Simulating Fluids in Real-World Still Images
- Supplementary Material -

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

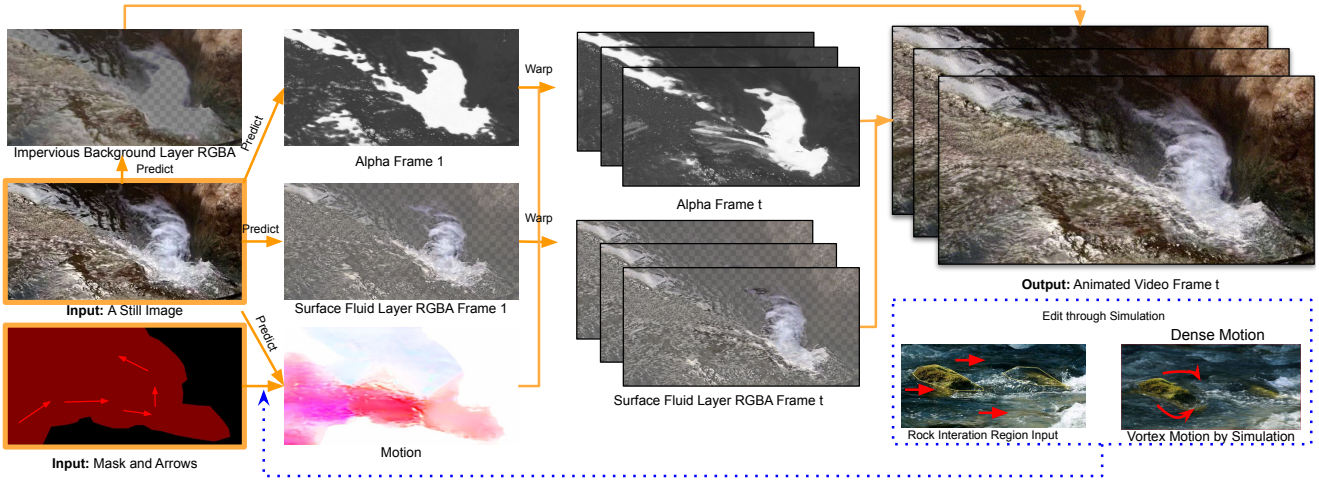

Figure 1. **Overview.** Given a still image and a coarse hint of motion as inputs, our model estimates fluid motion to generate animating videos. To be able to represent complex scenes like transparent fluid shown in the figure, we propose to learn a single background RGBA layer and per-frame surface fluid RGBA layer to compose each frame of the final animated video (solid arrows indicate the data flow). Besides, a simulation-based motion editing method (dot arrow in the figure) is introduced to generate realistic effects like fluid-rock interaction, which cannot be easily captured by the learning-based method only. The edited motion direction is represented as red arrows.

In this supplementary material, we provide: (1) detailed discussion on the implementation details; (2) the details of our newly collected CLAW dataset; (3) additional qualitative and quantitative experiments to further investigate the effectiveness of proposed designs; (4) the elaborate formula derivation for surface-only simulation. In addition, we also list a simplified framework overview at the beginning of the supplementary material, as shown in Figure 1, for better recalling the key aspects in the main paper.

A. Implementation Details

A.1. Networks for SLR

In this subsection, we provide more details of the network that predicting the Surface-based Layered Representation (SLR). Specifically, we first extent the detailed network architecture implementation along side the main paper. Then, we dive into the discussion about the challenges we will meet during optimizing the network since there is no oracle ground-truth supervision for fluid scene decomposition, as well as how we solve the problems through several losses and training strategy designs in practice.

Architecture. The network is under an encoder-decoder architecture. It consists three major components, an *Encoder* that maps the images to the corresponding background image, fluid layer features, α channel and Z channel, a *Decoder* that refines the warped features to final surface fluid layer image and the warped alpha to final alpha, and a *Translator* that refines the coarse velocity to the one with fine-grain.

For the encoder, we use 8 ResNet Blocks, which is the same as Synsin [11]. For the decoder, we use 8 ResNet Blocks and replace the convolution operator with partial convolution [6] along with mask input. The mask is vacated region of warped features or warped alpha. Partial convolution helps inpaint irregular blank areas caused by warping. For motion translator, we use U-net with 16 layers of convolution and use SPADE layer [9]instead of batch-norm. The

detailed structure can refer to Figure 2.

Optimizing the SLR As mentioned in the main paper, the training process is divided into three stages: (1) training the surface fluid branch; (2) training the background branch and (3) jointly training them together with α learning and surface fluid/ background branches finetuning.

For the first stage, we train the surface fluid branch with reconstruction losses at hand:

$$\begin{aligned}\mathcal{L}_{\text{image}} =&|I(T_i) - I_{gt}(T_i)|+\\ &\lambda_0\|\text{VGG}(I(T_i)) - \text{VGG}(I_{gt}(T_i))\|+\\ &\lambda_1 Disc(I(T_i))\end{aligned} \tag{1}$$

where $Disc$ denotes discriminative loss using a spectral normalized network updating simultaneously to distinguish the facticity of the output fluid image. The target of this stage is to enable the decoder to have the ability to memorize the texture of the fluids and hallucinate the texture in blank areas caused by warping. The background is viewed as black in this stage.

For the second stage, we train the background branch with warm-up supervision. 'Averaged image' of the whole sequence is used as guidance. The general idea behind this implementation is that for static textures in the video, averaging operation does not decline their quality. While for flowing textures, averaging will lead to blurring as well as mean static of texture, which is reasonable for under-surface liquid, as shown in the Figure 3 in the main paper. The loss could be written as:

$$\mathcal{L}_{\text{bg}} = |I_b - \frac{1}{n}\sum_i I(T_i)| \tag{2}$$

For the last stage, all the network components are trained together to update the α channels that composite the two layers, as well as to fine-tune other parts of the network. Despite the loss mentioned in the previous stages, we also make some restrictions on the predicted α that share a similar spirit with [7]. Considering $\mathbf{I}_{gt}(T_i)$ cannot be well aligned with $I(T_i)$ as motion is a pseudo ground truth, only supervised with image reconstruction loss will mislead alpha regressor to incorrectly prediction for scenes under complex motion variation. For example, splashes will always appear in the first frame then disappear in the next frame or do not exist in the first frame but appear in the next frame under a fluid collision scenario, as shown in the white splashes region in Figure 10(d). Such variation is hard to infer from previous source frames. For this reason, we constrain α with two loss terms to form $\mathcal{L}_\alpha$ as follows:

$$\begin{aligned}\mathcal{L}_\alpha =&|\left(\frac{\alpha_f(T_i)}{\alpha_f(T_i)+\alpha_b} - \alpha_{label}\right)\odot R_{M>\gamma}|+\\ &|(\alpha'_f(T_i) - \alpha_f(T_i))\odot R_{valid}|\end{aligned} \tag{3}$$

The first term is a L1 Loss under moving region $R_{M>\gamma}$ to provide a hint for alpha learning, where γ is a hyper-parameter that specifies the boundary of moving pixels and static pixels. $\odot$ is an element-wise product, α_{label} is generated with our newly labelled mask in the solid region and $\frac{\alpha_f(T_i)}{\alpha_f(T_i)+\alpha_b}$ is called the composited alpha[1] . The second term is a temporal-wise α consistency loss, which is applied between warped partial α'_f and its refinement complete result α_f under valid non-hole pixel regions in the target image . Then, the final loss terms in this stage is:

$$\mathcal{L} = \mathcal{L}_{\text{image}} + \lambda_{\text{bg}}\mathcal{L}_{\text{bg}} + \lambda_\alpha\mathcal{L}_\alpha \tag{4}$$

As described in Section 3.1 in the main paper, we train each part of the model separately and jointly train together afterwards. For the first and second stages, the learning-rate of the generator and discriminator is $5e-4$ and $2e-3$, with the loss weight of the components to be L1: 1.0, Perceptual: 10.0 and GAN: 1.0. For the final stage, with the previous trained network prior, the learning rate is $2.5e-4$ and $1e-3$, and a weighted L1 Loss with a weight 30.0 is added.

For training the translator network, we follow the training strategy in [8], but with some modifications. Specifically, we concatenate the source fluid image, initial dense motion map and the moving fluid region mask to form the input tensor and feed the tensor to the translator network. The output of the network is supervised with pseudo-ground-truth motion. The loss terms contain endpoint error, with weights 10 and GAN loss, with weights 1 . We use a learning rate of $5e-4$ and $2e-3$ to train the network and finetune with a fluid layer with a learning rate of $2.5e-4$ and $1e-3$.

Generating Labels for Alpha Values As mentioned in above, a few labels of transparency values are expected to initialize the learning of α. Thus, we re-annotate around 600 masks in the Holynski training set(note: we pick one image frame per scene). The definition of these masks is that the pixels contain solids that are overlapped with fluid regions. Then, for these masked regions, we set labels as $\alpha^{gt}_{bg} = 0.25$. For fluid regions with motion value greater than $0.1\times$ mean of motion speed, we set the labels as $\alpha^{gt}_{fluid} = 1$. The weight of this supervision gradually decays to zero during alpha training.

[1] It is abbreviated as "alpha" in the later experiment section for convenience, unless otherwise specified.

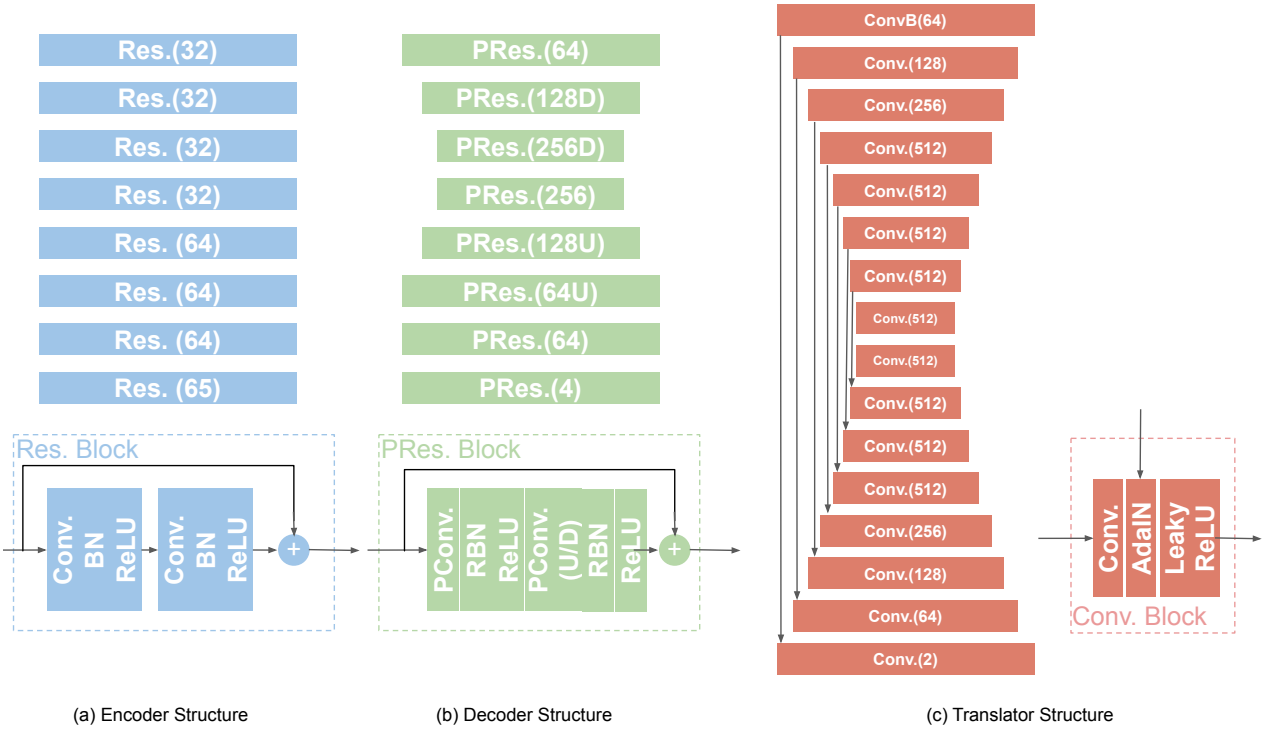

Figure 2. **Architectures.** (a) The structure of the encoder network, with 8 res-blocks. Each consists of a pair of convolutions, batch-normalization and relu layer. No upsampl or downsample is used here. The number in the bracket shows the output channels (b) The structure of the decoder network, with 8 partial res blocks. Each consists of a pair of convolutions, region-based batch-normalization(normalized on regions with positive mask), and relu layers. The upsample(U) and downsample(D) are done with strided convolutions or deconvolutions on the second unit of the res-block. (c) The structure of the translator transfers an image and guided flow map to a refined flow map. Each conv-block is built with a convolution, an adaptive instance normalization, with mean and variance acquired from guided dense flow needed to be refined. The activation is set to be Leaky-Relu to smooth the network.

B. Dataset

B.1. Holynski Dataset

The dataset proposed by [4] includes a variety of fluids under natural scenes, such as waterfalls, oceans, and fogs. We regard all the training samples (949 scenes, 5 short video clips for each scene on average) as the training set and perform testing on the validation set[2] (31 scenes). The main statistics of this dataset can be seen in Figure 3 and Figure 4 . Although diverse real-world fluids are covered in this dataset, most of the scenes are opaque.

B.2. Our Proposed CLAW Test Set

To further quantitatively evaluate our method on more complex scenes(*e.g.,* semi-/full transparent fluids) and facilitate future research of fluid animation, we collect a new test set named Complex Liquid Animation in-the-wild (CLAW) from StoryBlocks[3] with key words *fluid, waterfall, river etc.* and filter them manually to exclude unrelated videos. Videos from 122 scenes are finally provided. We follow two rules to select these videos. First, there must have transparent/semi-transparent fluid regions in the scene. Second, the motion of transparent fluid should be predicted reasonably by a pre-trained optical flow estimator, such that we can provide pseudo-ground-truth motion fields for network learning. We use flownet2 [5] in practice. The main statistic of our dataset are shown in Figure 6. Compared to the Holynski validation set, more challenging context relations with fluids are provided in the CLAW dataset, and the type of fluids and transparent regions are more balanced in CLAW. Some image samples of our test set are presented in Figure 7 .

[2]since [4] does not release ground-truth videos for their test set, we use the validation set for evaluation.

[3]https://www.storyblocks.com/

C. More Experiments

Quantitative Ablation. Table 1 shows the quantitative ablation among each training stage of our SLR model. Network input resolution is 768×768 to get higher resolution results during inference. For evaluation, We resize these 768×768 images to the size of the ground truth image(or

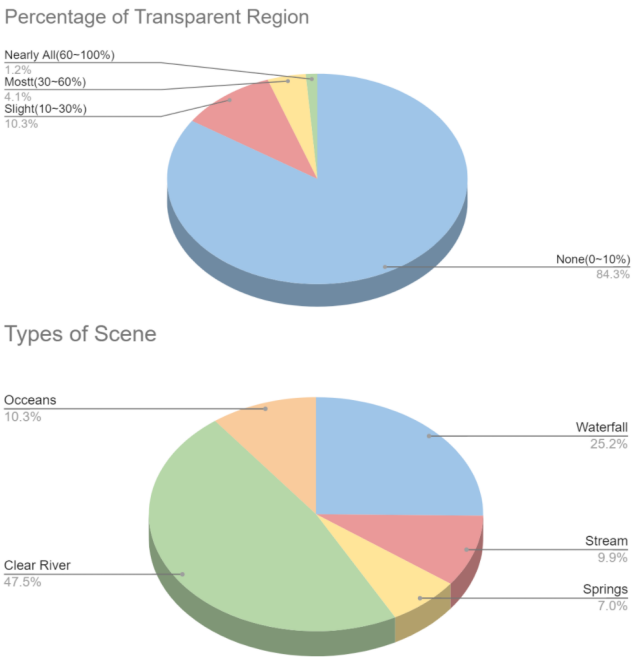

Figure 3. **Main statistics of Holynski's training set.**

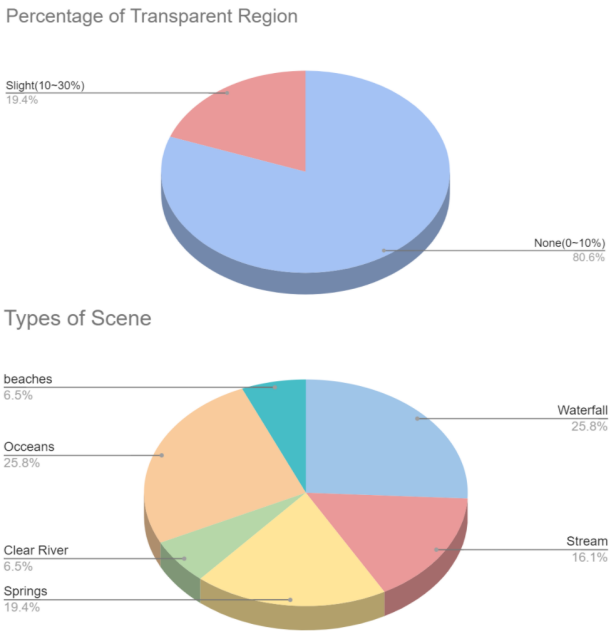

Figure 4. **Main statistics of Holynski's validation set.**

Figure 5. Evaluated scenes for components of motion calculation.

half of it in Holynski common validation set). The *Ours (stage 1)* is trained under the same setting as the model

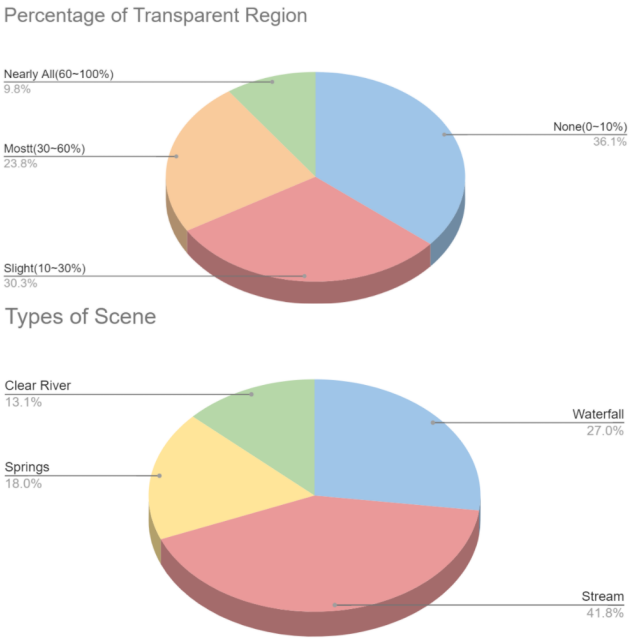

Figure 6. **Main statistics of our proposed CLAW test set.**

(*Modified Holynski(Baseline)* in Table 1 in the main paper but with lesser epochs. The prediction of *Ours (stage 2)* model is a uniform blending of the surface fluid image from the stage 1 model and background image from the background extractor trained in stage 2, telling us a naive combination will lead to blurry fluid synthesis in the non-transparent fluid region of the animating videos. The *Ours* is the final SLR model, which has comparable results with the best ones of Ours (stage 2) or Ours (stage 1) model in Holynski common validation set, and improves significantly in our CLAW testset.

Dataset	Methods	All Region			Fluid Region		
		LPIPS↓	PSNR↑	SSIM↑	LPIPS↓	PSNR↑	SSIM↑
Holynski	Ours(Stage 1)	**0.0782**	25.11	0.7772	**0.0650**	25.96	0.8026
Common	Ours(Stage 2)	0.0929	25.03	0.7612	0.0718	**26.59**	**0.8144**
Validation Set	Ours	0.0834	**25.14**	**0.7795**	0.0657	26.10	0.8030
Our	Ours(Stage 1)	0.2143	20.28	0.5926	0.2100	20.37	0.5933
CLAW	Ours(Stage 2)	0.2411	**21.09**	0.5674	0.2294	**21.06**	0.5738
Testset	Ours	**0.2040**	20.79	**0.6080**	**0.1975**	20.80	**0.6077**

Table 1. **Quantitative Abalation.** (a) Quantitative evaluation on Holynski's common validation set [4]. (b) Quantitative evaluation of CLAW test set. Unless otherwise specified, all settings are the same as Table 1 in the main paper.

Quantitative Ablation of Motion. Tab 2 shows the ablations for different motion representations in terms of end-point error metric (lower is better). Specifically, we conduct the ablation on a smaller test set, which contains representative scenarios such as collision, translucence, and different depths (as illustrated in Fig 5). The user input is restricted to one sparse annotation to justify the robustness of different motion representations under minimal input. The refine-

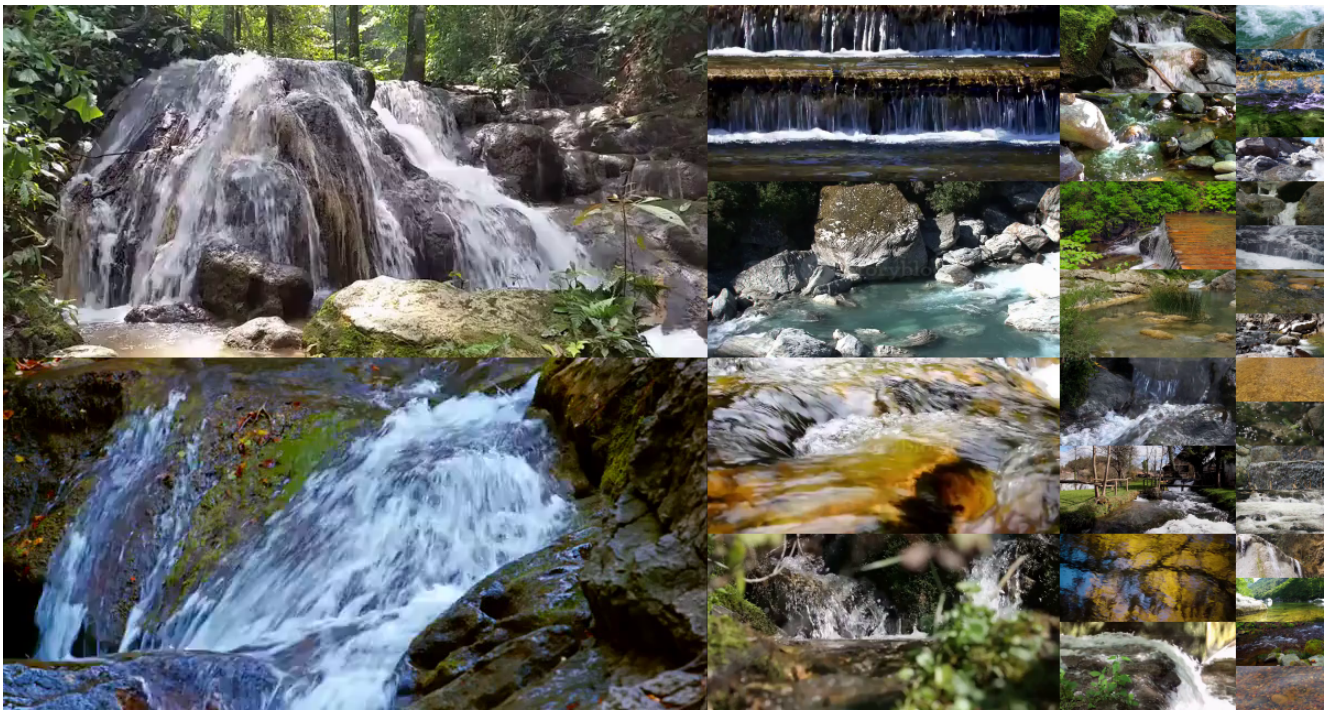

Figure 7. **Samples in our proposed CLAW test set.** We show some representative samples in our collected dataset. The CLAW covers diverse fluid scenes with different proportions of transparency, such as fluid, waterfall, river, and *etc*.

ment stage is not used in this setting [4]. The 2.5D simulation achieves the best performance on average. As for different types of scenes, in S1, differences are mainly centered on the small rock. In S2-S3, the 2.5D SFS outperforms 2D SFS and dense motion due to the ability to handle perspective distortion. In S4-S5, 2.5D SFS slightly falls behind the motion due to inconsistent depth estimation on multiple fluid surfaces.

Methods	S1	S2	S3	S4	S5	Avg.
DenseMotion1Hint	3.287	1.061	3.356	**0.959**	2.118	2.118
+2D Simulation	3.463	1.649	3.205	1.957	2.244	2.244
+2.5D Simulation	**3.184**	**0.816**	**2.749**	0.994	**1.963**	1.963

Table 2. **Quantitative Motion Ablation Study.** We use end-point error metric to evaluate the performance.

Decomposition Results. To help understand the network estimation for proposed representation in a more straightforward manner, we visualize a complex fluid case with each component predicated from our model, as shown in Figure 8. In the almost transparent fluid region (left side of dotted line in the figure), the surface fluid layer (g-h) carries more high-frequency fluid textures above rock compared with single-layer method (f), and the background layer removes most surface fluid textures compared with input image(a). alpha(i-j) is small but not zero in the region, associated with the static background and moving foreground, combines to meaningful transparency. As shown with pink arrow, the movement of rock under fluid is suppressed in our final prediction (d), while single-layer method (f) [4] animates improperly with both rock textures and fluid textures moving.

Influence of Different Losses to α Learning. As shown in Figure 10, for the learning of alpha factors, our labelled mask indicating the transparent region helps the alpha factors to converge to a semantic result. An absolute error (L1) on the output alpha with the labelled mask may extract sharp transparent factors than the squared mean (MSE). We can also find that total variance restriction strengthens the smoothness of the alpha channel to fill holes caused by warping, and warping consistency makes the alpha learning more aligned with input image textures, lessening the ghosting effects and blur in the final output.

Influence of Different Motion Hints. As shown in Figure 12, our system is not limited by the sparse velocities' quality. The reasons are two folds – First, it is not sensitive to the sparse number of velocity hints, as the sparse motion could be diffused (3rd column of the Figure) and further refined (4th column of the figure) to a reasonable dense motion field. Second, our system can rather generate diverse and realistic animations from different velocity inputs (Figure 12(a-b) and Figure 11). However, the mask quality is

[4]The refinement network is a plug-in module, that performs well at average. We exclude this module in Table 5)'s setting to concretely analyze the differences among Dense Motion, 2D simulation, and 2.5D simulation versions.

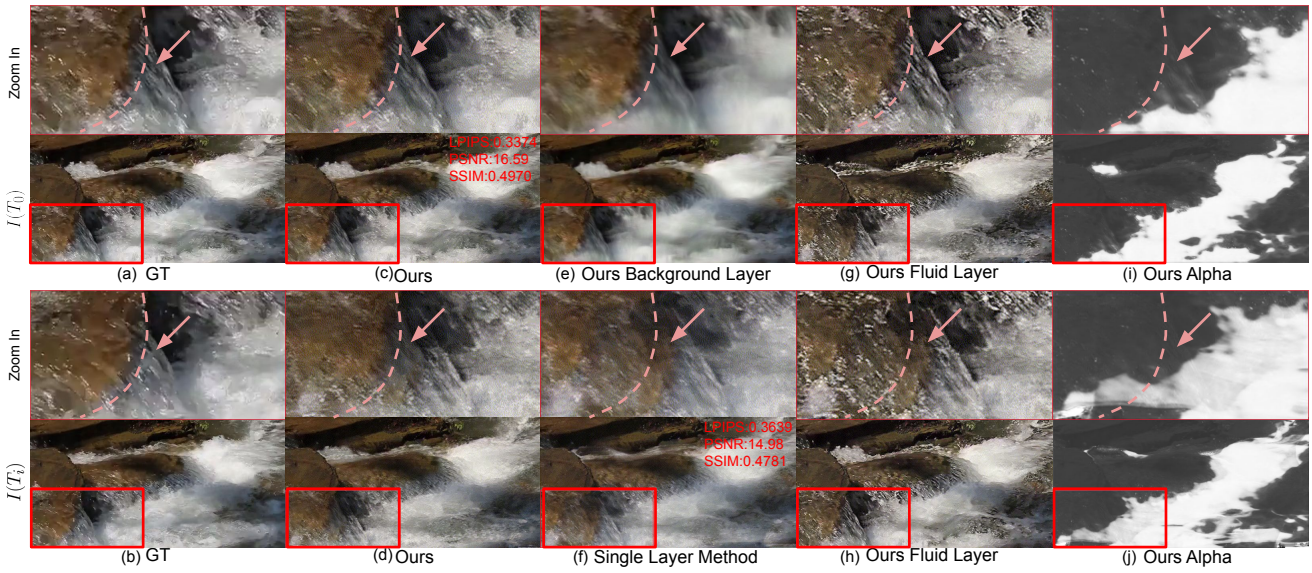

Figure 8. **Visualization of decomposition results.** The figure shows all components of our final prediction, explaining how our SLR leads to realistic fluid animation even under a complex transparent scene, with only moving fluid textures above rocks. (a-b)Ground-truth frames $I(T_0)$ and $I(T_i)$. (c-d) Ours final prediction for (T_i). (e) Background layer prediction. (f) Single-layer method [4] prediction. (i-j) Composited alpha prediction, alpha is 1 at surface fluid textures only region and is small at rock textures region. Pink arrows and dotted line point out the boundary of the transparent solid region and non-transparent fluid region. We can see the rock is moving in the single surface fluid layer (f) and (h), but keep static in Ours (d).

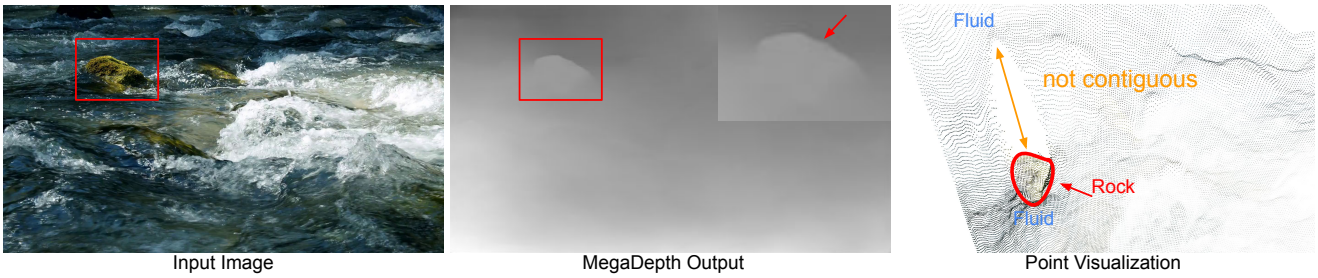

Figure 9. **Self-occlusion.** From left to right are the input image, the depth image predicted from MegaDepth, and the visualization of the corresponding fluid-solid collision scenario in point could form. Better zoom in for more details.

essential to delimit animation regions (Fig 12(b-c)). Since the dense motion is the diffusion result of sparse motion inside the mask region.

Influence of Depth Estimation. Since the 2.5D method relies highly on a depth estimation of fluid, we discuss two scenarios:

- Transparent fluids. The influence of depth error in this scenario is limited to final animated results in most cases. This is because transparent fluids are the most shallow water without visually distinct depth measurement differences, and the physics-based simulation is calculated on the fluid surface without considering the thickness.
- Self-occlusion. This exists in fluid-solid collision scenarios. Here we list a specific example for discussion. In Figure 9, part of fluid is occluded by collided rock, making real 3D fluid-solid boundary unseen in input view. However, as long as input view's depth around rock is reasonable and distinctive (as shown in the figure), the occluded fluid will be inpainted with isotropic remeshing. Thus, self-occlusion will not affect the final animation. In contrast, if depths of solid and fluid are contiguous, the 3D boundary will be shifted.

Influence of Different α to insert-object editing. In the main paper, we show a naive editing application for inserting objects. The visual quality could be further improved. One solution is that we can manually adjust the alpha blending weight of the inserted 3d object (Fig 14). The other alternative is to apply advanced shading techniques to the recovered 3D mesh for more realistic results. However, this

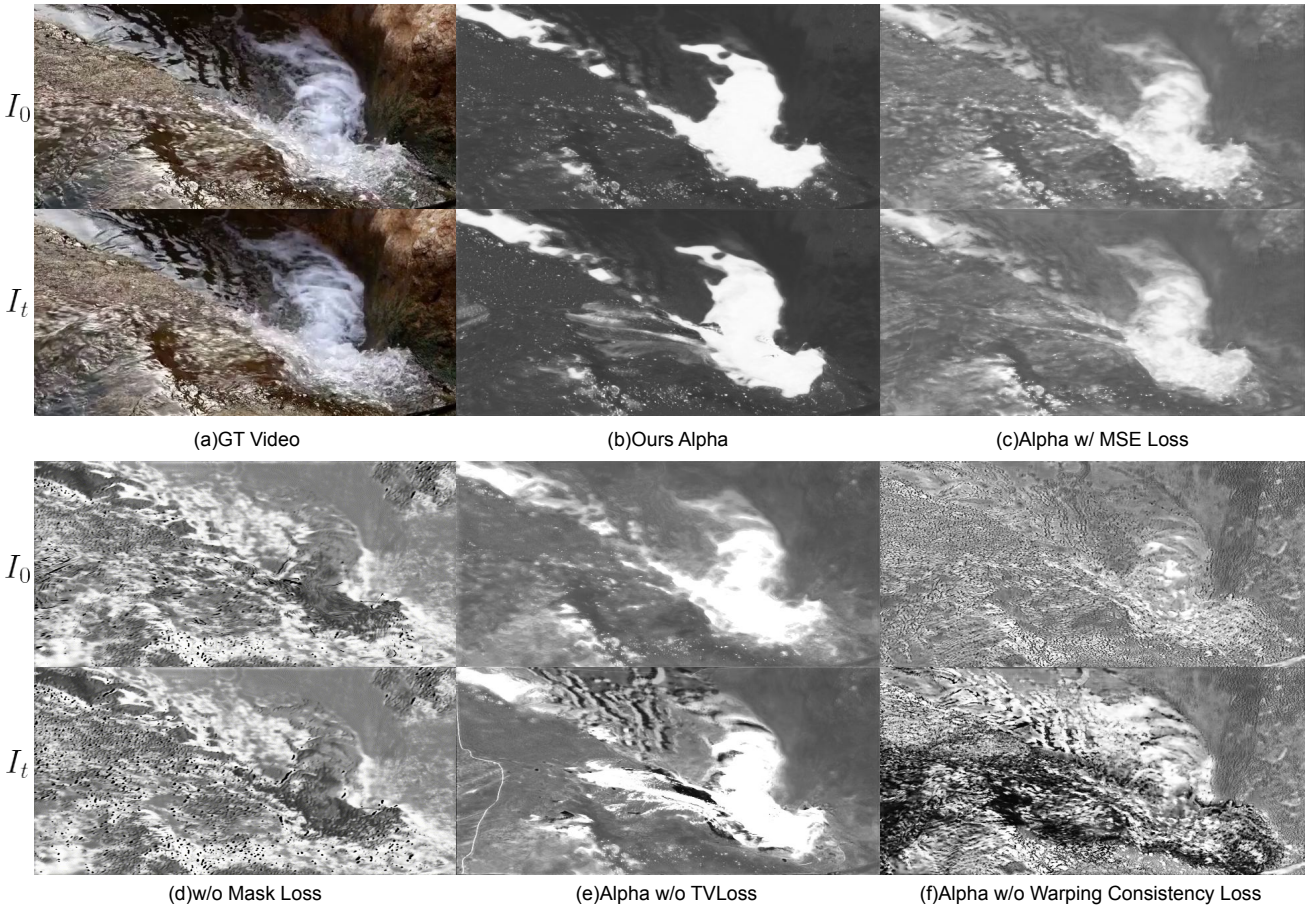

Figure 10. **Influence of Different Losses to α Learning.** From (b,c) can tell that L1 loss gives a more accurate alpha than MSE loss in our experiments. (d)Training alpha with reconstruction loss cannot predict an accurate alpha in the spray and splash region. (e)Alpha total variation loss alleviates alpha noise. (f)Training without alpha warping consistency loss cannot guarantee a temporally consistent output.

Figure 11. **Animations with different motion hints.** Our system generates different animations with realistic effects.

problem is beyond the scope of our current research.

Test on Fully Transparent Fluid Scenario. We also test our methods in dataset [10]. Note that, this dataset is originally used for evaluating undistort dynamic refractive effects, which captures images from above of a glass tank full of waved water and textured image under the tank. We test on this dataset to further probe the flexibility of our framework to scenarios with different scene scales. We did not fine-tune on this dataset. Fig 13 visualized the results of our method, where reasonable animation could be obtained. Moreover, we also find it is sufficient to use a simplified pipeline[5] to animate such in-door datasets from a single image. Since the wave patterns are relatively onefold.

Additional Qualitative Comparisons. Figure 15 shows

[5]Firstly, we generate wave-shape motion displacement (we use flownet2 in Fig 13). Then, we warp the image and inpaint holes.

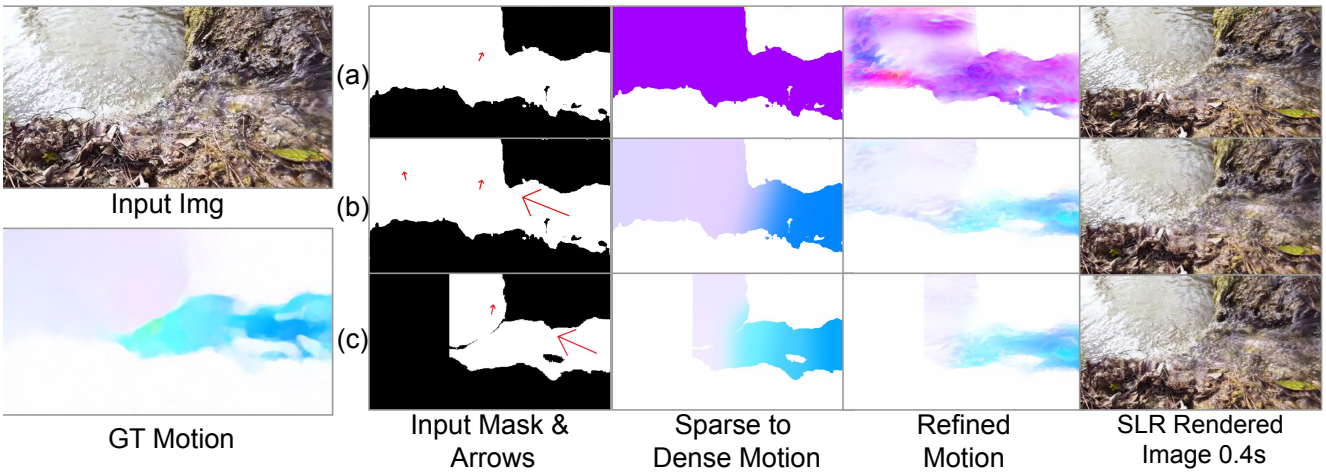

Figure 12. **Input quality**. (a) Input with GT mask and one hint. (b) Use three input hints. (c) Use an incorrect mask.

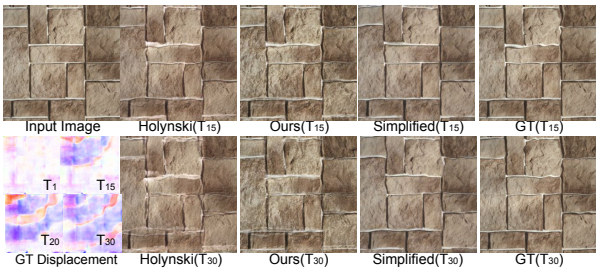

Figure 13. **Animating in the fully transparent fluid scenario.** To probe the flexibility of our framework to scenarios with different scene scale, we further evaluate our method on an in-door hand-made dataset.

more comparisons between the baseline (single-layer method) and our two-layer models. Our model successfully decouples rock textures and fluid textures above the rock. In this way, when the liquid moves after a period, the texture beneath the liquid could appropriately stay still in our representation, rather than moving with fluids in the single-layer model [4]. Moreover, we present more comparisons with the other two state-of-the-art methods [2, 3] in Figure 16. As shown in the Figure, Endo's method [2] struggles in predicting reasonable motion. For Tavi's results, although the per-image texture is visual-appealing, the temporal realism is lost in video sequences. The underlying reason is that Tavi's methods average motions from input hints rather than prediction. This design will also lead to ghosting effects (*e.g.,* rock moves underwater). Please refer to our supplementary video for better dynamic visual effect comparisons.

D. Editing Pipeline

As described in Section 5 in the main paper, one of byproducts of our work is interactive editing. We list an example in the main paper that editing the fluids with imaginary objects and changing the flow of the liquid regions.

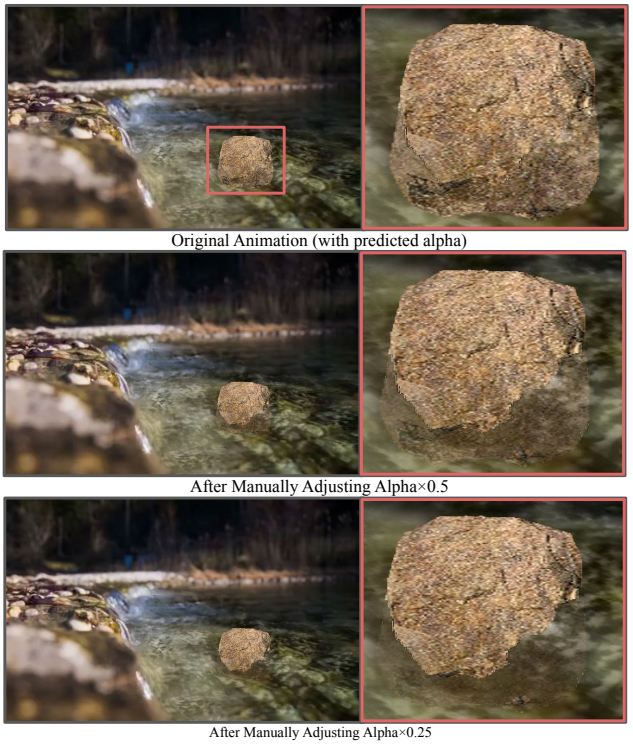

Figure 14. **3D object editing with the different alpha values of inserted 3d object.**

We detail the implementation process of such an editing application in this subsection.

As shown in Figure 4 in the main paper, the fluid mask (step 2) and monocular depth (step 4) is generated directly from the input image, and the initial dense velocity map (step 4) is generated with the user interactive sparse labelling (step 2). With the mesh (step 5) built upon the depth map and isotropic triangularization, we insert the cad model of the target object into the 3D scene at a proper position that crosses with the scene mesh(step 9). Then a Z-buffer al-

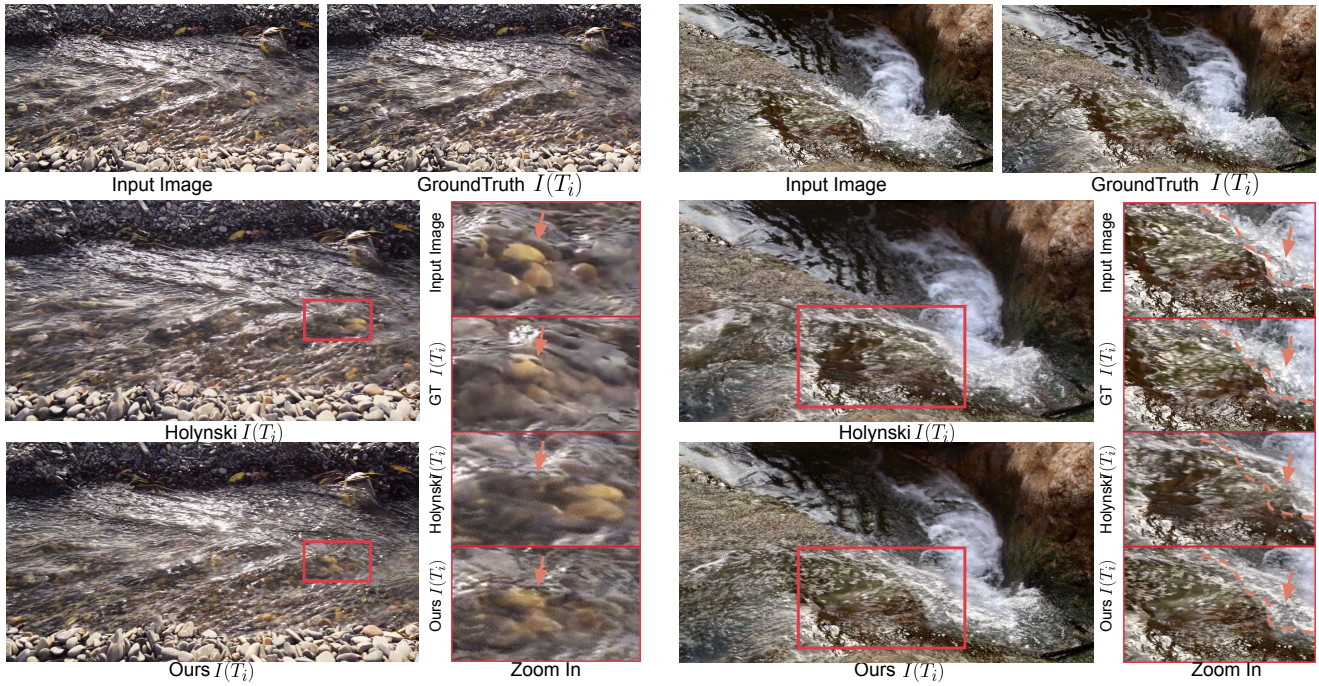

Figure 15. **Qualitative comparison with baseline method**. Two complex transparent fluid scenes are visualized. Left sample: rock under fluid is moving w/o our SLR method. Right sample: dotted line points out the boundary between the transparent solid region and the non-transparent fluid region. Two methods show comparable results in non-transparent region. While, for the transparent one, Holynski [4]'s method leads the texture of the intertidal zone to move with the fluid flow. In contrast, our method properly keeps the background still.

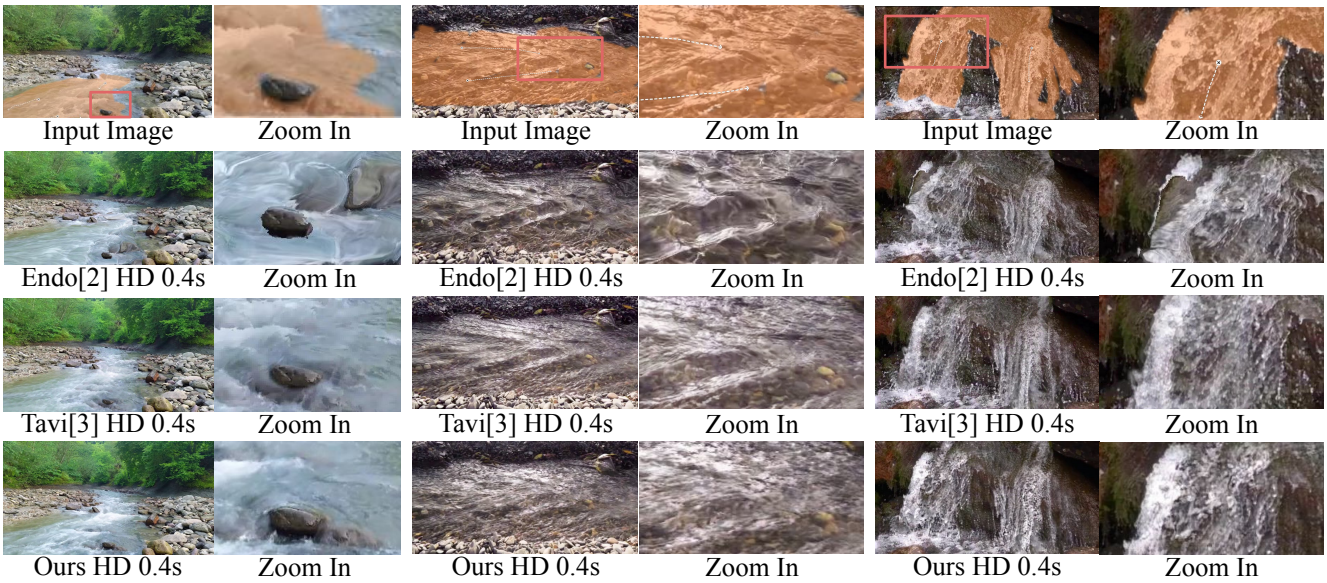

Figure 16. **More qualitative comparisons with state-of-the-art methods.**We compare our method with Endo [2], and Tavi [3] on three representative fluid scenes.

gorithm is applied to detect the occluded region of the scene mesh and the object (step 12). For velocity on mesh, we cut the mesh with the occluded area viewing as a solid boundary. The surface-only fluid simulation is performed with updated boundary conditions on the new mesh to achieve the effect of fluid colliding with solid, and the simulated velocity is refined with the pre-trained translator (step 13) to obtain the final motion fields. To obtain the final animated video, we need to render the edited scene. Specifically, we set the unoccluded region in front of the fluid layer to show the immersion effect (step 10) and the occluded one of the objects into the background layer (step 11). Then the warp-

ing is done the same as in the previous pipeline (step 14). With the help of our two-layer model and simulated velocity, we can edit the fluid image with various effects. Please refer to supplementary video for more details.

E. Formula for Surface-only Fluid Simulation

Traditional fluid simulations usually use grid-based sampler and simulate the fluids according to Euler Equation with no stickiness assumption. The overall equation is formulated as

$$\begin{aligned} \frac{\partial u}{\partial t} + u \cdot \nabla u &= -\frac{1}{\rho}\nabla p + g \\ \nabla \cdot u &= 0 \end{aligned} \tag{5}$$

where u is the fluid velocity, ρ is the density, p is the pressure inside the fluid, and g is the external force, where only gravity is considered here. The second equation presents the incompressibility properties.

The PIC method [12] can be applied to solve equation 5. This equation can be splitted by three steps: advection, apply forces and pressure projection(incompressibility) [1]. Since the key to our system is pressure projection and the other steps are not necessary, we only introduce how to solve this pressure projection equation:

$$\begin{aligned} \frac{\partial u}{\partial t} &= -\frac{1}{\rho}\nabla p \\ \nabla \cdot u &= 0 \end{aligned} \tag{6}$$

This equation can also be expressed as a Poisson's equation form, which is easier to solve . We addtionally add boundary conditions to the Poisson's equation [1]:

$$\begin{aligned} \nabla \cdot \nabla p &= \frac{\rho}{\Delta t}\nabla \cdot u \\ u \cdot n &= 0 \qquad at\ solid\ boundaries \\ p &= 0 \qquad at\ free\ surfaces \end{aligned} \tag{7}$$

where n is the normal of labelled solid and Δt is time step.

At the heart of our SFS is to solve this Poisson's equation on mesh, which will be detailed later. After solving the pressure p and its gradient ∇p, the velocity satisfying incompression can be calculatad by:

$$u^* = u - \frac{\Delta t}{\rho}\nabla p \tag{8}$$

where u is the fluid velocity corresponding to the motion map predicted by our translator network, and u^* is the fluid velocity after pressure projection. We use u^* to update our motion map.

Instead of performing previous steps on 2D or 3D grids on vertex, we perform the same procedure on a mesh layer representation, which has depth information, but only a slice of the surface is simulated. A toy example of the differences among proposed ones with traditional 2D and 3D equations is shown in Figure 17.

Specifically, in our framework, the mesh is built for the whole image with a mask region of fluids using mono-depth estimation and perspective projection. The vertices are calculated with depth as

$$\begin{aligned} x &= (u - cx)/fx \cdot d \\ y &= -(v - cy)/fy \cdot d \\ z &= d \end{aligned} \tag{9}$$

where x,y,z is the extracted vertex position, u,v is the pixel coordinate on the image, and camera parameters are set to be fx,fy,cx,cy as a perspective camera with a field of view(FOV) angle of 90 degrees in height.

With the given 3D surface and its projection information, we need to estimate the initial flow of the scene. A 2D flow field is generated with the help of the users' label, as described in Section 3.2 in the main paper. Assuming the projected velocity is consistent with input flow for each triangular surface, the equation is set to be

$$v_{\Delta abc} = \begin{bmatrix} \mu & \lambda \end{bmatrix} \begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \\ z_b - z_a & z_c - z_a \end{bmatrix} \tag{10}$$

where abc is the vertex indices for each triangular face, x, y, z is the 3D position, and v is the final velocity, the direction of the velocity must be parallel to the plane defined by Δabc, therefore a linear combination of the edges. We need to differentiate the projection equation , which is

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} u \\ v \end{bmatrix} &= \frac{d}{dt}\left(\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \end{bmatrix}\begin{bmatrix} x/z \\ y/z \end{bmatrix}\right) \\ &= \begin{bmatrix} fx & 0 \\ 0 & fy \end{bmatrix}\begin{bmatrix} 1/z & 0 & -x/z^2 \\ 0 & 1/z & -y/z^2 \end{bmatrix}\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix} \end{aligned} \tag{11}$$

where the velocity $v = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix}$ is restricted with 2 equations concerned with estimated 3D flow $\frac{du}{dt}$,$\frac{dv}{dt}$, combining with the parallel restrictions, we can derive the velocity for each triangle, with pixel velocity bilinear interpolated on the center of the projected position.

After 3D velocity is calculated, we sample 4 points on each triangular face to mimic as material points. Then we step one time interval to have the updated positions of each point. Since some of the points may move out of their initial position, we have to re-project the points back onto the fixed mesh surface. The projection is calculated as the minimal

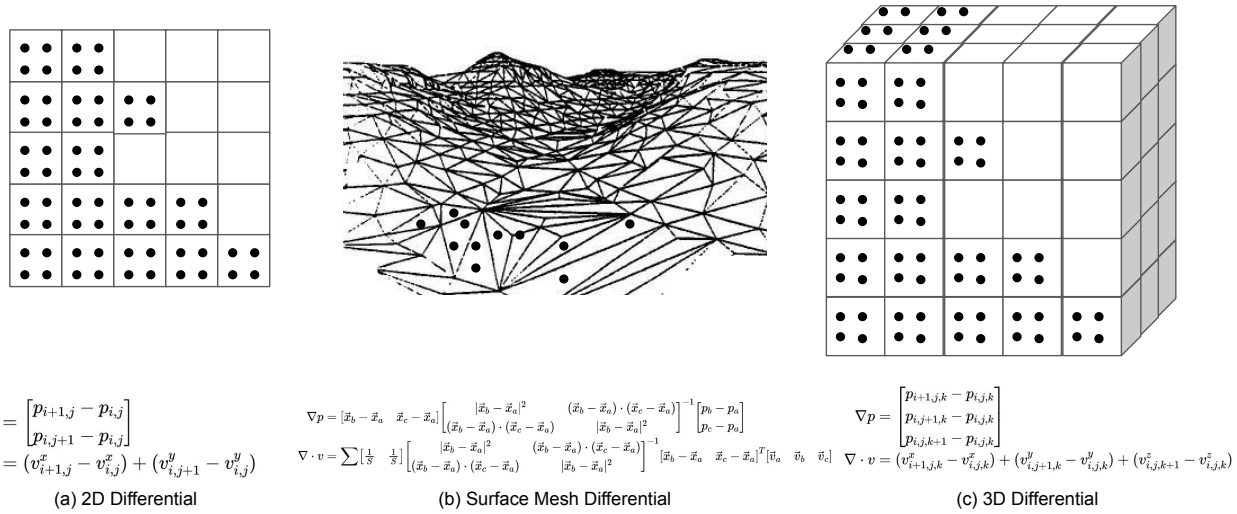

Figure 17. **A toy example of different formula settings.** Here we show the different formulas for gradient and divergence calculation in 2D/Mesh/3D. The sampled material points are regularly distributed on each unit. For 2D/3D version, the differentials are easy to be discretized as finite subtraction, while on the mesh-based surface, we need to use the vertex position to formulate a connection between the differential and the vector field. When the mesh grid is flattened on a 2D plane, we can derive the formula to be proportional.

position, which is

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ z_a & z_b & z_c \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} \tag{12}$$

and:

$$\begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \min_{\Delta abc} \min_{\sum \lambda_i = 1, \lambda_i \geq 0} \left\| \begin{bmatrix} x_n + dx \\ y_n + dy \\ z_n + dz \end{bmatrix} - \begin{bmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ z_a & z_b & z_c \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} \right\| \tag{13}$$

where we search for all Δabc to find the nearest projection in the face to the updated points, formulated by a positive homogeneous combination coefficients $\lambda_0, \lambda_1, \lambda_2$.

After the 3D locations of material points were updated, we calculated the velocity on each face as the average of each material point. Then we calculate the pressure on each vertex position as the divergence of the velocity as

$$\begin{aligned} v_{\Delta abc} &= \begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \\ z_b - z_a & z_c - z_a \end{bmatrix} \begin{bmatrix} \mu_{\Delta abc} \\ \lambda_{\Delta abc} \end{bmatrix} \\ p_a &= \text{div} v_a \\ &= \sum_{\Delta abc} \frac{\mu_{\Delta abc} + \lambda_{\Delta abc}}{S_{\Delta abc}} \end{aligned} \tag{14}$$

where we express the velocity v as the linear combination of triangle's edges and the coefficients are then added for all the triangles adjacent to a certain vertex a to calculate the divergence. $S_{\Delta abc}$ represents the area of the triangle. The formula is consistent for discrete Gauss theorem $\sum \text{div} v_a = \sum v_{\Delta abc} \cdot \vec{x}_{bc}$.

After vertex pressure is calculated, we apply the advection procedure to assure the velocity field is of no divergence, subscribing to the gradient operator of the pressure,

which is to solve the equation.

$$\nabla p_{abc} = \begin{bmatrix} \mu_p & \lambda_p \end{bmatrix} \begin{bmatrix} x_b - x_a & x_c - x_a \\ y_b - y_a & y_c - y_a \\ z_b - z_a & z_c - z_a \end{bmatrix}$$
$$\begin{bmatrix} p_a \\ p_b \\ p_c \end{bmatrix} = \nabla p_{abc} \cdot \begin{bmatrix} x_a & x_b & x_c \\ y_a & y_b & y_c \\ z_a & z_b & z_c \end{bmatrix} + \vec{c_p} \tag{15}$$

The pressure value on each vertex is estimated as a linear function of their position, with linear coefficients parallel to the triangle face, we can solve the parameter μ_p, λ_p to get the gradient value for that face, then the velocity is subscribed with this velocity as what we do in 2D projection.

F. RunTime

For a fair comparison, we list *Modified Holynski*, which has the same backbone network as our model, as the reference baseline. The tests are under a video length of 60 frames and run on a V100. Runtime/GFLOPS of the baseline are $5.3s$/1.63, and ours are $7.5s$/3.52. The 2.5D simulation takes additional 3.5s. Our system designs increase limited cost, while ensuring plausible performance and flexible downstream applications.

G. Limitation

Although our method could handle challenging scenarios (such as transparent fluids) and enable animating still images in physically plausible fluid behavior, there are several improvements that could be further explored in the future. (1) For splatting and inpainting, visual artifacts are non-negligible when the motion speed is too fast. Since it would raise too large holes to be inpainted. Any improvement in the splatting method or simply replacing the inpainting network with a larger receptive field design would ease the issue. (2) For SLR, alpha prediction sometimes might not be accurate enough due to imbalanced data distribution over transparent fluid scenes, which will result in frozen fluid textures in some cases. In these cases, a coarse mask of the non-transparent regions is required to reach the best visual effects. (3) For SFS, The 2.5D method requires at least a roughly accurate depth estimation of fluid, while depth estimation is sometimes inaccurate enough. We have included the detailed disscusion in Sec C. (4) The input masks might be inconvenient for users to draw when the scene context is sophisticated. To avoid the need for manual input, one possible way is to apply a segmentation model such as SAM with fine-tuning on real/synthesized fluid data, to segment fluid/rock regions to replace the mask input. Please refer to the project page to see how these issues affect the final animation results.

References

[1] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015. 10

[2] 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. 8, 9

[3] 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. 8, 9

[4] Aleksander Holynski, Brian L Curless, Steven M Seitz, and Richard Szeliski. Animating pictures with eulerian motion fields. In *CVPR*, 2021. 3, 4, 5, 6, 8, 9

[5] 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. 3

[6] 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. 1

[7] 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. 2

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

[9] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019. 1

[10] Simron Thapa, Nianyi Li, and Jinwei Ye. Dynamic fluid surface reconstruction using deep neural network. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 7

[11] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *CVPR*, 2020. 1

[12] Yongning Zhu and Robert Bridson. Animating sand as a fluid. *ACM TOG*, 24:965–972, 2005. 10