Computational In-betweening for Line Drawings in Animation

Anonymous Author(s)

1 INTRODUCTION

Hand-drawn animation is a timeless and beautiful medium for storytelling. However, its production is extremely time- and labor-intensive. Animators must draw each individual frame of their films, and even if they draw at a framerate lower than the standard 24 frames per second, they can produce up to hundreds of thousands of drawings. Luckily, not every drawing is equally important in defining motion: some are "key frames," which depict a movement's overall rhythm, while others are "in-betweens," which simply fill the gaps between key frames. *In-betweening*, therefore, is a pre-production task in which human animators take uncolored key frames and draw the in-betweens. One way to reduce animators' workload is to use a computational tool for in-betweening instead. Specifically, a computer can in-between by performing key frame interpolation with line drawings. Computer-assisted in-betweening is difficult, since images only give 2D information about objects that move and transform in 3D space, and systems must accommodate animation styles where objects' underlying 3D forms are inconsistent. Historically, this task has been done with stroke-based data, where line drawings are stored as a collection of vectors that depict information regarding position, color, etc. Corresponding strokes from successive key frames were matched either manually [Burtnyk and Wein 1971; Durand 1991] or based on similarity metrics, including those calculated from stroke lengths and enclosed areas [Whited et al. 2010] or strokes' relationships to those in neighborhoods around them [Xing et al. 2015]. Once strokes are matched, in-betweens are generated by mathematically interpolating the input strokes, fitting motion arcs to each stroke and deforming them along the arcs. However, many animators learn their craft on paper, which does not translate well to stroke-based programs.

Another way to represent drawings is to use raster data, where we can leverage the developing field for video interpolation. This currently finds state-of-the-art results with deep learning methods [Bao et al. 2019; Huang et al. 2022; Jiang et al. 2018; Niklaus and Liu 2020; Siyao et al. 2021], trained with large, *fully colored* and *textured* video datasets, which are able to produce high-quality interpolated frames. However, they do not work with the sparse visual content of black-and-white line drawings. The generated frames often look blurry, smeared, or unchanged from the input key frames themselves. The reason for this is that, without colors and textures, finding proper correspondence between frames becomes much more difficult in our domain.

We present a computational framework to do in-betweening for line drawings intently trained with three loss functions to help deal with these issues. In particular, we propose to use (a) a segmentation-style binary cross-entropy loss with logits as the reconstruction loss for the in-between, which is better suited for line drawings; (b) an adversarial loss to encourage the generated in-between to resemble line drawings; and (c) a consistency loss, inspired by cycle-consistency [Zhu et al. 2017], which further constrains the in-between to be consistent with the input frames, using backwards warping with interpolated flow. Preliminary results demonstrate that our framework achieves qualitatively superior in-between line drawings than current state-of-the-art video interpolation methods.

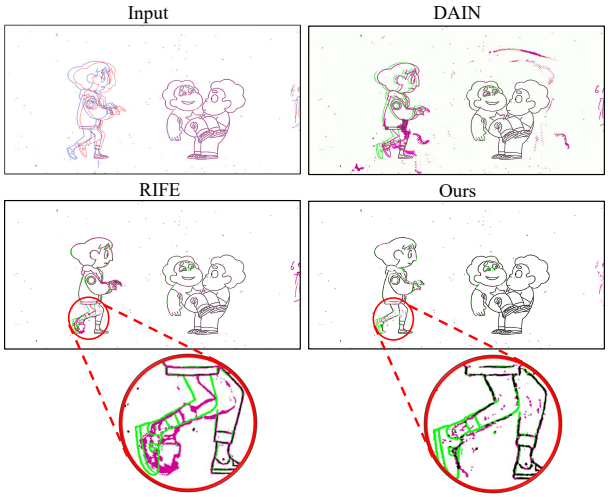

Figure 1. **Computational in-betweens for line drawings**. The two input frames are depicted overlayed in top left (red: starting frame, blue: ending frame). Results from video interpolation methods DAIN and RIFE, and preliminary results of our framework are shown. Legend for each result image: **black** pixels: true positives (correctly drawn in-between line); **purple** pixels: false positives (drawn line not present in ground truth); **green** pixels: false negatives (lines missed by the generator). Zoomed-in results for two best-performing methods are also shown, which highlights our strengths. Best viewed digitally and zoomed in. Original images ©James Baxter.

2 METHOD

Overview. Our framework is illustrated in Figure 2. Given input frames I_0 and I_1, our goal is to output a synthesized in-between $\hat{I}_t$, $0 < t < 1$. First, we calculate the forward flow $f_{0\to1}$ and backward flow $f_{1\to0}$ between I_0 and I_1 using the TVL1 algorithm [Pérez et al. 2013]. We choose this algorithm for optical flow over deep learning methods because it is edge-preserving and independent of available training data. We then feed the input frames and calculated flows into our in-between generator neural network, which returns an in-between $\hat{I}_t$, modeled as a foreground segmentation mask. This generated in-between is then fed to a discriminator neural network, and is also warped using the forward and backwards flows. Warping $\hat{I}_t$ gives estimated input frames $\tilde{I}_0$ and $\tilde{I}_1$, which are compared to the real input frames I_0 and I_1.

Data and pre-processing. There are few publicly available animation datasets, and none for our domain of black-and-white line drawings. Therefore, we generated data using Blender [Community 2018], rendering only object edges to mimic our intended domain (more details in the supplementary). To test our framework on our goal domain of real animation sequences of line drawings, we test

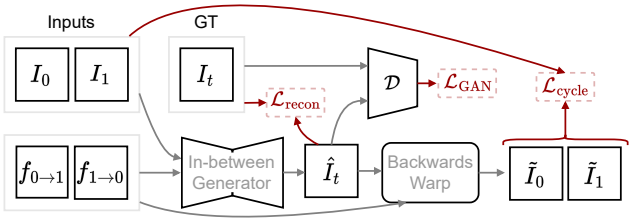

Figure 2. Illustration of our computational framework for in-betweening.

on such a sequence, called a "pencil test," by animator James Baxter from his YouTube channel [Baxter 2016].

2.1 Framework details

Our framework to generate high fidelity in-between line drawings has three core components: (a) an in-between generator network, (b) a discriminator network, and (c) a warping module that constrains our task, inspired cycle consistency. These modules result in three loss terms used to train our framework in an end-to-end fashion. We discuss these modules and losses below; training implementation details can be found in the supplementary document.

The main module in our framework is the in-between generation network. We propose to represent the generated in-between as a foreground segmentation mask instead of raw pixel data, which has two benefits. First, we can use binary cross-entropy loss with logits as the reconstruction loss ($\mathcal{L}_{\text{recon}}$ in Figure 2), which classifies each pixel in an image as either foreground (black line) or background (white background). Previous methods either use L1 loss, which favors majority values and therefore overwhelmingly returns white background pixels with our data, or L2 loss, which leads to line blurring. Second, the proposed representation better models our frames as "drawings" on "paper;" since the masks binarize the in-between, it helps synthesize crisper lines and mitigate the smearing or blurriness that usually occurs at motion boundaries in video interpolation. Our network uses a UNet with a ResNet34 underlying architecture, and was pre-trained on ImageNet segmentation [Yakubovskiy 2020].

The second module is a discriminator network trained using an adversarial loss [Goodfellow et al. 2014] ($\mathcal{L}_{\text{GAN}}$ in Figure 2). This classifies whether the in-between is "real" (i.e., from the original video data) or "fake" (i.e., generated by our generator). This encourages crisper line quality in our generated in-between, making it better resemble line drawings in our domain. Our network uses a DCGAN-based [Radford et al. 2016] discriminator.

Finally, our third module warps the generated in-between using estimated optical flow to reconstruct the two input frames. This enforces cycle consistency between inputs and their reconstructions. Since we do not have ground truth $f_{t\to1}$ or $f_{t\to0}$, we estimate these values by linearly interpolating $f_{0\to1}$ and $f_{1\to0}$, which is reasonable given small motion between inputs. We backwards warp to avoid holes, as is standard practice, and warp $\hat{I}_t$ using $f_{0\to t}$ to synthesize $\tilde{I}_0$, and use $f_{1\to t}$ to synthesize $\tilde{I}_1$. We then use a cycle-consistency reconstruction loss ($\mathcal{L}_{\text{cycle}}$ in Figure 2) between these synthesized and ground truth input images, which constrains our generated in-betweens to look "in between" the start and end frames. This added constraint introduces better structure to our task and reduces the space of plausible in-betweens, thus improving their quality. Following the first module, we again use binary cross-entropy loss with logits as our reconstruction loss.

3 PRELIMINARY RESULTS

We present qualitative results on a real line drawing animation sequence in Figure 1. More results, including those with Blender data and quantitative evaluation, are in the supplementary document. Our preliminary results are visually superior to two state-of-the-art video interpolation methods, DAIN [Bao et al. 2019] and RIFE [Huang et al. 2022]. DAIN struggles to generate "true" in-betweens in our domain. It produces numerous visual artifacts and does not deform shapes as required in an in-between. For example, notice the left character's leg where DAIN simply copies an input frame. Although RIFE produces fewer artifacts, it is still unable to deform the leg for the in-between when there are large movements (e.g., note the zoomed-in region), again copying the placement of the leg from an input frame. Furthermore, the clusters of purple pixels near character movement (on the hand and the back foot) in the RIFE result shows where the in-between is blurred. In contrast, our method has deformed the leg the most accurately to the ground truth, and shows much crisper line quality than the other two methods. Although our method was not able to render the whole foot, and the line quality around motion boundaries starts to be grainy, the position of the leg is truly "in between" the start and end frames, showing significant improvements over other video interpolation methods. There is more work to be done to continue to improve our framework so that deformed shapes are fully rendered, and so that line quality can improve further.

REFERENCES

W. Bao, W.-S. Lai, C. Ma, X. Zhang, Z. Gao, and M.-H. Yang. 2019. Depth-Aware Video Frame Interpolation. In *CVPR*.

J. Baxter. 2016. James Baxter's YouTube Channel. https://tinyurl.com/fdjwj5mj

N. Burtnyk and M. Wein. 1971. Computer-Generated Key-Frame Animation. *SMPTE* (1971).

B. O. Community. 2018. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam.

C. X. Durand. 1991. The "TOON" project: Requirements for a computerized 2D animation system. *Computers & Graphics* (1991).

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*.

Z. Huang, T. Zhang, W. Heng, B. Shi, and S. Zhou. 2022. Real-Time Intermediate Flow Estimation for Video Frame Interpolation. In *ECCV*.

H. Jiang, D. Sun, V. Jampani, M.-H. Yang, E. Learned-Miller, and J. Kautz. 2018. Super SloMo: High Quality Estimation of Multiple Intermediate Frames for Video Interpolation. In *CVPR*.

S. Niklaus and F. Liu. 2020. Softmax Splatting for Video Frame Interpolation. In *CVPR*.

J. S. Pérez, E. Meinhardt-Llopis, and G. Facciolo. 2013. TV-L1 Optical Flow Estimation. *Image Processing On Line* (2013).

A. Radford, L. Metz, and S. Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *ICLR*.

L. Siyao, S. Zhao, W. Yu, W. Sun, D. Metaxas, C. C. Loy, and Z. Liu. 2021. Deep Animation Video Interpolation in the Wild. In *CVPR*.

B. Whited, G. Noris, M. Simmons, R. W. Sumner, M. Gross, and J. Rossignac. 2010. BetweenIT: An Interactive Tool for Tight Inbetweening. In *Eurographics*.

J. Xing, L.-Y. Wei, T. Shiratori, and K. Yatani. 2015. Autocomplete Hand-drawn Animations. In *SIGGRAPH Asia*.

P. Yakubovskiy. 2020. Segmentation Models Pytorch. https://tinyurl.com/4stf98fb.

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. 2017. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *ICCV*.