Simulating Fluids in Real-World Still Images

Siming Fan[1,*], Jingtan Piao[1,3,*], Chen Qian[1], Hongsheng Li[2,3,4,✉], Kwan-Yee Lin[2,3,✉]
[1]SenseTime Research, [2]Shanghai AI Laboratory, [3]The Chinese University of Hong Kong, [4]CPII
{fansiming,qianchen}@sensetime.com, 1155116308@link.cuhk.edu.hk,
hsli@ee.cuhk.edu.hk, junyilin@cuhk.edu.hk

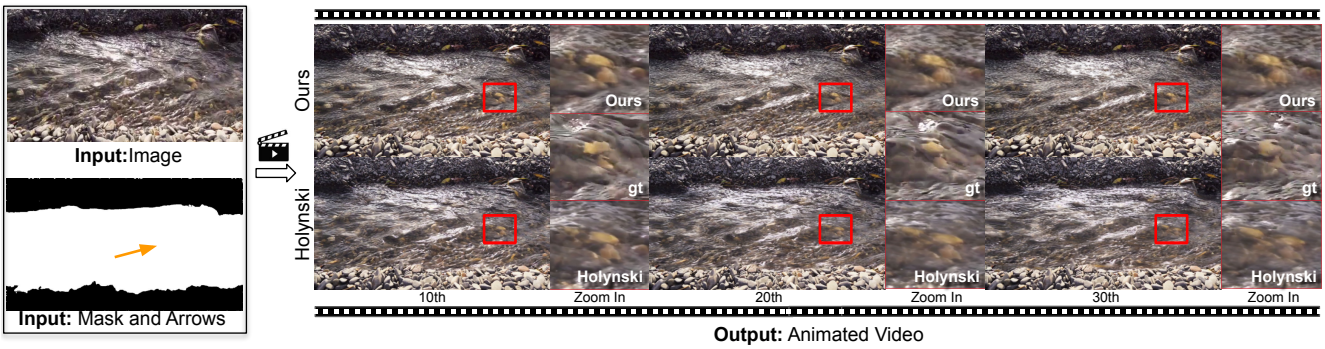

Figure 1. Given a still image and a coarse hint (mask and arrows) of motion as input (left), our model estimates fluid motion to generate an animating video (right). Holynski [11] is a state-of-the-art method. Please refer to the supplemental materials for video playing.

Abstract

In this work, we tackle the problem of real-world fluid animation from a still image. The key of our system is a surface-based layered representation, where the scene is decoupled into a surface fluid layer and an impervious background layer with corresponding transparencies to characterize the composition of the two layers. The animated video can be produced by warping only the surface fluid layer according to the estimation of fluid motions and recombining it with the background. In addition, we introduce surface-only fluid simulation, a $2.5D$ *fluid calculation, as a replacement for motion estimation. Specifically, we leverage triangular mesh based on a monocular depth estimator to represent fluid surface layer and simulate the motion with the inspiration of classic physics theory of hybrid Lagrangian-Eulerian method, along with a learnable network so as to adapt to complex real-world image textures. Extensive experiments indicate our method's competitive performance for common fluid scenes and better robustness and reasonability under complex transparent fluid scenarios. Moreover, as proposed surface-based layer representation and surface-only fluid simulation naturally disentangle the scene, interactive editing such as adding objects and texture replacing could be easily achieved with realistic results. Code, and dataset are publicly available.*

1. Introduction

Given an image of a scene containing liquid such as streams and waterfalls, humans can imagine how the liquid in the still image would move. The problem of animating fluid from a single image has become a blooming topic recently studied by a series of work [6, 11, 27, 9, 21]. Their general pipeline can be summarized as: (1) estimating optical flow that indicates the movement of liquid and (2) synthesizing future frame image based on the motion estimation. Although these methods achieve impressive visual effects on simple fluid regions, how to handle complex contents with challenging transparency, collisions, and thin structures in real-world scenes is still a challenging and open problem.

The key to the limitation of previous work is that the generation process of videos is modeled in a global manner. Specifically, the subsequent frames are generated based on warping existing textures on input images [3] or neural features [6]. Such operations regard the scene as a whole and warp all the contents in the scene together. For some complex transparent fluid cases, as shown in Figure 2(b), we can see through the water to observe the background rocks beneath the liquid. Textures on those static objects

*Equal contribution and joint first authorship.
✉Corresponding authors.

Project page: https://slr-sfs.github.io/

are also improperly deformed and result in unnatural animations. To decompose the influence of motion on fluids and surrounding static objects, we propose a two-layer representation for scenes with fluids, namely **S**urface-based **L**ayered **R**epresentation (SLR), which models the motion of the surface fluid and the rest background separately. To achieve such decomposition from the still image during inference, we propose a temporal-based training scheme, which facilitates the network to learn the decomposition with multiple supervisions over spatial and temporal dimensions of running fluid videos.

Another major difficulty in animating a single fluid image is predicting the motion flow of the fluid. There exist two types of methods. The first category is the interactive motion prediction methods. They require sparse hints (*ie*, labelling the velocity direction and relative amplitude) on random pixels of the liquid region as additional inputs. The feature clustering model [27] or convolutional network [11] is then used to generate a dense motion field. As the precision depends on the level of motion details a user can provide, such methods are time-costing in complex fluid scenes. The second category is automatic methods [6]. These works regard the task as a domain transformation problem, and use image-to-image translation techniques to predict the motion represented by 2-channel optical flows. Ambiguous velocity (e.g., a flat river surface can flow either to the left or to the right, as shown in Figure 2(a)) and tremendous changes of velocity in a period of time caused by the collisions with objects (as shown in Figure 2(c)) may pose great challenges to the above methods. To tackle the challenges, we introduce a simplified Navier-Stokes simulation into the motion estimation process, called **S**urface-only **F**luid **S**imulation (SFS). Specifically, the initial motion prediction of the liquid region is firstly calculated through sparse user inputs and interpolation inside the whole region. In parallel, a 3D mesh is built based on a monocular depth estimator. Then, the N-S equation is solved on the mesh surfaces and based on the initial motion to obtain a simulated motion field in a short period of time. Finally, since the real thickness of the fluid is neglected during the simulation, a refinement step is expected to compensate for the motion offset at height. Thus, a DNN-based motion translator is applied subsequently to obtain the refined motion field that adapts naturally to the input image. With SFS, we can enforce precise controls on the movement of fluid.

In summary, our work has three major contributions. (a) We propose a new and learnable representation, surface-based layered representation, which decomposes the fluid surface and the static objects in the scene to better synthesize the fluid animating videos from a single fluid image. (b) We design a surface-only fluid simulation to model the evolution of the image fluids with better visual effects. (c) Based on the proposed surface-based layered representation and surface-only fluid simulation, image editing with fluids is achieved with realistic and vivid results. In addition, We can easily extend our fluid animation system to downstream editing tasks, thanks to the SLR and SFS designs in our framework. For example, we can augment objects in the fluid region or background region, and simulate the interactions between the fluids and objects. We can also replace textures to different layers to create different scenes.

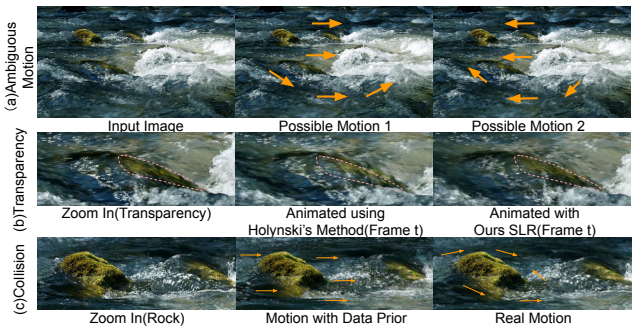

Figure 2. **Motivation**. (a) Given a still input image, the motion may be ambiguous. (b) Animating via previous SOTA method [11] causes background rocks beneath the liquid improperly deformed. (c) Motion prediction methods such as [11, 27] cannot well capture motion changes caused by the collisions with objects.

2. Related Work

2.1. Video Generation

In early studies, researchers focus on generating new video context by changing the attributes of recorded frames [32, 33]. Traditional methods of video motion generation with movable substances are based on stochastic process analysis [34] and color transfer [32]. Given a reference video indicating the temporal changes of a scene, texture movement and sequential images can be generated using a patch aligned method and energy-based optimizing [33]. Similar methods are applied to single image input [3]. However, obvious artifacts exist, and the smoothness of the video is not guaranteed. These drawbacks make the usage of such methods limited to typical types of scene samples.

In recent years, since the development of deep neural networks in image generation field [15], plenty of innovative efforts have been made to improve the synthesis quality of images [7, 16, 8, 35, 25]. Techniques such as GANs [8], perceptual loss [17], partial convolution [25] and cycle-consistency [35] have been widely used as standard components. Built upon the success of image generation, several attempts have created the precedents of neural video generation. A few methods are developed to force the network to learn intermediate scene representation for predicting the future frames or interpolating key frames, once the input sequences and camera trajectory are given (note: as we focus on single image input, we skip the review of

such multi-frames input trend). Given a single image as input, latent generative methods like [24] proposes an autoregressive model to generate infinite pixels forming sequential outputs. InfinityGAN [23] disentangles global appearances, local structures and textures with a structure synthesizer and texture synthesizer to generate images with spatial size and level of details not attainable before. [31] tries to use Fourier transforms in time variance showing the phase variations to represent the flow and generate moving videos. [20] spends efforts in digging into the semantic meanings in images and generates videos correspondingly to achieve a natural result. [6] decouples motion and appearance through learning intermediate flow fields and color transfer map. However, these methods are either focusing on spatial-wise consistency [20], or paying attention to coarse-level time-variance scene changes [31, 6]. None of them discusses or can handle the particularity of *fluid* with fine detail motions.

2.2. Fluid Simulation

Physical Fluid Simulation. Traditional methods originated from computer graphics to achieve accurate approximations on various types of fluids. Hybrid Lagrangian-Eulerian Methods [2, 36, 10] have successfully simulated the motion of incompressible fluids in a pre-defined 2D or 3D box environment, which solves pressure projection on Eulerian grid and advect on Lagrangian particles. To improve convergence ability and flexibility on complex shapes, material point method (MPM) [1] has been popular and more subsequent methods [18, 19] have been proposed to stabilize the convergence when iterating velocities. However, these methods are time-consuming in large scenes. To reduce the cost of simulating a large-range of fluid, methods have been proposed to simulate only a small range of fluid [4], or the surface of the fluid [5, 13] to produce a faster approximation by avoiding high computational cost in 3D volumetric solvers. Although these works achieve amazing simulation effects, they are still limited to synthetic scenarios due to scene-specific modelling.

Animating Fluids with Data Prior. Given a still image as animation target and a set of real-world water video candidates as animation bank, [31, 29] propose to retrieve a video or a group of videos from the bank that has similar content with the target image as the references. Then they transfer motion and appearance of references to each region of interest on target image to form the water animation video. These methods open the door of real-world fluid animation with data prior in-the-wild. However, the seamless alignment of animated results with source input image can not be well guaranteed and requiring lots of manual efforts. With the development of deep learning, recent works [11, 27] propose to automatically predict each frame of motion and appearance via learning from reconstruction loss with real world videos for training. [11] first simplifies motion estimation part as a single frame motion prediction via motion Eulerian integration and then proposes a deep feature warping technique to narrow down the size of blank areas caused by warping. Finally, it adaptively blends forward and backward features through a learnable parameter Z to generate animated fluid video. Based on [11], [27] proposes to regress fluid motion conditioned to user's sparse guidance and generate paired training data through motion speed clustering. The work further proposes to use multi-scale representation to capture different fluid speed in different resolution. Although these methods take a step further to real-world scenes with impressive results, it is hard for them to handle complex context relations over fluids due to the single representation to the entire scene that regards different objects and textures as equal.

Unlike previous work, our system marries the advantages of both physical simulation and learning-based pipelines. For textures, to avoid interplay between fluids and surroundings, we force the network to decouple the scene into a new two-layer representation **SLR**, that includes a static background and time-varies surface fluid layers[2]. Meanwhile, the network can hallucinate reasonable and photo-realistic textures in vacated regions thanks to the large-scale data prior. For motion, to skirt scene-specific modelling while keeping physic reasonability, we propose a three-step motion prediction. We first use sparse labels from the human interface to generate the initial rough motion and then calculate reasonable evolution in a short period by introducing a 2.5D surface simulation, **SFS**. Later, a smooth motion translator is used to refine the motion trend to better fit real-world examples with data prior. Moreover, as a consequence, we can generate natural fluid animations with the network training on plenty of real-world examples and flexibly edit the main liquid region and boundary condition to create various visual effects.

3. Method

Given a real-world image that contains liquid, our goal is to generate a video clip that animates vivid flowing of the fluid. To achieve this goal, a system is required to (1) represent complex context relations between fluids and surrounding environments; and (2) to estimate motion fields from single image input with reasonability and efficiency. We propose a novel system with two main designs that satisfy the above requirements. For texture, in contrast with previous methods that regard all elements in the scene as an entirety, our system learns a surface-based layered representation (SLR) (Sec. 3.1) that decomposes the scene into surface fluid layer and impervious background layer. Since the tex-

[2]This idea shares a similar spirit with [26] in separating a scene into both static and dynamic parts. Whereas, our **SLR** could handle transparent fluids, and only require a single image as input.

ture characteristics of surface fluid and rest background are assigned to different layers, such a design could overcome the negative influence of motion flows between fluids and surroundings (e.g., improper texture warping, as shown in Figure 2(b)). As for motion estimation, the system provides a three-step motion estimation alternative by introducing a surface-only fluid simulation (SFS) into the motion field prediction (Sec. 3.2). The motion prediction steps combine advantage of both physical- and learning-based motion estimation pipelines, thus ensuring physical reasonability and meanwhile skirting the scene-specific modelling problem. Each component is detailed in following subsections.

3.1. Learning of Decomposition of Images

In this subsection, we first introduce Surface-based Layered Representation (SLR). Then, we show how to obtain animated videos through SLR. Finally, we present how to learn SLR with a neural network (illustrated in Figure 3), and challenges we meet during optimizing the network since there is no oracle ground-truth supervision for fluid scene decomposition.

Surface-based Layered Representation. Given a still image $I(T_0)$ as input, a series of RGB frames $\mathbf{I} = \{I(T_0), \dots, I(T_n)\}$ are outputs to synthesize time-space variations of the scene with running fluids. As we aim to handle complex context of fluid scene through disentangling fluid motion and still objects beneath/interacting with surface fluid, we propose to learn a novel intermediate representation whose output is an impervious background layer image[3] I_b, a temporal sequence of surface fluid layer images $\mathbf{I}_f = \{I_f(T_0), \dots, I_f(T_n)\}$ and transparency factors α_b, $\boldsymbol{\alpha}_f = \{\alpha_f(T_0), \dots, \alpha_f(T_n)\}$, where $\alpha_* \in [0, 1]$. Transparency factors indicate the contribution of colors on each layer to the final image at each time. In practice, we utilize a convolution network to output the *simplification* of the representation: $F_\theta : (I(T_0), I(T_n), \mathbf{M}) \rightarrow (I_b, I_f(T_0), I_f(T_n), \alpha_b, \alpha_f(T_0), \alpha_f(T_n))$, where the network only outputs the estimated *first* and *last* frames of $\mathbf{I}_f$ as well as $\boldsymbol{\alpha}_f$. $\mathbf{M}$ is the estimated motion fields of T_0 to T_1 from the single input $I(T_0)$. We will detail the synthesis of $\mathbf{M}$ in Sec. 3.2.

We expect the background to be identical across all times, while the fluid surface deforms with realistic motions. To this end, F_θ is enforced to *automatically* map the original texture of non-liquid and static regions as well as the reasonable texture of under-liquid surface regions to the background layer (detailed in later subsection). As for fluid surface learning, recurrent estimation for all frames (RNN) is a possible alternative that can be tailored from video generation [30]. However, such an intuitive design might cause frame distortion as time goes on due to the accumulated errors and hard convergence (also demonstrated in [11]). In contrast, We use the linear combination of the first and last frames to construct the images at intermediate frames. This choice not only transforms the problem into interpolation rather than extrapolation, which is easier to converge, but also is able to generate an endless cyclic video rather than a video of a fixed interval. In detail, we tailor the symmetric softmax splatting, which is a pixel-warping strategy utilized in [11, 28], to our framework. We extend the splatting operation from a single image feature splatting into two levels – fluid surface feature splatting and fluid alpha splatting as follows :

$$D_f(T_i;\hat{\mathbf{x}}') = \frac{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha(T_i;\hat{\mathbf{x}}) \cdot D_f(T_j;\hat{\mathbf{x}}) \cdot \exp(Z_f(T_j;\hat{\mathbf{x}}))}{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha(T_i;\hat{\mathbf{x}}) \cdot \exp(Z_f(T_j;\hat{\mathbf{x}}))}, \quad (1)$$

$$\alpha'_f(T_i;\hat{\mathbf{x}}') = \frac{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha(T_i;\hat{\mathbf{x}}) \cdot \alpha'_f(T_j;\hat{\mathbf{x}}) \cdot \exp(Z_f(T_j;\hat{\mathbf{x}}))}{\sum_{\hat{\mathbf{x}}\in\mathcal{X}} \alpha(T_i;\hat{\mathbf{x}}) \cdot \exp(Z_f(T_j;\hat{\mathbf{x}}))}, \quad (2)$$

where $j \in \{0, n\}$ is the start index or last index of sequences, indicating forward splatting the frame 0 or backward splatting the frame n, which means the value j is :

$$j = \begin{cases} 0, & \text{if } \hat{x}' = M_{0\rightarrow t}(\hat{x}), \\ n, & \text{if } \hat{x}' = M_{0\rightarrow t-N}(\hat{x}). \end{cases} \quad (3)$$

$D_f(T_i;\hat{\mathbf{x}}')$ is the fused fluid feature value of target frame at pixel $\hat{\mathbf{x}}$, and $\alpha'_f(T_i;\hat{\mathbf{x}}')$ is the transparency value of target frame at pixel $\hat{\mathbf{x}}$. $D_f(T_i;\hat{\mathbf{x}})$ represents value of $D_f(T_i)$ at pixel $\hat{\mathbf{x}}$, $\alpha_f(T_i;\hat{\mathbf{x}})$ and $\alpha'_f(T_i;\hat{\mathbf{x}})$ are similar. During inference, we set $D_f(T_n) = D_f(T_0)$ and $\alpha_f(T_n) = \alpha_f(T_0)$ since only single image is available. $\mathcal{X}$ is the set of pixels satisfying $M_{0\rightarrow t}(\hat{\mathbf{x}}) = \hat{\mathbf{x}}'$ or $M_{0\rightarrow t-N}(\hat{\mathbf{x}}) = \hat{\mathbf{x}}'$, $M_{0\rightarrow t}$ is displacement map generated from $\mathbf{M}$ through euler integration [11, 28], and $\alpha(T_i;\hat{\mathbf{x}}')$ is a temporal coefficient to fuse features from forward warping and backward warping to ensure endless looping:

$$\alpha(T_i;\hat{\mathbf{x}}') = \begin{cases} \frac{t}{n}, & \text{if } \hat{x}' = M_{0\rightarrow t}(\hat{x}), \\ 1 - \frac{t}{n}, & \text{if } \hat{x}' = M_{0\rightarrow t-n}(\hat{x}). \end{cases} \quad (4)$$

Then a fluid feature decoder is applied to $D_f(T_i)$ to generate surface fluid image $I_f(T_i)$ and a alpha refinement network is applied to $\alpha'_f(T_i)$ to generate transparency alpha $\alpha_f(T_i)$ to avoid vacant alpha particles.

Animating Fluids with SLR. Once we obtain the final surface fluid layer $I_f(T_i)$ and transparency $\alpha_f(T_i)$, the reconstructed final image $I(T_i)$ can be acquired by adding the impervious background layer I_b back to the surface fluid layer $I_f(T_i)$ with α_b. Here we use separate α channels for background and fluid transparency predictions. This design helps to stabilize the training of decomposition. The core reason is that when a single fluid transparency α_f is warped according to the motion flow, it leaves a blank region on the boundary, and makes the gradients hard to be

[3] The word "impervious" is not rigorous, as background layer is not only formed from static object but also non-surface fluid which contains air molecules in real environment. However, we assume the fluid is incompressible, and thus a little bit abuse the word "impervious background" to indicate the non-surface fluid regions.

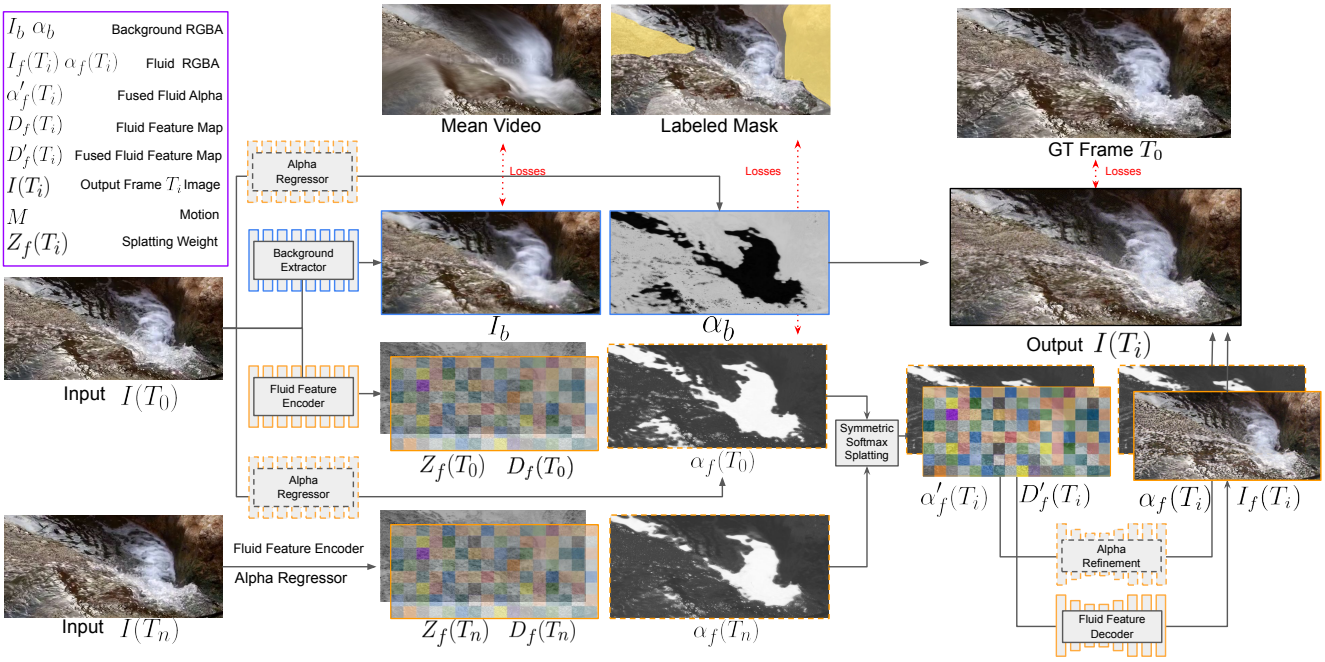

Figure 3. **Training and testing pipelines**. The background layer and background alpha are predicted only at the first frame. Both fluid features and fluid alpha are predicted at the first and last frames, and then symmetricly splatted and blended using the softmax splatting at frame T_i. A decoder is used to decode fluid features into images and refined alpha. For frame T_0, motion is calculated following pipeline Figure 4, and for frame T_i, motion is calculated through Euler integration. During inference, T_n is replaced by T_0 to create a cyclic video.

backpropagated. Although with the two-channel implementation, there still is a α_b on that vacated regions to fascinate the network refining the texture. In formula we have

$$I(T_i) = \frac{\alpha_f(T_i)I_f(T_i) + \alpha_b I_b}{\alpha_f(T_i) + \alpha_b} \tag{5}$$

For corresponding network architectures to achieve the above targets, as shown in Fig 3, the encoder is used to extract background texture and α channel as well as surface fluid layers' features and its α. The decoder is used to refine the fused fluid layer that combines the layers from the first and last frames to obtain the complete surface fluid layer, and the final animated image is reconstructed by compositing background and surface fluid layers.

Optimizing the SLR. The training process is divided into three parts, in which, we separately train (1) the surface fluid branch containing the encoder that predicts the fluid feature layer from image input and the decoder that reconstructs the warped fluid layer, (2) the background branch that predicts the background layer from the image input, and (3) jointly train them together with α learning and surface fluid/ background branches finetuning. For each training step, the paired training data $< (I(T_0), I(T_n)), I(T_i) >$ are randomly picked among 60-frames sequence without constraint on interval. We use dense optical flow estimation network [14] to extract reasonable movement of each video as pseudo motion ground-truth, and to ensure smoothness, we average the flow with window size of 30 frames. Due to space limits, we detail the training of each stage in the supplementary materials.

3.2. Motion Field Estimation

In the previous section, we have described the core necessities to learn an SLR but skipped the synthesis process of one of its input, motion fields $\mathbf{M}$. In this section, we discuss how to obtain $\mathbf{M}$.

During training, the motion of each frame can be generated from a conventional optical flow estimator [14]. During inference, as the input is a single image, we expect to synthesize a reasonable motion map from the observed scene. We propose a motion estimation process that separates the motion prediction into three parts taking advantage of both physical-based and learning-based pipelines for fluid simulation. First, interactive labelling of the sparse velocity of several pixels on the image is required to guide the generated motion directions of $I(T_0)$, along with a fusion with weights calculated based on Gaussian filtering of the distance maps from the labeled pixel to build a dense velocity initial map. Second, a surface-only fluid simulation is calculated based on initial motion and estimated scene depth to simulate a following convincing evolution of the liquid. Note that this step only calculates for the first frame. Last, a DNN-based pixel-to-pixel motion translator is utilized to refine the flow to create a more realistic and natural motion map. This translator could be considered as generating a compensation for a detailed motion map that is

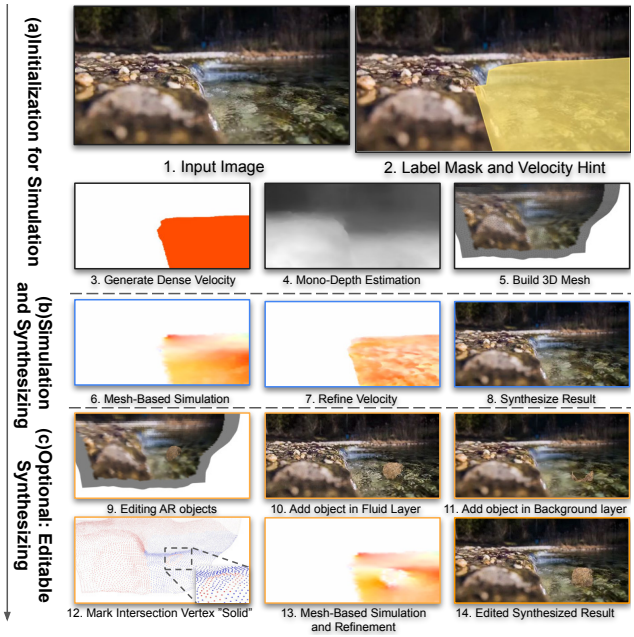

Figure 4. **Motion calculation pipeline**. (a) is initialization of fluid region, rock region, motion, and Mesh. (a,b) combine to form our 2.5D simulation and animation pipeline, where (4-5) is not included if using 2D simulation method. (a,c) combine to form our editable simulation and synthesis pipeline, where the user can edit 3D objects and create corresponding motion.

harmonic with fluid textures. The pipeline is illustrated in Figure 4.

Initial Motion from Interactive Sparse Labeling. To obtain determined as well as editable motion direction, we choose an interactive alternative to generate initial flow for image $I(T_0)$ following [27]. With no more than 10 discrete notations on images, and a fixed segmentation of the liquid region, we use nearest-neighbors-averaging to assign all the pixels that are inside the liquid region an initial velocity.

Surface-only Fluid Simulation. After the initial state is estimated, we tailor the traditional graphics pipeline with discretization and linear equation to simulate the follow-up evolution of the liquid. Intuitively, we can directly apply simulation on 2D grids that align with image pixels by initializing each pixel as an Eulerian grid and several Lagrangian particles near each pixel. However, since such perspective projection of the 3D scene may cause severe distortion when direct manipulating the motion in 2D, as shown in Figure 7, we thus turn to use a mono-depth estimator [22] to estimate the 3D surface of the liquid. After the 3D surface is constructed inside the region of the liquid, a mesh surface-based velocity is calculated, with the restriction of projected velocity :

$$\frac{d}{dt}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} fx & 0 \\ 0 & fy \end{bmatrix} \begin{bmatrix} 1/z & 0 & -x/z^2 \\ 0 & 1/z & -y/z^2 \end{bmatrix} \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix} \quad (6)$$

Where fx, fy, cx, cy is camera intrinsic parameters, u, v is the velocity on the projected image and x, y, z the 3D position in the scene. Since we have modelled the liquid as a surface, we replace traditional Laplacian operator with Laplacian-Beltrami operator on the mesh and use Mixed-Euler-Lagrange approach [36] implemented in Taichi [12] to simulate consequent motion of each vertex. *Detailed formula derivation can be seen in the supplementary*. The particle binding on the surface is randomly sampled according to the surface area. When a particle is moved outside the mesh surface, we use the nearest projection to force the position to fall onto the mesh. For the boundary of the mesh, we calculate the velocity of the boundary triangle, judging whether the velocity on the edge boundary is inward or outward. For outward edges, the particle is abandoned after moving out of the liquid region. For inward region, a particle source is built to maintain the overall number of particles to be roughly stable. With the texture of surface fluid layer binding to each vertex, we can warp fluid layer with a soft rasterizer to simulate flows. In this way, given arbitrary flow initial state, a stable velocity can be calculated to approximate real-world flow after a period of simulation.

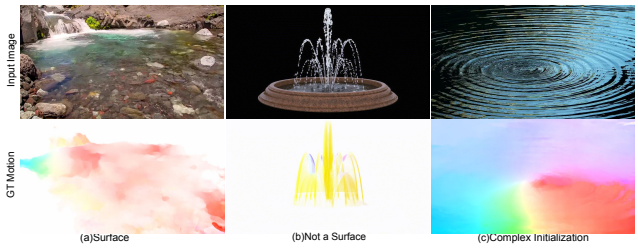

Figure 5. **Examples for Different Fluid Scenes**. (a) The surface thickness of the waterfall and river are gauzy. (b) The fountain cannot be considered a surface. (c) Fluid with vortex contains complex motion.

Smooth Motion Translator. With the help of above $2.5D$ fluid simulation, we can generate a time-variant velocity among the sequence. However, the simulated velocity is still a rough trend since it only works on fluid surface without considering thickness variation of fluids. Thus, the SFS can work well in cases like Figure 5(a). It fails in cases with distinct depth measurement differences, as shown in the Figure 5(b) due to the lack of height information during calculation. Also, It is fragile to cases with complex initial state as shown Figure 5(c) due to error-prone/time-consuming interactive initial motion labeling for such cases. Thus, we need extra scheme to compensate or rectify for detailed motion that is harmonic with the fluid textures. To this end, we use a convolution-based network to transfer the simulated motion into a more detailed and reasonable one, which is pixel-aligned with image textures. The network is trained as a traditional style-transfer task, with a L2 loss between output velocity and ground truth. An external patch discriminator is used to refine the pattern of the velocity distribution. All the settings are similar to the image translation task pipeline [15].

4. Experiments

In this section, we present experiments and detailed analysis to demonstrate the contributions of our method both quantitatively and qualitatively. **For all the cases shown in this paper, please refer to our demo video and project page for better dynamic visual effects.**

4.1. Quality of Fluid Video Synthesis

Implementation Details. To learn SLR, we train our network on a series of videos containing a large region of fluid movement and still background [11]. For training of motion refinement network, we use the same settings in [27]. All networks are trained at a resolution of 256×256.

Datasets and Evaluation Metrics. In addition to validation set in [11], we collect extra 126 scenes to form a new test set, which we called **C**omplex **L**iquid **A**nimation in-the-**W**ild (CLAW) test set, for evaluating the performance on both dense fluids (such as rivers and oceans) and sparse or transparent fluids (such as streams or splashes). Note that, in this subsection, the motion for all the methods to be compared is guided by the optical flow of the whole video to distinguish the performance of the synthesizing techniques. We use PSNR to show the whole average error, SSIM to show the errors in the region with plenty of textures and LPIPS, a perceptual based loss, to show visual quality difference. For more details, please refer to supplementary.

Dataset	Methods	All Region			Fluid Region		
		LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑
Holynski Common Val Set	Reproduced Holynski	0.0798	25.03	0.7787	0.0657	25.88	0.8007
	Modified Holynski(Baseline)	**0.0793**	24.75	0.7758	**0.0656**	25.72	0.8000
	Ours	0.0834	**25.14**	**0.7795**	0.0657	**26.10**	**0.8030**
Our CLAW Testset	Reproduced Holynski	0.2067	20.26	0.5955	0.2029	20.36	0.5961
	Modified Holynski(Baseline)	0.2078	19.97	0.5923	0.2041	20.10	0.5934
	Ours	**0.2040**	**20.79**	**0.6080**	**0.1975**	**20.80**	**0.6077**

Table 1. **Quantitative comparison.** (a) Results on Holynski's common val set [11], evaluating first to 60-th frames."Fluid Region" means static region is replaced by input image during metric calculation for focusing on the statistic of fluid. This could be done by replacing the non-fluid region with the corresponding textures of input images. (b) Results on CLAW test set.

Quantitative Comparisons with baseline. Table 1 shows the results between our method and state-of-the-art method [11], which is a single layer learning-based system, a typical method that animates the scene in a global manner. We re-implement [11] in Pytorch (*Reproduced Holynski*), since the source code is not public available. For other comparison baselines, *Modified Holynski* is built upon *Reproduced Holynski* but with the modification of replacing convolution to partial convolution in the fluid decoder. The third model (*Ours*) is our SLR model[4]. Three observations can be included from Table 1: (1) PSNR and SSIM metrics of "Modified Holynski" drop a little compared to "Reproduced Holynski" in both statistics on all- and fluid region. We infer that, although partial convolution could better capture context than conventional one to impaint reasonable texture in vacated regions in theory, simply replacing partial convolution into single-layer framework cannot *essentially* improve the image quality due to the wrong context raised from improper warping of both background and fluid. While, with SLR, the partial convolution can strengthen its advantage on context capturing for more proper fluid context textures provided in surface fluid layer. (2) The LPIPS of *Ours* model drops a little compared to the *Modified Holynski* model in the "All Region" of Holynski common validation set, but has nearly the same LPIPS in the "Fluid Region". For the other two metrics, our model leads to an uptick. *Since most regions of Holynski's validation set are static*, the blending of the fluid and background in the non-fluid region could lead to slight performance drops in terms of LPIPS. However, it could be eased by replacing the non-fluid region with the corresponding textures of input images, thus we care more about the performance in the "Fluid Region" instead of the "All Region". Such a phenomenon tells us that the two-layer decomposition design in *Ours* model does not harm the reconstruction ability on final images. To better unfold the analysis, we further conduct the close-up evaluations on Holynski's val set. Concretely, we crop the fluid region by motion threshold and only evaluate these visible fluid background regions. The LPIPS, PSNR, SSIM of Holynski's method are $0.230, 19.01, 0.565$, and Ours are $0.227, 19.64, 0.581$. This setting leads to larger performance margins with the baseline, which demonstrate the advantage of our system. (3) When evaluating only in the fluid region, we can see model *Ours* reaches a comparable result with the "Modified Holynski" model in Holynski validation set [11], but improves significantly in CLAW test set. The major reason behind this observation is that when it comes to complex scenes as included in our CLAW test set, our proposed two-layer representation suppresses the influence between surrounding and surface fluids. Thus we can achieve better results than single-layer baselines that regard all elements in the scene as an entirety during warping.

Comparisons with SOTA methods. In this paragraph, we provide comparisons with more state-of-the-art methods – Endo *et.al* [6], Tavi *et.al* [9], and Mahapatra *et.al* [27]. The first two methods are centered on animating general scenes, where Endo *et.al* [6] is designed for landscape, and Tavi *et.al* [9] focuses on animating periodic patterns in still images. These two methods could be applied to fluid scenarios. In contrast, Mahapatra *et.al* [27] is specialized for fluid scenarios, where a multi-scale representation is introduced to capture different fluid speeds in different resolutions. Concretely, as Endo's code is publicly available, we

[4]For *Ours*, we use (*Ours (stage 1)* in Table 1 in our supplementary material.) as surface fluid model initialization. Since our training contains three stages, and we train 100 epochs at the first stage to get the model (*Ours (stage 1)*), then decay learning rate and fine-tune 50 epochs for fluid at the last stage in practice, we also apply the same training strategy for the first two baselines to ensure fair comparisons.

compare our system with it both quantitatively and qualitatively. For other methods, we compare them qualitatively in Fig 6. Specifically, the result of Mahapatra is from official project page, and result of Tavi is from official app. Endo's method can only control motion with a latent code. Thus, we test ten different latent codes and select the best one to compare. We observe that these three methods suffer from animation distortions with different levels – (1) from the zoom-in perspective, all baselines improperly move the still stones with the fluids. Specifically, Tavi's results have significant ghosting artifacts, which might be caused by its one-fold periodic motion and alpha-blending designs. (2) From the full image perspective, Endo's method has difficulty in estimating fluid motion. The river's motion is incorrectly predicted to be from right to left, and the waterfall flows upwards to the right. The quantitative results (Tab 1) provide the demonstration from another aspect. This phenomenon is foreseeable, as Endo's framework considers motion in a global manner. In contrast, our two-layer method animates plausible fluid without affecting still stones. Please refer to the supplementary material for more comparisons and straightforward video effects.

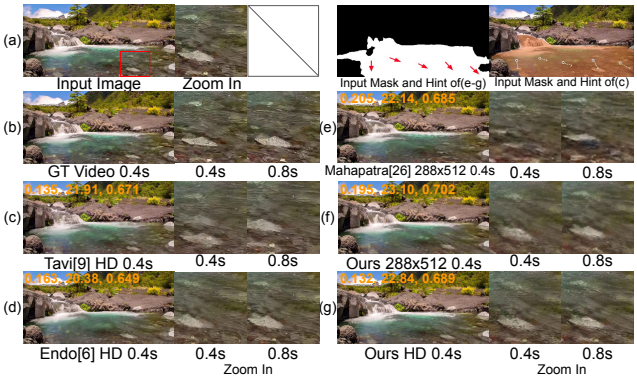

Figure 6. **Qualitatively Comparisons.** (e-f) are tested in 288 × 512, others are tested in W=1280. LPIPS/PSNR/SSIM of the whole video are colored in orange.

Methods	Holynski validation set						CLAW test set					
	All Region			Fluid Region			All Region			Fluid Region		
	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM
Endo[6] HD	0.357	16.27	0.351	0.120	22.31	0.741	0.357	15.92	0.296	0.347	16.38	0.331
Ours HD	**0.142**	**23.43**	**0.715**	**0.118**	**23.78**	**0.728**	**0.196**	**20.99**	**0.611**	**0.190**	**20.99**	**0.610**

Table 2. **Quantitative comparisons with Endo's method.**

User Study. We conduct a user study to compare visual effects between single layer baseline (*Modified Holynski*) and our two layers method(*Ours* in Table 1) on the full set of *CLAW testset*. For each scene, we animate 60 frames and output a two-second video. Animated videos are shown side by side without providing ground truth videos as reference to guide the participator focusing on the reasonability and photo-realistic. We ask the participants to select which one is better or neither is better, mainly according to three dimensions (photo-realistic, stereoscopic, and high-fidelity).

As shown in Table 3, the participators can make a distinction between the two methods over most samples, leaving 27.2% samples difficult to tell. Our method achieves better scores than the single-layer method [11] with 52.3% sample voting. This indicates the effectiveness of the proposed SLR in other aspects.

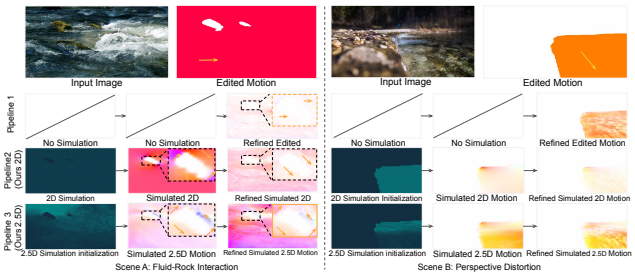

Figure 7. **Different simulations.** Orange arrows point out motion direction. All simulation methods use 75% weak incompression simulation instead of fully incompression for better visual effect. (row 1)Input image, sparse motion generated by sparse user hint required by each method. (row 2-4)ablation study of Mahapatra's method [27], 2D simulation and 2.5D SFS.

	Single layer	Ours	Neither
E-O-N	125(20.5%)	319(52.3%)	165(27.2%)

Table 3. **User study.** The subjective scores are voted on all samples of our CLAW testset, which includes 122 videos with various real-world fluids scenarios(*e.g.,* river, stream and waterfall) and complex context surrounding (*e.g.,* semi-/transparent liquid, collisions and thin structures).

4.2. Different Motion Prediction

Figure 7 shows different motion prediction pipelines. **Pipeline** 1**:** An interactive sparse labelling of the motion and then extending all the velocities to the whole moving region, which results in a constant flow that remains large gap with realistic motion. We can see that the motion appears reasonable global with a network refinement. However, for local regions around static rocks, the flow of liquid is improperly stiff and has no interactions with the rock. While in real-world scenes, there should be vortex and sprays caused by collisions. **Pipeline** 2**:** Built upon pipeline 1, while before the motion refinement stage, we additionally initialize grids with fluid, rock or air masks as well as placing particles around each grid, then advocate particles by edited motion for once. Later, a NS equation with the incompressible fluid assumption is solved on 2d grid to obtain the motion of the next frame, which we call "Simulated 2D Motion" in the figure. Considering pipeline 2 is simulating on a 2D image plane, a refinement step (using a smooth motion translator) is applied to compensate for the motion offset at height and fine details. **Pipeline** 3**:** instead of simulating on a 2D image plane, the N-S equation is solved in $2.5D$. Specifically, we regress the depth with a pretrained monocular depth estimator and consequently build a 3D mesh over the fluid. Then, we solve the N-S equation on the mesh surface. With

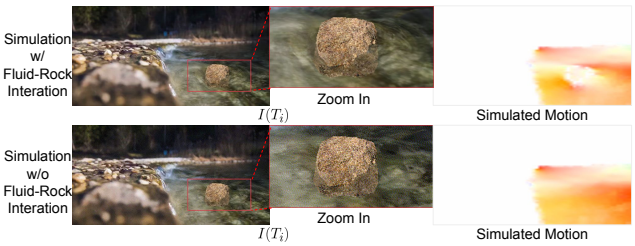

Figure 8. **Interactive Editing.** (Bottom) Initial all moving region as 'Fluid' statue. (Top) Initial 'Solid' statue at overlap area of mesh and new object.

the help of simulation, we can see in the left scene of the figure that the effect of turbulence around the rock can be animated, and the trend maintains the refined output. From the right scene in the figure, we can observe that the absolute value of the velocity is unnatural in Pipeline 2, which regards all particles on an identical plane. However, since the liquid surface leans in the picture, there should be a foreshortening effect on the projection. In contrast, pipeline3 samples particles on the surface and such simulation enable near-plane motion to carry more speed.

Quantitative Comparisons. Tab 4 shows the ablations of motion refinement in terms of endpoint error metric (lower is better). We evaluate all scenes in both two datasets with user input limited to five sparse hints. The results prove motion refinement is essential for achieving a better ability to represent GT motion. Refer to supplementary for quantitative metrics of the three motion prediction pipelines.

Methods	Dataset	
	Holynski Val Set	CLAW Test Set
DenseMotion	0.299	0.596
RefinedMotion	0.269	0.582

Table 4. **Quantitative Motion Refinement Ablation.**

5. Byproducts

Interactive Editing. We can augment fluid scene while keeping realistic fluid animation. For example, we can place an imaginary stone in a river that is originally slack water, and the flow could be updated to the new one with the effect of a vortex around the stone, as can be viewed in *Simulation w/ Fluid-Rock Interaction* of Figure 8. To achieve this, we only need to calculate interface of inserted stone mesh and fluid surface mesh. Then, we classify the part of stone above/beneath the river surface according to the depth. Later, fluid motion is updated with boundary conditions during simulation. Refer to supplementary for details.

Layer Replacement Editing. We can change texture attributes of both surface fluid and background by simply replacing either fluid layer or background layer with another reference texture. For example, as shown in Figure 9, the fluid layer can be replaced by/regrouped with a starry sky video with an endless loop to form dreamlike fluid effects that might remind the user of a Chinese classical poetry—*as if the Silver River fell from azure sky*. For a downstream task like such texture attribute replacement, accurately segmentation of all fluid regions is necessary to reach plausible visual effects. However, it is hard to be accomplished by brute force segmentation with either motion prediction (*Fluid Replacement Editing by Motion Threshold w/o SLR* in the figure) or manually labelling. Since real-world fluid scenes contain complex context relations such as semi-transparency, different motion behaviour among fluid regions in one scene and so on. On the contrary, since our proposed surface-based layer representation (SLR) naturally disentangles scene into surface fluid and background, we can achieve such editing easily.

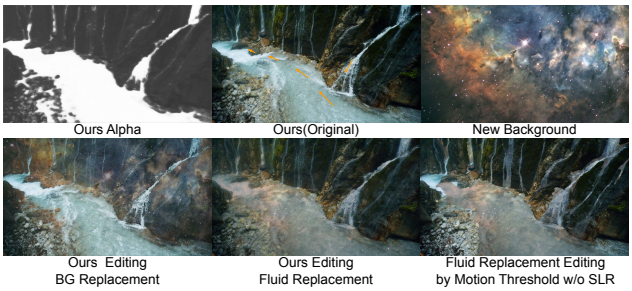

Figure 9. **Layer Replacement Editing**. First row show our SLR output and new background. The second row shows results of our background layer replacement effects, fluid layer replacement effects , and replacement effects using motion instead of alpha Arrows specify the motion direction and speed.

6. Conclusion

We propose a system for fluid animation from a still image. Specifically, for fluid texture, a neural network is designed to decompose the still image into surface liquids and background, which form a novel two-layered representation, SLR, to represent fluid scene. Compared to previous work, SLR can help better simulating the transparent fluids as well as other fluids scenes with complex surrounding. For motion estimation, we introduce surface-only fluid simulation (SFS), which provides a wider range of editing abilities on the fluid image. The system ensures physical reasonability and meanwhile skirting the scene-specific modelling problem. However, as our method works in a semi-supervised manner, without supervision from ideal decomposition or graphic-based rendering, the outputs may have some artifacts on boundaries. We expect more research on better disentangling and representing fluid surface with backgrounds, as well as more precise motion prediction.

Acknowledgements. This project is funded in part by National Key R&D Program of China Project 2022ZD0161100, by the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Commission (ITC)'s InnoHK, by General Research Fund of Hong Kong RGC Project 14204021. Hongsheng Li is a PI of CPII under the InnoHK.

References

[1] Scott G Bardenhagen and Edward M Kober. The generalized interpolation material point method. *CMES*, 5:477–496, 2004. 3

[2] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015. 3

[3] Yung-Yu Chuang, Dan B Goldman, Ke Colin Zheng, Brian Curless, David H Salesin, and Richard Szeliski. Animating pictures with stochastic motion textures. *ACM TOG*, 24:853–860, 2005. 1, 2

[4] Vincenzo Citro, Paolo Luchini, Filippo Giannetti, and Franco Auteri. Efficient stabilization and acceleration of numerical simulation of fluid flows by residual recombination. *J. Comput. Phys.*, 344:234–246, 2017. 3

[5] Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. Surface-only liquids. *ACM TOG*, 35:1–12, 2016. 3

[6] Yuki Endo, Yoshihiro Kanamori, and Shigeru Kuriyama. Animating landscape: self-supervised learning of decoupled motion and appearance for single-image video synthesis. *ACM TOG*, 38:175:1–175:19, 2019. 1, 2, 3, 7

[7] L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *CVPR*, 2016. 2

[8] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *NeurIPS*, 3:2672–2680, 2014. 2

[9] Tavi Halperin, Hanit Hakim, Orestis Vantzos, Gershon Hochman, Netai Benaim, Lior Sassy, Michael Kupchik, Ofir Bibi, and Ohad Fried. Endless loops: detecting and animating periodic patterns in still images. *ACM TOG*, 40:1–12, 2021. 1, 7

[10] Francis H Harlow. The particle-in-cell computing method for fluid dynamics. *Methods Comput. Phys.*, 3:319–343, 1964. 3

[11] Aleksander Holynski, Brian L Curless, Steven M Seitz, and Richard Szeliski. Animating pictures with eulerian motion fields. In *CVPR*, 2021. 1, 2, 3, 4, 7, 8

[12] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM TOG*, 38:201, 2019. 6

[13] Libo Huang, Ziyin Qu, Xun Tan, Xinxin Zhang, Dominik L Michels, and Chenfanfu Jiang. Ships, splashes, and waves on a vast ocean. *ACM TOG*, 40:1–15, 2021. 3

[14] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*, 2017. 5

[15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017. 2, 6

[16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2

[17] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 2

[18] ByungMoon Kim, Yingjie Liu, Ignacio Llamas, and Jaroslaw R Rossignac. Flowfixer: Using bfecc for fluid simulation. Technical report, Georgia Institute of Technology, 2005. 3

[19] Theodore Kim, Nils Thürey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM TOG*, 27:1–6, 2008. 3

[20] Pierre-Yves Laffont, Zhile Ren, Xiaofeng Tao, Chao Qian, and James Hays. Transient attributes for high-level understanding and editing of outdoor scenes. *ACM TOG*, 33:1–11, 2014. 3

[21] Thi-Ngoc-Hanh Le, Chih-Kuo Yeh, Ying-Chi Lin, and Tong-Yee Lee. Animating still natural images using warping. *TOMM*, 2022. 1

[22] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *CVPR*, 2018. 6

[23] Chieh Hubert Lin, Hsin-Ying Lee, Yen-Chi Cheng, Sergey Tulyakov, and Ming-Hsuan Yang. Infinitygan: Towards infinite-resolution image synthesis. *arXiv preprint*, arXiv:2104.03963, 2021. 3

[24] Andrew Liu, Richard Tucker, Varun Jampani, Ameesh Makadia, Noah Snavely, and Angjoo Kanazawa. Infinite nature: Perpetual view generation of natural scenes from a single image. In *ICCV*, 2021. 3

[25] Guilin Liu, Fitsum A Reda, Kevin J Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *ECCV*, 2018. 2

[26] Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, William T Freeman, and Michael Rubinstein. Omnimatte: associating objects and their effects in video. In *CVPR*, 2021. 3

[27] Aniruddha Mahapatra and Kuldeep Kulkarni. Controllable animation of fluid elements in still images. *arXiv preprint*, arXiv:2112.03051, 2021. 1, 2, 3, 6, 7, 8

[28] Simon Niklaus and Feng Liu. Softmax splatting for video frame interpolation. In *CVPR*, 2020. 4

[29] Makoto Okabe, Yoshinori Dobashi, and Ken Anjyo. Animating pictures of water scenes using video retrieval. *Vis. Comput.*, 34:347–358, 2018. 3

[30] Junting Pan, Chengyu Wang, Xu Jia, Jing Shao, Lu Sheng, Junjie Yan, and Xiaogang Wang. Video generation from single semantic label map. In *CVPR*, 2019. 4

[31] Ekta Prashnani, Maneli Noorkami, Daniel Vaquero, and Pradeep Sen. A phase-based approach for animating images using video examples. In *Comput Graph Forum*, 2017. 3

[32] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *Comput. Graph. Appl.*, 21:34–41, 2001. 2

[33] Arno Schödl, Richard Szeliski, David H Salesin, and Irfan Essa. Video textures. In *SIGGRAPH*, 2000. 2

[34] Meng Sun, Allan D. Jepson, and Eugene Fiume. Video input driven animation (VIDA). In *ICCV*, 2003. 2

[35] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 2

[36] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM TOG*, 24:965–972, 2005. 3, 6