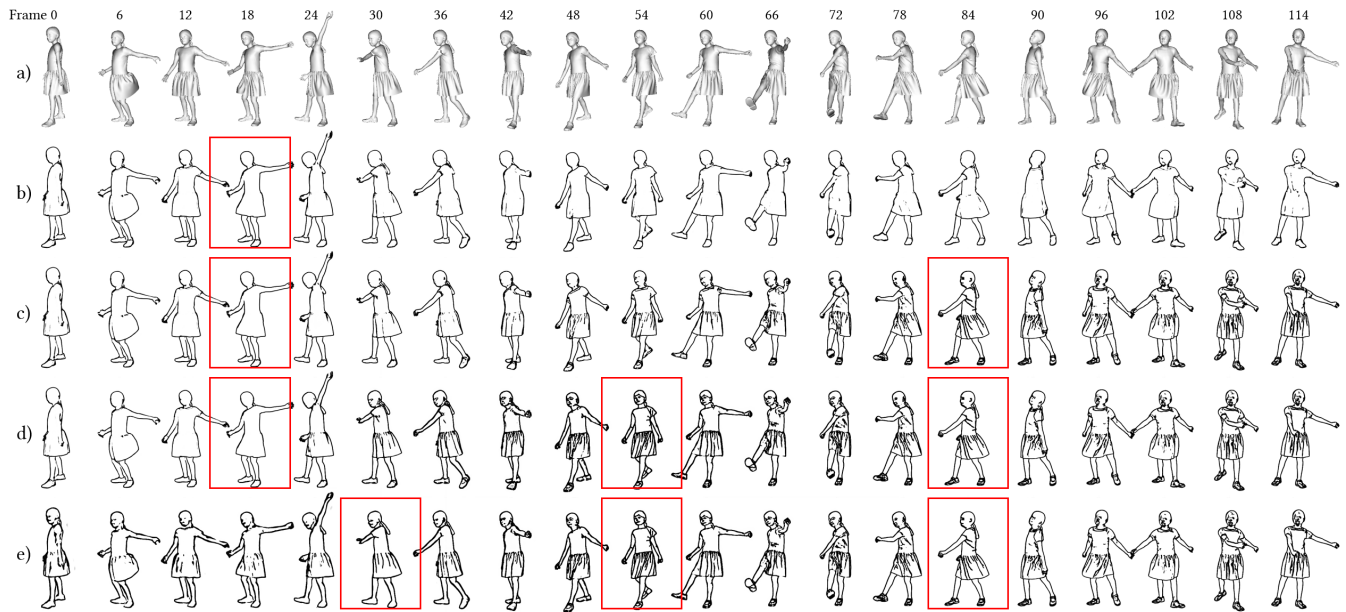
With the framework described above, we propose an interactive workflow for line drawing animation authoring. Starting from a 3D animation sequence, we first build our case-specific dataset using NPR methods over all frames of the sequence. With the synthetic dataset, we finetune (or train from scratch) our networks GeoNet ( $E_G$ ), StyleNet ( $E_S$ ), and Generator ( $G$ ) with the proposed loss terms. After the training/fine-tuning is done, the user then selects an example line drawing  $I_a$  from the NPR dataset to initialize the animation. We obtain the latent code  $z_a = E_S(I_a, M_a)$  and generate the line drawing animation for the whole sequence with  $I_k = G(E_G(M_k), z_a)$ , where  $k$  is the frame ID. Meanwhile, we add  $z_a$  at frame  $a$  to our keyframe set. The user may view the animation and decide which frames need to be modified. The user may select another keyframe  $b$  and create a line drawing  $I_b$ . We embed  $I_b$  into our latent space with the optimization framework to obtain  $z_b$  accordingly.  $z_b$  at frame  $b$  will be added into our keyframe set, after which we update the animation sequence with linear interpolation/extrapolation of  $z$  in the keyframe set. A keyframe can also be deleted and the animation will be updated similarly.

We invited five users with different levels of drawing expertise to test our prototype system. We show a typical case on Lilly in Figure 5. We observe that usually our system offers a plausible line drawing animation that matches users' intention after they select about 3% out of all frames as keyframes and create line drawings for these keyframes. The users recognize the value of our system as they no longer need to draw every single frame to create an animation. They also appreciate the intuitive control and interactivity that our system provides in contrast to other hard-to-edit results.



	Source	Interpolation				Target	Source	Interpolation				Target
No finetuning												
	0.102	0.209	0.223	0.227	0.145	0.131	0.233	0.284	0.273	0.186		
Finetuning for 500 iterations												
	0.092	0.141	0.162	0.161	0.124	0.127	0.176	0.214	0.223	0.172		
Finetuning for 1,000 iterations												
	0.077	0.127	0.151	0.158	0.113	0.110	0.160	0.186	0.194	0.158		
Ground truth												

**Figure 4:** Generalization to unseen scenarios. Our trained model can be adapted to a different scenario after a few iterations of fine-tuning. Here we show a model trained on *Mouse* and tested on *Lilly*, as well as one trained on *Lilly* and tested on *Mouse*. Since our networks learn non-local features for drawing interpolation, directly applying the pre-trained models ( $\sim 48$  hours) without fine-tuning generates artifacts (1<sup>st</sup> row). The performance improves rapidly after a few iterations ( $\sim 7$  to 15 minutes) of fine-tuning (2<sup>nd</sup> and 3<sup>rd</sup> rows). We report the  $L_1 + VGG$  error between each entry and the corresponding frame generated by the model trained on the target dataset (4<sup>th</sup> row).



**Figure 5:** An interactive pipeline for line drawing animation synthesis. Starting from the given 3D animation a), the user first selects a keyframe (frame 18) and creates a line drawing (highlighted in red box). Our method automatically propagates its style to the rest of the animation b). The user can iteratively add new keyframes (frame 84 in c), frame 54 in d), and frame 30 in e)) and edit line drawings at the keyframes. Our method smoothly blends the style between the keyframes. The user can also delete a keyframe (frame 18 in e)). Our system offers smooth interpolation between input line drawings at the keyframes and automatically updates the drawing animation after each edit.

## 6. Conclusion

In this paper, we have presented a method for interactive line drawing synthesis from animated 3D models by learning a latent style representation disentangled from the underlying geometry. Our approach efficiently learns a meaningful latent style space and results in a powerful line drawing generator, which can create clean drawings at a single frame and also interpolate the line drawings naturally along an animation sequence. This plays a critical role in establishing a prototype system that supports interactive authoring of line drawing animation. Our workflow worked well on various test cases and received positive feedback from our users.

In the future, we will extend our method to deal with more diverse stroke appearances and take hatching into consideration. Collecting line drawings for a data-driven based approach is always challenging, so we used a synthetic dataset in this work. Adopting more synthesis techniques may increase the diversity of our dataset, which can help reduce its gap to real human input. Since strokes are the fundamental element of a drawing, we will explore if using a vector representation will lead to better results. Finally, it is interesting to integrate sketch-based modeling techniques so users can edit the 3D animation sequence at the same time.

## Acknowledgments

This research began during an internship at Adobe Research and was sponsored in part by Adobe Research. We thank Aaron Hertzmann, Li-Yi Wei, Rubaiat Habib, Leonard McMillan, and Mianlun Zheng for the helpful discussions. This work was partially supported by National Science Foundation award #1942257.

## References

- [AQW19] ABDAL R., QIN Y., WONKA P.: Image2StyleGAN: How to Embed Images into the StyleGAN Latent Space? In *Proceedings of the IEEE International Conference on Computer Vision* (2019), pp. 4432–4441. 2
- [AQW20] ABDAL R., QIN Y., WONKA P.: Image2StyleGAN++: How to Edit the Embedded Images? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 8296–8305. 2
- [BCK\*13] BÉNAUD P., COLE F., KASS M., MORDATCH I., HEGARTY J., SENN M. S., FLEISCHER K., PESARE D., BREEDEN K.: Stylizing Animation by Example. *ACM Trans. Graph.* 32, 4 (jul 2013). 2
- [CAC22] CACANI PTE. LTD.: 2D Animation & Inbetween Software - CACANI. <https://cacani.sg/>, 2022. 2
- [CGL\*08] COLE F., GOLOVINSKIY A., LIMPAECHER A., BARROS H. S., FINKELSTEIN A., FUNKHOUSER T., RUSINKIEWICZ S.: Where Do People Draw Lines? *ACM Trans. Graph.* 27, 3 (Aug 2008), 88:1–88:11. 1
- [DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive Contours for Conveying Shape. *ACM Trans. Graph.* 22, 3 (July 2003), 848–855. 1, 2, 3
- [Fle05] FLEISCHER R.: *Out of the Inkwell: Max Fleischer and the Animation Revolution*. University Press of Kentucky, 2005. 2
- [JDA07] JUDD T., DURAND F., ADELSON E.: Apparent Ridges for Line Drawing. *ACM Trans. Graph.* 26, 3 (July 2007). 1, 2, 3
- [JSH09] JAIN E., SHEIKH Y., HODGINS J.: Leveraging the Talent of Hand Animators to Create Three-Dimensional Animation. In *Proceedings of the Symposium on Computer Animation* (New York, NY, USA, 2009), ACM, p. 93–102. 2
- [JSMH12] JAIN E., SHEIKH Y., MAHLER M., HODGINS J.: Three-Dimensional Proxies for Hand-Drawn Characters. *ACM Trans. Graph.* 31, 1 (Feb. 2012). 2
- [KLA19] KARRAS T., LAINE S., AILA T.: A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 4401–4410. 2
- [KLA\*20] KARRAS T., LAINE S., AITTALA M., HELSTEN J., LEHTINEN J., AILA T.: Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2020), pp. 8110–8119. 2, 4
- [LFHK21] LIU D., FISHER M., HERTZMANN A., KALOGERAKIS E.: Neural Strokes: Stylized Line Drawing of 3D Shapes. In *Proceedings of the IEEE International Conference on Computer Vision* (2021), pp. 14204–14213. 2
- [LLM\*19] LI M., LIN Z., MECH R., YUMER E., RAMANAN D.: Photo-Sketching: Inferring Contour Drawings From Images. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision* (2019), pp. 1403–1412. 1
- [LNHK20] LIU D., NABAIL M., HERTZMANN A., KALOGERAKIS E.: Neural Contours: Learning to Draw Lines from 3D Shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2020). 1
- [OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-Valley Lines on Meshes via Implicit Surface Fitting. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 609–612. 1, 2, 3
- [RAP\*21] RICHARDSON E., ALALUF Y., PATASHNIK O., NITZAN Y., AZAR Y., SHAPIRO S., COHEN-OR D.: Encoding in Style: A StyleGAN Encoder for Image-to-Image Translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2021), pp. 2287–2296. 2
- [SBF\*18] SU Q., BAI X., FU H., TAI C.-L., WANG J.: Live Sketch: Video-Driven Dynamic Deformation of Static Drawings. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2018), ACM, p. 1–12. 2
- [SGLT21] SARKAR K., GOLYANIK V., LIU L., THEOBALT C.: Style and Pose Control for Image Synthesis of Humans from a Single Monocular View, 2021. [arXiv:2102.11263](https://arxiv.org/abs/2102.11263). 2, 3
- [TAN\*21] TOV O., ALALUF Y., NITZAN Y., PATASHNIK O., COHEN-OR D.: Designing an Encoder for StyleGAN Image Manipulation. *ACM Trans. Graph.* 40, 4 (2021), 1–14. 2
- [TEB\*20] TEWARI A., ELGHARIB M., BERNARD F., SEIDEL H.-P., PÉREZ P., ZOLLHÖFER M., THEOBALT C.: PIE: Portrait Image Embedding for Semantic Control. *ACM Trans. Graph.* 39, 6 (2020), 1–14. 2
- [UVL18] ULYANOV D., VEDALDI A., LEMPITSKY V.: Deep Image Prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 9446–9454. 2
- [WLP\*17] WILLETT N. S., LI W., POPOVIC J., BERTHOUSOZ F., FINKELSTEIN A.: Secondary Motion for Performed 2D Animation. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (New York, NY, USA, 2017), ACM, p. 97–108. 2
- [WNS\*10] WHITED B., NORIS G., SIMMONS M., SUMNER R. W., GROSS M., ROSSIGNAC J.: BetweenIT: An Interactive Tool for Tight Inbetweening. *Computer Graphics Forum* 29, 2 (2010), 605–614. 2
- [WQF\*21] WANG Z., QIU S., FENG N., RUSHMEIER H., MCMILLAN L., DORSEY J.: Tracing Versus Freehand for Evaluating Computer-Generated Drawings. *ACM Trans. Graph.* 40, 4 (Aug. 2021). 2
- [WSFM19] WANG T. Y., SHAO T., FU K., MITRA N. J.: Learning an Intrinsic Garment Space for Interactive Authoring of Garment Animation. *ACM Trans. Graph.* 38, 6 (Nov. 2019). 2
- [XWSY15] XING J., WEI L.-Y., SHIRATORI T., YATANI K.: Autocomplete Hand-Drawn Animations. *ACM Trans. Graph.* 34, 6 (Oct. 2015). 2