Wassersplines for Stylized Neural Animation

PAUL ZHANG, Massachusetts Institute of Technology, USA
DMITRIY SMIRNOV, Massachusetts Institute of Technology, USA
JUSTIN SOLOMON, Massachusetts Institute of Technology, USA

Fig. 1. We apply our method on 3D data to generate smooth animations between keyframes (shown in orange) of varied geometry and topology.

Much of computer-generated animation is created by manipulating meshes with rigs. While this approach works well for animating articulated objects like animals, it has limited flexibility for animating less structured creatures such as the Drunn in “Raya and the Last Dragon.” We introduce Wassersplines, a novel trajectory inference method for animating unstructured densities based on recent advances in continuous normalizing flows and optimal transport. The key idea is to train a neurally-parameterized velocity field that represents the motion between keyframes. Trajectories are then computed by pushing keyframes through the velocity field. We solve an additional Wasserstein barycenter interpolation problem to guarantee strict adherence to keyframes. Our tool can stylize trajectories through a variety of PDE-based regularizers to create different visual effects. We demonstrate our tool on various keyframe interpolation problems to produce temporally-coherent animations without meshing or rigging.

CCS Concepts: • Computing methodologies → Motion processing; Point-based models.

Additional Key Words and Phrases: animation, trajectory inference, neural ODE

1 INTRODUCTION

In hand-drawn animation, a primary artist is tasked with laying out keyframes. These frames define the rough motion of an animation and occupy a fraction of the usual 12 drawings per second. The remaining frames are filled in afterwards to create smooth motion in a process called inbetweening. In the transition from hand-drawn animation to computer-assisted animation, much of the inbetweening process became automated [Lasseter 1987]. After an artist lays out keyframes as mesh rig displacements, in-between frames can be produced automatically using splines and other interpolation machinery. Secondary effects like elastic oscillation or fluids can be added afterwards through physical simulation. This process is largely responsible for modern character animation and has had major success for articulated objects like humans and animals.

While methods for articulated animation via meshes and rigs are abundant, research on animation of objects such as the Drunn in Walt Disney Animation Studio’s “Raya and the Last Dragon” is far less common. These animations are characterized by abstract, amorphous boundaries and fluid-like motion. As with classical animation, keyframes are still provided by an artist to coarsely define the desired motion. Due to the lack of form between keyframes, however, we denote such animations as unstructured. Rig-based methods are insufficient in this case, since unstructured animations can tear and recombine. Simultaneous to their freedom of movement, their trajectories must accurately reach keyframes, so that an artist is able to convey the appropriate gestures in a scene.

Unstructured animation can be found in various media over the last several decades. In 2018, Supergiant’s “Hades” animated trailer depicts a hand drawn sequence of smoke assembling into the form of Zeus. In 1994, Nickelodeon’s “The Secret World of Alex Mack” depicts Alex dissolving into a fluid and reforming. In as early as 1991, “Terminator 2: Judgment Day” depicts visual effects of liquid metal transitioning into various geometries. The unstructured animation style even pre-dates computer animation and can be found in the stylized motions of Cruella’s cigarette smoke in Disney’s 1961 “One Hundred and One Dalmations.” That is to say, interest in this style...
is abundant, but methods are largely manual, highly specific to the scene, and undocumented.

In this paper, we present Wasserstsplines for unstructured animation. We encode keyframes as point clouds or probability measures, allowing us to capture arbitrary geometries without the limitations of a mesh or rig. Trajectories are encoded using a coordinate multilayer perceptron (MLP) to produce a velocity field in space-time. A rough animation is then produced by advecting points through the velocity field. We propose a Wasserstein barycenter interpolation step to guarantee keyframe adherence. Using partial differential equation– (PDE–) based regularizers on the coordinate MLP, we affect various stylizations of the animation without extra keyframes. We demonstrate our method on 2D and 3D examples, showing adherence to keyframes and flexibility in the interpolations.

2 RELATED WORK

Normalizing Flows. Normalizing flows map a prescribed initial density such as a Gaussian through a set of invertible functions to a target posterior distribution [Papamakarios et al. 2021; Rezende and Mohamed 2015]. Neural ordinary differential equations (ODE) take this concept to the limit by parameterizing the state derivative with a deep neural network. Continuous normalizing flows (CNF) integrate the neural ODE to produce the target posterior distribution [Chen et al. 2018b]. In this context, Grahwohl et al. [2018] use the Hutchinson’s trace estimator to compute posterior density values through a neural ODE.

Various strategies decrease training time for CNFs. Finlay et al. [2020]; Kelly et al. [2020] regularize the spatial variation of the state derivative and its higher-order time derivatives. Tancik et al. [2020] use random Fourier features (RFF) for faster training of high-frequency state derivatives. Hertz et al. [2021] sequentially unmask RFFs in order of increasing frequency, decreasing sensitivity to the initial RFF sampling. Poli et al. [2020] learn auxiliary networks for faster ODE integration, a costly step in CNFs.

Optimal Transport. Optimal transport models compute the cheapest map from a source distribution onto a target distribution via a linear program [Kantorovich 2006]. The cost of this map defines the Wasserstein distance between distributions; see [Peyré et al. 2019; Santambrogio 2015; Solomon 2018] for general discussion.

Adding entropic regularization to the optimal transport linear program yields an efficient and easily-implemented optimization technique known as Sinkhorn’s algorithm or matrix rebalancing [Cuturi 2013]. While cheaper to compute, entropically-regularized optimal transport biases the Wasserstein metric so that the distance from a distribution to itself is nonzero. This issue is fixed in the definition of the Sinkhorn divergence by adding a de-biasing term to entropic optimal transport [Genevay et al. 2018]. Efficient large-scale implementations of Sinkhorn divergence and other optimal transport routines are available through “GeomLoss” [Feydy et al. 2019] and “POT: Python Optimal Transport” [Flamary et al. 2021].

Dynamical optimal transport provides an alternative means of computing Wasserstein distances when the ground metric is quadratic in geodesic distance. Instead of computing a map between the source and target distributions, it computes a kinetic energy-minimizing velocity field that advects the source distribution into the target [Benamou and Brenier 2000]; recent algorithms accelerate solution of the relevant variational problems and explore alternative mesh-based and neural parameterizations [Lavenant 2021; Lavenant et al. 2018; Papadakis et al. 2014; Tong et al. 2020].

Image/Shape Registration. Registrations between images or shapes can be built from velocity field–induced diffeomorphisms. Hug et al. [2015] regularize the velocity field in dynamical optimal transport and prove existence of minimizers with a velocity gradient regularizer. Eisenberger et al. [2019] compute static volume-preserving, velocity fields for mesh registration. Feydy et al. [2017] use unbalanced OT to build diffeomorphic registrations in medical imaging.

Measure-valued Splines. Higher-order interpolations can be computed through an ordered sequence of distributions by minimizing acceleration instead of kinetic energy [Benamou et al. 2019; Chen et al. 2018a]. Benamou et al. [2019] compute solutions as distributions over cubic splines via a multi-marginal transport problem. Chewi et al. [2021] show that such solutions do not allow for deterministic trajectory inference and instead compute optimal transport plans between consecutive pairs of point clouds followed by classical spline interpolation.

Our problem also aims to interpolate an input sequence of distributions. A major difference, however, is that we parameterize the trajectory of our interpolation with a velocity field. This difference allows us to regularize based on spatial derivatives of the velocity rather than just time-derivatives like acceleration. Furthermore, our aim is not to globally minimize any particular time derivative but rather to provide a palette of effects to stylize a trajectory.

Trajectory Inference. Lavenant et al. [2021]; Schiebinger et al. [2019] use Waddington OT for inference of cellular dynamics by concatenating OT interpolations between consecutive keyframes. The approach used by Rice [2021] to animate Drunn is similar in that it also computes OT matchings between consecutive point clouds. A key difference, however, is that the point clouds used in animation of Drunn are automatically generated by sampling densities evolved via fluid simulation. This provides an abundance of data, mitigating artifacts of a piecewise-smooth trajectory. In the context of 2D stylized fluid simulation, Browning et al. [2014] interpolate keyframes by matching image patches between keyframes. To get more smooth trajectories, however, they allow deviation from the provided keyframes.

3 PRELIMINARIES

For completeness, we overview relevant developments in optimal transport and continuous normalizing flows. For detailed coverage, see [Genevy et al. 2018; Papamakarios et al. 2021; Peyré et al. 2019].

3.1 From Wasserstein Distance to Sinkhorn Divergence

Given probability measures $\mu$ and $\nu$ on $X \subset \mathbb{R}^n$, let $\Pi(\mu, \nu)$ denote the set of joint probability measures on $X \times X$ with marginals $\mu$ and $\nu$. The squared 2-Wasserstein distance between $\mu$ and $\nu$ is defined as

$$W_2^2(\mu, \nu) := \inf_{\pi \in \Pi(\mu, \nu)} \int_{X \times X} \|x - y\|^2 d\pi(x, y).$$

The coupling solving Equation 1 is the optimal transport plan $\pi$. 

Entropic regularization simplifies solving Equation 1. Entropically-regularized transport distance is defined via the convex program
\[
\text{OT}_e(\mu, \nu) := \inf_{\pi \in \Pi(\mu, \nu)} \int_{X \times X} ||x - y||^2 d\pi(x, y) + \epsilon KL(\pi || \mu \otimes \nu). \tag{2}
\]
Unlike Equation 1, \(\text{OT}_e(\mu, \nu)\) can be computed efficiently by Sinkhorn’s algorithm [Cuturi 2013]. This efficiency comes at the cost of entropic bias, i.e., \(\text{OT}_e(\mu, \mu) \neq 0\). The bias becomes especially problematic when one is interested in Wasserstein gradient flows to transform a source measure \(\nu\) into a target measure \(\mu\). One could implement the flow \(\dot{\nu} = -\nabla_v \text{OT}_e(\mu, \nu)\), but it would converge to a solution where \(\nu \neq \mu\) [Feydy et al. 2019]. To address this bias, Genevay et al. [2018] build the Sinkhorn divergence:
\[
S_e(\mu, \nu) := \text{OT}_e(\mu, \nu) - \frac{1}{2} \text{OT}_e(\mu, \mu) - \frac{1}{2} \text{OT}_e(\nu, \nu). \tag{3}
\]

3.2 Neural ODEs

Given an initial state \(z(t_0) \in \mathbb{R}^n\) and parameterized state derivative function \(f_0(z, t) \in \mathbb{R}^n\), one can solve the ODE \(\dot{z} = f_0(z, t)\) for \(z\) at time \(t_1\) as
\[
z(t_1) = z(t_0) + \int_{t_0}^{t_1} f_0(z(t), t) dt. \tag{4}
\]
When \(f_0(z, t)\) is parameterized by a deep neural network, it is referred to as a coordinate MLP; \(\dot{z} = f_0(z, t)\) is a neural ODE. Given a loss function \(L(z(t_1))\), the gradient \(\nabla_{\theta} L(z(t_1))\) is computed by the adjoint method using black box ODE solvers [Chen et al. 2018b].

Neural ODEs can build CNFs in the following way. Let \(v\) be a measure with simple parametric density on \(\mathbb{R}^n\), \(\mu\) be the measure of a target density, and \(\phi\) be a map from \(z(t_0)\) to \(z(t_1)\). Then \(\phi_{\mu, v}\) denotes the measure obtained by flowing \(v\) according to \(f_0\) from \(t_1\) to \(t_2\), i.e., the pushforward of \(v\) by \(\phi\). The CNF generating \(p\) is obtained by minimizing Kullback-Leibler divergence \(D_{\text{KL}}(p||\phi_{\mu, v})\):
\[
D_{\text{KL}}(p||q) = \int_{\mathbb{R}^n} \log \frac{dp}{dq} dp \tag{5}
\]
for measures \(p, q\) over \(\mathbb{R}^n\) [Chen et al. 2018b; Papamakarios et al. 2021].

4.4 Fitting Loss

CNF methods like [Chen et al. 2018b; Grathwohl et al. 2018; Tong et al. 2020] use KL divergence \(D_{\text{KL}}(X^i_t | X^i_j)\) as a fitting loss. For unstructured animation, however, a KL loss is unsuitable because animation keyframes often have compact support; in particular, pushforward measures like \(X^i_t\) are unlikely to overlap with their targets \(X^j_t\) early in training. Furthermore, computing KL divergence requires density access, which is expensive to estimate. These situations leave the KL divergence undefined or infinite [Arjovsky et al. 2017].

Instead, we use Sinkhorn divergence as a trajectory fitting loss:
\[
L^t_{\text{fit}} = S_e(X^t_i | X^t_j). \tag{8}
\]

A Sinkhorn divergence of 0 guarantees that \(X^t_i = X^t_j\). Unlike KL divergence, Sinkhorn divergence has no dependence on overlapping support between its measures. In addition, its gradient brings non-overlapping measures together. Finally, Sinkhorn divergence can be computed with only sample access from its input measures.

Our total trajectory fitting loss is then
\[
L_{\text{fit}} = \sqrt{\sum_{t=1}^{T-2} L^t_{\text{fit}} + L_{fit}^{1+T}}. \tag{9}
\]

\(L_{\text{fit}}\) is constructed through Equation 8 applied to consecutive pairs of keyframes. This choice is motivated by “teacher-forcing” in training recurrent neural networks (RNNs), where one inserts ground-truth data into the network to decrease training time [Williams and Zipser 1989]. We also use a square root in Equation 9 in preparation to balance our fitting loss against a regularization energy. Recall that
Within the space of trajectories that minimize the fitting loss, we can identify different types of motion with mathematical quantities when interpolating between a square and rectangle reduces the increase in area of the shapes along the trajectory (circled in red).

\[
\| \cdot \|_F \text{ denotes Frobenius norm. This can also be interpreted as a Killing vector field energy [Ben-Chen et al. 2010] or, from the perspective of continuum mechanics, } f_0 \text{ is the displacement gradient and } l_{\text{rig}} \text{ is the magnitude of the linear strain tensor. Movements generated by a velocity field where } l_{\text{rig}} = 0 \text{ will appear rigid and not squash or stretch keyframes.}
\]

Compressibility. A closely related concept to rigidity is compressibility. Consider a quivering block of gelatin. While its movements are non-rigid, squashing in height will cause bulging in width. Compressibility can be captured by divergence:

\[
l_{\text{div}} = \nabla \cdot f_0 = \text{Tr} (\nabla z f_0).
\]

Movements of a keyframe generated by a velocity field where \(l_{\text{div}} = 0\) will preserve area or volume, i.e., they are incompressible.

Swirliness. One of the most visually salient features of an animation or fluid is its swirliness. In fluid dynamics, this is called vorticity and is naturally captured by \(\text{curl}\): \(\nabla \times f_0\), a vector valued quantity. Its direction points in the axis of rotation, and its magnitude is twice the angular velocity. Given a target curl vector \(c\), we can quantify how close \(f_0\) is to meeting that target with

\[
l_{\text{curl}} = \| \nabla \times f_0 - c \|_2.
\]

For example, if we want a 2D animation where the keyframe rotates a full circle clockwise in four seconds, then we can measure \(l_{\text{curl}}\) with \(c = [0, 0, -\pi]^T\).

User-Directed Alignment. We can explicitly incentivize keyframes to move in certain directions during the animation via a user-provided metric \(A(z, t) \in \mathbb{R}^{d \times d}\) by minimizing

\[
l_A = \| f_0 \|_A = \sqrt{f_0^\top A f_0}.
\]

If the user wants \(f_0\) to align with unit vector field \(v\), they can choose the metric \(A = I - vv^\top\). Conversely, the user can also disincentivize alignment of \(f_0\) to \(v\) with \(A = vv^\top\).

Smoothness. Finally, we mention some regularizers from the neural ODE literature. ODE solvers can be slow if \(f_0\) varies too much spatially or temporally [Finlay et al. 2020; Kelly et al. 2020]. One can try to make \(f_0\) smoother with the following:

\[
l_{\text{grad}} = \| \nabla z f_0 \|_F
\]

\[
l_{\text{vel}} = \| f_0 \|_2, \quad l_{\text{acc}} = \| f_0 \|_2, \quad l_{\text{jerk}} = \| f_0 \|_2.
\]

\(l_{\text{grad}}\) quantifies spatial variation of \(f_0\), while \(l_{\text{vel}}, l_{\text{acc}},\) and \(l_{\text{jerk}}\) measure orders of temporal variation, i.e., speed, acceleration, and jerk.

\[ Fig. 3. \] Comparison between trajectories obtained using our method and optimal transport for interpolating between the Chinese character for "horse" and an image of a horse (rows 1 and 2) and between two images of horses in different poses (rows 3 and 4). The OT interpolation exhibits more spatial discontinuities (circled in red).

\[ Fig. 4. \] Effect of rigidity and compressibility regularization on our trajectories. Increasing the coefficient of the rigidity regularizer yields a less wobbly interpolation between two straight bars (top). Regularizing compressibility when interpolating between a square and rectangle reduces the increase in area of the shapes along the trajectory (bottom).

\[ L_{\text{fit}}^{i+1} \] is a squared Wasserstein distance with entropic and de-biasing modifiers and that gradients of squared quantities decrease with the magnitudes of their arguments. This square-root maintains the magnitude of the fitting loss gradient even as the fitting loss approaches 0, so that gradients are not out-competed by regularizers.

4.3 Controlling the Trajectory

Within the space of trajectories that minimize the fitting loss, we can encourage paths with desirable features by regularizing the velocity field \(f_0\). This process is straightforward, since motion types have natural vector calculus or continuum mechanics counterparts. Here, we identify different types of motion with mathematical quantities based on vector field \(f_0\) in a pointwise manner. We then show how to integrate these quantities in space-time to build a regularizing loss. These regularizers enable the user to select from a palette of options to control their animation.

Squash and Stretch. We begin with one of the basic principles of animation: “squash and stretch” [Thomas et al. 1995]. A mathematical measure for this type of movement is rigidity:

\[
l_{\text{rig}} = \| \nabla z f_0 + \nabla z f_0^\top \|_F,
\]

where \(\| \cdot \|_F\) denotes Frobenius norm. This can also be interpreted as a Killing vector field energy [Ben-Chen et al. 2010] or, from the perspective of continuum mechanics, \(f_0\) is the displacement gradient and \(l_{\text{rig}}\) is the magnitude of the linear strain tensor.
Integrating Along the Trajectory. We have defined various point-wise quantities based on $f_0$. We will denote all of them with $\mathcal{L}_2(z, t)$, where $\square$ can be replaced with any of the previously-mentioned quantities. To regularize $f_0$ with $\mathcal{L}_2(z, t)$ we use the loss

\[
\mathcal{L}_2 = \frac{1}{2} \sum_{i=1}^{T-2} \int_{t_i}^{t_{i+1}} \int_{\mathbb{R}^d} \mathcal{L}_2^2(z, t) d(X_t^i + X_{iz}) dt
\]

that integrates $\mathcal{L}_2$ over the trajectory. For each regularizer, we add $\mathcal{L}_2$ to our loss with coefficient $\lambda_2$. As with Equation 9, Equation 16 only considers trajectories obtained by flowing keyframes to their consecutive neighbors. Doing this avoids applying regularization unnecessarily at space-time locations that come from accumulated error in ODE integration. We show in §6 the isolated effects of these regularizers and how they can be used to stylize an animation.

4.4 Wasserstein Barycenter Interpolation

After training the neural ODE, we are not yet guaranteed that the trajectory strictly adheres to keyframes, as the neural ODE balances the fitting loss and regularizing losses. If $L_{\text{fit}} \neq 0$, the neural ODE trajectories deviate from keyframes. We correct deviation by applying the following Wasserstein barycenter interpolation step. Given a query time $t \in [t_i, t_{i+1}]$, our in-between frame is defined as

\[
X(t) = \arg\min_{\alpha} (t_{i+1} - t) W_2^2(X_t^i, \alpha) + (t - t_i) W_2^2(\alpha, X_{t+1}^i).
\]

In this way our output trajectories are guaranteed to adhere to the keyframes: $X(t_i) = X_i$. This step of our pipeline is similar to [Solomon et al. 2015], who use Wasserstein barycenters for shape interpolation, but their barycenters are computed directly between keyframes while our barycenters are computed between ODE-advected keyframes. When $L_{\text{fit}} = 0$, Equation 17 becomes trivial with $X(t) = X_t^i = X_{t+1}^i$. Our interpolation is illustrated schematically in Figure 2.

5 IMPLEMENTATION

Keyframes have been presented so far as measures over $\mathbb{R}^d$ with sample access. Our implementation mirrors this and samples $N$ points per keyframe in every training iteration. We increase $N$ by a factor of 1.26 every 50 training iterations. For 2D (3D) examples, $N$ is initialized to 300 (1000) points. All keyframes are jointly normalized to lie within $[-1, 1]^d$. Our state derivative $f_0$ is a multilayer perceptron with normally sampled random Fourier features [Tancik et al. 2020] of standard deviation $\sigma_{z,t} = 6.7$ in spatial and temporal dimensions. We use three hidden layers of size 512 each. All nonlinearities are Tanh except for the final layer, which is Softplus.

We also employ incremental unmasking of random Fourier features during training, as described by Hertz et al. [2021], with a modified rate so that all features are unmasked when 80% of training iterations are finished instead of their 50%. All models were trained for 300 iterations with a learning rate of $10^{-4}$. We use Adam [Kingma and Ba 2014] with default parameters and a learning rate scheduler that halves the learning rate on plateau with a minimum learning rate of $10^{-7}$.

The Sinkhorn divergence in Equation 8 is computed via “Geo-\text{omLoss” [Feydy et al. 2017] with entropic regularization weight $\epsilon = 10^{-4}$. The space-time integral in Equation 16 is computed each
Fig. 7. Effects of various regularizers on a four-keyframe animation. Employing rigidity, user-directed alignment, acceleration, and swirliness regularization as well as modifying the standard deviation of the RFF results in trajectories with varying path qualities and shapes. We also compare to trajectories obtained using optimal transport and [Chewi et al. 2021].
the trajectory slightly over-rotates just past the head of the fish in keyframe 3; \( \lambda_{\text{acc}} \) increased the swirliness of the animation.

User-Directed Alignment and Cyclic Trajectories. Figure 6 demonstrates the effect of \( L_{\text{acc}} \). We interpolate from a butterfly to a cat and finally to a caterpillar. The base trajectory is shown on the left. In the middle, we add \( \lambda_{\text{acc}} = 10^{-1} \) regularizer that penalizes alignment of the velocity field to the unit radial vector field. As a result, the trajectory takes a round circuitous path. On the right, we compute a cyclic trajectory by repeating the first keyframe at the end of the keyframe list. We round temporal RFF coefficients to the nearest values that produce cyclic signals with a 3 second period. Finally, we add the same \( \lambda_{\text{acc}} = 10^{-1} \) regularizer as in the middle to encourage a circular trajectory. The result is a perfectly loopable animation depicting the fictitious life cycle of a butterfly.

Regularizing Trajectories. Figure 7 demonstrates the isolated effects of various regularizers on the trajectory of a four-keyframe animation. The goal is to transform from a witch, into a pumpkin, into a cat, and finally into a bat. By employing different regularizers, we achieve varying effects on the animation.

In the top left, we show a baseline trajectory with just \( \lambda_{\text{peak}} = 10^{-2} \) as a regularizer. Here, in-between frames maintain a mostly upright posture, i.e., the top of each keyframe is mapped to the top of the following keyframe. When the aspect ratios of the keyframes differ, this trajectory squashes keyframes into one another.

Next, we impose an additional \( \lambda_{\text{rig}} = 10^{-1} \). While there is no truly rigid interpolation between these keyframes, the cat rotates sideways into the bat, yielding a more rigid trajectory than the base case. The orange arrows indicate the path from the cat to the bat.

We then replace the rigid regularizer with \( \lambda_{\text{acc}} = 10^{-1} \), where the metric is chosen to incentivize alignment of \( f_b \) to the unit radial vector field. The origin is plotted in red. Due to \( \lambda_{\text{acc}} \), the trajectory of the pumpkin to the cat is pulled towards the origin, creating a bouncing effect.

As a comparison, on the top right, we show the trajectory from concatenated OT maps. This results in extremely sharp turns at the keyframes, which is expected since trajectories are built piece-wise.

On the bottom left, we impose \( \lambda_{\text{acc}} = 10^{-1} \) on top of the base. The shape of the trajectory at the pumpkin keyframe is smoother compared to the trajectory taken in the base case.

Next we replace \( \lambda_{\text{acc}} \) with \( \lambda_{\text{curl}} = 10^{-1} \) where again, the curl is incentivized to be \( c = [0,0,-\pi]^T \). As a result, the trajectory makes almost a full \( 2\pi \) rotation.

Then we remove all regularizers and increase the RFF standard deviation from \( \sigma_{\text{E}} = 6.7 \) to 11.1. The increased magnitude of RFF coefficients and lack of regularization yield a much noisier trajectory.

Finally on the bottom right, we show [Chewi et al. 2021] for contrast. It produces a smooth path of cubic splines that are qualitatively close to the \( \lambda_{\text{acc}} = 10^{-1} \) case. Similar to Figure 5, the trajectory does not rotate keyframes, squashing them to get the right aspect ratio.

Volumetric Results. In Figure 1 we demonstrate application of our method to several 3D animations. In the first row, we build in-between frames for an animation from an open hand, to a partially closed hand, and finally to a cat. This animation is regularized with \( \lambda_{\text{acc}} = \lambda_{\text{rig}} = 10^{-1} \). In the second row, we interpolate from a sphere to a cow to a torus. In the last row, we interpolate through five keyframes of rings at different angles. The first, third, and fifth keyframes are rings with helical patterns carved into them, while the second and fourth keyframes are normal tori. We regularize this animation with \( \lambda_{\text{acc}} = \lambda_{\text{rig}} = 10^{-1} \) to produce a smoother trajectory and include in supplementary materials a version without this extra regularization.

7 DISCUSSION AND CONCLUSION

Unstructured animations appear in various forms of media ranging from hand-drawn to video games and film. These animations share a fluid morphing capability that is mesmerizing to watch but challenging to construct. Our work identifies unstructured animation as a density interpolation problem and builds automatic solutions through the machinery of optimal transport, neural ODEs, and PDE-based regularizers for intuitive and varied stylization.

Future work might consider discontinuous parameterizations of the velocity field. Depending on the context, spatially discontinuous trajectories may be desirable but integration of a smooth velocity field will always produce a diffeomorphism. Discontinuous velocity fields provide added flexibility, but also pose challenges to gradient based optimization and ODE integration. A natural regularizer in this setting is vectorial total variation [Goldluecke and Cremers 2010], which measures vector field smoothness but does not diverge near discontinuities.

Another avenue for further exploration might be to treat the fitting loss as a constraint. If the fitting loss can reach exactly 0, Wasserstein barycenter interpolation would no longer be necessary. The final rendering quality of our animations depend in part on the number of samples used to build the trajectory. Since the Wasserstein barycenter interpolation is computed independently per in-between frame and scales in expense with the number of points, it would be ideal to skip computing barycenters altogether.

Mesh- and rig-based animation are approachable for beginners through the abundance accessible tools and tutorials. Unstructured animation is the opposite: Almost no documented computational tools exist enabling its design. This paper represents a step toward bridging the gap between rig-based animation and unstructured animation. We hope that the graphics community will discover more exciting approaches toward unstructured animation to further improve its ease of construction and accessibility.

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In International conference on machine learning. PMLR, 214–223.

Mirela Ben-Chen, Adrian Butscher, Justin Solomon, and Leonidas Guibas. 2018. On discrete killing vector fields and patterns on surfaces. In Computer Graphics Forum, Vol. 29. Wiley Online Library, 1701–1711.

Jean-David Benamou and Yann Brenier. 2000. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. Numer. Math. 84, 3 (2000), 375–393.

Jean-David Benamou, Thomas O Gallouët, and François-Xavier Vialard. 2019. Second-order models for optimal transport and cubic splines on the Wasserstein space. Foundations of Computational Mathematics 19, 5 (2019), 1113–1143.

Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich. 2011. Displacement interpolation using Lagrangian mass transport. In Proceedings of the 2011 SIGGRAPH Asia Conference. 1–12.
