Skeleton-Driven Inbetweening of Bitmap Character Drawings

KIRILL BRODT and MIKHAIL BESSMELTSEV, Université de Montréal, Canada

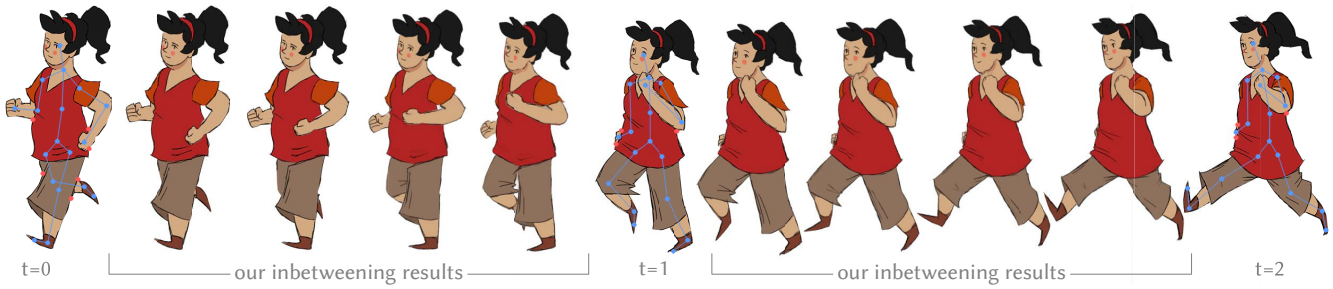


Fig. 1. For given bitmap keyframes of a character animation (marked as $t = 0, 1, 2$) and the desired skeletal motion (blue skeleton overlays), with minimal annotation (red circles), our system generates a series of high-quality in-between frames smoothly interpolating the input images. Our system does not depend on pointwise correspondences and therefore does not constrain the geometry nor topology of the input drawings, effectively handling *distant* keyframes with occlusions (arms, right leg). Running girl © thornpuck.

One of the primary reasons for the high cost of traditional animation is the inbetweening process, where artists manually draw each intermediate frame necessary for smooth motion. Making this process more efficient has been at the core of computer graphics research for years, yet the industry has adopted very few solutions. Most existing solutions either require vector input or resort to tight inbetweening; often, they attempt to fully automate the process. In industry, however, keyframes are often spaced far apart, drawn in raster format, and contain occlusions. Moreover, inbetweening is fundamentally an artistic process, so the artist should maintain high-level control over it.

We address these issues by proposing a novel inbetweening system for bitmap character drawings, supporting both *tight* and *far* inbetweening. In our setup, the artist can control motion by animating a skeleton between the keyframe poses. Our system then performs skeleton-based deformation of the bitmap drawings into the same pose and employs discrete optimization and deep learning to blend the deformed images. Besides the skeleton and the two drawn bitmap keyframes, we require very little annotation.

However, deforming drawings with occlusions is complex, as it requires a piecewise smooth deformation field. To address this, we observe that this deformation field is smooth when the drawing is lifted into 3D. Our system therefore optimizes topology of a 2.5D partially layered template that we use to lift the drawing into 3D and get the final piecewise-smooth deformaton, effectively resolving occlusions.

We validate our system through a series of animations, qualitative and quantitative comparisons, and user studies, demonstrating that our approach consistently outperforms the state of the art and our results are consistent with the viewers' perception.

Authors' address: Kirill Brodt, kirill.brodt@umontreal.ca; Mikhail Bessmeltsev, bmpix@iro.umontreal.ca, Université de Montréal, 2900 Edouard Montpetit Blvd, Montréal, QC, H3T 1J4, Canada.

0730-0301/2024/12-ART246 $15.00
https://doi.org/10.1145/3687955

Code and data for our paper are available at http://www-labs.iro.umontreal.ca/~bmpix/inbetweening/.

CCS Concepts: • **Computing methodologies** → **Animation**; **Mesh geometry models**.

Additional Key Words and Phrases: cartoon inbetweening, 2D animation, mesh deformation

ACM Reference Format:
Kirill Brodt and Mikhail Bessmeltsev. 2024. Skeleton-Driven Inbetweening of Bitmap Character Drawings. *ACM Trans. Graph.* 43, 6, Article 246 (December 2024), 19 pages. https://doi.org/10.1145/3687955

1 INTRODUCTION

Traditional hand-drawn animation, despite its expressiveness and visual appeal, has largely been replaced by cheaper digital animation. One of the main reasons of this transition is the cost of the inbetweening process, where artists need to draw *in-betweens*, which are intermediate frames creating an illusion of smooth movement between keyframes. Inbetweening not only accounts for a significant portion of the total time and cost of animation production, but it also demands great precision and skill. Overall, inbetweening is a challenging and expensive process that requires specialized training, making high-quality traditional 2D animation barely accessible to smaller studios or hobbyists.

Automating inbetweening remains a central challenge in 2D animation, as most existing solutions have not adopted by the industry. These solutions often attempt to establish dense correspondences between two keyframes, a task that is inherently difficult due to its discrete nature. This challenging problem forces many methods to resort to *tight* inbetweening, used when two keyframes are very close. However, typical keyframe pairs are not close, and may significantly differ both geometrically and topologically: some motions are quick, and many keyframes contain occlusions (Fig. 1). Moreover, some existing methods require vector input [Whited et al. 2010; Yang et al. 2018], while many artists create keyframes in raster format; vectorizing those keyframes is an open problem [Guţan et al. 2023].

A promising alternative line of deep learning methods explores a related task of video frame interpolation, often used to increase video frame rate [Huang et al. 2022; Yu et al. 2022]. Inbetweening distant keyframes, however, is a more intricate task than increasing frame rate and should not be fully automated. Inbetweening is an artistic process, as it requires precise *artistic control* over the motion between keyframes.

Frame interpolation methods, trained on high-FPS video, tend to produce unpleasant artifacts when naïvely applied to distant animation keyframes (Fig. 3). We observe, however, that for character animation, many large movements can be explained by the motion of the skeleton. The secondary deformations, such as facial expressions, movements of clothing, or muscles jiggles are often smaller in amplitude and thus can be effectively interpolated by frame interpolation pipelines.

We thus propose a novel method for inbetweening bitmap character drawings, where an artist can specify the target 3D skeletal motion. In a nutshell, our method deforms the two given keyframes according to the skeletal motion into the same pose, and then performs nonlinear or out-of-plane blending using existing video frame interpolation methods. Essentially, we sidestep the challenging dense correspondence problem by leveraging the sparse correspondences provided by the skeleton, and relax all requirements on geometry and topology of the input keyframes by using a deep learning–based raster interpolation method.

The key technical contribution of our system is the deformation of raster character drawings via given skeletal motion. In the absence of occlusions, for purely 2D motion, this task can be effectively accomplished using standard deformation techniques, such as As-Rigid-As-Possible deformation [Sorkine and Alexa 2007] or Linear Blend Skinning. However, in the presence of occlusions, such techniques typically fail, as they target smooth deformation fields, whereas for drawings with occlusions, deformation fields are typically only *piecewise smooth* (Fig. 2). Our key observation, in line with previous work, is that this piecewise smooth deformation field is a projection of a *smooth* deformation field over a 3D character shape, with the projection itself creating the discontinuities (Fig. 2). We therefore propose an algorithm to automatically construct a 2.5D non-manifold template that we use to lift the drawing into 3D, resolving the occluded regions and enabling smooth deformation (Fig. 1).

To achieve this, we propose a novel optimization formulation and a custom discrete-continuous solve strategy that optimizes the topology of the template while aiming for smooth deformation under the constraints of the skeletal motion. We then perform *blending optimization* by generating a series of possible interpolations using an existing deep learning–based solution and picking the interpolation frames with the optimal quality while producing a smooth motion via an optimization over a graph. Our optimizations leverage insights into the structure of cartoon drawings, as well as deep learning techniques to resolve occlusions.

We validate our method on a gallery of keyframe animation sequences, successfully inbetweening character motions with and without occlusions (Sec. 6), demonstrating that our method is capable of inbetweening both close and distant keyframes with vastly different geometry and topology, thus performing both tight and far inbetweening. We qualitatively and quantitatively compare our method to the state-of-the-art alternatives, and validate our algorithmic choices via a series of user studies.

2 RELATED WORK

Our work is inspired by progress in three areas: video frame interpolation, inbetweening, and deformable models.

Frame Interpolation. Video frame interpolation aims to synthesize new frames between existing ones in a video [Bao et al. 2019]. Often drawing inspiration from optical flow methods [Horn and Schunck 1981], these deep learning methods interpolate [Bao et al. 2019; Chen and Zwicker 2022; Huang et al. 2022; Kalluri et al. 2023; Niklaus and Liu 2020; Siyao et al. 2023, 2021; Yu et al. 2022] video frames in videos, including cartoons. Typically, those methods are applied to videos with at least 15 frames per second (FPS) to increase the frame rate. As a result, when applied to distant key frames, these methods often yield unsatisfactory results (Fig. 3).

More importantly, traditional cartoon inbetweening is not merely a technical task, but an art form crucial to the quality of the final animation [Williams 2001]. Clearly, artists need to have control over the inbetweens, yet frame interpolation methods typically offer no such control. In contrast, our inbetweens are controlled by the provided 3D skeletal animation, allowing the artist to concentrate on the expressiveness of the motion and let the algorithm handle the details of inbetweening.

We compare our algorithmic results with several frame interpolation methods in Sec. 6 and Supplementary materials.

Vector and Raster Inbetweening. In computer graphics, inbetweening of two vector or raster images has been a longstanding problem, with some approaches being common and others different between the vector and raster worlds.

In vector-based inbetweening, early works [Catmull 1978; Kort 2002] identified finding correspondences as one of the key challenges. Generally, the search for correspondences is often posed as a discrete problem, which is difficult to solve efficiently. The vector structure of the drawings aids in formulating the problem as correspondence between strokes, thereby reducing the search space. Unfortunately, stroke connectivity can be an unreliable cue. To further simplify the search, some previous methods rely on additional user input, such as sparse point correspondences [Reeves 1981], guidelines [Carvalho et al. 2017] or require user corrections [Miyauchi et al. 2021a; Yang et al. 2018]. Others limit the topology of the keyframes [Whited et al. 2010] or match only closed regions [Zhu et al. 2016]. Only a few approaches venture to solve the general problem of finding correspondences between drawings with different topology, such as Yu et al. [2012] or Liu et al. [2011]. Our method targets raster keyframes, which are often challenging to vectorize with reasonable accuracy [Guţan et al. 2023].

One way to facilitate the search for correspondences is to relax it into a continuous problem, searching for a displacement field instead of discrete correspondences, reminiscent of the optical flow mentioned above. For instance, Sýkora et al. [2009] apply deformable image registration to cartoon drawings, a technique that can be

Fig. 2. In contrast with an occlusion-free drawing where a typical deformation between $t = 0$ and $t = 1$ is smooth (left), deforming a drawing with occlusions (running girl on the right) requires a piecewise-smooth deformation field. The vector field discontinuities are often at the occlusion contours (blow-up). The displacement vector fields are normalized and coloured by the polar angle for visualization. Running girl © thornpuck.

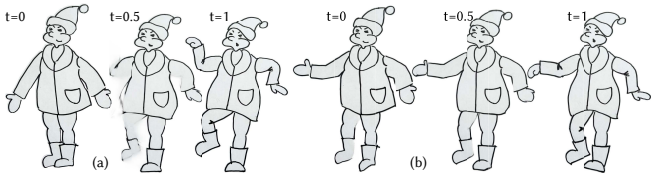


Fig. 3. Video frame interpolation methods, such as [Huang et al. 2022], trained on videos with relatively high frame rate, often struggle to convincingly interpolate distant frames (a). However, for characters in similar poses, these methods can effectively blend textures, resulting in a smooth transition (b). In both examples, $t = 0, t = 1$ are the input frames, with $t = 0.5$ being interpolated.

adapted to produce inbetweens [Noris et al. 2011]. Another relaxation approach involves finding correspondences through functional maps, recently extended to vector sketches [Myronova et al. 2023]. However, such methods generally do not handle occlusions, as occlusions require searching for a piecewise smooth displacements fields or functional maps — a highly non-trivial task.

A notable exception is the method by Li et al. [2022], which predicts occlusion regions directly and uses them to estimate the flow and synthesize new frames from user-provided sketches. The method produces impressive results, but requires the user to draw a complete sketch for each in-between frame, which can be time-consuming. Instead, our system is controlled by a skeletal animation, which normally takes less time to create than a series of precise drawings. Skeletal animation is widely accepted as a standard means of specifying a character motion.

Starting from the pioneering work of Burtnyk and Wein [1976] and some of the earliest animation software packages [Kitching 1977], skeletons, whether extracted automatically or provided by a user, have been used to facilitate inbetweening [Miyauchi et al. 2021b]. However, the correspondences or even the final inbetween frames of these methods are typically *confined* to the skeletal motion. In contrast, our method only uses the skeletal motion as a guide to improve the inbetweening quality and as a constraint to satisfy. Due to our use of a frame interpolation system, we can interpolate various motions without such restrictions.

Even et al. [2023] propose an alternative workflow to the inbetweening problem, when the user iteratively selects, deforms, and overdraws groups of vector strokes; then the system performs interpolation. Instead, we focus on the classical workflow where two keyframes are drawn in an unrestricted manner.

We are inspired by ToonSynth [Dvorožňák et al. 2018], which also uses skeletal motion — not for inbetweening, but to drive a single raster drawing. The core of our algorithm is computing a template, visually similar to the one they create *manually*. Our template is somewhat different: it is not intended to be anatomically correct, instead it is optimized to be smoothly deformed by the given animation. Their method requires a whole sequence of images, often dozens, aligned to a reference motion, and a full and precise manual puppet construction, as discussed in [Hinz et al. 2022].

Hinz et al. [2022] propose a generative model that can be trained with just a few images (8-12) and generate animation keyframes controlled by a few keypoint locations. Despite showing impressive results, they do not interpolate the drawn keyframes, instead generating *similar* images, which is typically unacceptable in inbetweening. Their input is closer to our method's input (images, skeletons and simple annotation), but they require a dozen labelled images and take hours to train per example even for a modest resolution (250x250). In contrast, our method is resolution independent and takes 32 seconds for a pair of keyframes on average. We compare with [Hinz et al. 2022] and [Dvorožňák et al. 2018] in Sec. 6 (Fig. 16).

A recent work [Mo et al. 2024] performs tight inbetweening via simultaneous tracing and finding correspondences between two digital line drawings with no texture. Despite working with raster images, they require a high-quality vectorization with stroke order of one of the keyframes. It is unclear whether their system can be applied to complex drawings with different line styles and textures (e.g., 16). Our system can work for arbitrary line styles and textures and does not require any vector input. We compare with [Mo et al. 2024] in Sec. 6 (Fig. 15).

Even if the correspondences are known, interpolating vector drawings in not trivial. Rough sketches are difficult to interpolate or synthesize [Ben-Zvi et al. 2016; Chen et al. 2023]. Most importantly,

however, interpolating drawings with varying topology, typical in the presence of occlusions, is a challenging task [Dalstein et al. 2015; Jiang et al. 2022]. We are also inspired by the method of Zhu et al. [2017] that animates planar shapes by using annotations to create a *multimesh* structure supporting topological changes. In contrast, their system is not targeting character animations with occlusions.

Cartoon Deformable Models. Our template-based deformation is related to deformable models used for cartoon animation. While the single-layered mesh is standard [Bai et al. 2016; Jacobson et al. 2011], occlusions and out-of-plane rotations are known issues of such approach. Previous work has proposed using multi-layered models [Catmull 1978; Fan et al. 2018; Fukusato and Maejima 2022; Poursaeed et al. 2020; Rivers et al. 2010], adding depth to each cartoon region [Sýkora et al. 2010], local layering [McCann and Pollard 2009], 3D proxies [Bessmeltsev et al. 2015; Jain et al. 2012; Sýkora et al. 2014], or even full 3D models [Dvorožňák et al. 2020]. Our deformable template is not necessarily anatomically correct, nor does it target to have correct 3D shape as it is never directly rendered, but is designed to be smoothly deformed by the specified skeletal animation and carry the drawn keyframe as a texture. Therefore, our template is a connected partially layered 2.5D mesh structure that can be non-manifold (Sec. 3). Our template structure is related to video meshes [Chen et al. 2011], however, our template topology is driven by the skeletal animation; occluded parts that do not move are not cut, which helps us preserve texture quality. We express the piecewise smooth 2D motion of a character with occlusions as a projection of the smooth motion of our template (Fig. 2). We compare our system to the one of [Fan et al. 2018] in Sec. 6 (Fig. 16).

Our system is loosely inspired by the work of Bai et al. [2016], where they use a single-layered mesh with specified handles and their motions to enable both keyframe animation and simulation of a drawn image without occlusions. Instead of deformation, [Sýkora et al. 2014] create impressive global illumination effects by inflating a mesh at each frame of a dense animation, which may lead to temporal inconsistencies, as noted by the authors. In contrast, we optimize for the topology of a template per keyframe, resulting in a smooth, consistent deformation of the texture and a smooth final inbetweening. Note that these methods do not perform inbetweening, i.e., smooth interpolation of arbitrary keyframe images.

3 OVERVIEW AND OBSERVATIONS

Observations. In the absence of occlusions, typical motion of a character can be represented as a smooth deformation vector field in 2D (Fig. 2a). In this case, the interior of the input character, i.e., the image foreground (textured in Fig. 2ab), can be used as the deformation domain. Once the domain is triangulated, one can simply find a smooth deformation field over that domain satisfying the skeletal motion. Such deformation can be done via standard techniques, such as As-Rigid-As-Possible [Sorkine and Alexa 2007], linear blend skinning, or others.

In general, however, drawings contain occlusions. A deformation vector field over such drawing, for many skeletal motions, will be only piecewise-continuous (Fig. 2d), so the naïve deformation strategy fails. We observe that if a full 3D character model were known, the sought skeletal deformation would be still smooth — it only becomes piecewise-smooth under projection onto the screen. Reconstructing a precise 3D character shape from a single drawing, however, is a complex task. We note that under orthographic or weak perspective projection often used in sketches, the true depth of the 3D character is irrelevant for the projection, and only the depth order matters for the final result.

We are inspired by the concept of branch cuts in complex analysis, where, intuitively, a function defined on a plane can have multiple values, but the function is continuous along each sheet; sheets connect along curves called branch cuts (Fig. 5a). In the same way, we can represent the deformable template as multiple sheets, or layers, stacked upon each other, each corresponding to an occluded or occluding body part, connecting along some curves (Fig. 5b). Compared to the traditional layering where all the layers are completely independent, our template is non-manifold but simply connected allowing for a simple definition of function smoothness.

Intuitively, this domain can be thought of as a non-planar cross-section of an unknown 3D character model through its occlusion contours (Fig. 5b). Reconstructing an anatomically correct template, however, is a non-trivial task, especially for characters with non-standard geometry or topology. Instead, we look for a template topology that has multiple layers at every detected occlusion region, and that can be smoothly deformed by the given skeletal motion (Fig. 5b). As the key step in designing such template, we need to introduce cuts into the foreground mask (Fig. 4c, foreground mask is grey).

In general, the cuts we need to introduce into the template are consistent with the drawn occlusion contours (cf. Fig. 4a and Fig. 4d), so an alternative strategy would be to detect the occlusion contours are, and cut the mesh along them. Unfortunately, such naïve strategy is bound to fail, as artists often omit parts of the occlusion contours or draw textural strokes that are barely distinguishable from occlusion contours (Fig. 4a, folds). Moreover, such task requires a human-annotated dataset of visible and invisible contours on real drawings, which might be difficult to create consistently. Instead, we use the distortion induced by the skeletal animation as the main indicator of where the cuts should be. We therefore predict occlusion masks automatically and ask for a simple annotation of T-junctions around these occlusion regions. The occlusions masks and the T-junctions make the template topology optimization efficient and robust.

We outline the following requirements for the deformation template:

- *Connected.* The template has to be connected, i.e., have a single connected component. This is a technical requirement simplifying optimization of smooth deformations over the template.
- *Locally layered.* In each occlusion region the template has to have two layers resolving the occluding body parts.
- *Admitting a smooth deformation.* The template should be deformed smoothly given the user-provided 3D skeletal deformation as constraints.

Our algorithm below is designed to reconstruct a template satisfying these three requirements.

Once such deformation template is found for each of the two given keyframes, we can deform them to each of the intermediate

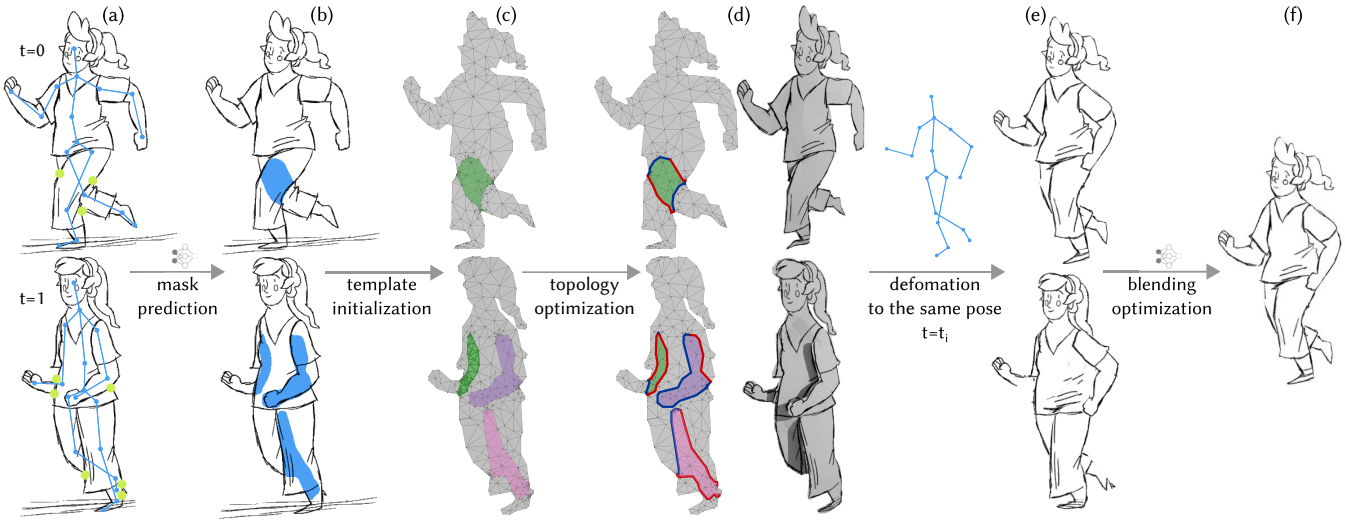


Fig. 4. Method overview: Starting with a pair of bitmap key frames, a corresponding 3D skeletal animation, and a set of T-junctions (a), for each keyframe we first predict occlusion regions (b, blue), as well as foreground mask (c, grey), using a deep neural network (b), then initialize the deformable template (c), optimize its topology, and texture it (d). For each intermediate frame, we then deform two textured templates into the same skeletal pose (e), and use a frame interpolation neural network to blend those into the final inbetween frame (f).

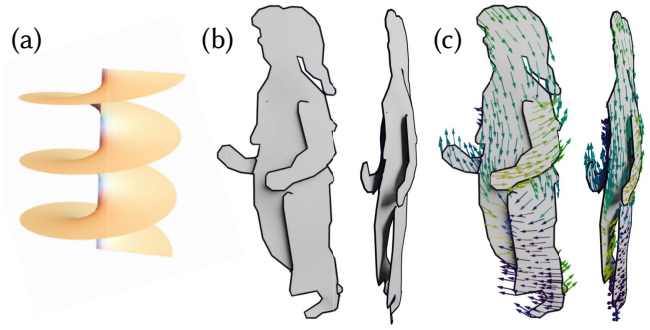


Fig. 5. Inspired by the concept of branch cuts in complex analysis (a), we lift the drawing onto a multi-sheet domain – our template (b), where the deformation is smooth. The multi-sheet domain can be thought of as 2.5D structure with partial layers, connected along the boundary of each occlusion region. The projection of the smooth deformation field onto the plane then gives the sought piecewise-smooth deformation field (c).

poses in the input skeletal animation. Having two deformed frames in the same pose but with different textures, we can blend them using a pretrained frame interpolation method. Such blends at each intermediate time frame create a smooth animation connecting the two input keyframes with the specified skeletal motion.

Unfortunately, drawings containing occlusions exhibit another challenge: we cannot reliably texture the template with a given keyframe, as some parts were occluded and therefore contain no texture. Even with modern inpainting methods, if those occluded parts become visible during animation, this might lead to visible artifacts in the final inbetweening. We propose a *blending optimization* approach to tackle this challenge, where we use more of the keyframe that contains less occlusions while simultaneously picking the best possible frame interpolation, leading to less artifacts.

Overview. Our input is two bitmap keyframes, the desired 3D skeletal motion in the image space, and a small set of T-junctions (Fig. 4a, Sec. 4). We first use our deep network to predict foreground and occlusion masks on both keyframes (Fig. 4b, Sec. 4.1). Leveraging these masks and the provided T-junctions, we initialize (Sec. 4.2) and then optimize for the deformation template topology (Fig. 4c, Sec. 4.3), which allows for a smooth deformation of the drawing despite the occlusions. Even though formally the problem is NP-hard, we use occlusion regions–specific observations to formulate a more efficient optimization problem, linear in the number of occlusion masks boundary edges (Sec. 4.4). We then use the optimized template to deform the two keyframes into the target pose at each time frame (Fig. 4d, Sec. 5.1). Finally, now that the two character drawings are in the same pose at each time frame, we perform blending optimization (Fig. 4e, Sec. 5.2). To that end, we use a pretrained frame interpolation network to generate a series of possible interpolation frames. Leveraging these series, we pick a sequence of frames that are using the frame containing less occlusions, while still providing a smooth interpolation (Sec. 5.2), producing the final smooth animation (Fig. 4f).

4 ALGORITHM

The input to our algorithm is two bitmap keyframes for times $t = 0$ and $t = 1$, and the corresponding animation of the 2D or 3D skeleton, represented as a hierarchy of joints $i = 1, \ldots, N$ with trajectories $x_i(t) \in \mathbb{R}^2$ or $\mathbb{R}^3$. If full automation is required, one can predict the 3D character pose from the keyframe directly using, e.g., Sketch2Pose [Brodt and Bessmeltsev 2022] and interpolate the skeletal poses via modern pose inbetweening methods [Starke et al.

2023]; those are complementary to our system (see Sec. 6). In the rare case where a drawing contains occlusion regions with more than two local layers, i.e., where a body part is occluded by more than one other body part, we require manually annotated occlusion masks, one per layer, sorted by depth (an example in Fig. 9). A full 3D skeleton is not required, our algorithm only needs a 2D skeletal motion and, if there are occlusions, depth order of joints near those occlusion regions.

To disambiguate occlusions, we also require the user to specify T-junctions around the occlusion regions (Fig. 4a). Usually there are fewer than 7 points per drawing, which are easy to select once we display the occlusion regions. We found all the algorithmic solutions, including deep learning architectures, unreliable to infer T-junctions robustly: Drawings often contain many T-junctions that are coincidental and are not indicative of any actual occlusions or depth ordering (Fig. 4, T-shirt sleeves). In our experiments, placing the T-junctions takes a user a few seconds, compared to minutes drawing a single image.

4.1 Foreground and Occlusions Masks

We predict background/foreground (Fig. 4, foreground masks are grey) and occlusion masks (Fig. 4b, occlusion masks are blue) using a deep neural network with two prediction heads, trained on their respective datasets. For both, we use a U-Net, the encoder-decoder architecture segmentation model [Ronneberger et al. 2015], predicting a pixel heatmap of the foreground or occlusion regions. To alleviate the need for a large dataset, as a backbone encoder we use the pre-trained DINOv2 model [Oquab et al. 2024] producing reliable visual features, namely `vit_large_path14_dinov2`. This setup achieves the intersection over union (IoU) score of 0.65. A different backbone `efficientnetv2_m` trained from scratch achieves the score of 0.56. We consider the pixel inside the foreground or occlusion mask if the corresponding head outputs a value greater than 0.8. We then convert the boundaries of the occlusion masks to polylines.

Once the occlusion masks are predicted, we leverage the specified T-junctions to refine the masks' shapes. Precisely, we first identify the specified T-junctions with the occlusion masks by proximity, and then deform each occlusion mask such that its boundary passes through the corresponding T-junctions. To this end, we triangulate the interior of each occlusion mask and deform the triangulation via minimizing symmetric Dirichlet energy (Eq. 2). For each T-junction, we find the nearest vertex on the occlusion mask border and pull the mask towards the T-shaped connection by these vertices fixing the other points far away from T-junctions, both expressed using the same energy terms as (Eq. 2), see Sec. 4.3 for more information.

Dataset and Training. We hired a third-party who annotated occlusion masks for 981 drawings from the sketch dataset in [Brodt and Bessmeltsev 2022]. We additionally annotated 10 high quality sketches for validation. We extended our dataset by adding 2432 synthetic images, renders of various poses of the SMPL model [Loper et al. 2015] along with their occlusion masks, in a non-photorealistic style with occlusion contours highlighted in Blender. We trained the network with binary cross entropy loss for 15 epochs using AdamW optimizer [Loshchilov and Hutter 2019]. We used gradually increasing learning rates for each transformer block of layers: $1.25 \cdot 10^{-6}$ for layers 1-8, $2.5 \cdot 10^{-6}$ for layers 9-16, $5 \cdot 10^{-6}$ for 17-24 and 10^{-5} for 24-32. Without this technique the final accuracy degrades significantly, resulting in a lower score of 0.62 instead of our score 0.65. During training we resize all images to 512×512px resolution and use a series of standard image augmentations. The model trains for 2 hours on a single NVIDIA A100 GPU 40GB with the batch size of 2. The synthetic augmentation improves the score by 0.01.

4.2 Template initialization

Once the foreground and the occlusion masks are predicted, we create the base mesh that will serve as the initialization for our template optimization (Fig. 4c, grey). To this end, we mesh the foreground mask via Triangle library [Shewchuk 1996] by constructing a conforming Delaunay triangulation where we enforce edges along the boundaries of the occlusion masks and the foreground masks. For quality, we additionally constrain the minimum triangle angle to be $\geq 20^\circ$. We similarly mesh each occlusion region.

We then initialize the template (Fig. 4c, colored regions; Fig. 6a) by overlaying each meshed occlusion region on top of the meshed foreground mask, connecting them at each occlusion region boundary — note that the boundaries of the occlusion masks are meshed consistently. Simply put, each occlusion region forms a 'pocket' on the main mesh, 'sewn' from all sides. The following optimization partially cuts off these pockets along their boundaries to minimize the distortion (Fig. 4c, Fig. 6).

Point Constraints. To simplify implementing the skeletal deformation as point constraints, we additionally insert handle vertices into the mesh corresponding to each joint and the middle of each bone, which will 'pull' our template into the new pose. For point handles outside the occlusion regions, we simply insert that point as a vertex at that location in the mesh. For the point handles inside an occlusion region, we insert the point as a vertex into an *arbitrary* layer of the two mesh layers inside the region. The subsequent optimization (Sec. 4.3) finds the topology of the template and thus implicitly selects which layers is 'forward' and which one is 'back'. We refer to this set of constrained vertices as *Constraints*.

4.3 Optimizing the template topology

Our goal is to create the 2.5D template from a key frame, such that its deformation, induced by the skeletal animation, is smooth.

In the following, we segment the boundary of each occlusion region into edges that are 'visible' occlusion contours (Fig. 6c, red), 'hidden contours' (Fig. 6c, blue), and non-manifold edges (Fig. 6c, orange). Once detected, the initial template pockets will be cut along the visible and hidden contours, forming two locally disconnected layers. For the visible contours, we connect the main, i.e., non-occluded, layer with the back layer of the occlusion mask; for the hidden contours, we connect it with the front layer. The visible contours thus allow for a discontinuity after the projection onto the screen. We do not cut the non-manifold edges, thus allowing for two layers to locally merge (Fig. 5). Note that visible or hidden occlusion contours do not necessarily correspond to the contours explicitly drawn or omitted from the drawing (see Sec. 3).

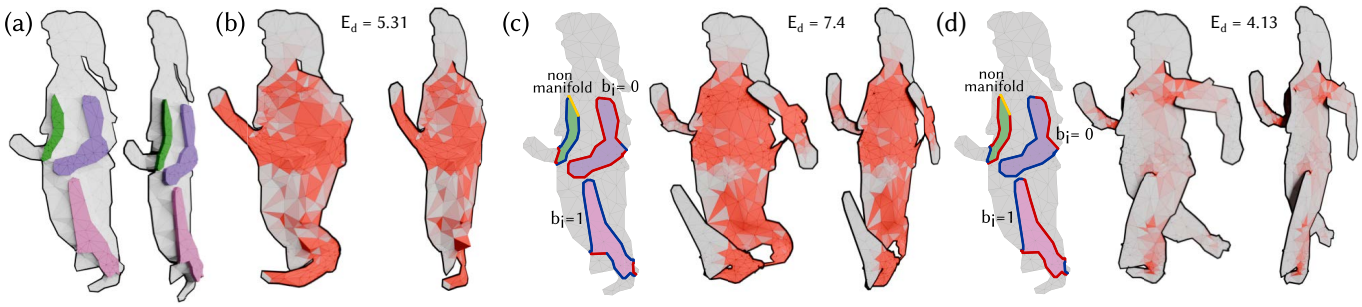


Fig. 6. Optimizing the template connectivity: the initial topology of the template with 'pockets' for each occlusion region (a), deformation of the initial template (b), a visualization of suboptimal assignment of the binary variables and its deformation into the target pose (c), and the optimal assignment with its deformation (d). We visualize the Symmetric Dirichlet energy per triangle from grey (low) to red (high). In 2D views, blue parts of the contour correspond to the 'visible' occlusion contours, and red to the hidden ones.

Requiring a user to manually annotate visible and invisible parts of the contours is not only tedious and cumbersome, but also requires precision. Small annotation mistakes can manifest as noticeable artifacts in the final result (see Fig 8). Instead, we optimize the topology as follows.

We first identify non-manifold edges where different layers of the final template will merge (Fig. 6d). Heuristically, we choose multi-valence joints, such as chest or hip joints, and pick the closest boundary edge of an occlusion region that intersects some bone in 2D.

Inspired by OptCuts [Li et al. 2018], to each other edge of an occlusion region boundary we assign a binary variable $b_i \in \{0, 1\}$, where $b_i = 0$ indicates that the edge belongs to a hidden occlusion contour, and $b_i = 1$ indicates the edge is visible. The visible edges are depth discontinuities, so for each $b_i = 1$ we cut the occlusion region along that edge from the rest of the template; for $b_i = 0$, we cut off the occluded region (Fig. 6, $b_i = 0$ are marked red, $b_i = 1$ blue).

We then formulate our problem as follows:

$$\min_{b_i=\{0,1\}, x\in\mathbb{R}^{2n}} E_d(b, x) + E_{pose}(b, x), \tag{1}$$

where E_d is symmetric Dirichlet energy of the deformation of the template with the given new joint positions. More concretely, for a fixed set of b_i and thus fixed template,

$$E_d(x) = \sum_{f\in F} A_f(||\mathbf{J}_f(x)||_F^2 + ||\mathbf{J}_f^{-1}(x)||_F^2)$$

$$E_{pose}(x) = \frac{\mu}{2} \sum_{i\in Constraints} ||x_i - \hat{x}_i||_2^2,$$

where $\hat{x}$ are the point constraints. Here f iterates over all the faces of the mesh, A_f is the area of f in the original mesh, and $\mathbf{J}_f(x)$ is 2x2 Jacobian of the affine transformation between the original face and the deformed mesh face f. We normalize the areas such that $\sum_{f\in F} A_f = 1$. The constant μ controls the balance between the desired distortion and the precision of the specified joint locations; in our experiments we kept it as $\mu = 200$.

Continuous Optimization. For a fixed set of binary variables, we optimize the continuous energy using Laplacian-preconditioned quasi-Newton method, or Sobolev gradient method in Sobolev space H^1, similar to [Kovalsky et al. 2016]. We use the standard cotangent Laplacian discretization [Pinkall and Polthier 1993] and precompute its Cholesky decomposition once, as the matrix does not change during each continuous optimization. Note that for different assignment of binary variables, the topology of the mesh changes, so we need to recompute the Laplacian. Laplacian is a singular matrix with a translation as the 1-dimensional null-space, so we project the null space out to avoid ill-conditioned gradients. We optimize until the gradient norm is less than a threshold or the maximum number of iterations is reached (0.01 and 5000 respectively in our implementation). We use backtracking line search with Armijo and Wolfe conditions and the constants suggested in Nocedal and Wright [1999].

All the point constraints are in 2D; if a 3D skeleton is supplied, we use its orthographic projection. Note that in contrast with typical parameterization problems, in our setup non-bijective deformations can be perfectly valid. For instance, any new occlusion that was not present in the original frame is a source of non-bijectivity. However, we do need local injectivity, i.e., we need to avoid folded triangles (Fig. 13), as this would cause issues in texturing. To enforce local injectivity, we use the line search upper bound from Smith and Schaefer [2015].

4.4 Efficient Formulation

Solving this nonlinear mixed-integer optimization directly, however, is NP-hard, and practically infeasible: Each edge on the occlusion region boundary gets a binary variable, so a typical number of binary variables in our setup would be on the order of hundreds for a typical drawing.

Instead, our key observation, which we refer to as *the switching rule*, consistent with the previous literature [Karpenko and Hughes 2006], is that boundary of each occlusion region switches between visible and hidden parts only at two kinds of points: T-junctions and cusps (Fig. 7). Precisely, since T-junctions are often formed by one body part occluding another, those are the points where the visibility of the occlusion contour switches. Similarly, at a cusp, located in the regions of negative Gaussian curvature on the body [Koenderink and van Doorn 1982], such as shoulders or hips, a visible contour

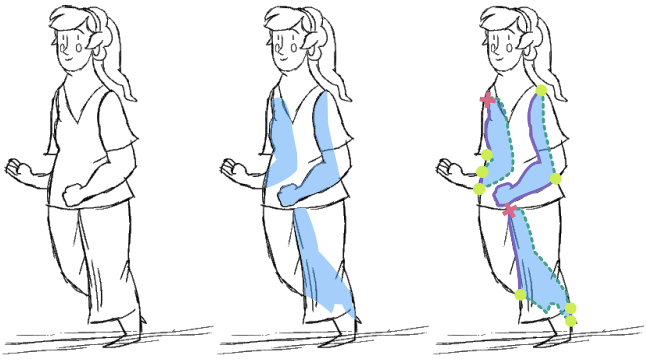


Fig. 7. T-Junctions (yellow-green circles) and cusps (red crosses) are often the only locations where a contour of an occlusion region changes its visibility. We ask the user to specify the T-junction locations and find the cusps automatically.

is occluded by the surface itself and becomes hidden. While we do not know the locations of cusps, we observe furthermore that in all examples we analyzed, an occlusion region boundary may not contain more than one cusp, as typically there is no body part connected to hips and shoulders simultaneously.

These observations allow us to significantly simplify the problem. Knowing that to one side of each T-junction the occlusion region boundary is necessarily visible, and to the other side it is hidden, for each region we add equality constraints around the T-junctions, enforcing the switching rule, and add an inequality constraint allowing for maximum one cusp, i.e., a maximum one switch away from T-junctions.

The final optimization problem is then as follows:

$$\begin{aligned} \min_{b_i=\{0,1\}, x\in\mathbb{R}^{2n}} & E_d(b,x) + E_{pose}(b,x), \\ \text{s.t.} & \sum_i |b_i - b_{i^+}| \le 1, \text{for } \; i, i^+ \; \text{not separated by a T-junction} \\ & b_i = 1 - b_j, \text{if } i, j \; \text{are separated by a T-junction,} \end{aligned} \tag{2}$$

where i^+ is the next edge along the closed loop, occlusion region boundary, i.e., $i^+ = i + 1 (\text{mod } n)$.

With these constraints, instead of exponential, the problem becomes worst-case *linear* in the number of edges. With k denoting the number of annotated T-junctions for a region, depending on the parity of k, the total number of valid assignments of b_i, is either 2 (only two) or $2(n-k)$ for k even and odd respectively. Indeed, if k is even, one can easily verify that by a simple parity argument, there can be no cusp. If k is odd, there will be necessarily one cusp. Therefore, for an even k, we test exactly two combinations: Setting $b_i \in \{0, 1\}$ for an arbitrary edge, by the switching rule, defines the binary values for all the other edges. For an odd k, we must find the location of the cusp, which can be located at any of the vertices except T-junctions, $(n-k)$ it total; for each location, there are two valid assignments.

Solver Mechanism. We note that each occlusion region is independent of the others. We therefore further simplify the optimization by optimizing each region separately, keeping the others fixed.

For contours with an even number of T-junctions, we simply check both valid combinations by running the nonlinear optimization twice and choosing the minimal energy. For the odd cases, while one can use a branch-and-bound nonlinear mixed integer solver, in our implementation we use an automatic hyperparameter optimization framework optuna [Akiba et al. 2019] in their default configuration, implementing the Tree-Structured Parzen estimator to reduce the search space [Watanabe 2023] (see Algorithm 1). In our setup, for each occlusion region we typically have $k \le 4$. We further reduce the number of edges n by simplifying the boundary of the occlusion masks with Douglas-Peucker algorithm with the distance $\varepsilon = 2$px. In the end, we typically have fewer than 60 edges for all the occlusion regions in an input image (Table 2).

ALGORITHM 1: Template topology optimization

```
Function template_optimization(mesh, occlusion_masks) is
    forall occlusion_mask in occlusion_masks do
        b_assignments := valid_combinations(occlusion_mask);
            // returns a set of valid assignments of b
            from Eq. (2)
        meshes := ∅;
        forall b in b_assignments do
            m := mesh;
            forall contour in occlusion_mask.contours do
                m := arg min_{m_j} energy(m_j);
                // where m_j = cut(j, m, contour,
                    occlusion_mask, b)
            end
            meshes := meshes ∪ {m};
        end
        mesh := mesh with minimal energy from meshes;
    end
    return mesh;
end
Function cut(j, mesh, contour, occlusion_mask, b) is
    [v_1, ..., v_n] := contour.vertices;
    b_1 := b[v_1]; // returns b_1 for edge associated to v_1
    b_n := b[v_n];
    if b_1 = b_n then
        j := n;
    end
    vertices := [v_1, ..., v_j];
    occlusion_vertices := occlusion_mask.vertices[b_1]; // visible
        vertices for b_1 = 1 and hidden ones for b_1 = 0
    mesh := incise(mesh, vertices, occlusion_vertices); // cuts the
        mesh along the vertices using vertex reindexing
        from occlusion_vertices
    if b_1 ≠ b_n then
        mesh := incise(mesh, [v_{j+1}, ..., v_n],
          occlusion_mask.vertices[b_n]);
    end
    return mesh;
end
```

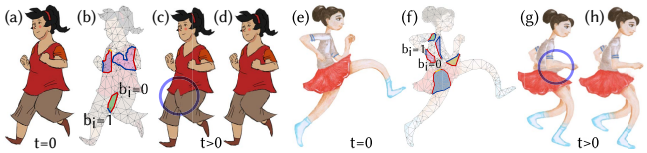


Fig. 8. Relying on user annotation for template topology is problematic, as this extra annotation requires precision, while minor mistakes in the template topology (b, f) can lead to visible artifacts in the final result (c, g). Visible occlusion contours are blue, hidden are red.

5 INBETWEENING

Once the two template meshes are computed at the initial and final key frames respectively (Fig. 4d), we now can use these templates to deform the input images. For any intermediate time $0 < t < 1$, we leverage the input skeletal motion $x_i(t)$ to deform both templates into the same pose (Sec. 5.1, Fig. 4e). Then, in Sec. 5.2, we perform the blending optimization, where we leverage those deformed frames to generate series of potential interpolations via a frame interpolation method, and solve an optimization problem on a graph to find the optimal inbetweening frame sequence.

5.1 Texturing and Animation

We first texture each template mesh using the input image. If the input key frame does not have any occlusions, then all the triangles in the template mesh are visible, so we use the input image directly as the texture for the overlaid template mesh. As the mesh and the texture are already in the common coordinate frame, we use the (x, y) coordinates of the mesh as the (u, v) texture coordinates.

If there is at least one occlusion region, however, some triangles in the template mesh will be occluded. By construction, no template mesh edge can intersect the boundary of an occlusion region, so each template triangle (except perhaps its edges) is either fully inside the occlusion region or fully outside. All the triangles outside occlusions regions get textured as above.

Inside each occlusion region, containing at least two layers, we need to first identify the depth order of layers. We leverage the template, as well as the point constraints (Sec. 4.2), and the depth order of bones or joints in the occlusion region to estimate average depth of each layer within the occluded region. To this end, we use Bounded Biharmonic skinning, which works on the non-manifold meshes [Jacobson et al. 2011]. Once the skinning weights are computed, we use them to assign the depth value for each vertex as simply the sum of the depth values for involved joints multiplied by their skinning weight. We then average these values over each layer of the occluded region, and select the layer with the smaller depth as the front layer. The front layer is visible, so we texture it as previously by simply projecting the input image.

Each occluded layer of triangles, however, is not visible, so the current key frame does not contain its texture. To texture it, we first attempt to use the texture from the other key frame: we deform the template to the other pose using our nonlinear optimization in Eq. 1, and test whether it falls into any occlusion region of the other key frame. If it is fully outside, we texture that triangle using that other key frame. If, however, even after the deformation, the triangle is either fully or partially occluded, we cannot texture it, so we use a state-of-the-art inpainting algorithm [Suvorov et al. 2021] which works for images with high resolution.

We then deform both textured skinned templates for each intermediate time step $0 < t < 1$ using the same optimization in Eq. 1, resulting in two bitmap frame sequences $I^{\rightarrow}(t)$ and $I^{\leftarrow}(t)$ for forwards and backwards animation sequences respectively (Fig. 4e). The deformation can be also done with the computed skinning weights; the difference between those deformations is not very significant for our final results.

5.2 Blending Optimization

Two deformed image sequences, $I^{\rightarrow}(t)$ and $I^{\leftarrow}(t)$ follow the skeletal motion, so at every time step they are in the same pose. However, these deformed templates carry different textures, one from each keyframe only. In this section, we find an optimal interpolation between these sequences that becomes our final inbetweening.

First, for each time intermediate time step $0 \leq t \leq 1$, we compute a sequence of frames interpolating the pair of images $I^{\rightarrow}(t)$ and $I^{\leftarrow}(t)$ using a pre-trained frame interpolation model RIFE [Huang et al. 2022] with their default parameters. Even though it is not currently state of the art, in all our experiments it performed better than the newer methods. One of the core comporenents of Huang et al. [2022] is predicting an optical flow vector field deforming one image into another.

More specifically, at every time step the model interpolates two images, producing an image $I(\alpha, t) = F(I^{\rightarrow}(t), I^{\leftarrow}(t), \alpha)$. The $0 \leq \alpha \leq 1$ is the interpolation parameter, such that $I(0, t) = I^{\rightarrow}(t)$, and $I(1, t) = I^{\leftarrow}(t)$. We thus get a matrix of images $I(\alpha, t)$ (Fig. 9a).

We now need to select a sequence of frames $I(\alpha_i, t_i), t_{i+1} > t_i$ connecting the keyframes $I(0, 0)$ and $I(1, 1)$, corresponding to a path in the image matrix connecting the top-left and bottom-left images. A naïve choice would be as follows: set $\alpha = t$, i.e., with increasing time we increase the weight of the final keyframe. This naïve strategy, corresponding to the red diagonal of the image matrix in Fig. 9a, even though it formally interpolates the two drawings, may create ghosting artifacts and flickering (Fig.9c, bottom and Supplementary video). There are two main sources of these issues: the quality of RIFE interpolation and the quality of texture inpainting. RIFE, especially for two significantly different drawings, even deformed to the same pose, often fails to find a reasonable optical flow, creating ghosting artifacts. Similarly, inpainting algorithms are also often unreliable and can lead to artifacts.

We seek to minimize the visual artifacts of that sort. Our main observation is that if one of the input keyframes contains less occlusions, we can use more of it in the final optimization to get a better quality inbetweening. For instance, if the first keyframe in Fig. 9, corresponding to $I(0, 0)$ contains less occlusions than the second frame, our path should stay closer to the left side of the matrix. Furthermore, the quality of RIFE interpolation is different for different time steps; our path should prefer moving horizontally for the t values with better interpolations.

Following these observations, we define two metrics, *texture quality*, measuring how much inpainting could affect that frame, and *interpolation quality*, measuring the quality of RIFE output. At a

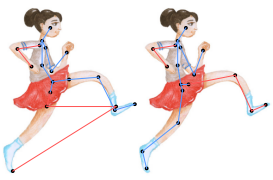
time step t, for both $I^{\leftarrow}(t)$ and $I^{\rightarrow}(t)$, we define texture quality $q(I)$ as simply intersection over union (IoU) of the inpainted area in that pose. Note that $q(I^{\rightarrow}(0)) = q(I^{\leftarrow}(1)) = 1$, because in the original keyframes none of the inpainted areas are visible. As we deform the keyframes, however, some initial occluded areas become visible, decreasing q, indicating the 'loss' of texture quality. We thus set $q(0,t)$ and $q(1,t)$ to these values and set $q(\alpha,t)$ using linear interpolation of those.

To measure the interpolation quality of RIFE, we prefer how *incompressible*, for instance, purely translational, optical flows, deform the texture. We therefore measure $Q(t)$ as $\int_I ||\operatorname{div} v(t)||^2$, where $v(t)$ is the RIFE optical flow at time t, represented as a 2D vector field, and div is the divergence operator. Note that it does not depend on α, only on t, so it is a constant per row in the image matrix.

Equipped with those two metrics, we form a graph where each vertex corresponds to an image in the matrix and edges connect all the vertices between t and $t+1$ that have $\alpha_i - \alpha_j < \varepsilon$. Here the parameter ε controls the smoothness of the final animation; in our experiments we set it to 0.02. We set a cost of each vertex as $q(I)$, and cost of each edge as $0.5(Q(t) + Q(t+1))$. We then find the shortest path in that graph connecting $I^{\rightarrow}(0)$ with $I^{\leftarrow}(1)$ using Dijkstra's algorithm, trivially converting the vertex costs into an additional term in the edge costs. This graph represents the best quality interpolation between the two keyframes, completing the stage of blending optimization, and forming our final result (Fig. 9c, top).

6 RESULTS AND VALIDATION

So far we have shown a few examples of in-between frames, algorithmically generated from two given bitmap character drawings, skeletal motions, and T-junctions (Fig. 1, 4). Our system allows to generate inbetween frames for *close or distant* key frames, operating with unrestricted bitmap drawings, but fully controlled by the given skeletal animation — a task inaccessible to previous work. Our novel optimization allows us to successfully resolve occlusions typical for character drawings, see, e.g., Fig. 10, Fig. 11 for additional results. Note that we support both planar motions (e.g., santa, snowman in Fig. 10) and complex 3D motions (e.g., pirate in 15). Most of these input keyframes contain occlusions, such as crossed legs or arms; our algorithm successfully resolves all of them. A few of our inputs contain complex occlusions, where more than two body parts occlude each other in the same 2D location (Fig. 9, boy's right arm is behind the body that is behind his left arm, snowman in Fig. 11). Our method successfully handles these challenging situations. For all the examples, our method convincingly creates smooth animations (see the supplementary video), consistent with the drawn keyframes and the 3D skeletal animation.



We generate the skeletons directly from the bitmap keyframes using Sketch2Pose [Brodt and Bessmeltsev 2022] with their default parameters. Often the skeletons their method outputs need minor adjustments (see inset), like changing the position of a few joints or switching left to right. These adjustments normally take 10-20 seconds on average per keyframe. Skeletal motion between those keyframes can be either generated automatically or animated using standard animation software, such as Autodesk Maya.

We run our algorithm with the same set of parameters on all inputs. The least robust part of our algorithm is the occlusion masks prediction network, which may be inaccurate for some drawings. We use the automatic prediction as-is for almost all the results; we manually adjusted the predicted masks for the girl from [Dvorožňák et al. 2018] (Fig. 16), the frog (Fig. 10) and Santa (Fig. 11). Each adjustment took 15-30s on average (see Supplementary). In total, after a simple training, a non-professional can do all the annotations for a pair of input images, i.e., adjust the Sketch2Pose skeleton, add T-junctions, and possibly adjust the masks in less than 90 seconds on average.

We validate the key aspects of our method in a number of ways. The questionnaires used in the evaluations, detailed results, and complete comparisons are included in our supplementary.

Ground Truth comparisons. We have compared our algorithmic results with ground truth, manual inbetweening results made by professional animators. As ground truth, we took a classical animation feature film, Aladdin and His Wonderful Lamp [Studios 1939], one of rare examples of manually inbetweened 2D animations in public domain. Since there is no way of knowing which frames were keys in the original animation, we chose seven sets of three consecutive frames and created skeletal motions for each triplet. We then use the first and the third frames and generate the second one ($t = 0.5$) with our system. We show comparison results on a single triplet of frames in Fig. 17. We note that even though the ground truth has somewhat sharper lines, our result is an equally plausible inbetweening. We also observe that professional inbetweeners added some motion blur into the animation (Fig. 17, bottom example, $t = 0.5$, fists). While by default our system prefers a sharp inbetweening, one can use the velocities of the template mesh vertices to generate such motion blur if desired. Those can be considered examples of tight inbetweening with our system. The frame interpolation methods (Fig. 17, bottom) are unable to faithfully inbetween these frames, resulting in either blurred or anatomically incorrect in-betweens. Note that this is precisely the scenario for which those systems are designed; the only difference is that videos they are trained on often have higher frame rate. Our deformable template helps preserve anatomical correctness, and our novel blending optimization helps minimize the blur due to frame interpolation.

We manually removed the background on each input image in the ground truth animation. Being only trained on paper drawings with white or grey background, our foreground segmentation network (Sec. 4.1) is *not* designed to remove finished cartoon backgrounds. This is intentional, since in traditional animation character animations are typically drawn separately from the background and overlaid over backgrounds at the end.

We have additionally performed a quantitative evaluation of our algorithmic results, comparing it with the frame interpolation methods on the standard metrics, such as PSNR (Peak signal-to-noise ratio), SSIM (Structural similarity), LPIPS (Learned Perceptual Image Patch Similarity) [Zhang et al. 2018] based on deep learning image classification features, and CD (Chamfer Distance) [Chen

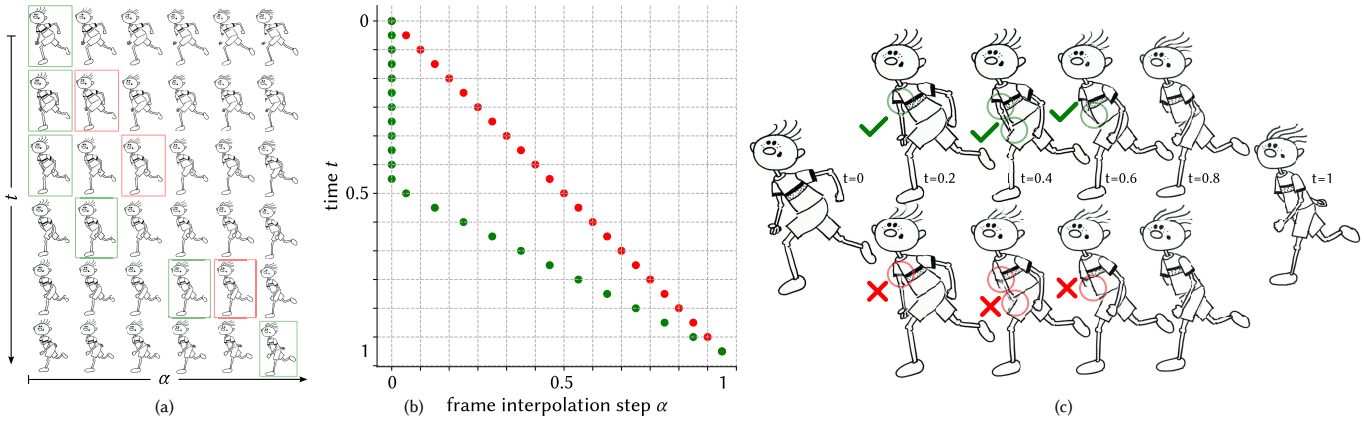


Fig. 9. Starting with the deformation sequences of both keyframes (corresponding to leftmost and rightmost columns in (a)), we use a pre-trained model to create interpolation frames at every time step, yielding an image matrix. We then define image quality and interpolation quality metrics and find an optimal path connecting the first keyframes (top-left image) with the second keyframe (bottom-right) ((b), green path and (c), top). Note that a naïve strategy of picking the diagonal (b, red) often leads to ghosting artifacts (c, bottom).

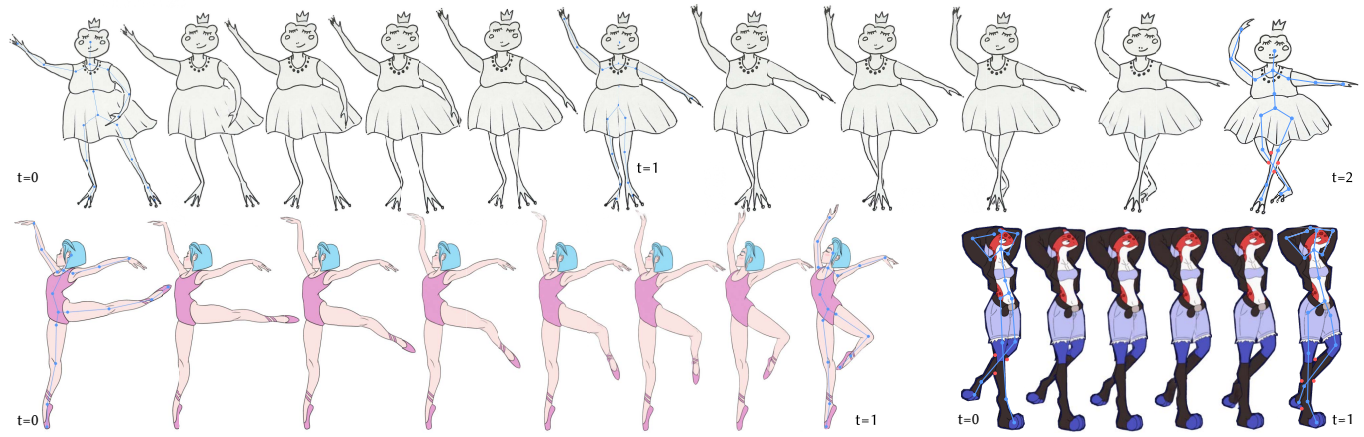


Fig. 10. Results gallery. The images marked as $t = 0, 1, 2$ are input. Lizard lady © Jared Dillon.

and Zwicker 2022]. The results, averaged over the seven triplets of frames, are presented in Table 1; our method performs the best on all these metrics. We note, however, that none of these metrics, except for, perhaps LPIPS, are based on human perception, and therefore are not reliable indicators of inbetweening quality.

Comparison to Vector Inbetweening. We additionally compare our inbetweening with a state-of-the-art vector inbetweening methods of Siyao et al. [2023] (Fig. 14) in a few scenarios. For comparison with [Siyao et al. 2023], we vectorized the input images (marked as $t = 0, t = 1$) with a line drawing vectorization method [Bessmeltsev and Solomon 2019] using their default parameters (Fig. 14, left and right in the second row) and ran the inbetweening system of [Siyao et al. 2023]. We can see that the vector-based method [Siyao et al. 2023] struggles to in-between the two drawings (second row), perhaps due to their drastically different topology and number of strokes. To exclude the influence of the artifacts in the automatic vectorization, we then vectorized the input images manually by tracing over the image with pixel accuracy and correct topology and repeated this experiment (third row). This results shows that even on manual vectorizations, their system is not robust enough, perhaps because it targets very tight inbetweening only, or perhaps because it was trained on synthetic data and does not generalize well to natural drawings. We ran our system on the raster drawings directly (top row). The input drawings contain no occlusions, so we did not specify any T-junctions. Our method does not depend on the drawing topology and successfully inbetweens these two drawings.

Comparison to Raster Inbetweening. In Figure 13 we compare our results to some of the modern video frame interpolation systems, including AnimeInterp [Siyao et al. 2021], FILM [Reda et al. 2022], RIFE [Huang et al. 2022], SAFA [Huang et al. 2024], and EISAI [Chen and Zwicker 2022]. We run all the methods with default parameters. We could not run SAFA in the original resolution, since it requires

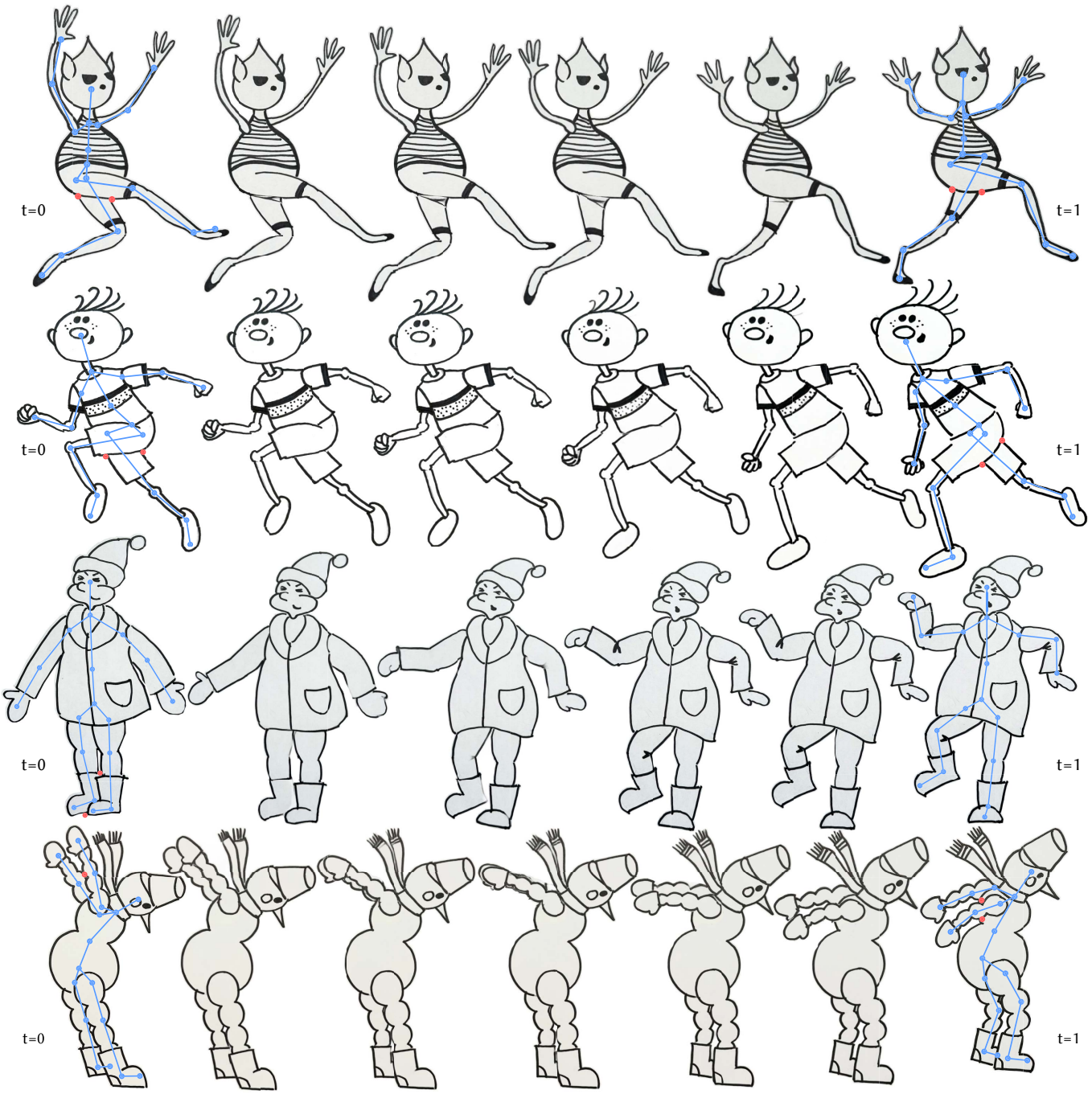


Fig. 11. Additional results gallery. The images marked as $t = 0, 1$ are input.

more GPU RAM that we had, so for that method we downscaled the input images to 512×512px. As the figure demonstrates, together with Fig. 3, frame interpolation methods, trained on high FPS videos, are not designed to perform inbetweening for distant keyframes. Often the interpolations they produce contain blur, ghost images, and other artifacts. In contrast, our method (top) produces sharp inbetweens even for such complex cases with occlusions. We include more comparison results in the supplementary materials.

We compare our method with the inbetweening method of [Mo et al. 2024] (Fig. 15). Their method targets inbetweening of clean bitmap line drawings, requiring a high-quality input vectorization

Fig. 12. The optimized template can be manipulated by the user and textures are interpolated automatically. On the first row the user modifies the frame at $t = 0$. On the second row the user modifies the intermediate frame at time $t > 0$ (see the supplementary video).

as additional input. They do not provide code, so we can only run their inputs in our system. As the figure demonstrates, our method, despite not requiring any vector input, is capable of producing an inbetweening of comparable quality. Note that a simple example on the bottom shows that our method is also easily generalizable to non-humanoid characters; only the mask prediction networks need to be retrained for such case, the rest of the pipeline remains unchanged.

We additionally compare our method with [Dvorožňák et al. 2018; Fan et al. 2018; Hinz et al. 2022] (Fig. 16). Our results have quality comparable with ToonSynth [Dvorožňák et al. 2018]. As discussed in Sec. 2, Dvorožňák et al. [2018] requires a whole sequence of images and a full and precise manual puppet construction [Hinz et al. 2022]. In contrast, we only require two keyframes with a few junction points — the only annotation, apart from the skeleton. Furthermore, their results are limited to an animation of a puppet with the fixed topology. In contrast, our method is capable of interpolating images with vastly different topologies and geometries (e.g. Fig. 9). We cannot run their method on our inputs (no code available).

In Fig. 16, we also compare with the method of Hinz et al. [2022] which we run with their pretrained models for the 'red-haired boy' and the 'lady in the pink top' (250x250px) and train our model for Jafar (720x720px) on 11 frames with default parameters. Their method does not interpolate the given keyframes, they can only generate images that are similar to the keyframes, which is incompatible with a typical animation pipeline (notice the distorted details on keyframes). Their results on the images we tried replicate the overall style of the drawings, but are often anatomically incorrect and blurry.

Finally, we compare our method to the one of [Fan et al. 2018]. They register a manually annotated layered puppet to new frames, and hence use a fixed texture; do not target inbetweening. We run our method on one of their results, demonstrating comparable quality (Fig. 16, bottom right).

Qualitative Comparison. We validate the quality of our results by comparing them to the alternative via a comparative perceptual study. As the alternative methods we chose Dvorožňák et al. [2018], Mo et al. [2024], and Huang et al. [2022]. We chose the latter because, despite not being formally the state of the art in the frame

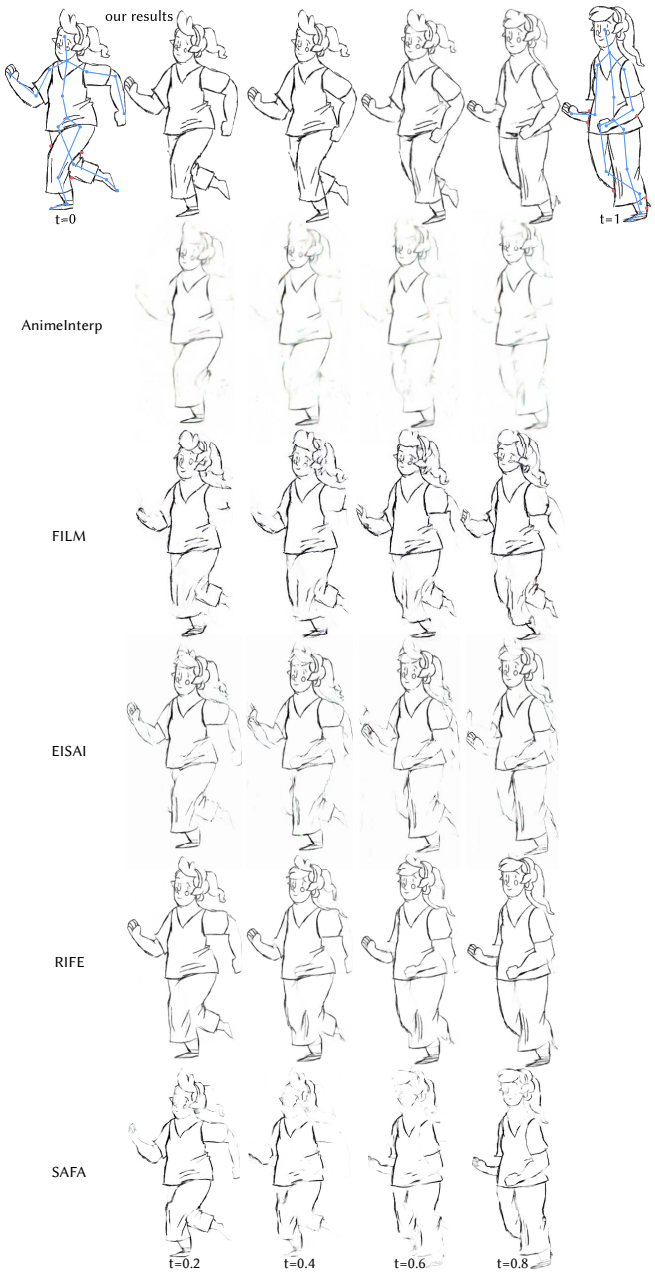


Fig. 13. Comparisons of our inbetweening results (top) with video frame interpolation methods: AnimeInterp [Siyao et al. 2021], FILM [Reda et al. 2022], RIFE [Huang et al. 2022], SAFA [Huang et al. 2024], and EISAI [Chen and Zwicker 2022].

interpolation, it visually performed the best on our inputs among its competitors. Study participants were shown input sketches, together with our algorithmic in-betwening result image and an alternative frame interpolation method result for the same time $t = 0.5$. Since neither [Mo et al. 2024] nor the frame interpolation methods do not offer artistic control via a skeleton, we did not show the desired

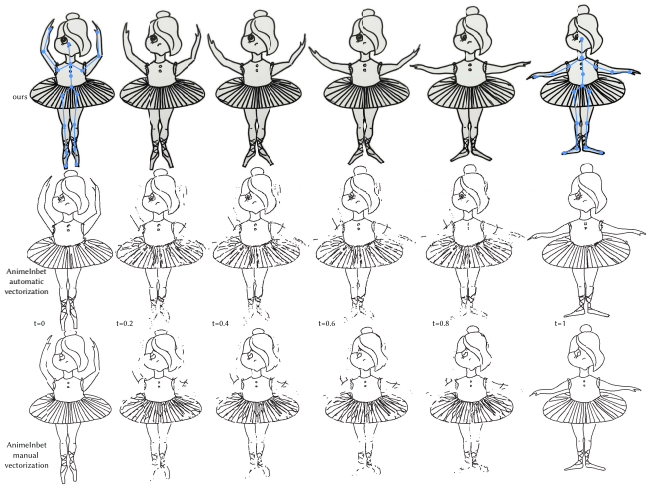


Fig. 14. Vector-based inbetweening systems, such as AnimeInbet [Siyao et al. 2023] (middle and bottom) often struggle to perform inbetweening for two vectorized drawings, due to significant topological changes. Whether the drawing is carefully vectorized by an artist (bottom) or automatically via a standard algorithm (middle) has little effect on the final result. Our system (top) works directly with raster drawings and has no constraints on topology of input drawings (top).

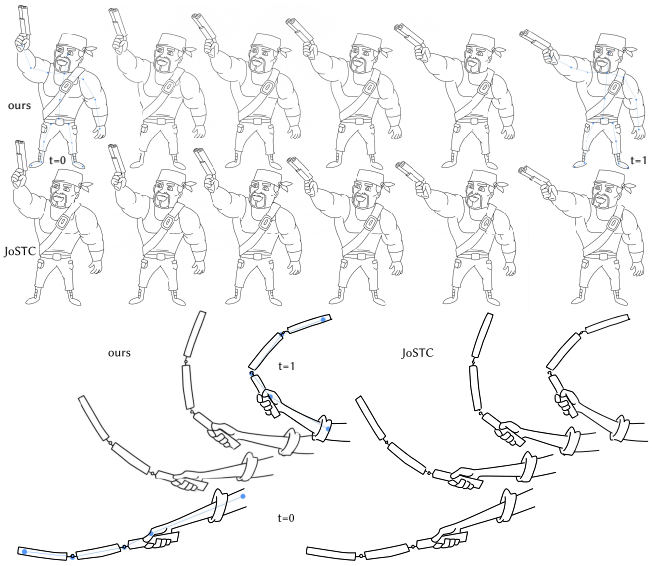


Fig. 15. Compared to [Mo et al. 2024], our method can produce similar results. Their method, however, requires a clean vectorization as an input, which is hard or impossible to get for more complex images with texture; we directly work with raster images.

skeletal pose to the participants, to avoid bias. The two input images were shown on the top, marked as A and B, and the two inbetweening results were placed at the bottom in random order and marked as "C" and "D". Participants were then asked "Which of the images below, C or D, more accurately captures a frame of an animation between A and B? If both are equally acceptable, choose 'Both'. If neither, select 'Neither'". We included 20 questions. We collected answers for each query from 17 different participants, including 10 males and 7 females, age ranging from 23 to 50 years; 3 were artists. The full layout, as well as the study data, is presented in the supplementary.

Fig. 18 summarizes the results. Participants preferred our results over the one of competitor methods 71.5% of the time, ranked our methods on par 25.7% of the time, and preferred the alternative only 0.6% of the time. This study convincingly demonstrates that the inbetweening results we produce are more consistent with viewer expectation than the ones produced by previous approaches.

Qualitative Evaluation. We asked 3 artists and 14 non-professionals to comment on the results of our algorithm. We showed them each pair of input and our algorithmic result and asked to comment on the following statement, separately for each pair, "This in-between frame captures the artist intended animation frame for the given pose.", with 5 Likert-type reply options: "Strongly disagree" (-2), "Disagree" (-1), "Neither disagree nor agree" (0), "Agree" (1), "Strongly Agree" (2). On average, the respondents agreed with the statement ($avg = 1.55, std = 0.2$). The layout of the study and the results are presented in the Supplementary.

Interactive Manipulation. Once the template is computed, a user can interactively manipulate the template by changing the given skeletal animation and seeing the results in real time (Fig. 12 and the accompanying video). For efficient visualization purposes, we use the diagonal texture interpolation strategy (Sec. 5.2).

Performance. We have implemented our system in a combination of C++, Python with PyTorch and OpenGL/GLSL, and measured the performance on our desktop machine (single Intel® Core™ i7-9700K CPU @ 3.60GHz with NVIDIA® GeForce® RTX 2080Ti). On average, full inbetweening of a pair of bitmap images takes roughly half a minute. The performance depends on the number of occlusion regions, the number of specified T-junctions, and the number of edges in each region. The complete performance statistics are presented in Table 2. The optimization is done on a single CPU core, only mask prediction and frame interpolation require a GPU.

	PSNR (↑)	SSIM (↑)	LPIPS (↓)	CD (↓)
AnimeInterp	17.076	0.808	18.735	32.710
FILM	16.781	0.796	17.423	32.689
EISAI	16.843	0.797	16.900	32.866
RIFE	17.087	0.803	16.793	32.492
SAFA	16.108	0.786	20.252	32.706
ours	**18.594**	**0.823**	**13.809**	**23.611**

Table 1. Quantitative evaluation of our method and a few video frame interpolation methods, compared with ground truth in-betweens, on a range of standard quality metrics. All the scores are averaged over 7 chosen frames in Alladin and his wonderful lamp. PSNR is Peak Signal-To-Noise Ratio, SSIM is structural similarity, LPIPS [Zhang et al. 2018] measures perceived image quality, CD is Chamfer distance between dark pixels [Chen and Zwicker 2022]. LPIPS is scaled by 10^2, CD by 10^5.

Fig. 16. Compared to ToonSynth [Dvorožňák et al. 2018], our results have comparable quality; in contrast to their system that requires often dozens of images and fully manual puppet construction, we only need two and minimal annotation; the template is optimized. CharacterGAN [Hinz et al. 2022] does not perform inbetweening in the sense of interpolating the input images, they synthesize *similar* images, which may not be acceptable. Overall, their results often have inferior quality. ToonCap [Fan et al. 2018] does not perform inbetweening; our method can produce similar results on their input data.

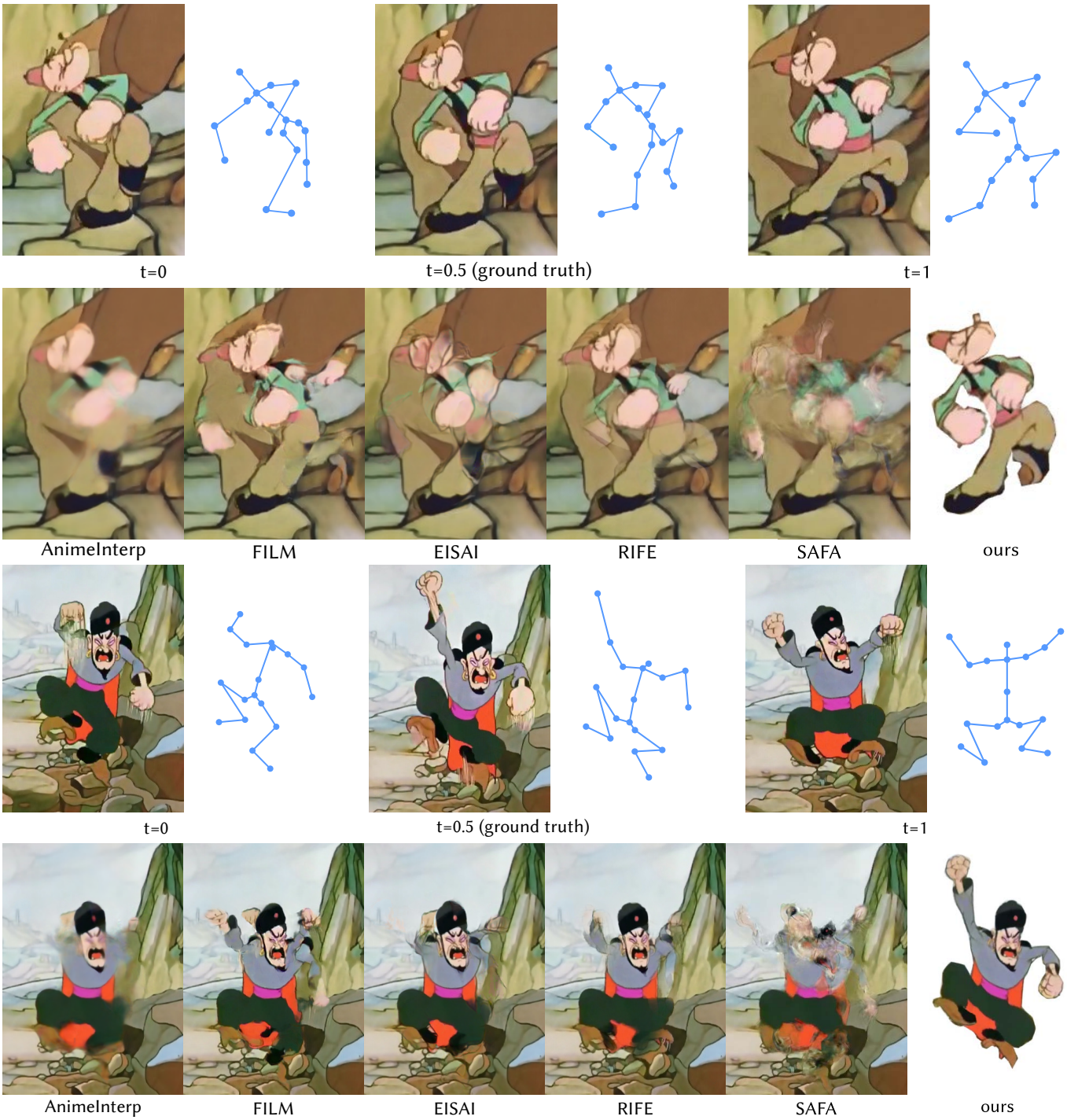


Fig. 17. A comparison of our method and video frame interpolation methods with the ground truth inbetweening, made by professional animators. While professional animators produce sharper frames, our results are visually plausible and consistently better than the state of the art.

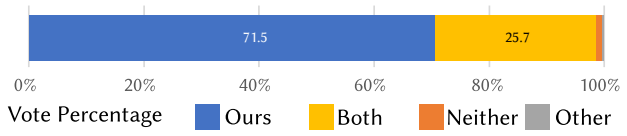


Fig. 18. Comparative preferences in our perceptual study. Participants consistently preferred our results over the one of competitor methods.

	occ	odd	cut	$\|V\|$	$\|F\|$	time (s)
Running girl	1	1	20	382	543	
	3	2	57	580	863	31
Lizard lady	1	1	14	296	417	
	1	1	24	328	470	18
Onion person	3	1	35	778	1129	
	1	0	0	374	504	62
Running boy	1	0	0	463	631	
	1	0	0	481	646	41
Santa	1	0	0	426	598	
	0	0	0	281	368	35
Frog	1	0	0	427	568	
	0	0	0	291	364	33
Ballerina in a tutu	0	0	0	227	300	
	0	0	0	232	308	10
Snowman	1	1	36	412	591	
	1	0	0	470	666	29

Table 2. Performance statistics for the input images. Columns: the number of occlusion masks ('occ'), number of segments with different end types ('odd'), number of edges for such contours ('cut'), mesh vertices ($|V|$), mesh faces ($|F|$), total time. For each character, we include only the first pair of input images in the statistics. The time includes our optimization and deforming the meshes. Frame interpolation and rendering is real time for image resolution 500 × 500 px. (Sec. 5.2).

Limitations. Even though many of the motions we demonstrate are inherently non-planar and include some 3D rotation (e.g., the pirate in Fig. 15, Jafar in Fig. 17), in general we rely on frame interpolation backbone to deal with out-of-plane motion, which might lead to some unnatural inbetweening (the face in Fig. 19).

For optimal quality we assume that each body part that is visible during the animation, is drawn on at least one keyframe; otherwise, it will be inpainted, sometimes leading to suboptimal results. This can be resolved by converting one of the inbetween frames into a keyframe and editing that part of the drawing.

The quality of our inbetweens is also limited by the frame interpolation network and the inpainting method we use; progress in these areas will improve our final results.

7 CONCLUSIONS AND FUTURE WORK

We have presented and validated a novel method for bitmap character inbetweening, offering artistic control via a 3D skeleton animation. Our system combines modern deep learning components

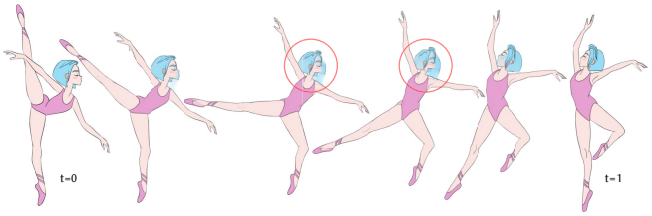


Fig. 19. For a strong out-of-plane rotation, our system relies on frame interpolation backbone only, which might lead to unnatural inbetweening.

with an optimization inferring the topology of the deformable template. Our method successfully performs inbetweening for distant keyframes, with or without occlusions, with vastly different geometry and topology — a task of the complexity mostly inaccessible to previous methods.

Our work raises many questions for future research. An interesting exploration would be supporting strongly out-of-plane motions with significant perspective foreshortening, which might require a fully 3D deformable template. Another direction would be to generalize our method to supporting characters with props or multiple characters in the same frame.

ACKNOWLEDGMENTS

We would like to thank Karl-Étienne Bolduc for his help with the figures. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No.: RGPIN-2019-05097 ("Creating Virtual Shapes via Intuitive Input"), RGPIN-2024-04968 ("Modelling and animation via intuitive input"), and the NSERC - Fonds de recherche du Québec - Nature et technologies (FRQNT) NOVA Grant No. 314090.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Yunfei Bai, Danny M. Kaufman, C. Karen Liu, and Jovan Popović. 2016. Artist-directed dynamics for 2D animation. *ACM Trans. Graph.* 35, 4, Article 145 (jul 2016), 10 pages. https://doi.org/10.1145/2897824.2925884

Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. 2019. Depth-Aware Video Frame Interpolation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

N. Ben-Zvi, J. Bento, M. Mahler, J. Hodgins, and A. Shamir. 2016. Line-Drawing Video Stylization. *Computer Graphics Forum* 35, 6 (2016), 18–32. https://doi.org/10.1111/cgf.12729 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12729

Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. 2015. Modeling Character Canvases from Cartoon Drawings. *Transactions on Graphics (2015)* 34, 5 (2015). https://doi.org/10.1145/2801134

Mikhail Bessmeltsev and Justin Solomon. 2019. Vectorization of Line Drawings via Polyvector Fields. *ACM Trans. Graph.* 38, 1, Article 9 (Jan. 2019), 12 pages.

Kirill Brodt and Mikhail Bessmeltsev. 2022. Sketch2Pose: Estimating a 3D Character Pose from a Bitmap Sketch. *ACM Transactions on Graphics* 41, 4 (7 2022). https://doi.org/10.1145/3528223.3530106

N. Burtnyk and M. Wein. 1976. Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation. *Commun. ACM* 19, 10 (oct 1976), 564–569. https://doi.org/10.1145/360349.360357

Leonardo Carvalho, Ricardo Marroquim, and Emilio Vital Brazil. 2017. DiLight: Digital light table – Inbetweening for 2D animations using guidelines. *Computers & Graphics* 65 (2017), 31–44. https://doi.org/10.1016/j.cag.2017.04.001

Edwin Catmull. 1978. The Problems of Computer-Assisted Animation. *SIGGRAPH Comput. Graph.* 12, 3 (aug 1978), 348–353. https://doi.org/10.1145/965139.807414

Jiawen Chen, Sylvain Paris, Jue Wang, Wojciech Matusik, Michael Cohen, and Frédo Durand. 2011. The video mesh: A data structure for image-based three-dimensional video editing. In *2011 IEEE International Conference on Computational Photography (ICCP)*. 1–8. https://doi.org/10.1109/ICCPHOT.2011.5753118

J. Chen, X. Zhu, M. Even, J. Basset, P. Bénard, and P. Barla. 2023. Efficient Interpolation of Rough Line Drawings. *Computer Graphics Forum* 42, 7 (2023), e14946. https://doi.org/10.1111/cgf.14946 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14946

Shuhong Chen and Matthias Zwicker. 2022. Improving the Perceptual Quality of 2D Animation Interpolation. In *Proceedings of the European Conference on Computer Vision*.

Boris Dalstein, Rémi Ronfard, and Michiel van de Panne. 2015. Vector Graphics Animation with Time-Varying Topology. *ACM Trans. Graph.* 34, 4 (July 2015).

Marek Dvorožňák, Wilmot Li, Vladimir G. Kim, and Daniel Sýkora. 2018. ToonSynth: Example-Based Synthesis of Hand-Colored Cartoon Animations. *ACM Transactions on Graphics* 37, 4, Article 167 (2018).

Marek Dvorožňák, Daniel Sýkora, Cassidy Curtis, Brian Curless, Olga Sorkine-Hornung, and David Salesin. 2020. Monster Mash: A Single-View Approach to Casual 3D Modeling and Animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2020)* 39, 6 (2020), 214.

Melvin Even, Pierre Bénard, and Pascal Barla. 2023. Non-linear Rough 2D Animation using Transient Embeddings. *Computer Graphics Forum* 42, 2 (2023), 411–425. https://doi.org/10.1111/cgf.14771 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14771

Xinyi Fan, Amit H Bermano, Vladimir G Kim, Jovan Popović, and Szymon Rusinkiewicz. 2018. Tooncap: A layered deformable model for capturing poses from cartoon characters. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*. 1–12.

Tsukasa Fukusato and Akinobu Maejima. 2022. View-Dependent Deformation for 2.5-D Cartoon Models. *IEEE Computer Graphics and Applications* 42, 5 (2022), 66–75. https://doi.org/10.1109/MCG.2022.3174202 Publisher Copyright: © 1981-2012 IEEE..

Olga Guțan, Shreya Hegde, Erick Jimenez Berumen, Mikhail Bessmeltsev, and Edward Chien. 2023. Singularity-Free Frame Fields for Line Drawing Vectorization. *Computer Graphics Forum* 42, 5 (2023), e14901. https://doi.org/10.1111/cgf.14901 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14901

Tobias Hinz, Matthew Fisher, Oliver Wang, Eli Shechtman, and Stefan Wermter. 2022. CharacterGAN: Few-Shot Keypoint Character Animation and Reposing. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 1988–1997.

Berthold K.P. Horn and Brian G. Schunck. 1981. Determining Optical Flow. *Artificial Intelligence* 17, 1 (1981), 185–203. https://doi.org/10.1016/0004-3702(81)90024-2

Zhewei Huang, Ailin Huang, Xiaotao Hu, Chen Hu, Jun Xu, and Shuchang Zhou. 2024. Scale-Adaptive Feature Aggregation for Efficient Space-Time Video Super-Resolution. In *Winter Conference on Applications of Computer Vision (WACV)*.

Zhewei Huang, Tianyuan Zhang, Wen Heng, Boxin Shi, and Shuchang Zhou. 2022. Real-Time Intermediate Flow Estimation for Video Frame Interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (jul 2011), 8 pages. https://doi.org/10.1145/2010324.1964973

Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. 2012. Three-dimensional proxies for hand-drawn characters. *ACM Trans. Graph.* 31, 1, Article 8 (Feb. 2012), 16 pages.

Jie Jiang, Hock Soon Seah, and Hong Ze Liew. 2022. Stroke-Based Drawing and Inbetweening with Boundary Strokes. *Computer Graphics Forum* 41, 1 (2022), 257–269. https://doi.org/10.1111/cgf.14433 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14433

Tarun Kalluri, Deepak Pathak, Manmohan Chandraker, and Du Tran. 2023. FLAVR: Flow-Agnostic Video Representations for Fast Frame Interpolation. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2070–2081. https://doi.org/10.1109/WACV56688.2023.00211

Olga A. Karpenko and John F. Hughes. 2006. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph.* 25, 3 (jul 2006), 589–598. https://doi.org/10.1145/1141911.1141928

Alan Kitching. 1977. Antics—Graphic animation by computer. *Computers & Graphics* 2, 4 (1977), 219–223. https://doi.org/10.1016/0097-8493(77)90018-8

Jan J. Koenderink and Andrea J. van Doorn. 1982. The Shape of Smooth Objects and the Way Contours End. *Perception* 11, 2 (1982), 129–137. https://doi.org/10.1068/p110129 arXiv:https://doi.org/10.1068/p110129 PMID: 7155766.

Alexander Kort. 2002. Computer Aided Inbetweening. In *Proceedings of the 2nd International Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) *(NPAR '02)*. Association for Computing Machinery, New York, NY, USA, 125–132. https://doi.org/10.1145/508530.508552

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Trans. Graph.* 35, 4, Article 134 (jul 2016), 11 pages. https://doi.org/10.1145/2897824.2925920

Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. 2018. OptCuts: Joint Optimization of Surface Cuts and Parameterization. *ACM Transactions on Graphics* 37, 6 (2018). https://doi.org/10.1145/3272127.3275042

Xiaoyu Li, Bo Zhang, Jing Liao, and Pedro V. Sander. 2022. Deep Sketch-Guided Cartoon Video Inbetweening. *IEEE Transactions on Visualization and Computer Graphics* 28, 8 (aug 2022), 2938–2952. https://doi.org/10.1109/TVCG.2021.3049419

Dongquan Liu, Quan Chen, Jun Yu, Huiqin Gu, Dacheng Tao, and Hock Soon Seah. 2011. Stroke Correspondence Construction Using Manifold Learning. *Computer Graphics Forum* 30, 8 (2011), 2194–2207. https://doi.org/10.1111/j.1467-8659.2011.01969.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01969.x

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34, 6 (Oct. 2015), 248:1–248:16. https://doi.org/10.1145/2816795.2818013

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7

James McCann and Nancy Pollard. 2009. Local Layering. In *ACM SIGGRAPH 2009 Papers* (New Orleans, Louisiana) *(SIGGRAPH '09)*. Association for Computing Machinery, New York, NY, USA, Article 84, 7 pages. https://doi.org/10.1145/1576246.1531390

R. Miyauchi, T. Fukusato, H. Xie, and K. Miyata. 2021a. Stroke Correspondence by Labeling Closed Areas. In *2021 Nicograph International (NicoInt)*. IEEE Computer Society, Los Alamitos, CA, USA, 34–41. https://doi.org/10.1109/NICOINT52941.2021.00014

Ryoma Miyauchi, Yichen Peng, Tsukasa Fukusato, and Haoran Xie. 2021b. Skeleton2Stroke: Interactive Stroke Correspondence Editing with Pose Features. In *SIGGRAPH Asia 2021 Technical Communications* (Tokyo, Japan) *(SA '21)*. Association for Computing Machinery, New York, NY, USA, Article 6, 4 pages. https://doi.org/10.1145/3478512.3488612

Haoran Mo, Chengying Gao, and Ruomei Wang. 2024. Joint Stroke Tracing and Correspondence for 2D Animation. *ACM Transactions on Graphics* (2024).

Mariia Myronova, William Neveu, and Mikhail Bessmeltsev. 2023. Differential Operators on Sketches via Alpha Contours. *ACM Trans. Graph.* 42, 4, Article 69 (jul 2023), 15 pages. https://doi.org/10.1145/3592420

Simon Niklaus and Feng Liu. 2020. Softmax Splatting for Video Frame Interpolation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Jorge Nocedal and Stephen J. Wright (Eds.). 1999. *Numerical Optimization*. Springer-Verlag, New York. https://doi.org/10.1007/b98874

G. Noris, D. Sýkora, S. Coros, B. Whited, M. Simmons, A. Hornung, M. Gross, and R. W. Sumner. 2011. Temporal Noise Control for Sketchy Animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering* (Vancouver, British Columbia, Canada) *(NPAR '11)*. Association for Computing Machinery, New York, NY, USA, 93–98. https://doi.org/10.1145/2024676.2024691

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. 2024. DINOv2: Learning Robust Visual Features without Supervision. *Transactions on Machine Learning Research* (2024). https://openreview.net/forum?id=a68SUt6zFt

Ulrich Pinkall and Konrad Polthier. 1993. Computing Discrete Minimal Surfaces and Their Conjugates. *Experimental Mathematics* 2, 1 (1993), 15–36. https://doi.org/10.1080/10586458.1993.10504266 arXiv:https://doi.org/10.1080/10586458.1993.10504266

Omid Poursaeed, Vladimir Kim, Eli Shechtman, Jun Saito, and Serge Belongie. 2020. Neural puppet: Generative layered cartoon characters. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3346–3356.

Fitsum Reda, Janne Kontkanen, Eric Tabellion, Deqing Sun, Caroline Pantofaru, and Brian Curless. 2022. FILM: Frame Interpolation for Large Motion. In *European Conference on Computer Vision (ECCV)*.

William T. Reeves. 1981. Inbetweening for Computer Animation Utilizing Moving Point Constraints. *SIGGRAPH Comput. Graph.* 15, 3 (aug 1981), 263–269. https://doi.org/10.1145/965161.806814

Alec Rivers, Takeo Igarashi, and Frédo Durand. 2010. 2.5D Cartoon Models. *ACM Trans. Graph.* 29, 4, Article 59 (jul 2010), 7 pages. https://doi.org/10.1145/1778765.1778796

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. *CoRR* abs/1505.04597 (2015). http://dblp.uni-trier.de/db/journals/corr/corr1505.html#RonnebergerFB15

Jonathan Richard Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Selected Papers from the Workshop on Applied Computational Geometry, Towards Geometric Engineering (FCRC '96/WACG '96)*. Springer-Verlag, Berlin, Heidelberg, 203–222.

Li Siyao, Tianpei Gu, Weiye Xiao, Henghui Ding, Ziwei Liu, and Chen Change Loy. 2023. Deep Geometrized Cartoon Line Inbetweening. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7291–7300.

Li Siyao, Shiyu Zhao, Weijiang Yu, Wenxiu Sun, Dimitris Metaxas, Chen Change Loy, and Ziwei Liu. 2021. Deep Animation Video Interpolation in the Wild. In *CVPR*.

J Smith and S Schaefer. 2015. Bijective Parameterization with Free Boundaries. *Acm Transactions on Graphics* 34, 4 (2015), 9. https://doi.org/10.1145/2766947

Olga Sorkine and Marc Alexa. 2007. As-Rigid-As-Possible Surface Modeling. *Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), 109–116. https://doi.org/10.1007/s11390-011-1154-3 ISBN: 9783905673463.

Paul Starke, Sebastian Starke, Taku Komura, and Frank Steinicke. 2023. Motion In-Betweening with Phase Manifolds. *Proc. ACM Comput. Graph. Interact. Tech.* 6, 3, Article 37 (aug 2023), 17 pages. https://doi.org/10.1145/3606921

Fleischer Studios. 1939. Aladdin and His Wonderful Lamp. https://archive.org/details/popeye-meets-aladdin.

Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. 2021. Resolution-robust Large Mask Inpainting with Fourier Convolutions. *arXiv preprint arXiv:2109.07161* (2021).

Daniel Sýkora, John Dingliana, and Steven Collins. 2009. As-Rigid-as-Possible Image Registration for Hand-Drawn Cartoon Animations. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (New Orleans, Louisiana) *(NPAR '09)*. Association for Computing Machinery, New York, NY, USA, 25–33. https://doi.org/10.1145/1572614.1572619

Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. 2014. Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters. *ACM Transaction on Graphics* 33, 2 (2014), 16.

D. Sýkora, D. Sedlacek, S. Jinchao, J. Dingliana, and S. Collins. 2010. Adding Depth to Cartoons Using Sparse Depth (In)equalities. *Computer Graphics Forum* 29, 2 (2010), 615–623. https://doi.org/10.1111/j.1467-8659.2009.01631.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01631.x

Shuhei Watanabe. 2023. Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. arXiv:2304.11127 [cs.LG]

Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W. Sumner, Markus Gross, and Jarek Rossignac. 2010. BetweenIT: An Interactive Tool for Tight Inbetweening. *Computer Graphics Forum* 29, 2 (2010), 605–614. https://doi.org/10.1111/j.1467-8659.2009.01630.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01630.x

Richard Williams. 2001. *Animators Survival Kit: A Working Manual Of Methods Principles And Formulas For Computer* (7th printing edition ed.). Faber & Faber, London.

Wenwu Yang, Hock-Soon Seah, Quan Chen, Hong-Ze Liew, and Daniel Sýkora. 2018. FTP-SC: Fuzzy Topology Preserving Stroke Correspondence. *Computer Graphics Forum* 37, 8 (2018), 125–135.

Jun Yu, Wei Bian, Mingli Song, Jun Cheng, and Dacheng Tao. 2012. Graph based transductive learning for cartoon correspondence construction. *Neurocomputing* 79 (2012), 105–114. https://doi.org/10.1016/j.neucom.2011.10.003

Zhiyang Yu, Yu Zhang, Xujie Xiang, Dongqing Zou, Xijun Chen, and Jimmy S Ren. 2022. Deep Bayesian Video Frame Interpolation. In *European Conference on Computer Vision*. Springer, 144–160.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *CVPR*.

Haichao Zhu, Xueting Liu, Tien-Tsin Wong, and Pheng-Ann Heng. 2016. Globally Optimal Toon Tracking. *ACM Transactions on Graphics (SIGGRAPH 2016 issue)* 35, 4 (July 2016), 75:1–75:10.

Yufeng Zhu, Jovan Popović, Robert Bridson, and Danny M. Kaufman. 2017. Planar Interpolation with Extreme Deformation, Topology Change and Dynamics. *ACM Trans. Graph.* 36, 6, Article 213 (nov 2017), 15 pages. https://doi.org/10.1145/3130800.3130820