Pacific Graphics (2022)
N. Umetani, E. Vouga, and C. Wojtan (Editors)

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

Zeyu Wang[1], Tuanfeng Y. Wang[2] and Julie Dorsey[1]

[1]Yale University [2]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*

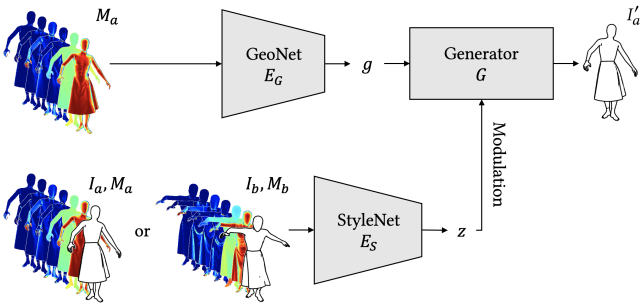

Figure 1: *Our network architecture. GeoNet E_G learns a 2D geometric feature g for the underlying geometry M_a captured at frame a. StyleNet E_S encodes a drawing I_a at frame a or one with the same style I_b at any frame b into a 1D latent style code z. A StyleGAN-based generator G [KLA*20] is initialized 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_κ, used in suggestive contours [DFRS03]; 4) positive first principal curvature κ_1 and 5) negative first principal curvature κ_2, used in ridges and valleys [OBS04]; and 6) view-dependent curvature κ_t, used in apparent ridges [JDA07], with a resolution of $H \times W$. Each curvature is normalized using its 90^{th} 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 $g \in \mathbb{R}^{H_s\times W_s\times 512}$:

$$g = E_G(M_a), \tag{1}$$

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 $z \in \mathbb{R}^{1\times 1\times 2048}$:

$$z = E_S(I_a, M_a). \tag{2}$$

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:

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

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:

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

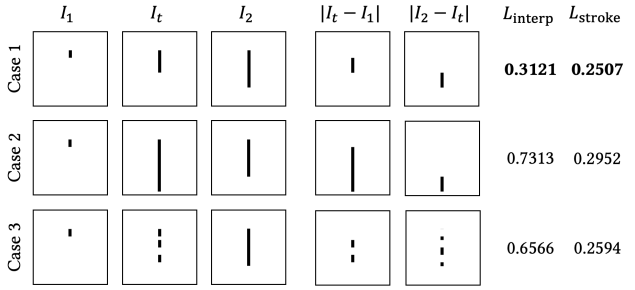

Figure 2: *A toy example explaining our interpolation and strokeness losses. Given two line drawings I_1 and I_2, Case 1 shows natural inbetweening, i.e., gradually growing the stroke from I_1 to I_2 in a certain direction. Case 2 and Case 3 present two counterintuitive examples. In Case 2, the stroke 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) strokeness 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:

$$L_{\text{recon}} = \left\| I_a - I'_a \right\|_1 \tag{5}$$

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

where $\text{VGG}_k(\cdot)$ is the k^{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:

$$L_{\text{intrinsic}} = \left\| z_{a,x} - z_{b,x} \right\|_2^2, \tag{7}$$

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.

$$L_{\text{sparsity}} = \left\| I'_a \right\|_1. \tag{8}$$

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:

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

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

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,

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

$$+ \left\| GauSm_k(|I'_t - I'_1|) \right\|_{0.5}, \quad (12)$$

where $GauSm_k(\cdot)$ is the 2D Gaussian smoothing operator with a kernal 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

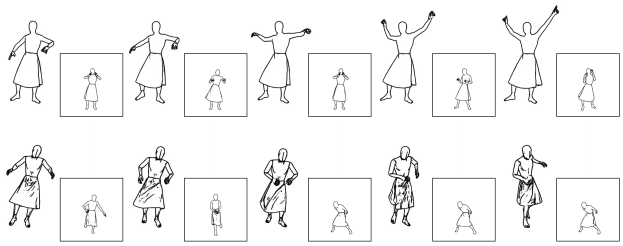

Figure 3: *Generalization to unseen frames. We evaluate our trained model on an unseen subsequence from the same animation. In each row, the style is properly propagated to unseen frames, compared to the nearest neighbor from the training set (black box).*

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.

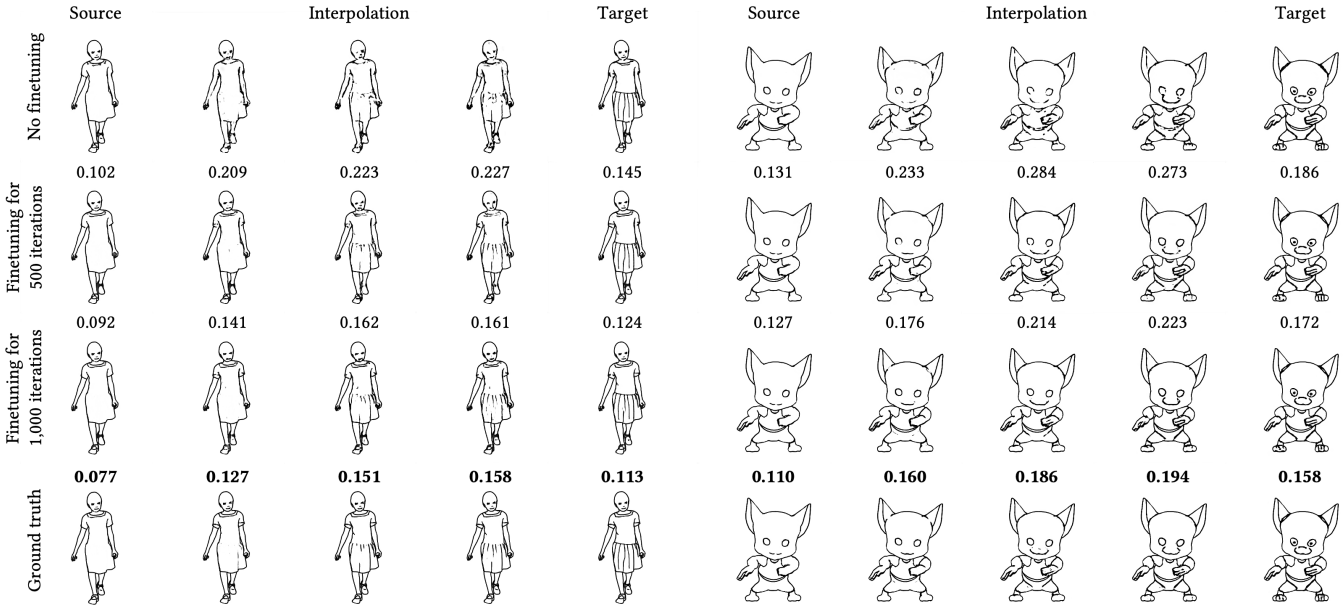

Figure 4: *Generalization to unseen scenarios. Our trained model can be adapted to a different scenario after a few iterations of fine-tuning. Here we show a model trained on* `Mouse` *and tested on* `Lilly`*, as well as one trained on* `Lilly` *and tested on* `Mouse`*. Since our networks learn non-local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

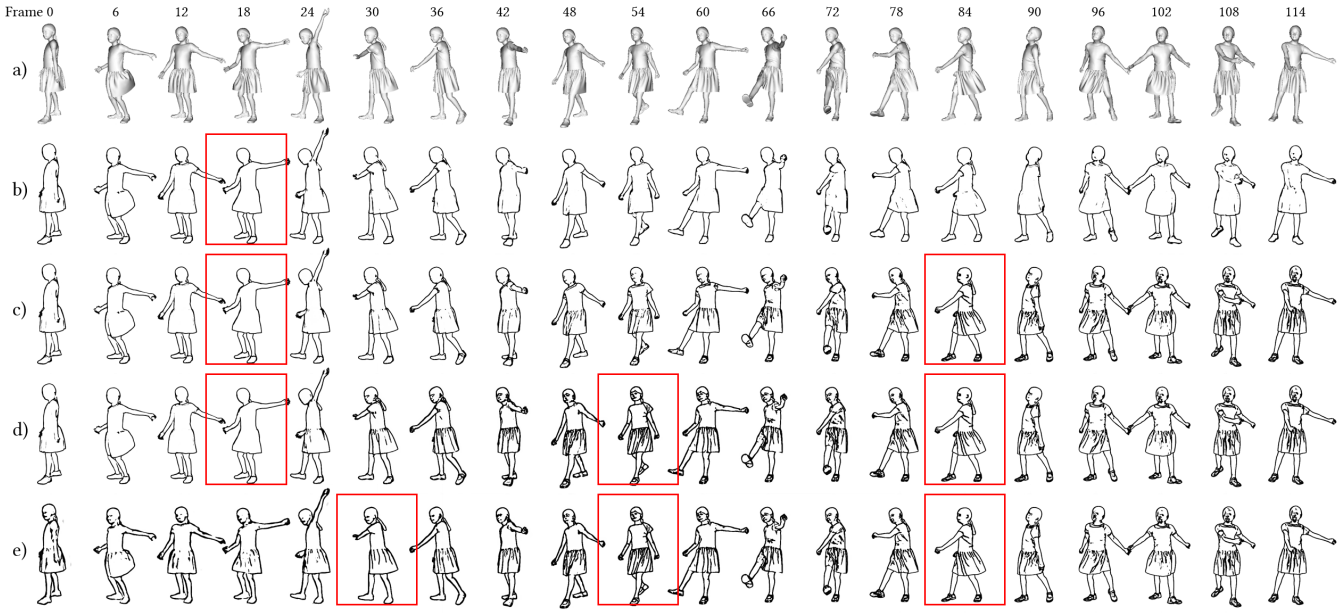

Figure 5: *An interactive pipeline for line drawing animation synthesis. Starting from the given 3D animation a), the user first selects a keyframe (frame 18) and creates a line drawing (highlighted in red box). Our method automatically propagates its style to the rest of the animation b). The user can 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] BÉNARD 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., HELLSTEN 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. 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., BERTHOUZOZ 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

Pacific Graphics (2022)
N. Umetani, E. Vouga, and C. Wojtan (Editors)

Learning a Style Space for Interactive Line Drawing Synthesis from Animated 3D Models (Supplementary Material)

Zeyu Wang[1], Tuanfeng Y. Wang[2] and Julie Dorsey[1]

[1]Yale University [2]Adobe Research

1. Overview

In the supplementary material, we describe the details of input drawing embedding at run time, dataset preparation, network architecture, and training strategy. We provide a systematic evaluation with various test cases and show that our approach learns a powerful latent style space for effective line drawing animation synthesis given different types of user input. We then validate our system design choices via ablation studies.

2. Input Drawing Embedding

At run time, we allow the user to create a line drawing at a keyframe in the animation. We need to obtain a style code from the input drawing that can faithfully reconstruct itself via our generator. The source of the input drawing can be 1) a NPR rendering of the keyframe with a set of user-specified parameters; 2) manual editing based on 1), e.g., removing or adding strokes; and 3) drawing from scratch over a reference image of the underlying geometry. Due to the huge space of drawing variation and limited capacity of NPR drawing generation, our StyleNet (E_S) only works well with input from 1). To enable an interactive editing workflow with full user control, during test time, we adopt an optimization-based approach to embed an input drawing I_a at keyframe a into the latent style space. Specifically, we compute geometry feature g when a keyframe is selected. We freeze the weights of our generator G and optimize a latent code z^* so that the generated image, $I'_a = G(g;z)$, is similar to the input drawing. Since the gradient of our binary drawing can be unstable for optimization, we adopt a pyramid structure with different levels of blurring, i.e.,

$$z^* = \arg\min_z \sum_{k=1,33,65} \left\| GauSm_k(I_a) - GauSm_k(I'_a) \right\|_1, \quad (1)$$

where k is the kernel size of the Gaussian smoothing operator in pixels. We initialize the optimization with the projection of StyleNet, i.e., $z_0 = E_S(I_a, M_a)$.

3. Implementation Details

3.1. Training Data

We evaluate our approach on five animation sequences: `Mouse` [ZZCB21] (480 frames), `Lilly` [3DP22] (1,200 frames), `Human` [Ado22] (3,000 frames), `Michelle` [Ado22] (200 frames), and `Vegas` [Ado22] (123 frames). For `Human`, we use the first 2,000 frames for network training and the rest of the sequence is only used for generalization test on unseen frames. For each scenario, we use NPR [DFRS03] to build a synthetic dataset for network training. For each frame, we use three NPR methods, i.e., suggestive contours, ridges and valleys, and apparent ridges, and sample four thresholds for each. We combine outputs from these methods and generate 64 ($4 \times 4 \times 4$) NPR line drawings for each frame and 4 additional Canny edge maps [Can86] generated with different thresholds. For each frame, it takes about 15 seconds on average to generate all 68 line drawings with a resolution of $W, H = 512, 512$. We apply commonly used techniques for 2D data augmentation, including translation, rotation, scaling, and flipping. We show samples training data for `Human` in Figure 1. Figure 2 shows an example of the input channels that represent the underlying geometry at each frame with a set of geometric properties.

Figure 1: *Sample training data for the `Human` character. Our synthetic dataset consist of line drawings generated with different non-photorealistic rendering methods with varying parameters.*

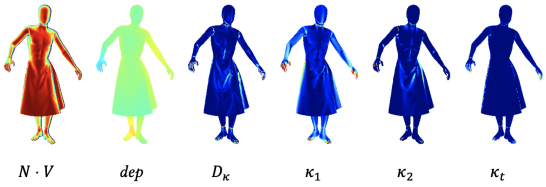

Figure 2: *Input to GeoNet E_G. We concatenated six geometric properties, i.e., shading, depth map, and four types of surface curvature. Blue represents low values and red represents high values.*

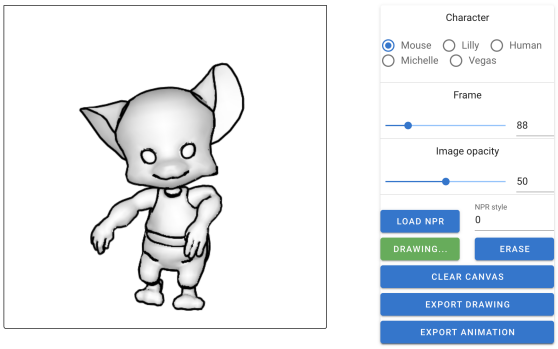

Figure 3: *User interface for drawing editing and interactive animation authoring. It allows the user to load an NPR line drawing, edit an existing drawing, or draw from scratch. We show a shading image of the geometry at the target frame in the background with adjustable opacity as a reference. In the animation mode, user can play the sequence, add, edit, and delete keyframes.*

3.2. Data Collection for Evaluation

We collect line drawings from three sources to evaluate our system.

1. Similar to the synthetic dataset generation pipeline, we generate unseen NPR line drawings for selected frames from the animation sequence with random sampled NPR parameters. We denote these drawings as NPR.
2. We allow users to edit an NPR line drawing by adding new strokes or erase existing ones. We build a user interface for this purpose as shown in Figure 3. It allows users to select a frame and load an NPR line drawing from the training set or from NPR. We show a shading image of the geometry at the selected frame in the background with adjustable opacity. Users can draw and erase strokes on the left canvas. We denote these drawings as NPR w/ edits.
3. More experienced users can directly create line drawings from scratch by looking at the reference shading image. We denote these drawings as Freehand.

3.3. Network Architecture and Training Strategy

Our GeoNet, E_G, is a neural feature extractor built with convolutional layers followed by Parametric Rectifying Linear Unit (PRelu) activation [HZRS15] and batch normalization. Starting from the multi-channel geometric signal map $M_a \in \mathbb{R}^{512\times512\times6}$, our network gradually decreases the dimension of the output to 256, 128, 64, and 32, while increases the number of channels gradually to 32, 64, 256, and 512 after each layer. This results in a 2D geometric feature, $g \in \mathbb{R}^{32\times32\times512}$. Our StyleNet, E_S, adopts a similar architecture but maps the input line drawing $I_a \in \mathbb{R}^{512\times512\times1}$ into a 1D style code, $z \in \mathbb{R}^{1\times1\times2048}$ with five more layers. The architecture of our generator, G, follows StyleGAN2 [KLA*20] including bilinear upsampling, equalized learning rate, noise injection at every layer, variance adjustment of residual blocks and leaky ReLU. The final output of this pipeline is a line drawing image $I'_a \in \mathbb{R}^{512\times512\times1}$. We trained our network with a learning rate of 0.02 using four NVIDIA V100 GPUs with a batch size of 4. It takes about 48 hours on average to converge in our experiments.

During the drawing embedding step, we implement the optimization using the *optim* package from PyTorch [Fac20]. We choose LBFGS algorithm [LN89] as our optimizer with a learning rate of 0.1. The optimization takes place on a single NVIDIA V100 GPU for 100 steps for all the cases discussed in this paper. This optimization takes about 30 seconds for 50 steps on average.

4. Evaluation

We evaluate our system with respect to line drawing synthesis, style interpolation, latent code embedding, and interactive editing. Experiments show that our approach outperforms vanilla style transfer and style interpolation baselines. Our approach produces plausible output that follows the properties of user input and transitions naturally in the animation. Although our network is trained in a case-specific manner, we show that it is easy to generalize to new cases with quick finetuning. Users of our system think the synthesized line drawings are consistent with their style and are useful for efficient and controllable animation authoring.

Latent style space embedding. We first evaluate the latent space embedding for a given line drawing. As discussed in the paper, the style code directly predicted from an input drawing may not be accurate enough for an edited NPR drawing or one drawn from scratch. In Figure 4, we see that the drawing generated from the learned latent style code z is similar to the target input in general but missing quite some details especially for NPR w/ edits and Freehand cases. With latent code optimization, the generated drawing can recover strokes missing from the initial projection. Our pyramid strategy facilitates gradient propagation and leads to better reconstruction compared to the one without the pyramid as shown by the $L_1 + \text{VGG}$ error. In case b), the pyramid strategy helps recover strokes that are missing in the optimization without the pyramid.

Disentanglement between style and geometry. Since we cast the style as an intrinsic property that depicts the relationship between stroke placement and geometric features, same drawing style at different frames in a sequence should be mapped to the same place in the latent style space. In other words, no matter which frame the style is learned from, the latent style code should all produce the same drawing for the target frame. Our NPR dataset provides consistent style across frames in a sequence. In Figure 5, given a target line drawing I_a at a certain frame a, we randomly pick two other frames b and c from the same sequence. The corresponding drawings with the same style are denoted as I_b and I_c. We show the style code extracted from I_b and I_c can be used to faithfully reconstruct the target drawing I_a.

Style interpolation. Our latent style space is learned from a set of separated NPR line drawings. The dataset itself cannot provide supervision on smooth style transition due to the nature that some NPR parameters are not interpolatable. Instead, our interpolation loss and strokeness loss provide self-supervision for this purpose. Here, we evaluate the performance of our method on interpolating between two projected latent style codes. Starting from style interpolation at a fixed frame, we learn the geometric features g and keep it unchanged during the interpolation. We embed the source

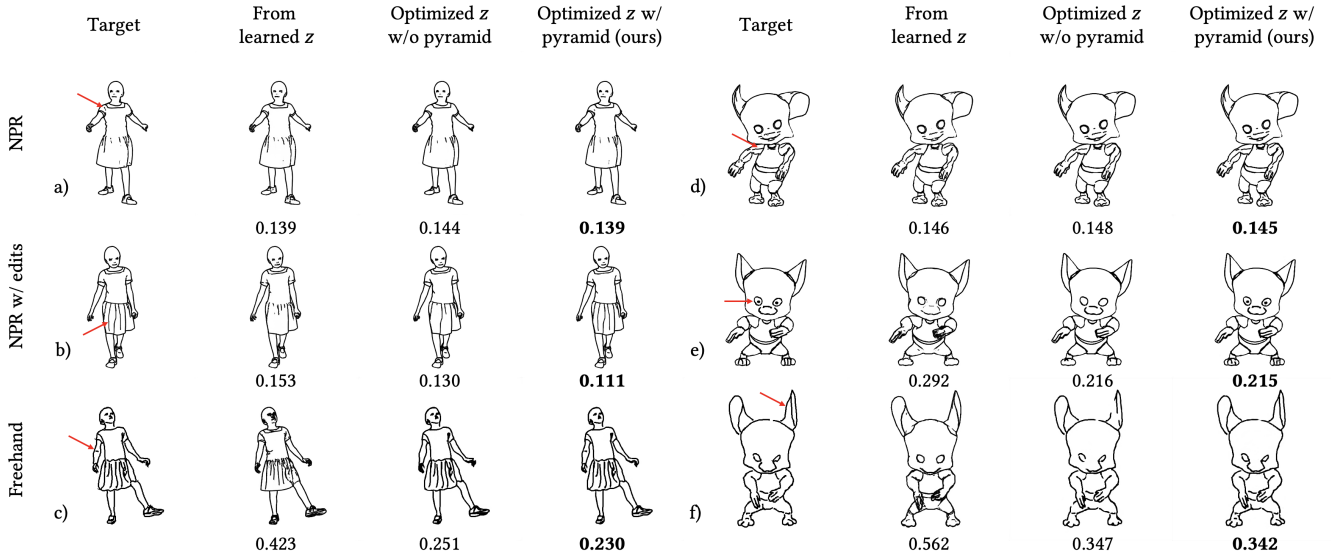

Figure 4: *Evaluation of latent style code embedding. We test our style embedding approach on different types of line drawings, i.e., NPR, NPR w/ edits, and Freehand. We show generated drawings from 1) learned latent style code z, 2) optimized z without the pyramid strategy, and 3) optimized z with the pyramid strategy (ours). Our approach achieves best reconstruction from the latent embedding. Red arrows highlight challenging areas. Ours achieves the lowest $L_1 + VGG$ error as shown by the values below each entry.*

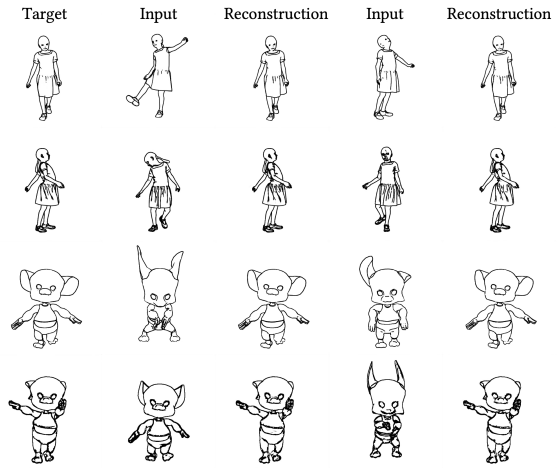

Figure 5: *Disentanglement between geometry and style. For each target drawing (1^{st} col.), we learn the latent style code z from one rendered with the same NPR parameters but at a different frame (2^{nd} and 4^{th} cols.) The drawing reconstruction from the learned style codes is almost identical at the target frame (3^{rd} and 5^{th} cols.)*

and target line drawings into latent space to obtain the corresponding latent style codes z_{source} and z_{target}. We perform linear interpolation between z_{source} and z_{target} to generate the transitioning line drawings accordingly, as shown in Figure 6.

An alternative baseline method for this task is to adopt a vanilla StyleGAN [KLA*20]. In the even rows of Figure 6, we train a vanilla StyleGAN with the same NPR dataset and perform the drawing embedding with our pyramid-based optimization. Instead of performing interpolation in the style space, we directly perform interpolation in the W space for the vanilla StyleGAN to generate corresponding output. The optimization-based embedding performs reasonable for reconstructing target drawings in the W space [KLA*20]. However, without an explicit disentanglement of style and geometry, the interpolation results using the vanilla StyleGAN have heavy artifacts.

Table 1: *Statistics of style interpolation on static frame in Figure 6. We calculate sparsity loss, interpolation loss, and the strokeness loss along the interpolated sequence. We show our method outperforms the baseline approach on style interpolation.*

		Lilly			Mouse		
		L_{sparsity}	L_{interp}	L_{stroke}	L_{sparsity}	L_{interp}	L_{stroke}
NPR	Ours	**0.069**	**0.044**	**0.025**	**0.098**	**0.073**	**0.042**
	Baseline	0.082	0.207	0.113	0.130	0.237	0.181
NPR w/ edits	Ours	**0.054**	**0.047**	**0.020**	**0.075**	**0.097**	**0.034**
	Baseline	0.059	0.275	0.147	0.099	0.190	0.148
Freehand	Ours	0.073	**0.096**	**0.047**	**0.072**	**0.126**	**0.052**
	Baseline	**0.072**	0.283	0.148	0.090	0.244	0.168

In this experiment, we test line drawings from NPR, NPR w/ edits, and Freehand. We show that our method learns a robust latent style space for all the three types of drawings. In Table 1, we report the sparsity loss, interpolation loss, and the strokeness loss along the interpolated sequence. We show that our method achieves lower loss which agree with the qualitative comparison.

Next, we elaborate on the experiment setup for an animation sequence rather than a static frame. The geometric features g are

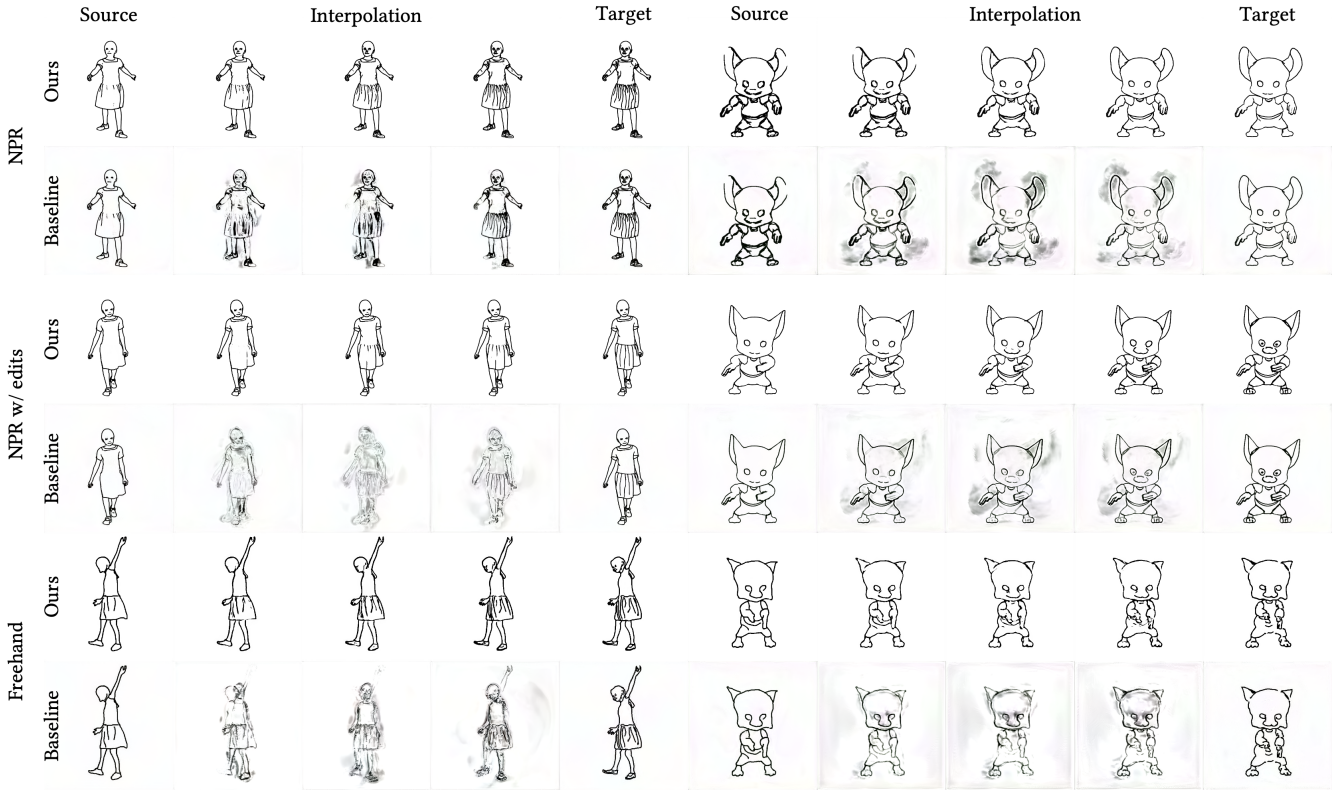

Figure 6: *Style interpolation at a static frame. For each case, the source line drawing (1^{st} or 6^{th} column) and the target (5^{th} or 10^{th} column) are given and embedded into the style space. Linearly interpolated style codes are used to generate transitioning line drawings (2^{nd}–4^{th} or 7^{th}–9^{th} columns). We compare our approach (odd rows) with a baseline method (even rows), vanilla StyleGAN trained with the same dataset. Our approach outperforms the baseline method due to geometry/style disentanglement and drawing-specific supervision during training.*

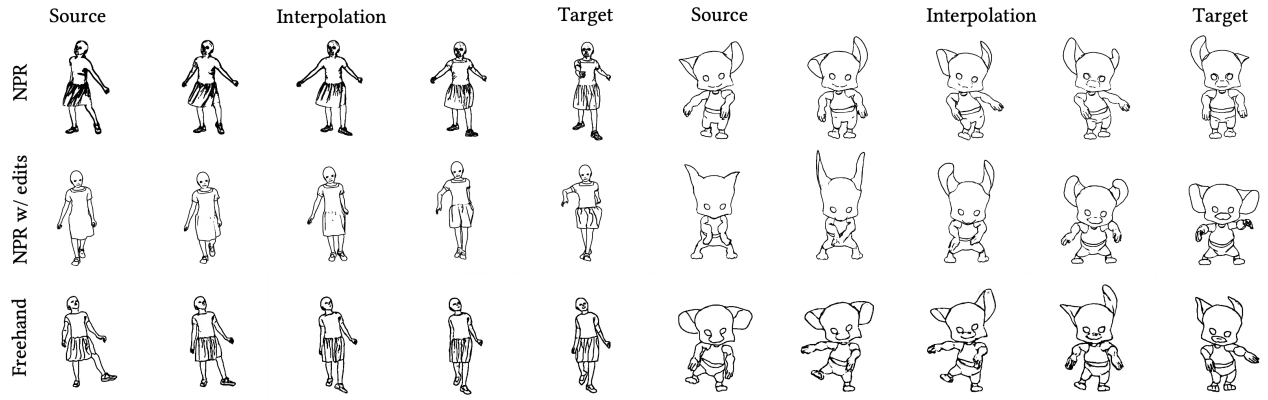

Figure 7: *Style interpolation between dynamic frames. We evaluate our system on animation sequences with dynamic frames using the same experiment setup from Figure 6.*

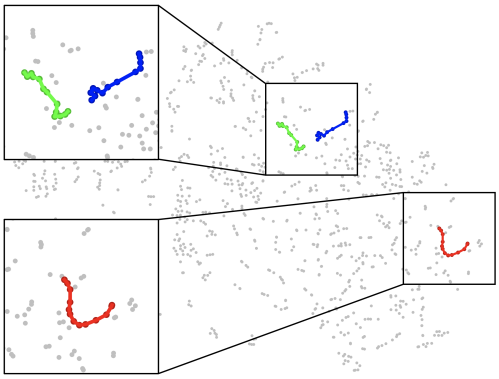

Figure 8: *t-SNE visualization of style interpolation between dynamic frames. For the* `Lilly` *dataset, we calculate the VGG features for* 10*k randomly sampled NPR line drawings in the training dataset and embed them into a 2D space via t-SNE [VdMH08]. We then calculate the VGG features for each frame in the three clips shown in Figure 7. We visualize the three sequences (NPR: red, NPR w/ edits: green, Freehand: blue) with the same 2D embedding. The trajectories indicate that the predicted style interpolation between frames is perceptually smooth and generalized well to unseen styles (NPR w/ edits and Freehand).*

learned from the underlying geometry at each frame accordingly, while the style code z is still interpolated linearly between the source and target line drawings. As shown in Figure 7, our learned latent space supports different types of line drawings for interpolation between the frames. The generated line drawings have a smoothly transitioning style along the sequence. We also visualize the drawing sequence as a trajectory in the 2D embedding space as shown in Figure 8. Specifically, for the test animation, we randomly sample 10k NPR line drawings from our dataset and compute the 512-dimensional VGG feature for each drawing. We apply t-distributed stochastic neighbor embedding (t-SNE) [VdMH08] for the VGG features. For the sequences in Figure 7, we embed the VGG feature for each entry into the same t-SNE domain. We see that the line drawing animation with linear interpolated style forms a smooth trajectory in the perceptual embedding space. This backs up our observation that the transition between source and target line drawings is smooth and natural.

5. Ablation Study

We use an ablation study to validate our design choices. Starting from a source edited NPR line drawing, we perform style interpolation on a static frame towards a target edited NPR line drawing. Figure 9 shows the effect of the loss terms used in network training. We see that the three losses proposed to self-supervise drawing interpolation, i.e., sparsity loss, interpolation loss, and strokeness loss, are functioning as expected as the toy examples discussed in the paper. We observe a smoother transition from the source to the target in our full pipeline. We also observe that those losses provide supervision over the gaps between samples in the NPR dataset, which helps the method generalize to unseen input line drawings.

We adopt the same style interpolation setup to evaluate the effect of latent space dimension. We train the same network but reduce the dimension of the latent style space from 2048 to 1024 to 512. After training with the same number of epochs, we compare their style interpolation performance in Figure 10. We observe that the quality of line drawing synthesis is improved along with the increase of the latent space dimension, where the model size and training time for each epoch are increased as well. Therefore, our choice of a 2048-dimensional latent style space is a reasonable tradeoff between training efficiency and network performance.

References

[3DP22] 3DPEOPLE: 3D People for Your Visualizations and Animations. https://3dpeople.com/, 2022. 1

[Ado22] ADOBE: Mixamo. https://www.mixamo.com, 2022. 1

[Can86] CANNY J.: A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 8*, 6 (Nov 1986), 679–698. 1

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive Contour Software: rtsc. https://rtsc.cs.princeton.edu, 2003. 1

[Fac20] FACEBOOK: torch.optim. https://pytorch.org/docs/stable/optim.html, 2020. 2

[HZRS15] HE K., ZHANG X., REN S., SUN J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision* (2015), pp. 1026–1034. 2

[KLA*20] KARRAS T., LAINE S., AITTALA M., HELLSTEN 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, 3

[LN89] LIU D. C., NOCEDAL J.: On the Limited Memory BFGS method for Large Scale Optimization. *Mathematical Programming 45*, 1 (1989), 503–528. 2

[VdMH08] VAN DER MAATEN L., HINTON G.: Visualizing Data Using t-SNE. *Journal of Machine Learning Research 9*, 11 (2008). 5

[ZZCB21] ZHENG M., ZHOU Y., CEYLAN D., BARBIC J.: A Deep Emulator for Secondary Motion of 3D Characters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2021), pp. 5932–5940. 1

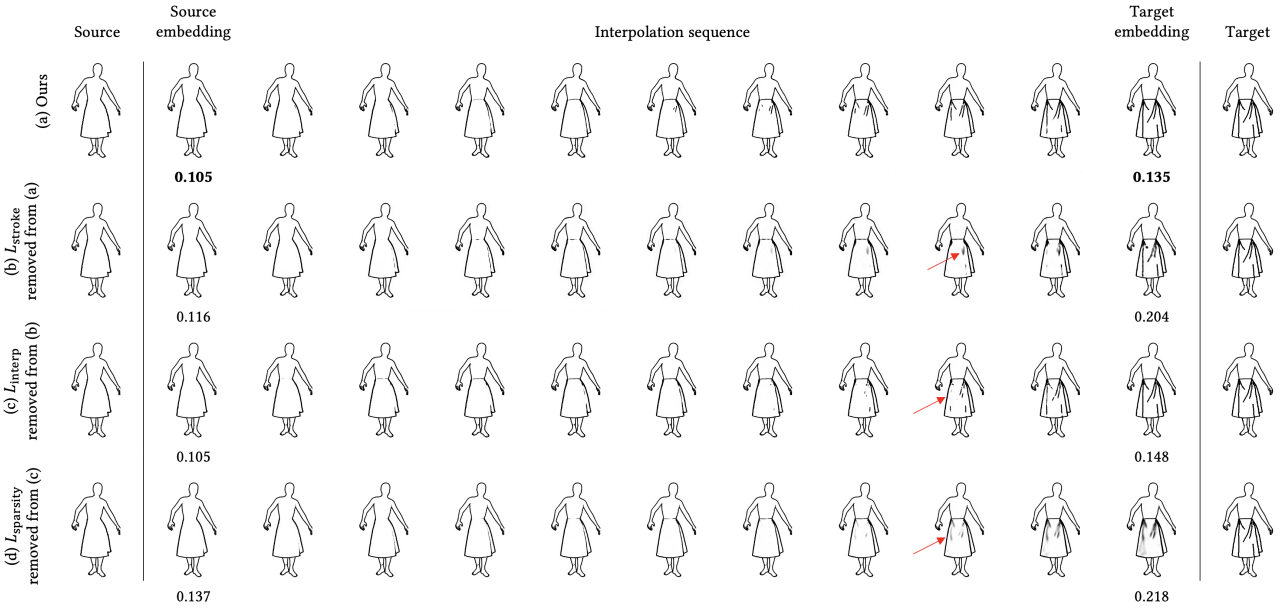

Figure 9: *Ablation study of loss terms. Given the same source and target line drawings, we first embed the drawings to obtain their latent style codes. We linearly interpolated the style codes to generate the in-between drawings. From top to bottom, we first show our method (a), and gradually remove strokeness loss (b), interpolation loss (c), and sparsity loss (d). Reconstruction error is reported for the embedding of source and target drawings. Red arrows highlight the artifacts in the interpolation. We show that, with all the loss terms, our approach performs the best for style interpolation and reconstruction of unseen input line drawings.*

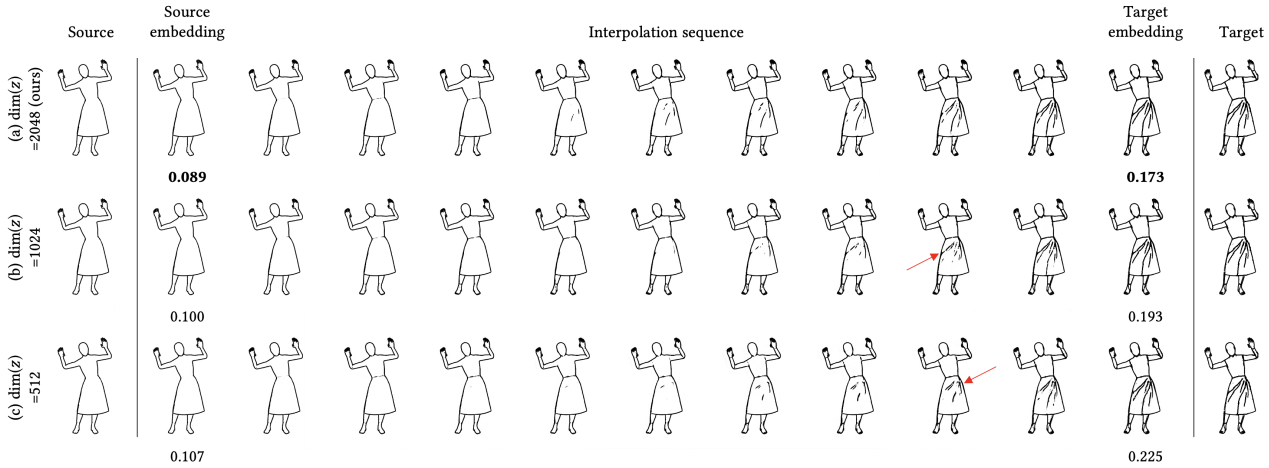

Figure 10: *Ablation study of latent space dimension. We train the same network with a latent space dimension gradually decreasing from 2048 to 1024 to 512. With an experiment setup similar to Figure 9, we show that a 2048-dimensional latent space (ours) is more capable of learning diverse drawing styles. Decreasing latent space dimension worsens the performance in drawing embedding and style interpolation.*