

# Learning an Implicit Physics Model for Image-based Fluid Simulation

## Supplementary Material

### 1. Implementation Details

Our training process consists of two stages: first, the physics-informed neural dynamics model is trained for 120k iterations, followed by the animation module for 250k iterations. Both stages use the Adam optimizer with a  $1e-4$  learning rate and  $(0, 0.9)$  for betas. Training is performed on Nvidia RTX 6000 Ada GPUs with a batch size of 1. Weight decay is applied every 50k iterations with a decay rate of 0.5.

As illustrated in the main paper, depth maps of the input image and fluid-area masks are also needed. We use [7] for depth prediction. Fluid area masks are manually specified by users.

For quantitative and qualitative comparisons, we follow this full training setup. However, in the first ablation study, both stages are trained for only 25k iterations on a training subset that is  $10\times$  smaller than the original dataset. Evaluations are still conducted on the full validation set. In the second ablation study, all the alternatives are trained for 120k iterations, as in the full training setup.

### 2. User Hints

Our network can be extended to incorporate sparse flow hints from users, as shown in Figure 2. These hints provide valuable information, such as the general direction of fluid flow and help to reduce ambiguity in velocity prediction (e.g., a river could flow both leftward or rightward). Therefore, we append them in our evaluations for all the methods.

In the user hint setting, we first convert the sparse hints into a dense hint map, which is then appended to the inputs of our physics-informed neural dynamics.

### 3. Model Architecture

Our model consists of two parts: physics-informed neural dynamics and the animation part. The animation part contains an inpainter to inpaint each LDI, an encoder to extract features for each LDI, and a decoder to decode the feature maps rendered by 3D Gaussians. Similar to 3D-Cinematography [4], we use the inpainter from [8]. The feature encoder mostly follows the architecture of ResNet34, while the decoder adopts the U-Net architecture.

For our physics-informed neural dynamics, the architecture will be detailed in the following sections. The model comprises three components: a convolutional network that extracts features from the inputs, an MLP that processes the 4D coordinates along with the extracted features to predict

the velocity of the points, and an additional network that predicts the external forces for each image. We refer to these three parts as Feature Network, Velocity Network, and External Force Network, respectively.

#### 3.1. Feature Network

Our feature network processes the source image, monocular depth map, and fluid area mask to generate a feature map. In the scenarios involving user interaction, the network also incorporates a user hint mask as an additional input. Figure 1 illustrates the architecture of our feature network, which is based on the design of SPADE [6]. We find the residual connection blocks in SPADE very useful as they allow the network to retain input information while processing it incrementally. In the encoder section, shown at the top of Figure 1, we use Leaky ReLU activation with a negative slope of 0.2. We set the kernel size to 4 and the stride to 2 for all the convolution layers in the encoder. For the decoder section, shown at the bottom of the figure, ReLU activation is applied. We set the kernel size to 3 and the stride to 1 for convolution layers in the decoder section. We set padding to 1 for all the convolution layers in the feature network. Additionally, we downsample all inputs to a resolution of  $(512, 512)$  for faster processing.

#### 3.2. Velocity Network

Our velocity network is composed of five fully connected layers with ReLU activation applied after each layer except the final one. The dimensions of the intermediate layers are sequentially set to 2048, 1024, 256, and 64. To prevent the prediction of overly smooth velocity fields, we apply positional embedding [5] to the input 4D coordinates  $(x, y, z, t)$ . To ensure all coordinates fall within the range  $[-1, 1]$ , we first transform the 3D spatial locations  $(x, y, z)$  using the following equations:

$$(x^p, y^p, d) = Project((x, y, z), P), \quad (1)$$

where  $Project(\cdot)$  indicates projecting the world coordinates to the camera plane and  $P$  is the projection matrix of the input view. We then re-scale and shift  $(x^p, y^p, d)$  by:

$$x' = \frac{2x^p}{W - 1} - 1, \quad (2)$$

$$y' = \frac{2y^p}{H - 1} - 1, \quad (3)$$

$$z' = \frac{2}{\max(d, 1)} - 1, \quad (4)$$

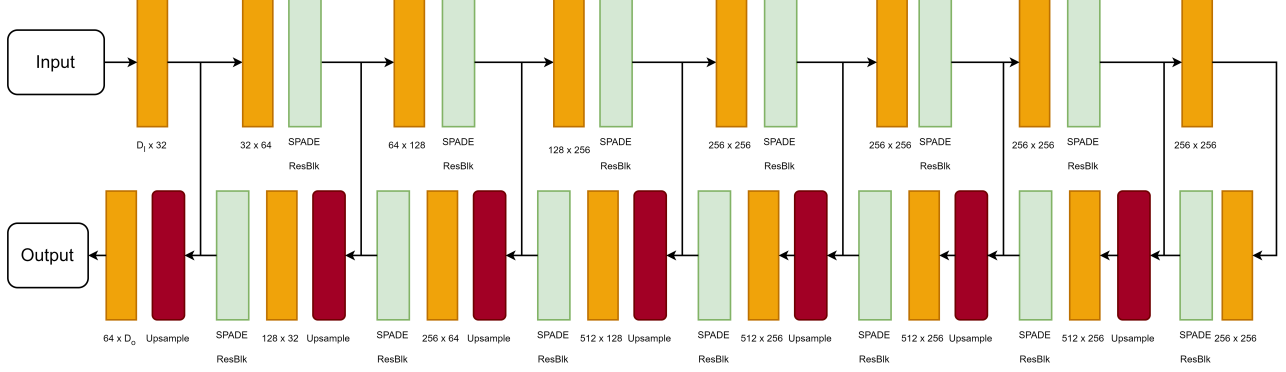


Figure 1. **Detailed Architecture of our feature network.** Our feature network is designed based on SPADE [6]. The encoder is shown at the top of the figure, while the decoder is displayed at the bottom. Leaky ReLU activation is applied after each convolution layer in the encoder (indicated by orange rectangles), whereas ReLU activation is used after each convolution layer in the decoder.



Figure 2. An example of user hints.

where  $H$  and  $W$  are the height and width of the input image. For the time  $t$  in the 4D coordinates, we transform it by:

$$t' = \frac{t}{T}, \quad (5)$$

where  $T$  is the total seconds of the output video.

The resulting transformed coordinates  $(x', y', z', t)$  are then processed with positional embedding and passed to the Velocity Network.

### 3.3. External Force Network

Using the feature maps produced by our feature network, we utilize an additional convolutional network to predict external forces for each image. This external force network comprises 7 convolution layers and 4 fully connected layers. The feature dimensions of the convolutional layers are 64, 32, 32, 32, 32, 32, and 32, respectively, while the fully connected layers have feature dimensions of 1024, 256, and 64. The convolution layers include residual connections, with BatchNorm applied to each layer. ReLU activation is used for all layers throughout the network. We set the kernel size to 3 and the stride to 2 for all convolution layers.

## 4. Additional Results

### 4.1. Video Results

We include video results generated by our methods and baseline methods in the supplementary materials. These results comprise: (a) video outputs from the Holynski *et al.* [3] validation set, showcasing both original and novel viewpoints, presented alongside comparisons with baseline methods; (b) video results from edited images, compared with 3D-Cinemagraphy [4]; and (c) further comparisons with Stable Video Diffusion(SVD)-XT [2] and CogVideoX-5B [9] on the Holynski *et al.* [3] validation set.

### 4.2. Comparison with diffusion-based methods

We include several videos generated by SVD-XT [2] and CogVideoX-5B [9] using images from the Holynski *et al.* [3] validation set. As demonstrated in the video results, diffusion-based methods struggle with complex fluid motion, leading to unreasonable fluid motions and noticeable jittering artifacts in animations.

### 4.3. Image Editing

An image editing example is included in the supplementary material. In this example, we add a stone to the seashore, in which we expect the stone to block and separate the waves. As shown in the video results, our method accurately simulates the intended behavior, with the waves splitting around the stone. In contrast, 3D-Cinemagraphy [4] fails to account for the occlusion, resulting in waves unrealistically passing beneath the stone. Additionally, the results from 3D-Cinemagraphy exhibit other noticeable artifacts, such as holes and dot patterns.

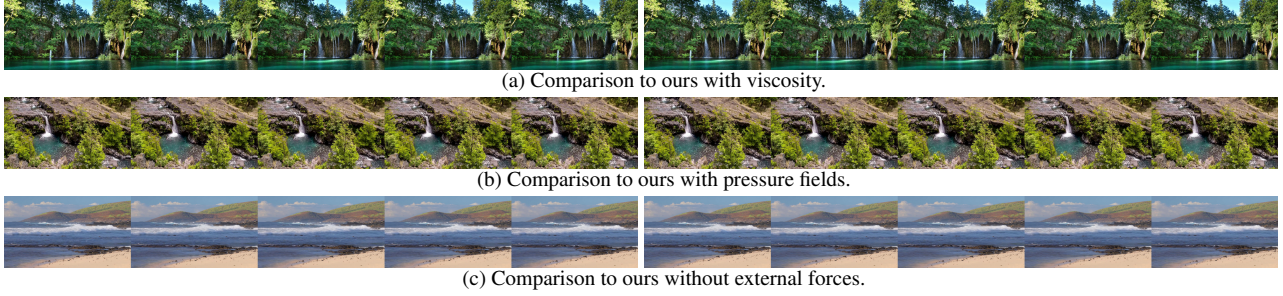


Figure 3. Qualitative ablations for Navier-Stokes equation simplification

	Method	SA $\uparrow$	PC $\uparrow$	Time(s) $\downarrow$
Input	3D-C	0.8742	0.6936	91
	Holynski <i>et al.</i>	<b>0.8900</b>	<b>0.7171</b>	16
	Ours	0.8739	0.7110	<b>13</b>
Novel	3D-C	0.8638	0.6622	90
	Make-it-4D	<b>0.8702</b>	0.6753	<b>24</b>
	Ours	0.8618	<b>0.6814</b>	29

Table 1. VideoPhy scores and the computation cost for all methods.

#### 4.4. Qualitative ablations for Navier-Stokes equation simplification

In our ablation study, we present a quantitative analysis supporting our choice of physics guidance. Here, we further provide qualitative ablations in Figure 3 to illustrate the visual artifacts produced by alternative designs. Corresponding video results are also available on our project website.

#### 4.5. VideoPhy Scores

Recent studies have introduced metrics to directly evaluate how closely generated videos adhere to physical common sense. In Table 1, we report results using the VideoPhy metric [1]. As shown in Table 1, the scores across all methods are very similar, suggesting that the current VideoPhy metric may lack the sensitivity to capture fine-grained physical inconsistencies in generated videos.

#### 4.6. Computation Cost

Computation cost is also an important consideration for downstream applications. We report the generation time for a 60-frame 1280×720 video on an H100 GPU in Table 1.

### 5. Limitations

Our method focuses on natural fluids and may be less effective in some scenarios. In particular, the lack of pressure fields limits its ability to capture interactions such as river merging. Static fluid masks also pose challenges for scenes with evolving boundaries. Novel-view videos may contain artifacts due to depth prediction inaccuracies. In the future,

we will try to eliminate the N-S simplifications and extend it to compressible fluids, by collecting data with pressure, viscosity, and density measurements. We will add this to the paper.

### References

- [1] Hritik Bansal, Zongyu Lin, Tianyi Xie, Zeshun Zong, Michal Yarom, Yonatan Bitton, Chenfanfu Jiang, Yizhou Sun, Kai-Wei Chang, and Aditya Grover. Videophy: Evaluating physical commonsense for video generation. *arXiv preprint arXiv:2406.03520*, 2024. 3
- [2] Andreas Blattmann, Tim Dockhorn, Sumith Kulal, Daniel Mendelevitch, Maciej Kilian, Dominik Lorenz, Yam Levi, Zion English, Vikram Voleti, Adam Letts, et al. Stable video diffusion: Scaling latent video diffusion models to large datasets. *arXiv preprint arXiv:2311.15127*, 2023. 2
- [3] Aleksander Holynski, Brian L Curless, Steven M Seitz, and Richard Szeliski. Animating pictures with eulerian motion fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5810–5819, 2021. 2
- [4] Xingyi Li, Zhiguo Cao, Huiqiang Sun, Jianming Zhang, Ke Xian, and Guosheng Lin. 3d cinematography from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4595–4605, 2023. 1, 2
- [5] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1
- [6] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019. 1, 2
- [7] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12179–12188, 2021. 1
- [8] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3d photography using context-aware layered depth inpainting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1

- [9] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024. [2](#)