Computational In-betweening for Line Drawings in Animation

Lillian Huang[1], Soumik Mukhopadhyay[1], Max Ehrlich[1,2], Abhinav Shrivastava[1]
{lilhuang,soumik}@umd.edu, mehrlich@nvidia.com, abhinav@cs.umd.edu
[1]University of Maryland [2]NVIDIA

Abstract

We present a framework for computer-aided in-betweening that is specifically designed for line drawings in animation. We separate training into two distinct parts. The first consists of a two-stream motion refinement module, which models intermediate animated movement with limited context, using the input frames and forward and backward reference flows as input. The second is the in-between generator, which takes the learned motion representations from the first module and uses them to guide the synthesis of in-between frames, modeled as foreground line segmentation. This results in better generated image quality and state-of-the-art performance when tested on real-world line-drawn animated content. In particular our method outperforms prior frame synthesis techniques which heavily depend on color information for additional motion context.

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 slower than the standard 24 frames per second, they can produce 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 task in which human animators take key frames and draw the in-betweens. We note that traditional in-betweening is a pre-production task—this process is applied to line drawings only, before frames are colored and composited with backgrounds.

Given that in-betweening is a relatively mundane task, one way to reduce animators' workload is to use a computational tool for assistance. Specifically, a computer can in-between by performing key frame interpolation on animated data. For this algorithm to be successful, we must balance two competing goals. First, the algorithm must preserve the artists' creative control. In animation, timing and spacing is imperative to get right in order to fully capture the feeling of (often exaggerated) motion that an artist means to evoke. Animators refine timing and spacing during pre-production, working with only black and white line drawings so that sequences can be easily changed. Only when these line drawings are finalized can production commit to compositing colors, backgrounds, light, and special effects into the frames for the final render. Therefore, prioritizing artists' creative control means that we must work with black and white line drawings. Second, the algorithm must produce accurate, crisp in-betweens which are, ideally, indistinguishable from what a human animator would have produced.

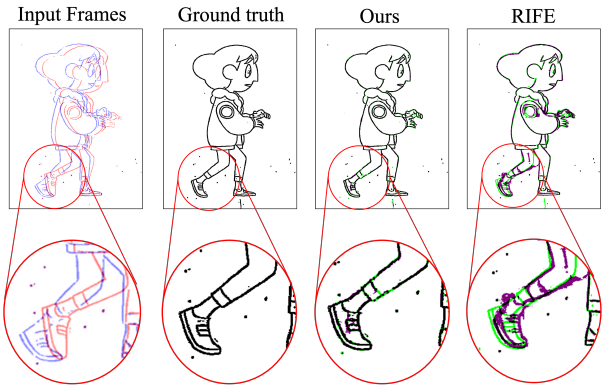

Figure 1. **Computational in-betweens for line drawings**. The two input frames are depicted overlayed in top left (**red**: starting frame, **blue**: ending frame). Results from video interpolation methods AnimeInterp and RIFE, and results of our framework are shown. Legend for each result image: **black** 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). Original images Copyright ©James Baxter.

While in-betweening appears at first to be an instance of frame interpolation, a well-studied problem in deep learning [2, 3, 10, 11, 14, 15, 21, 22, 29], it has numerous unique challenges which warrant the development of a novel algorithm. Our algorithm is designed with these challenges in mind and is motivated by the lack of satisfactory results us-

ing off-the-shelf frame interpolation solutions on this task. The first major issue is with motion modeling. Traditional frame interpolation tasks are designed for use on fully colored and textured live action video. As such, these methods can leverage dense flow fields computed using commodity techniques (TV-L[1] [24], RAFT [26], *etc*.). For in-betweening, we have only 2D black-and-white line drawings depicting the focus of a scene; dense flow methods often outright fail on these data. Furthermore, animation is a stylized medium where objects do not move and deform according to strict mathematical and physical laws as they do in live-action video. While this in itself does not necessarily preclude a traditional flow-based solution in the presence of color data [25], these complex deformations exacerbate the failure modes caused by our line data. Finally, we note that traditional solutions use l_1 or l_2 regression losses, something which is sensible when the goal is to produce a full color scene. However, since our data consist of sparse lines, this tends to regress to simple majority solutions which are either entirely background (white pixels) or gray smears where lines are averaged.

We address all of these issues in our method. Our algorithm operates in two phases which are trained end-to-end: motion refinement and in-between generation. The goal of the motion refinement network is to learn accurate motion representations. We do this with a two-stream network. Each network consumes one of the two input frames as well as the ground-truth flow field that relates the two frames (either a forward or backward flow). Since we cannot use traditional dense flow-fields, we use sparse flows computed with a point matching network. The two-stream network in the motion refinement module predicts flow fields which relate the input frames to the in-between frame. Although the motion refinement network predicts flow fields, its true goal is to learn a rich and dense motion feature map that captures the complex, stylistic, and sparse motions in the animated frames. We use this feature map as input to the in-between generator. The in-between generator consumes these motion features and predicts the in-between. Crucially, instead of treating this as predicting the pixel values of the lines, we instead supervise this network as "line-presence detection," where each pixel is either 0 for background or 1 for line. We use the binary cross-entropy loss function to train this, making the prediction task more like segmentation or classification.

In summary, our three major contributions are:

1. A method for producing in-between frames which works on line drawings to preserve the artists' creative control over their animation.

2. A method for producing rich motion features which helps deep networks leverage sparse motion information.

3. A method for accurate line production that is based on line-presence detection in the intermediate frames.

2. Related Works

Since in-betweening is a relatively tedious task, this process has been a popular one for graphics researchers to fully or semi-automate in the animation pipeline. Skeleton-based pose interpolation has been widely implemented for 3D animation, using algorithms such as splines and bezier curves to follow an object or character's motion in smooth arcs from one pose to another [23]. For 2D animation, such algorithms can be used to interpolate stroke-based drawings as well, as they are vector representations of images and can be mathematically manipulated. These vectors can depict a variety of information regarding position, color, pen pressure, time drawn, and more. However, 2D data does not hold the inherent scene information of a 3D model. Thus, to in-between 2D drawings, it is imperative that strokes are properly matched from one frame to the next, so that each frame's generated strokes properly depict the underlying object's form. Early iterations of this research required users to input the stroke correspondence manually [5, 8, 13, 19], but eventually systems were able to match strokes on their own, at least as an initial step before additional artist guidance. These matching algorithms were based on similarity metrics, calculated from stroke lengths and enclosed areas [27] or strokes' relationships to those in neighborhoods around them [28]. Once strokes were matched, in-betweens were generated by mathematically interpolating the input strokes, fitting motion arcs to each stroke and then deforming them along the arcs. There are some drawbacks to using vector-based data, however. Many animators learn their craft on paper, which does not translate well to stroke-based programs. Strokes often simplify or smooth out the characteristic irregularity of an artist's line, which can be difficult to work with in real-time—the resulting image might not look how the artist intended, and often is not as appealing. Thus, our method works with entirely raster data.

In order to process and make sense of raster image data, we turn to the field of live-action video interpolation. This task has been well-studied with deep learning methods. Beginning in 2016, [15] took a naive approach, simply feeding two input frames to a convolutional neural network and training it to match the RGB values of the in-between frame. To improve on this, [14] introduced optical flow as a way to guide interpolation. Liu et al. calculated both forwards and backwards flow between the input frames, then synthesized an in-between frame using bilinear interpolation between corresponding pixel values from the input frames. Since optical flow tracks the position of each input pixel, flow helps find correspondences between the two input images, mirroring the stroke matching step present in graphics-based

in-betweening methods. Thus, most methods now use optical flow to guide interpolation. One apparent problem to this method was the presence of muddy flow fields at motion boundaries. When objects move, they often obscure other objects in the scene, leading to problems with occlusion reasoning. When this happens, pixels from one frame cannot be mapped to any corresponding pixels in the other, so simple bilinear interpolation leads to "smeared" motion boundaries. Other researchers addressed this in various ways. [11] predicted visibility maps to try to rectify this, [2] predicted depth maps to prioritize one frame's pixel values others, [21] used context to better synthesize intermediate frames, [10] directly predicted intermediate flows, and [22] also predicted depth, but with translational variance. However, unlike the arc motion that characterized stroke-based in-betweening, all of these methods assumed linear motion between input frames. This is due in part to the smaller time steps between frames in live-action videos, which allows arcs to be approximated by a many-piecewise linear function. [29] puts arcs back into the model, assuming nonlinear motion between frames and synthesizing intermediate frames as a quadratic function of the flow. All of these methods mentioned are relatively high-performing, but they are extensively trained on large, fully colored and textured live-action datasets. They do not work when applied to line-drawn animation. Generated frames often look blurry, smeared, or unchanged from the input frames. Without color or textures, as well as the relatively exaggerated motions found in animated content, pinpointing proper correspondence between frames is difficult.

Some deep learning methods have already been used to try to improve video interpolation for animated data. Notably, [6, 25] adapts flow-based deep learning methods to better interpolate animated videos, addressing both the texturally sparser data and the more exaggerated motions. However, their methods depend on using colored data, which does not work for line drawings. In contrast, [20] designs their method for untextured, uncolored animation data. Their method applies a distance transform to the input line drawings, which simulates texture in each image. This improves the conditions to find pixel-level correspondence, leading to more robust flow fields. Our method, in contrast, does not warp input images at all, instead using flow to simply learn a deep representation of line motion between frames, and using that to guide in-between synthesis.

3. Method

Our framework is illustrated in Figure 2. Given input frames I_0 and I_2, our goal is to output a synthesized in-between $\hat{I}_1$. First, we calculate the forward flow $f_{0\to2}$ and backward flow $f_{2\to0}$ between I_0 and I_2 using the method detailed in Section 3.1. We then feed our input frames and flows into our two-stream motion refinement module. Each stream is a network which takes one frame and its corresponding flow field as input and outputs the intermediate flow. Specifically, the first stream takes I_0 and $f_{0\to2}$ as input and outputs an estimated $\hat{f}_{0\to1}$, while the second stream takes I_2 and $f_{2\to0}$ as input and outputs an estimated $\hat{f}_{2\to1}$. We supervise the training of this module with ground truth intermediate flows $f_{0\to1}$ and $f_{2\to1}$ using l_2 loss. The features from these two streams, denoted $F_{0\to1}$ and $F_{2\to1}$, are then fed as inputs to the in-between generator network, which returns an estimated in-between $\hat{I}_t$, modeled as a foreground segmentation mask. Our framework is trained end-to-end.

3.1. Optical Flow Generation

For real-world animated line drawing sequences, true optical flows are not readily available, therefore, we must generate the optical flows. Because our data consist of only black-and-white line drawings, the optical flow field are also difficult to compute. Traditional methods Lucas-Kanade [18] or TV-L^1 [24] and even newer deep-learning-based methods like RAFT [26] fail on these data. These methods heavily leverage color and texture data for matching pixels across frames, two features which are missing from our data.

Instead, we found that the Persistent Independent Particles (PIPs) approach [9] works well when tracking points in our data over time, even though the frames are visually sparse. PIPs independently focuses on each particle being tracked and iteratively updates the position and features across all timestamps using occlusion aware local deep feature correlation at multiple scales. Empirically, it has produced the best pixel-to-pixel correspondence between consecutive frames of all the methods we explored.

We use PIPs to generate *sparse* flows for our data. These flows track the movement of only the lines in our data since the white background is static. Note that PIPs does not provide flows in the standard format, only a list of points and their positions tracked over time. Thus, we calculate the sparse forward ($f_{0\to2}$ and $f_{0\to1}$) and backward flows ($f_{2\to0}$ and $f_{2\to1}$) by storing the difference between the position of the tracked points over subsequent frames.

3.2. Framework details

Our framework to generate high fidelity in-between line drawings has two core components: (a) a two-stream motion refinement module, and (b) an in-between generator network. Each module has its own distinct loss term, and both loss terms are used to train our framework in an end-to-end fashion. We discuss these modules and losses below.

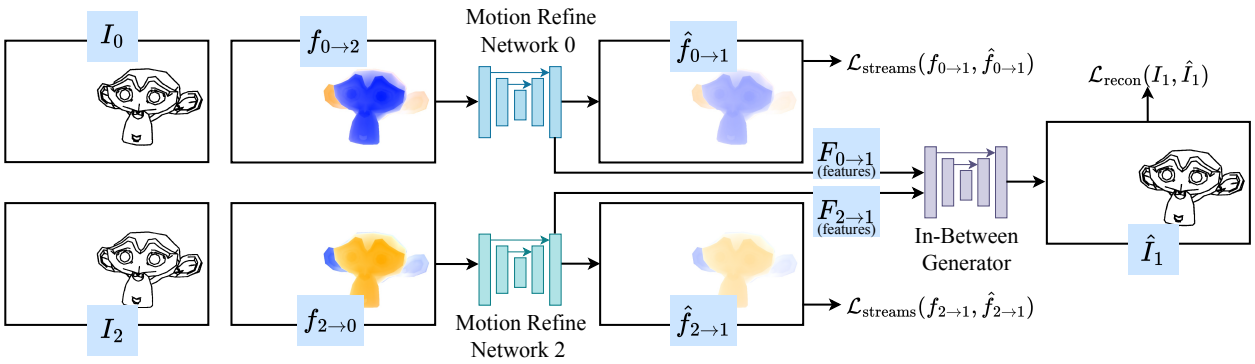

Figure 2. Network Architecture (refer to Section 3 for more details).

3.2.1 Motion Refinement Module

The purpose of the motion refinement module is to better understand the motion between the two key frames, and to be able to extrapolate what the intermediate motion might be. Given the appearance of a key frame, as well as a flow field that shows how lines move over a large time step, we want to learn how individual parts or pixels of the frame move at smaller time steps, *i.e.* the granularity at which we aim to generate in-betweens. We note that the motion between animated key frames is often nonlinear, especially because animations tend to exaggerate movement. In addition, animated key frames are often further apart in time than their live-action counterparts, since they are drawn at a lower frame rate than a traditional camera records. Thus, instead of approximating the intermediate flows by calculating a simple linear combination of the overall flows, we train the module to predict the intermediate flow instead, supervised by either ground truth intermediate flow for our Blender data, or intermediate flow generated with the method described in Section 3.1 for the pencil test. Our loss function for intermediate flow supervision is l_2 loss. In addition, animation interpolation is subject to artistic stylization and line imperfections, so it is important that we calculate intermediate flows in both directions in order to capture such effects and define the overall motion accordingly. Hence, we adopt the two-stream architecture of this module, one for each direction of movement.

As we teach this module how to calculate intermediate motion, the network learns a representation that captures all the difficult aspects of line-drawn animated motion. We note that this representation holds more complex information than if it was learning motion representation from live-action frames—while live-action videos have physical constraints, animated content must also work with motion that is at least partly spurred by artistic decisions. Because of this, we train a separate module where its sole task is to understand motion, instead of training a single network to perform both motion refinement and in-between generation. This gives us a more explicit, targeted representation of the motion in animated data, and therefore can better predict where lines should be drawn for the in-between frame.

3.2.2 In-between Generator

The in-between generator takes the intermediate motion representations from both streams of the motion refinement module and uses it as input to generate an in-between frame. We represent generated in-betweens as a foreground segmentation mask instead of RGB pixel values, and denote black lines as the "foreground" of an image. This turns our task into one of classification, or "line detection," as we are only asking if a given pixel is part of a drawn line or not. This is both a more efficient representation of our black-and-white data than RGB pixel values, as well as a better model our in-betweening task, since the generator is essentially being asked to "draw" where the foreground is in a frame.

This representation of our frames as foreground segmentation masks means that we use binary cross-entropy as our reconstruction loss. Previous methods use l_1 or l_2 loss, but these lead to various artifacts (see Section 5) when rendering an in-between in our domain. Specifically, l_1 loss favors majority values and therefore overwhelmingly returns white background pixels with our data, while l_2 loss leads to line blurring. Our segmentation-style loss helps synthesize crisper lines and mitigates the smearing or blurriness that usually occurs at motion boundaries in video interpolation.

4. Experiments

4.1. Data and preprocessing

There are few publicly available animation datasets, and none for our domain of black-and-white line drawings.

Therefore, we generated data using Blender [7], rendering only object edges to mimic our intended domain. We generated 120 videos in Blender, each with 242 frames and rendered at 24 frames per second for a total of 58,080 frames. These videos depict a Blender primitive of a monkey's head, called Suzanne, moving in space, spinning, and scaling up and down. In addition to only rendering the form's edges to resemble line drawings, we also ran the difference of Gaussians (DoG) algorithm [17] on each image to binarize our data into foreground segmentation masks. From each video, we took successive triplets of frames as our training data. We did this at two granularities: 1 intermediate frame (e.g. network inputs are frames 000 and 002 of a video, and the network output tries to match frame 001) and 3 intermediate frames (e.g. network inputs are frames 000 and 004 of a video, and the network output tries to match frame 002). Finally, we cropped our training images to 512×512 squares, both to augment our data and to make training more efficient. We also generated ground truth flow between frames for network input and supervision.

To test our framework on our goal domain of real animation sequences of line drawings, we test on such sequences, called a "pencil test," by animator James Baxter from his YouTube channel [4]. We broke this video into 24 individual frames per second, and once again ran the DoG algorithm on the frames to obtain foreground segmentation masks. We also divided these frames into triplets at the aforementioned granularities (1 and 3 intermediate frames). Although we did not have ground truth flow for this data, we used [9] to generate both *intermediate* flows (key frame to an in-between), for supervision for the motion refinement networks, and *reference* flows (key frame to key frame).

4.2. Training details

Our framework was trained end-to-end. Our two-stream motion refinement module consisted of two UNets, each with a ResNet34 as its underlying architecture [30]. Our in-between generator is also a UNet, with a modified and deepened ResNet34 as its underlying architecture. All three UNets were initialized with random weights and trained on our data. We used the Adam optimizer to train all of our networks [12]. Both UNets in our motion refinement module started training with a learning rate of 10^{-4}, which changed for subsequent epochs with cosine annealing [16]. The in-between generator network was trained starting with learning rate 10^{-3}. This learning rate was held constant for 200 epochs, and then began decreasing with cosine annealing. For all network optimizers, we set $\beta_1 = 0.5, \beta_2 = 0.999$. Our final loss $\mathcal{L}$ was calculated from the sum of the losses from each stream of the motion refinement module and the loss from the in-between generator, resulting in the following loss:

$$\mathcal{L} = \mathcal{L}_{\text{streams}} + \mathcal{L}_{\text{recon}}$$

5. Results

We present the results of our framework in this section. We show both qualitative and quantitative results on both the Blender-generated dataset and James Baxter's pencil test in comparison to two other video interpolation frameworks. We compare against RIFE [10], which currently gives state-of-the-art performance in live-action video interpolation, and AnimeInterp [25], which uses methods tailored for fully rendered and colored animated content.

Our quantitative results are shown in Tables 1 and 2. We evaluate on four standard perception metrics: Peak signal-to-noise ratio (PSNR), structural similarity (SSIM), chamfer distance, and learned perceptual image patch similarity (LPIPS). We show that we beat AnimeInterp for all metrics we evaluate and on both datasets, which is meaningful because AnimeInterp was also designed specifically for animated data. We also show that we achieve better metrics on RIFE for PSNR, SSIM, and chamfer distance when evaluating on the pencil test. RIFE still gives better quantitative results for our Blender-generated dataset. However, the Blender-generated dataset does not show complex motion—the movement of the Blender primitive is slow, there are no deformations or occlusions, and there is only one object in the frame. Thus, this dataset achieves relatively high-performing results for all methods we use for evaluation, and does not give an accurate sense of the true performance of each method for our domain. In fact, if we look at our qualitative results, shown in Figure 4, we see that although our quantitative metrics do not measure up to RIFE, our image quality is just as clean as their output for the same dataset.

On the other hand, we achieve state-of-the-art results on all but one of the metrics we calculated when we evaluate on a real-world pencil test. This data is a better representation of our target domain than Blender-generated data, given that there is more complex motion shown in the sequence, and more distinctly shows the benefits of using our system on line drawings. We can see that although our metrics in Table 2 show slight improvements over RIFE, our qualitative results in Figure 4 are vastly cleaner than those generated from either of the baselines. In particular, we point to the characteristic blurring around motion boundaries with RIFE. We can also see evidence of "ghosting" with other methods, or synthesizing an in-between by simply stacking the two input frames—observe the third example from the top in Figure 4. In contrast, our lines are much more crisp, and lines are properly deformed to show an intermediate pose that is consistent with underlying 3D geometry. For example, the left-hand character's leg can actually be found in a position that is "in between" those of the input frames.

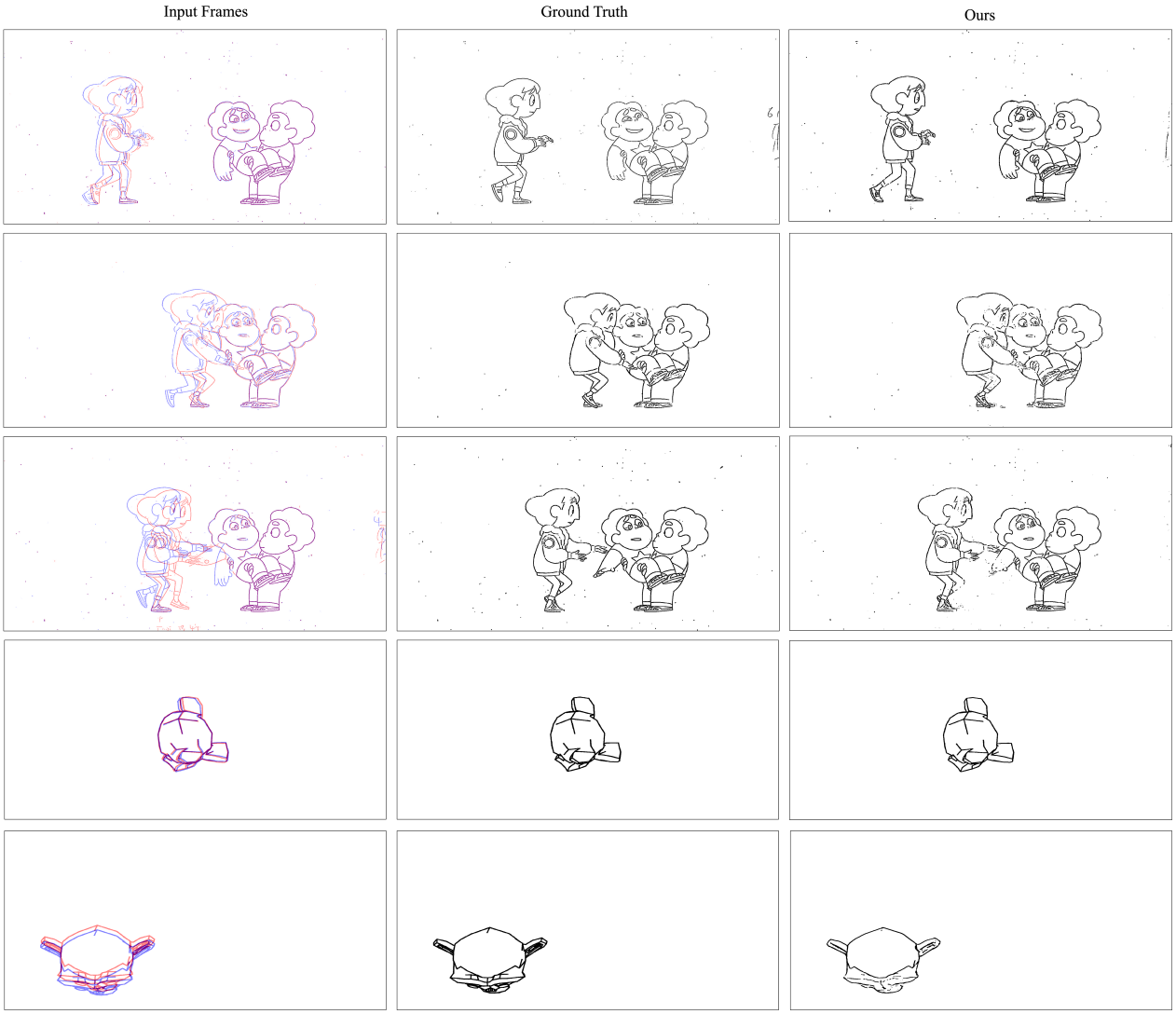

Figure 3. **Qualitative results**. The following columns are referred to left-to-right, while rows are referred to top-down. The inputs to our framework are depicted in the leftmost column; the starting frame is shown in red, while the ending frame is overlaid in blue. The second column shows the ground truth frame. The last column shows our framework's results. The first three rows show results from James Baxter's pencil test, while the last two show results on our Blender-generated dataset. Original images ©James Baxter.

Method	PSNR↑	SSIM↑	Ch. dist↓	LPIPS↓
AnimeInterp	24.1862	0.9362	9.7225	0.2837
RIFE	**32.4101**	**0.9936**	**3.7378**	0.0079
Ours	28.1267	0.9898	7.7505	**0.0064**

Table 1. Metrics with Blender dataset. State of the art results are **bold**, second-best are underlined.

Method	PSNR↑	SSIM↑	Ch. dist↓	LPIPS↓
AnimeInterp	20.2265	0.9459	28.0739	0.1121
RIFE	21.1639	0.9583	28.3712	**0.0482**
Ours	**21.4970**	**0.9632**	**25.1816**	0.0806

Table 2. Metrics with pencil test. State of the art results are **bold**, second-best are underlined.

5.1. Additional Qualitative Results

We present qualitative results in Figure 5 from additional pencil tests, taken from animator James Baxter's YouTube

Figure 4. **Qualitative comparison with baselines**. The following columns are referred to left-to-right, while rows are referred to top-down. The results to our network are shown in the leftmost column. The second and third columns show results on AnimeInterp and RIFE respectively. The first three rows show results from James Baxter's pencil test, while the last two show results on our Blender-generated dataset. We see that there are significantly more visual artifacts when generating in-betweens than with our method. Original images ©James Baxter.

channel [4] and from a YouTube channel called *Time and Spacing*, which shows animator Milt Kahl's work [1]. All pencil tests were preprocessed in the same way as described in Section 4.1.

5.2. Ablation

We present ablation studies to show the necessity of our design choices.

5.2.1 Reconstruction loss function

We ran experiments with our Blender-generated data, trained on first l_1 and then l_2 loss instead of binary cross-entropy with logits loss. We show a number of comparisons of in-between quality in Figure 6. We see that in column 1, where we used l_1 loss, the generated intermediate frames look "wiped," as if the lines have been almost entirely erased. This is because l_1 loss favors the majority, and since white pixels overwhelmingly outnumber black pixels in each frame, the generated outputs tend to have this effect. With l_2 loss, seen in the middle column, we see that there is

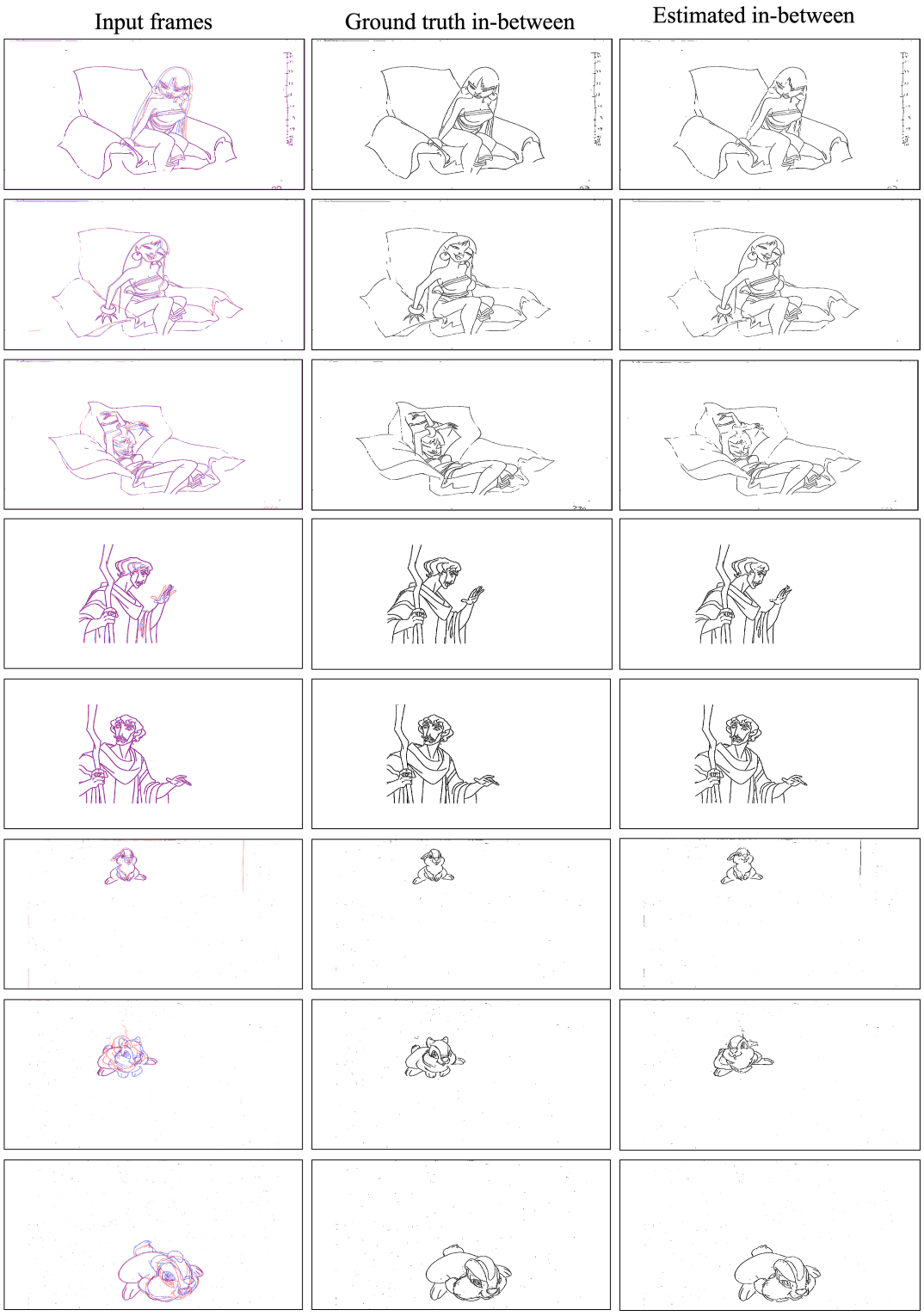

Figure 5. Additional results from pencil tests found on YouTube.

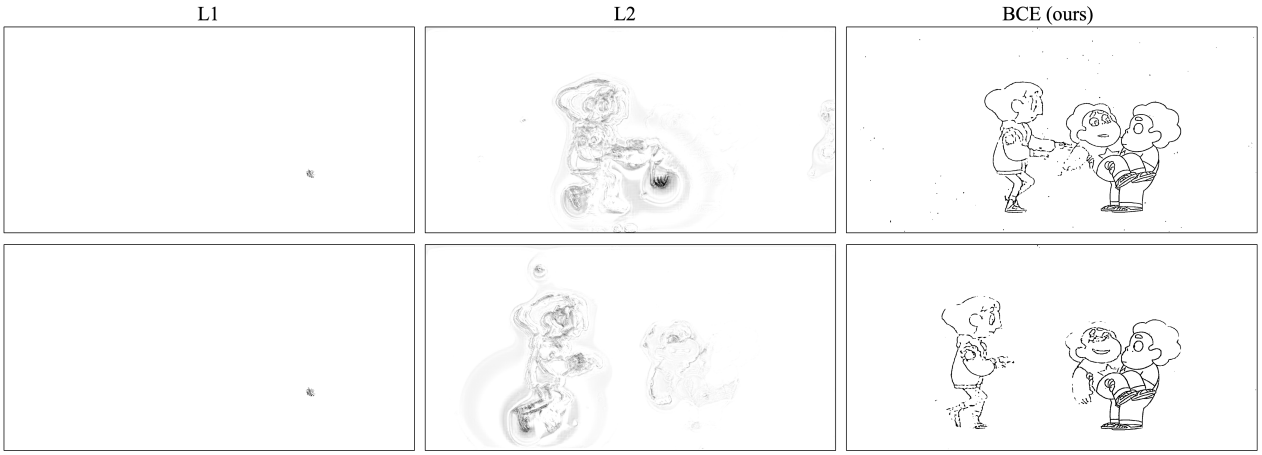

Figure 6. Ablation on various reconstruction losses with our Blender-generated dataset. We see that with l_1 loss (shown in the leftmost column), many lines are omitted from the frame, to the point of looking almost completely blank. With l_2 loss (shown in the middle column), the frames look extremely blurry. In comparison, our method of using BCE loss produces much cleaner and more accurate frames than both "traditional" reconstruction loss functions.

a smudging or smoothing effect that happens around some of the lines. This is a well-documented problem, as optimization using l_2 loss is equivalent to fitting a Gaussian distribution, which is unimodal, to data that is often multimodal.

5.2.2 No motion refinement

We also ran an experiment without using the motion refinement module, training an in-between generator alone. We used both the input frames and the overall flows as input to this in-between generator, and asked it to output a foreground segmentation mask of the estimated in-between. In this experimental setup, we have ostensibly given the network enough information to find correspondences between the images and to deform lines accordingly to generate an in-between. However, in using only one network, this setup combines motion understanding and line generation into one task for the system to solve.

We ran this experiment on the pencil test data, as the Blender-generated data depicts only very simple motion and does not deform any objects. We can see the results of this experiment in Figure 7. With only a single network, our results look extremely smudged—almost unrecognizable from the ground truth in-between, or either of the input frames. We therefore see that in-between generation is too complex a task for a single network. It is necessary to break down the task into two smaller, easier parts, and to train a network specifically to perform each part. Furthermore, it is important for the system to have explicitly learned the motion of the lines between the frames, as this helps place lines correctly and clearly in an intermediate frame.

5.3. Refinement Flow Results

We present image results from the sparse intermediate flow reconstruction efforts of the two-stream motion refinement network. Although we do not use the output intermediate flows themselves for subsequent in-between generation, their accuracy ensures the saliency of the motion representations we do use. We compare the output flows to the ground truth output flows in Figure 8. The color and shape of the recovered flows indicates that the motion refinement module correctly learned accurate magnitude and direction of motion. Moreover, the position and sparsity of these flows shows that the network is able to accurately distinguish background and foreground shapes.

6. Conclusion

Our in-between generation framework achieves state-of-the-art qualitative and quantitative results on real-world line-drawn animated content. Our two-stream motion refinement module first finds a deep representation for intermediate line motion, both in the forwards and backwards directions. This embedding space aids the in-between generator in detecting where a line may fall in the in-between frame.

Unfortunately, there are no standard metrics to evaluate generated in-between frames depicted only with lines on a white background. Although we report findings on various perception metrics from past literature, they still do not fully encapsulate the performance of our in-between generator. One future work of our research is to explore metrics suitable for line drawings. In addition, we will explore user studies with traditional animators, who work with raster line

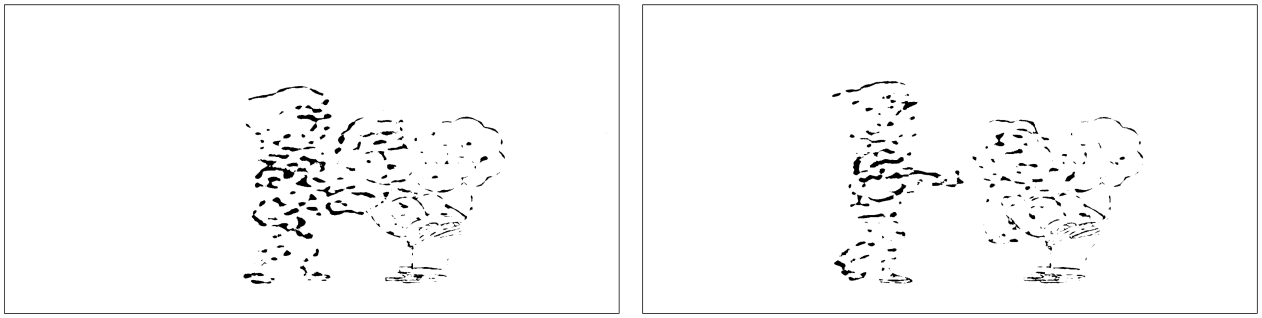

Figure 7. Examples of generated in-betweens with no motion refinement. We see that without the motion refinement module, the network does not understand how to properly deform lines, and the results look "smudged."

drawings. Finally, we want to build a more robust line drawing dataset which captures more complex animated motion. We note that the work presented in this paper is essentially a *proof of concept*. We show results on rudimentary data only, with no plans to capitalize on the models we have currently trained. If we were to proceed with this work, we would build a custom dataset from scratch for our task, and would make sure to properly inform and compensate any artists involved for their contributions.

References

[1] Time and spacing youtube channel, 2019. 7

[2] Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Depth-aware video frame interpolation. In *CVPR*, 2019. 1, 3

[3] Wenbo Bao, Wei-Sheng Lai, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Memc-net: Motion estimation and motion compensation driven neural network for video interpolation and enhancement. *PAMI*, 2018. 1

[4] James Baxter. James baxter's youtube channel, 2016. 5, 7

[5] N. Burtnyk and M. Wein. Computer-generated key-frame animation. *SMPTE*, 1971. 2

[6] Shuhong Chen and Matthias Zwicker. Improving the perceptual quality of 2d animation interpolation. In *Proceedings of the European Conference on Computer Vision*, 2022. 3

[7] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 5

[8] Charles X. Durand. The "toon" project: Requirements for a computerized 2d animation system. *Computers & Graphics*, 1991. 2

[9] Adam W Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle video revisited: Tracking through occlusions using point trajectories. In *ECCV*, 2022. 3, 5

[10] Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. Real-time intermediate flow estimation for video frame interpolation. In *ECCV*, 2022. 1, 3, 5

[11] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *CVPR*, 2018. 1, 3

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 5

[13] Marc Levoy. A color animation system: Based on the multiplane technique. In *4th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH*. ACM, 1977. 2

[14] Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *ICCV*, October 2017. 1, 2

[15] Gucan Long, Laurent Kneip, Jose M. Alvarez, and Hongdong Li. Learning image matching by simply watching video, 2016. 1, 2

[16] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 5

[17] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999. 5

[18] Bruce D Lucas, Takeo Kanade, et al. *An iterative image registration technique with an application to stereo vision*, volume 81. Vancouver, 1981. 3

[19] Takeo Miura, Junzo Iwata, and Junji Tsuda. An application of hybrid curve generation: Cartoon animation by electronic computers. In *AFIPS'67*. ACM, 1967. 2

[20] Rei Narita, Keigo Hirakawa, and Kiyoharu Aizawa. Optical flow based line drawing frame interpolation using distance transform to support inbetweenings. In *IEEE*, 2019. 3

[21] Simon Niklaus and Feng Liu. Context-aware synthesis for video frame interpolation. *CoRR*, abs/1803.10967, 2018. 1, 3

[22] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *CVPR*, 2020. 1, 3

[23] Rick Parent. *Computer animation: algorithms and techniques*. Newnes, 2012. 2

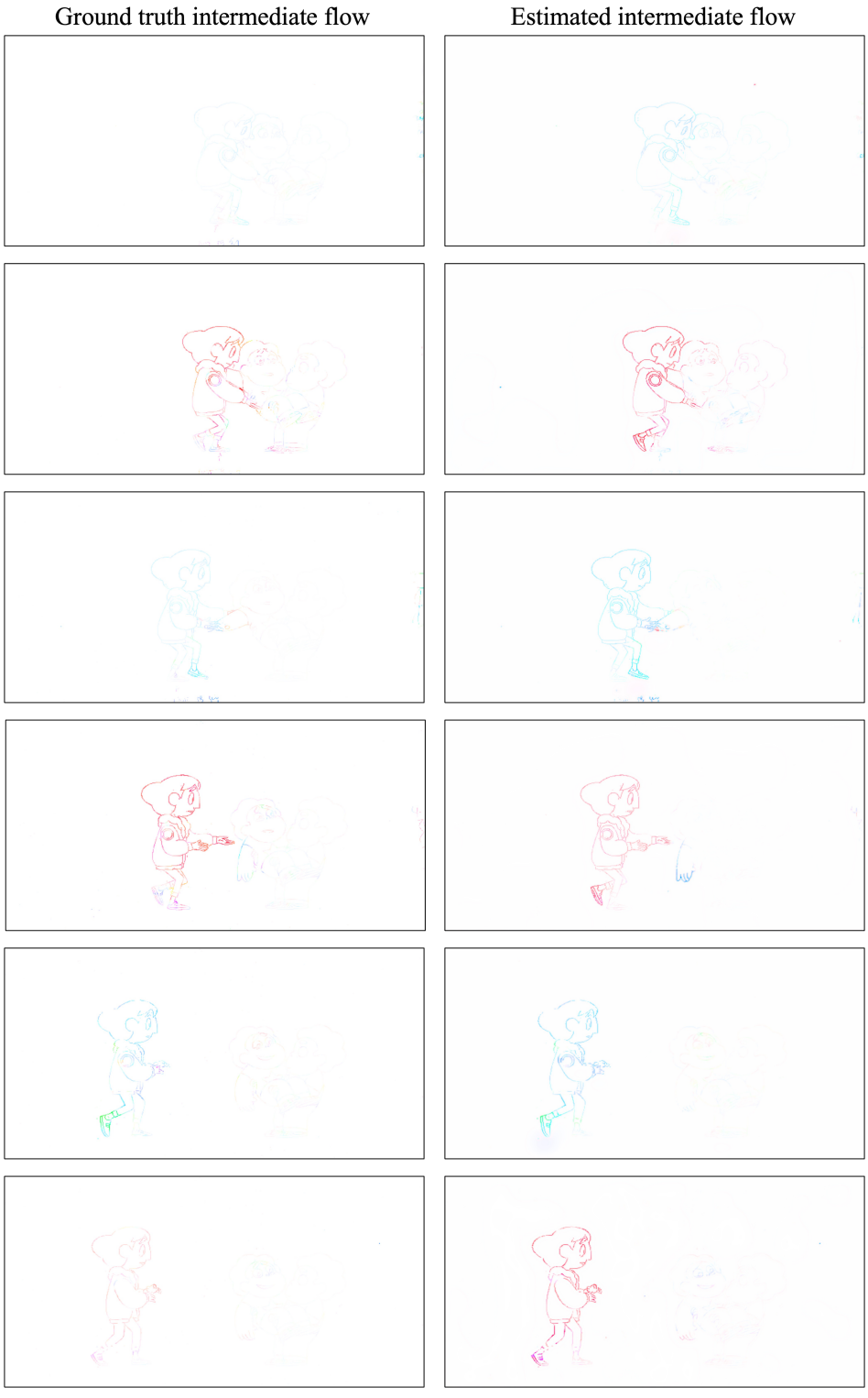

Figure 8. Comparison of ground truth intermediate flows with the motion refinement module's output flows.

[24] Javier Sánchez Pérez, Enric Meinhardt-Llopis, and Gabriele Facciolo. TV-L1 Optical Flow Estimation. *Image Processing*

On Line, 2013. 2, 3

[25] Li Siyao, Shiyu Zhao, Weijiang Yu, Wenxiu Sun, Dimitris Metaxas, Chen Change Loy, and Ziwei Liu. Deep animation video interpolation in the wild. In *CVPR*, 2021. 2, 3, 5

[26] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. *CoRR*, abs/2003.12039, 2020. 2, 3

[27] Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W. Sumner, Markus Gross, and Jarek Rossignac. Betweenit: An interactive tool for tight inbetweening. In *Eurographics*, 2010. 2

[28] Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. Autocomplete hand-drawn animations. In *SIGGRAPH Asia*, 2015. 2

[29] Xiangyu Xu, Li Siyao, Wenxiu Sun, Qian Yin, and Ming-Hsuan Yang. Quadratic video interpolation. In *NeurIPS*, 2019. 1, 3

[30] Pavel Yakubovskiy. Segmentation models pytorch. https://tinyurl.com/4stf98fb, 2020. 5