Learning Meaningful Controls for Fluids

MENGYU CHU, Max Planck Institute for Informatics, SIC , Germany
NILS THUEREY, Technical University of Munich, Germany
HANS-PETER SEIDEL, Max Planck Institute for Informatics, SIC, Germany
CHRISTIAN THEOBALT, Max Planck Institute for Informatics, SIC, Germany
RHALEB ZAYER, Max Planck Institute for Informatics, SIC, Germany

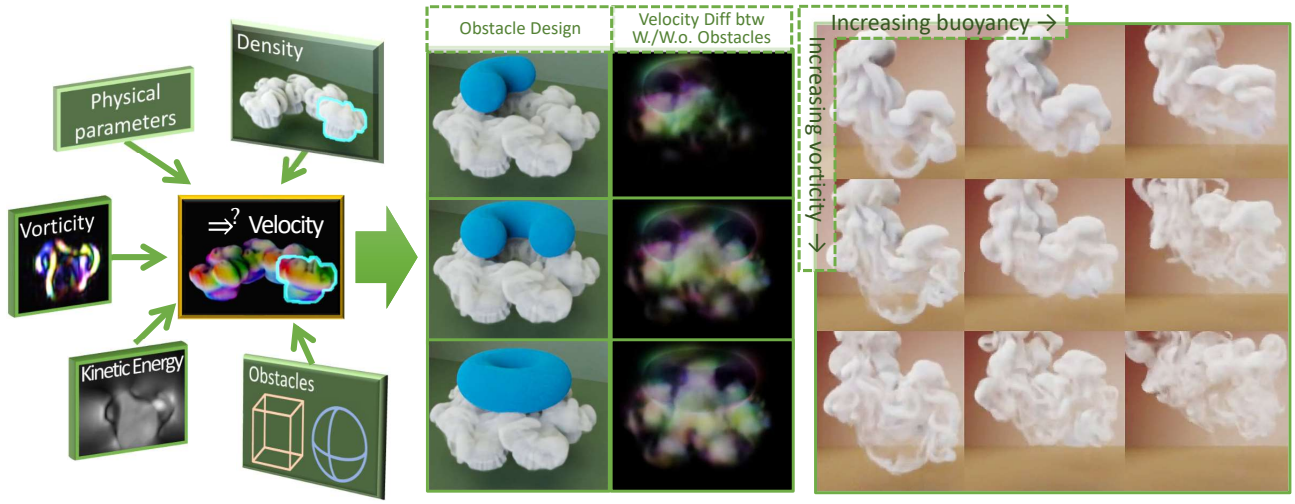


Fig. 1. Learning to infer velocity from only a density input while taking into account obstacles and other physical quantities, we present a novel method for generating fluid flows. It allows for flexible modifications via semantic controls as well as traditional physical parameters. The conditional generation of velocity takes less than 1 second for a 3D volume with a resolution of 256^3. Using our adversarial training with cyclic mapping, we achieve results that change sensitively with user modifications.

While modern fluid simulation methods achieve high-quality simulation results, it is still a big challenge to interpret and control motion from visual quantities, such as the advected marker density. These visual quantities play an important role in user interactions: Being familiar and meaningful to humans, these quantities have a strong correlation with the underlying motion. We propose a novel data-driven conditional adversarial model that solves the challenging and theoretically ill-posed problem of deriving plausible velocity fields from a single frame of a density field. Besides density modifications, our generative model is the first to enable the control of the results using all of the following control modalities: obstacles, physical parameters, kinetic energy, and vorticity. Our method is based on a new conditional generative adversarial neural network that explicitly embeds physical quantities into the learned latent space, and a new cyclic adversarial network design for control disentanglement. We show the high quality and versatile controllability of our results for density-based inference, realistic obstacle interaction, and sensitive responses to modifications of physical parameters, kinetic energy, and vorticity. Code, models, and results can be found at https://github.com/RachelCmy/den2vel.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Physical simulation**.

Additional Key Words and Phrases: Generative adversarial network, Fluid Simulation, User Interaction.

ACM Reference Format:
Mengyu Chu, Nils Thuerey, Hans-Peter Seidel, Christian Theobalt, and Rhaleb Zayer. 2021. Learning Meaningful Controls for Fluids. *ACM Trans. Graph.* 40, 4, Article 100 (August 2021), 18 pages. https://doi.org/10.1145/3450626.3459845

1 INTRODUCTION

The design of art-directable fluid simulations [Shi and Yu 2005; Kim et al. 2008; Nielsen and Bridson 2011] remains a highly challenging task. Despite growing hardware performance and substantial algorithmic improvements, achieving a desired outcomes with a physical simulator often requires a tedious, iterative trial-and-error process. Therefore, there is a strong need for user-friendly tools that allow for intuitive manipulations in fluid simulation workflows. Visual effects typically aim to achieve results defined by visible entities that have obvious semantic meanings for humans, such as

Authors' addresses: Mengyu Chu, mchu@mpi-inf.mpg.de, Max Planck Institute for Informatics, SIC, Germany; Nils Thuerey, nils.thuerey@tum.de, Technical University of Munich, Department of Computer Science, Munich, Germany; Hans-Peter Seidel, hpseidel@mpi-sb.mpg.de, Max Planck Institute for Informatics, SIC, Germany; Christian Theobalt, theobalt@mpi-inf.mpg.de, Max Planck Institute for Informatics, SIC, Germany; Rhaleb Zayer, rzayer@mpi-inf.mpg.de, Max Planck Institute for Informatics, SIC, Germany.

0730-0301/2021/8-ART100
https://doi.org/10.1145/3450626.3459845

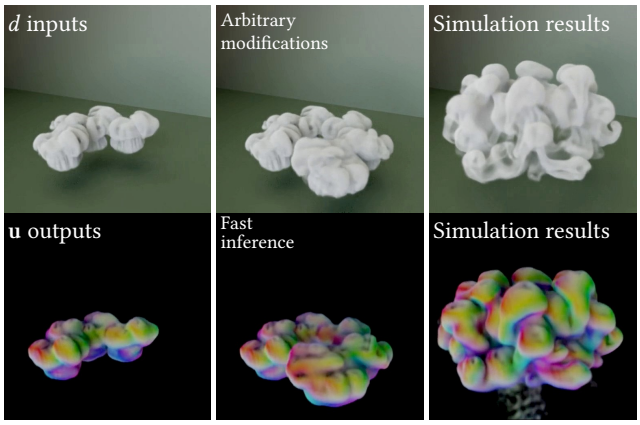


Fig. 2. Learning the relationship between density and velocity, our method generates natural plume simulations from density inputs, allowing for *what-you-see-is-what-you-get* modifications of the density field.

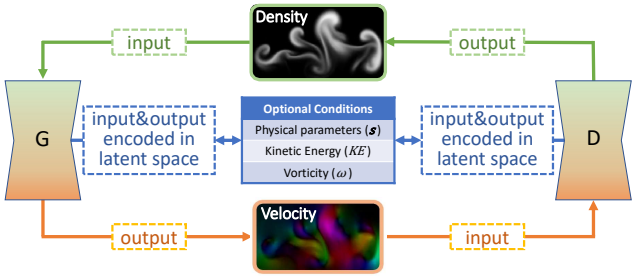


Fig. 3. In our GAN training, we employ a generator G to translates densities (in green) into velocities (in orange). In contrast to classical discriminators that are classifier-based, we propose a discriminator that checks velocities by inferring densities. Important physical quantities are further encoded in the latent spaces (in blue) of the generator and discriminator. In this way, cyclic links are formed between them and we are able to supervise the generator to fulfill quantitative conditions explicitly. Through this cyclic adversarial training, sensitive quantitative control is achieved.

the shape of a smoke cloud or the "swirliness" of its motion. Thus, it is advantageous for artists to have a workflow with intuitive knobs and controls to work with these semantic entities, instead of having to tune potentially abstruse physical parameters.

Existing methods rely on established physical models, e.g., the Navier-Stokes equations to drive visible quantities, such as a smoke density, with the motion induced by surrounding fluid. In these works, the density is primarily considered as passively transported quantity. Granted little influence over the underlying physics, it is nigh impossible for users to realize their mental image by direct modifications of the density fields. However, there is a strong correlation between the underlying motion and spatial configurations of the advected marker density in a fluid simulation. This poses an interesting challenge: can we obtain a realistic motion only from a given density configuration? While the problem is highly ill-posed in the context of traditional physical models, it is an excellent setting for data-driven methods.

We propose a novel deep-learning-based algorithm to encode the physical relationship between a visual quantity, such as the smoke density, and realistic motions as specified by a training dataset. We show that we can train a conditional generative model to infer velocity fields based on single-frame density inputs. In this way, a user can generate and modify fluid simulations directly through arbitrary density modifications. A use case is demonstrated in Fig. 2, where simple density modifications, e.g., moving, rotation, and duplication of the densities in a scene directly yield complex and realistic changes in terms of the velocity.

Given such a density-based velocity inference, a fluid simulation step can be performed by applying it in conjunction with an advection step. We make use of this approach to present a simulation method by alternatively applying the velocity-prediction and density-advection operations in a loop. In contrast to classical simulation methods, which focus mostly on velocity, the proposed simulation method is density-conditioned and likewise yields natural fluid motion over time. In addition, our model is inherently differentiable and thus lends itself to inverse problems, e.g., fluid control via user-defined keyframes [McNamara et al. 2004] or image-based style transfer tasks [Kim et al. 2019a].

Our approach is based on Generative Adversarial Networks (GANs) [Goodfellow et al. 2014], which were shown to be powerful tools to learn high-dimensional data distributions, e.g., for natural images [Isola et al. 2017; Karras et al. 2020]. However, despite their success, these algorithms have fundamental shortcomings in the presence of multi-model data and often *mode-collapse* to a simple subset of the desired outputs [Srivastava et al. 2017; Salimans et al. 2016]. To enable plausible control and mitigate mode collapse, we propose to deeply embed physical quantities in the learned latent-spaces of a GAN, in addition to a cyclic adversarial network design. As we will demonstrate below, this prevents the coupled non-linear minimization that needs to be solved during training from converging to undesirable states that largely ignore inputs, as, e.g., exhibited by existing conditional GAN algorithms [Mirza and Osindero 2014; Isola et al. 2017; Marzouk et al. 2019]. Our approach, as visualized in Fig. 3, makes it possible to train networks that learn to synthesize large spaces of flow behavior while at the same time being responsive to changes of a conditioning on semantically meaningful parameters, including buoyancy, kinetic energy, vorticity, and boundary conditions. An example of the flexible modifications supported by our networks is shown in Fig. 4.

To summarize, our work makes the following contributions:

- a novel fluid generation method for velocity inference from single-frame density snapshots,
- a unified method simultaneously enabling multiple controls including obstacles, physical parameters, kinetic energy and vorticity fields,
- adversarial learning via cyclic mappings for sensitive interactions alongside conditional modifications,
- and a flexible differentiable simulation model with support for a wide range of forward and inverse problems.

With these contributions, we arrive at a fluid generation method with intuitive semantic controls and natural simulation results, as we will demonstrate with a wide range of examples below.

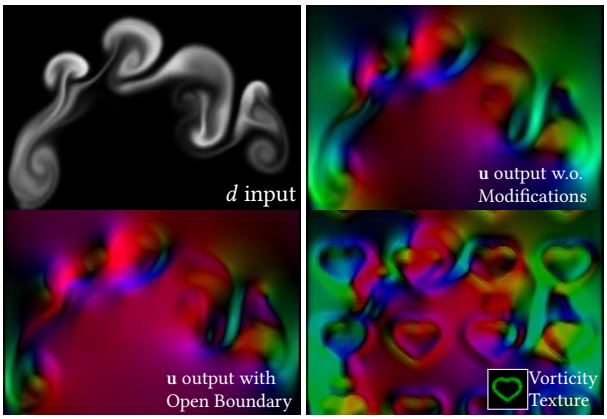


Fig. 4. Conditioned on density (d) and optional physical quantities including parameters, kinetic energy and vorticity, our model offers flexible manipulations. The first row shows the velocity(**u**) generated from the single density input. With an open boundary, the velocity changes accordingly (bottom left). The bottom right shows the result when vorticity is modified with a heart-shape texture.

2 RELATED WORK

Fluid Simulation Methods in computer graphics have a long history. Methods that solve the Navier-Stokes equations on Eulerian grids [Stam 1999], particle systems [Monaghan 1992], or hybrid systems [Zhu and Bridson 2005; Stomakhin et al. 2013] are popular choices. We refer readers to Bridson[2015], Koschier et al.[2019], and Jiang et al.[2016] for overviews of these methods respectively. While these simulation algorithms yield impressive and realistic results for a wide range of phenomena, they typically need to be extended to address the demands of artistic manipulation.

Aiming to support flexible fluid manipulations for users, a variety of algorithms have been proposed in the direction of fluid control and guiding. E.g., many fluid control methods focus on matching desired density distributions at keyframes [McNamara et al. 2004; Pan et al. 2013; Pan and Manocha 2017]. Based on optimization, these methods are usually computationally expensive. Furthermore, designing a set of keyframes allowing realistic transitions requires extensive field expertise. There are also fluid guiding methods that generate high-resolution velocity fields following sparse skeletal flows [Thürey et al. 2009; Inglis et al. 2017; Forootaninia and Narain 2020]. These methods constrain velocity outputs with desired coarse velocity samples. Providing a different perspective, our work focuses on the relationship between velocity and a variety of physical quantities including density, obstacles, kinetic energy, vorticity. More specifically, our aim is not to match prescribed shapes or embody geometric forms, but to understand semantic meaning of visible quantities in order to influence velocity in a physically-plausible manner.

Detail synthesis and stylization are active directions in fluid editing. Based on a low-resolution simulation, many techniques synthesize details from textures [Ma et al. 2009; Jamriška et al. 2015], numerical procedures [Kim et al. 2008], and high-resolution simulations [Chu and Thuerey 2017; Sato et al. 2018]. Browning et al.[2014] animate stylized keyframes on a given fluid simulation. Kim et al. [2019a] transfer stylized features from natural images to smoke simulations with the help of neural style transfer techniques. As a simulation method, our work supports editing via density and vorticity. We generate realistic velocity results for artificial density inputs, e.g., 2D drawings and 3D meshes, as well as texture-based vorticity modifications.

Machine learning algorithms are also proposed for fluid animation purpose. Some of them [Tompson et al. 2017; Sanchez-Gonzalez et al. 2020] approximate the complex physical solutions with neural networks to reduce computational cost. Reinforcement learning [Ma et al. 2018] and differentiable fluid solvers [Holl et al. 2020; Hu et al. 2019] are being explored for fluid control.Generative models were proposed for velocity synthesis from a set of reduced physical parameters [Kim et al. 2019b], temporal prediction [Wiewel et al. 2019], liquid surface reconstruction from refracted images [Thapa et al. 2020], and splash modeling by user sketches [Yan et al. 2020]. Deep generative models show strong potential in revealing relationships between datasets. Related to our work, generative adversarial models have been explored for super-resolution [Xie et al. 2018]. In contrast, our models do not receive any velocities as input. They represent a unified system that considers influences from multiple physical quantities, providing sensitive and flexible controls for velocity generation.

Deep Learning Algorithms have opened up new possibilities for intuitive user manipulations in image and video applications. E.g., learning from image pairs allows users to generate natural images from sketches or semantic images [Isola et al. 2017]. Trained on portrait images with a variety of synthesized lighting conditions, interactive relighting results can be achieved using a single input portrait [Sun et al. 2019]. Beyond images, 3D shapes including human body geometry can be reconstructed from single 2D views [Arsalan Soltani et al. 2017; Alldieck et al. 2019]. While regular losses such as L^1 and L^2 differences offer good performance for training on datasets with surjective mapping [Sun et al. 2019], GANs improve the perceptual quality significantly for a wide range of multimodal problems, e.g., image in-painting [Kaplanyan et al. 2019], fluid super resolution [Xie et al. 2018] and city modeling [Kelly et al. 2018] tasks. In the following, we first discuss related work on general unconditional GAN, and then discuss conditional GAN (cGAN), which is widely used in paired and unpaired image translation tasks.

GANs. Achieving state-of-the-art performance, mode collapse is still one of the largest challenges for GANs. Starting from the vanilla GAN with a cross entropy loss [Goodfellow et al. 2014], a series of loss functions have been proposed for GAN training, including the least-square GAN [Mao et al. 2017], the Wasserstein GAN [Gulrajani et al. 2017], and the relativistic GAN [Jolicoeur-Martineau 2018]. BourGAN [Xiao et al. 2018] extends the input domain with a mixture of Gaussians in order to support multimodality of the target domain. On the other hand, InfoGAN [Chen et al. 2016], VEEGAN [Srivastava et al. 2017], and BEGAN [Marzouk et al. 2019] propose improvements on the self-supervision. Both InfoGAN and VEEGAN use additional classifiers to reconstruct the latent code that generates the fake samples, while InfoGAN focuses on the

latent-code disentanglement and VEEGAN focuses on their distributions. The discriminator of BEGAN is extended with an auxiliary decoding part to restore ground-truth targets, which then offers a pixel-wise supervision for the generation. In these methods, the risk of mode collapse is reduced with improved supervision. Sampling from noises, high-quality and diverse samples can be generated with general GANs [Karras et al. 2020].

cGANs. While general GANs sample from regular distributions, cGANs focus on conditional generation, e.g., image translation tasks. Aiming to learn conditional distributions, generators and discriminators in cGANs usually take conditional attributes as additional inputs. While this provides a level of user control, it usually yields blurry results due to a compromise between different modes and falls behind general GANs in terms of quality and diversity. With paired data, Pix2Pix [Isola et al. 2017] and Pix2PixHD [Wang et al. 2018] achieves high quality results, while the diversity of results cannot be easily extended. For unpaired data domains, CycleGAN [Zhu et al. 2017] proposes to establish cycle consistency between domains and MUNIT [Huang et al. 2018] decomposes image representations into content space and style space.

While our work shares similarity with BEGAN, InfoGAN, and CycleGAN, we propose to use conditional attributes as outputs of discriminators. In this way, a cyclic link is formed between the conditional attributes and the target. Targeting conditional supervision in our work, we will show the superiority of the proposed method over typical conditional adversarial learning methods below.

3 VELOCITY GENERATION FROM SINGLE-FRAME DENSITY

Typically, fluid animations in computer graphics are generated by solving the incompressible Navier-Stokes (NS) equations:

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla \mathbf{u} + \mathbf{f} \,, \\ \nabla \cdot \mathbf{u} &= 0 \,, \end{aligned} \tag{1}$$

where $\mathbf{u}, p$ denote velocity, pressure, and an external force function $\mathbf{f} = \mathbf{f}(d, \mathbf{u}, \mathbf{s})$ that depends on marker density, velocity and a set of physical parameters $\mathbf{s}$. For simplicity, we have dropped the ν term with explicit viscosity calculations in the rest of the paper but rely on the numerical viscosity of the discretization. Focusing on velocity fields, traditional solvers compute an update in the following way:

$$\mathbf{u}_t = \mathcal{A}(\mathbf{u}_{t-1}, \mathbf{u}_{t-1}) + \mathbf{f} - \frac{1}{\rho} \nabla p \,, \tag{2}$$

where $\mathcal{A}$ denotes an advection step and t denote the time step. Other quantities including marker density are similarly advected with the underlying flow, i.e., obey $\partial d / \partial t + \mathbf{u} \cdot \nabla d = 0$ or $\mathbf{d}_t = \mathcal{A}(\mathbf{d}_{t-1}, \mathbf{u}_{t-1})$.

In order to support density interactions, we now change the perspective and consider density as the primary quantity of interest, solving for a velocity output that fulfills the density conditions. From before we have $\mathbf{u}_t = \mathcal{F}(\mathbf{u_{t-1}})$, with function $\mathcal{F}$ derived from Eq. 2. However, we'd instead like to have $\mathbf{u}_t = \mathcal{G}(d_t)$, with an unknown function $\mathcal{G}$. As mentioned previously, manually deriving the relationship between d_t and $\mathbf{u}_t$ according to NS equations is impractical: $\mathcal{G}$ is strongly under-constrained, and d by itself does not allow us to draw concrete conclusions about the underlying motion $\mathbf{u}$. Hence we approach this problem in a data-driven manner: based on a simulation dataset, we aim for finding the most suitable $\mathbf{u}$ to explain an observed density configuration d.

First, a large dataset is prepared by solving NS equations with a wide range of initial settings and physical parameters. Represented as $\{d_t, \mathbf{u}_t\}_n$, the dataset defines a non-linear, high-dimensional, and multimodal mapping between d and $\mathbf{u}$ with n pairs of data samples obeying the NS equations. With the help of such a dataset, we can now approximate the complex mapping $\mathcal{G}$ with a neural network: $G(d_t; \theta) \approx \mathbf{u}_t$. The network improves its variables θ during training, until a local minimum is found for chosen loss function $\mathcal{L}(\mathbf{u}_t, G(d_t; \theta))$. This not only makes it possible to find an approximation of $\mathcal{G}$, but the trained NNs also allow for very fast runtimes, which is important for user-centered applications.

While density plays an important semantic role for users, different tasks typically require different viewpoints of a problem and hence also different control knobs. Thanks to the flexibility of data-driven algorithm, it is possible to include influence from other quantities, e.g., the kinetic energy (KE) and the vorticity (ω), when paired data is provided. Thus, by extending the training dataset to $\{[d, \mathbf{s}, \mathrm{KE}, \omega]_t^\top, \mathbf{u}_t\}_n$, we train $G : [d, \underline{\mathbf{s}, \mathrm{KE}, \omega}]_t^\top \rightarrow u_t$ to learn a multivariate mapping. Denoted with an underline, $\mathbf{s}, \mathrm{KE}$ and, ω are *optional inputs* and a flag value of -1 will be used when not given. A purely density-based velocity generation is offered when all other quantities are -1. This design allow users to combine optional conditions freely and use the trained model to generate a velocity output that handles all given conditions. With this formulation, an update step of our novel density-based simulation via $\mathbf{u}_t = G(d_t; \theta)$ is summarized the as:

$$\begin{aligned} d_t &= \mathcal{A}(d_{t-1}, \mathbf{u}_{t-1}) \,, \\ \mathbf{u}_t &= G(d_t, \underline{\mathbf{s}_t, \mathrm{KE}_t, \omega_t}; \theta) \,. \end{aligned} \tag{3}$$

When approximating the mapping defined by the training dataset, the network accuracy depends on the loss functions and architectures. Especially when learning a multimodal relationship, like the one between density and velocity, it is hard to avoid blurry results as a flat local minimum when using regular losses including $\mathcal{L}^2$ norms. The adversarial learning of GANs is powerful in preventing convergence to these undesirable minima. For a GAN, we jointly train a pair of networks: a generator and a discriminator. While the generator learns to produce data samples that match the target domain, the discriminator is trained to learn the distribution of the ground-truth dataset and to supervise generated samples accordingly. Learning the data distribution of the ground truth helps to avoid undesired mode interpolations. While unconditional GANs, e.g., StyleGAN [Karras et al. 2020], generate realistic data from random samples of a synthetic distribution like Gaussian noise, conditional GANs learn a generation process conditioned on inputs that allow users to control.

Facing a conditional generation task from density to velocity, we propose to train our network based on conditional GANs. The training process for a typical cGAN could be formulated as:

$$\arg \max_{\theta_G} \min_{\theta_D} \mathcal{L}_p(\mathbf{u}, d; \theta_D) - \mathcal{L}_n(G(d; \theta_G), d; \theta_D) \,, \tag{4}$$

where $\mathcal{L}_p$ is a function of $D(\mathbf{u}, ...)$ measuring the classification distances on the positive side, and $\mathcal{L}_n$ is a function of $D(G(d), ...)$ for

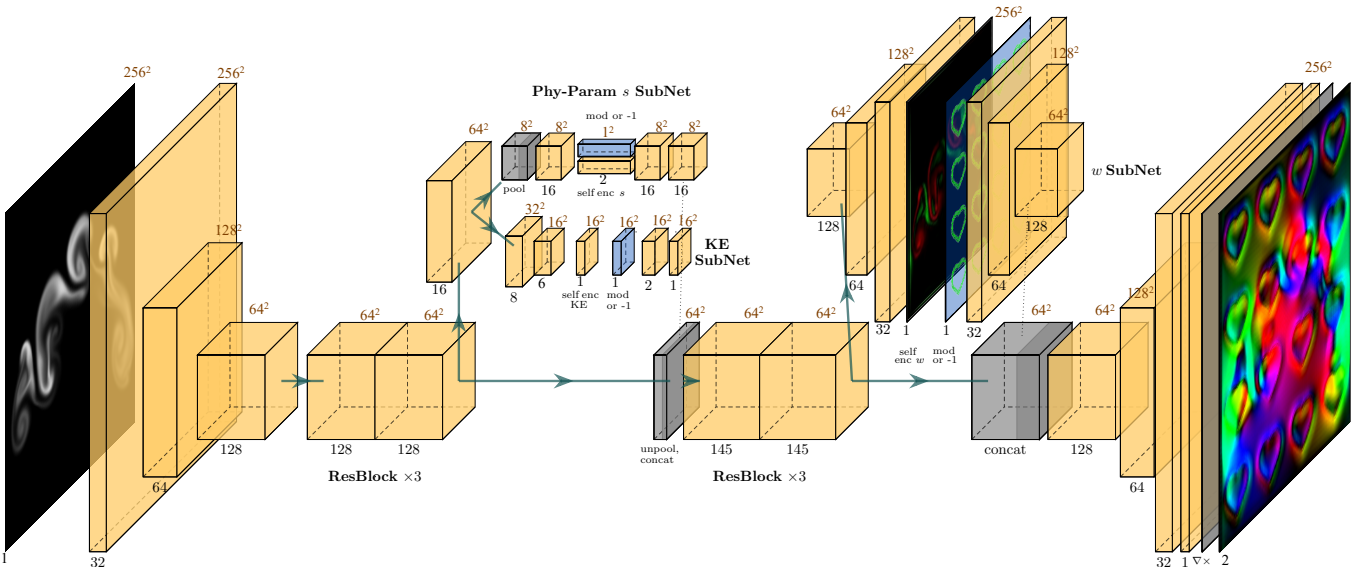


Fig. 5. A ResNet-based encoder-decoder generator, with $\mathbf{s}$, ω, and KE sub-networks encoding corresponding quantities in the bottleneck. Our discriminator shares a very similar architecture with a swapped density (as output) and velocity (as input). Yellow, blue, and gray boxes stand for normal or transposed convolutional layers, optional inputs, and pooling or un-pooling layers, respectively. Black numbers below stand for channel depth, while orange numbers on top represent spatial resolutions for the 2D case with a density input in shape of $256 \times 256 \times 1$. For 3D cases and cases with obstacles, please refer to the appendix.

the negative side. Playing the min-max game, the discriminator tries to make a clear separation between $D(\mathbf{u})$ and $D(G(d))$, while the generator tries to decrease $\mathcal{L}_n$ by generating realistic samples. In this and following equations, all physical quantities share a consistent subscript of the current time step t, so it has been discarded. Our method can be seen as a special form of a conditional GAN, a visual outline for which is given in Fig. 3. On the generative side, a ResNet-based encoder-decoder architecture is used for $G : d \rightarrow \mathbf{u}$. Although there is a strong multimodality between d and $\mathbf{u}$, a valid estimation of physical quantities can largely reduce this ambiguity and point towards a most likely mode for the decoding side. Thus, we propose to encode physical quantities in the latent space, as will be explained in Sec. 3.1. On the discriminative part, we propose to use an architecture that closely resembles the generator. The only difference is that the velocity, either generated or ground-truth, is now the input while density forms the output. With $D : \mathbf{u} \rightarrow d$, our approach is different to other cGANs where discriminators take conditional attributes, e.g., density, as input and produces scores as classification outputs, i.e. $D : \mathbf{u}, d \rightarrow$ (scores or distances). We show in Sec. 3.2 that our cyclic mapping with $G : d \rightarrow \mathbf{u}$ and $D : \mathbf{u} \rightarrow d$ leads to an improved generation of small-scale structures. Details of $\mathcal{L}_p$ and $\mathcal{L}_n$ are given in Sec. 3.2. Like the generator, our discriminator also encodes physical quantities in the latent space. This further allows supervision with modified conditions, as introduced in Sec. 3.3. The next sections will provide details of our loss formulation.

3.1 Latent-Space Control Using Physical Quantities

Our generator network $G : [d, \underline{\mathbf{s}, \mathrm{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \mathrm{KE}, \omega, \mathbf{u}]^\top$ takes several optional inputs, as indicated by an underline. It is forced to always estimate $\mathbf{s}$, KE, and ω as outputs in the latent-space, which is crucial for an accurate synthesis of the desired output $\mathbf{u}$. This latent-space encoding shares similarities with the environmental map encoding proposed by Sun et al. [2019] for portrait relighting. We propose the use of sub-networks for each quantity. This not only offers additional user controls, but also simplifies the many-to-many mapping between d and $\mathbf{u}$ with the many-to-one mapping between $(d, \mathbf{u})$ and $\mathbf{s}$, which effectively improves the overall inference accuracy.

A detailed architecture of our generator can be seen in Fig. 5. It transfers density into velocity with an encoder-decoder network, where a curl operation is used as the output layer in order to achieve fully incompressible velocity as output [Kim et al. 2019b]. Meanwhile, physical quantities are inferred from densities in the bottleneck part. Specifically, the physical parameters $\mathbf{s}$, containing buoyancy and boundary conditions, are learned with a $\mathbf{s}$ sub-net, while a coarse kinetic energy and a detailed vorticity are encoded with KE and ω sub-networks individually. While we have $\mathbf{s}$ in shape of $1 \times 1 \times 2$, a $16 \times 16 \times 1$ KE, and a $256 \times 256 \times 1$ ω for the regular 2D case, the method can be easily extended to support other shapes. More details of the architecture are given in the appendix.

The architecture of these sub-networks is quite straightforward when they are considered individually: From the density input to the encoded $\mathbf{s}$, we have an encoding network with convolutional layers, pooling layers, and a fully connected layer in the end. KE is generated with fully convolutional layers. Besides the encoding part, ω sub-net uses convolution-transpose layers to decode. Once these important quantities are predicted, we concatenate them with user modifications or flag values, since both self-encoded values and user modifications are important for the velocity estimation. In Fig. 5, these optional inputs are colored in blue. With decoding layers, the second halves of our sub-networks connect these latent code back into the original position of the main network.

3.2 Adversarial Learning with Cyclic Mapping

Although the embedding of physical quantities in the latent space of the generator via sub-networks significantly helps to reduce the training difficulty, it is still hard to avoid problems of mode collapse when training with regular GAN losses. According to Eq. 4, GANs are usually trained with the following losses:

$$\begin{cases} \mathcal{L}_{D,\text{adv}} & = \mathcal{L}_p(D(\mathbf{u},...)) - \mathcal{L}_n(D(G(d),...)), \\ \mathcal{L}_{G,\text{adv}} & = \mathcal{L}_n(D(G(d),...)), \end{cases} \tag{5}$$

The original cGAN modifies this setup and uses a conditional attribute as an additional input of the generator and discriminator [Mirza and Osindero 2014]. Using $D : \mathbf{u}, d \rightarrow (0, 1)$ for Eq. 5 with cross entropy losses as $\mathcal{L}_p$ and $\mathcal{L}_n$, its goal is to learn the conditional data distribution using discriminators. This, however, is a very complex task and requires stable training with large amount of data under varying conditions.

In practice, several distance functions have been proposed for better convergence. E.g., BEGAN and cBEGAN [Berthelot et al. 2017; Marzouk et al. 2019] extend the discriminator as an auto-encoder. With $D : \mathbf{u}, d \rightarrow \mathbf{u}$, the discriminator of cBEGAN tries to restore only the ground-truth samples with

$$\mathcal{L}_p = \|\mathbf{u} - D(\mathbf{u}, d)\|; \; \mathcal{L}_n = \|\mathbf{u} - D(G(d), d)\|. \tag{6}$$

In our work on the other hand, the conditional attributes, such as d, are considered to be an output rather than an input of the discriminator. Compared to Eq. 6 where the target data $\mathbf{u}$ is restored, our discriminator $D : \mathbf{u} \rightarrow d$ thus tries to infer the conditioning d correctly for ground-truth samples via:

$$\mathcal{L}_p = \|d - D(\mathbf{u})\|; \; \mathcal{L}_n = \|d - D(G(d))\|. \tag{7}$$

In this way, our conditional supervision is established through a cycle link between the generator and the discriminator: During training, the discriminator focuses on retrieving the conditioning from the target velocity $\mathbf{u}$, while the generator should fool the discriminator by keeping features that correspond to the correct conditioning. Trained to minimize $\mathcal{L}_n$ in Eq. 7, the generator is punished for results corresponding to wrong conditions in d, such as blurry results interpolated from different modes. Although it is still a complex task to learn the conditional data distribution, our architecture allows the discriminator to check for the fulfillment of the specified conditions in a more explicit manner.

Before introducing additional conditional attributes as inputs to the discriminator, we compare the discriminators of cBEGAN and our conditional adversarial learning using Eq. 6 and Eq. 7 respectively in the first row of Fig. 6. Suffering from mode collapse problems, Eq. 6 generates sub-optimal results while our adversarial approach yields significantly more detailed results.

Similar to the objective for the generator, the task for the discriminator can be simplified by including physical quantities. Thus, using almost the same encoder-decoder architecture, we propose to likewise encode $\mathbf{s}, \omega$, and KE using bottleneck layers of the discriminator. In this way, we arrive at a generator and discriminator in the form of $G : [d, \underline{\mathbf{s}, \text{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \text{KE}, \omega, \mathbf{u}]^\top$ and $D : [\mathbf{u}, \underline{\mathbf{s}, \text{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \text{KE}, \omega, d]^\top$. Hence, the discriminator can offer conditional supervision on physical quantities according to its self-encoded values.

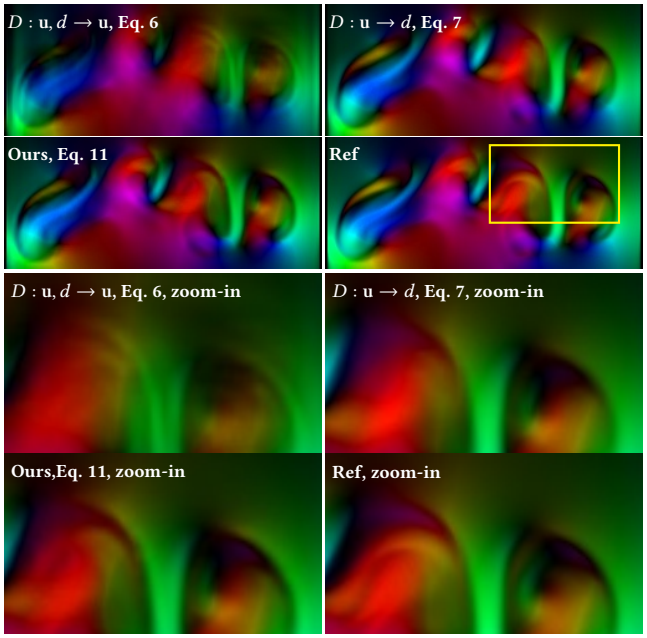


Fig. 6. Taking density as a input for the discriminator, cBEGAN produces blurry results with ringing artifacts in the top left. Via density as the discriminator's output, results are improved on the top right. Our final model at the bottom left further includes physical parameters as discriminator outputs, and yields realistic results that closely resemble the reference at the bottom right. Generator architectures are the same for all cases.

With

$$\begin{aligned} \mathcal{L}_p &= \|[d, \mathbf{s}, \omega, \text{KE}]^\top - D(\mathbf{u})\|; \\ \mathcal{L}_n &= \|[d, \mathbf{s}, \omega, \text{KE}]^\top - D(G(d, \underline{\mathbf{s}, \text{KE}, \omega}))\|, \end{aligned} \tag{8}$$

our discriminator is trained to infer the correct condition of $[d,\mathbf{s},\omega,\text{KE}]^\top$ for ground-truth velocities, so that the generated outputs have to fulfill these conditions accordingly.

Following previous work [Berthelot et al. 2017], it is additionally beneficial to maintain the balance between the generator and the discriminator dynamically during training using a hyper-parameter $\gamma = \mathbb{E}[\mathcal{L}_p(\mathbf{u})]/\mathbb{E}[\mathcal{L}_n(G(d))]$. To summarize, the learning objective for the adversarial part of our network is:

$$\begin{cases} \mathcal{L}_{D,\text{adv}} = & \|[d, \mathbf{s}, \omega, \text{KE}]^\top - D(\mathbf{u})\| \\ & -k\|[d, \mathbf{s}, \omega, \text{KE}]^\top - D(G(d, \underline{\mathbf{s}, \text{KE}, \omega}))\|, \\ \mathcal{L}_{G,\text{adv}} = & \|[d, \mathbf{s}, \omega, \text{KE}]^\top - D(G(d, \underline{\mathbf{s}, \text{KE}, \omega}))\| \\ k = & \text{MovingAVG}\ (\gamma) \end{cases} \tag{9}$$

During training, the optional inputs for $\mathbf{u}, \mathbf{s}, \omega$ and KE for the generator and discriminator are randomly selected to contain the flag (-1) or the reference value provided in the training data.

In line with our full discriminator, $D : [\mathbf{u}, \underline{\mathbf{s}, \text{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \text{KE},\omega,d]^\top$, a full cBEGAN model can be trained with $D : [\mathbf{u}, d, \mathbf{s}, \text{KE}, \omega]^\top \rightarrow \mathbf{u}$. Compared to the first row of Fig. 6 with models conditioned by density only, a full conditioning via $(d, \mathbf{s}, \text{KE}, \omega)$ offers better accuracy. In the rest of our paper, the cBEGAN model and ours refers to the versions with full conditioning attributes.

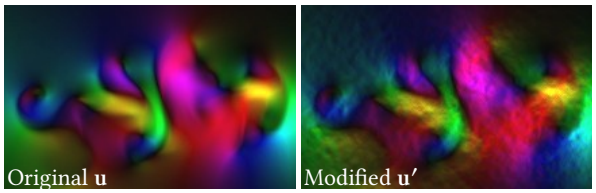


Fig. 7. An example of modified $\mathbf{u}'$ based on the wavelet turbulence [Kim et al. 2008] method. Its KE and ω are used as the modified conditions in the proposed training with data augmentation.

3.3 Control Disentanglement

When receiving density as the only input, the generator first predicts the physical quantities $\mathbf{s}$, ω, and KE. Afterwards, it can infer realistic velocity content based on the predicted conditions. As there are no further user modifications and all conditions are consistent with each other, we denote this as a *"restorative"* generation task. While this case is handled properly using our cyclic adversarial training as explained so far, things become more difficult when user modifications are introduced. In a *"modified"* generation task, the results are expected not only to be different to the restorative version, but also to faithfully and sensitively deliver the quantitative modifications as specified by the user. Interestingly, a trained network makes significant errors for such tasks if it has never seen these modified conditions at training time. As the physical quantities are correlated in the training data, the generative model tends to place more attention on directly related conditions. For instance, once trained, modifications of KE lead to clear differences in the generated velocities, but when KE is kept unmodified, the model largely ignores changes only in $\mathbf{s}$, as these would likewise lead to a change in KE according to simulations in the training dataset.

In order to make user controls *as-disentangled-as-possible*, we propose to train the generative model with randomly perturbed parameters as data augmentation. We obtain modified physical parameters $\mathbf{s}'$ by randomly sampling a uniform distribution from a pre-defined range for each parameter. As adding random perturbations to the original KE and ω could result in unnatural results, we choose to first synthesize a modified $\mathbf{u}'$ and calculate its KE and ω as the modified conditions. More specifically, we use wavelet turbulence [Kim et al. 2008] with a randomized strength. With $\mathbf{y}(\mathbf{X})$ and e_f denoting curl noise evaluation and frequency-based weighting, respectively, it can be summarized as:

$$\begin{cases} \mathbf{u}_{LR} & = \mathrm{DownSample}(\mathbf{u}, 0.25) \\ \mathbf{u}' & = \mathrm{UpSample}(\mathbf{u}_{LR}, 4.0) + 2^{-\frac{5}{6}} e_\mathrm{f} \cdot \mathbf{y}(\mathbf{X}) \\ \mathrm{KE}', w' & = \frac{1}{2}\mathbf{u}'^2,\ \nabla \times \mathbf{u}' \end{cases}$$

A visual sample is given in Fig. 7

When conditions are modified, the generated results should be supervised accordingly. While the fulfillment of KE and ω conditions can be improved via loss terms $\|KE' - \frac{1}{2} G_\mathbf{u}(d, KE')^2\|$ and $\|\omega' - \nabla \times G_\mathbf{u}(d, \omega')\|$, the conditioning of $\mathbf{s}$ can only be guaranteed with the help of our discriminator. Specifically, we use a loss function $\|s' - D_\mathbf{s}(G_\mathbf{u}(d, s'))\|$. With a binary mask M randomly deciding which conditions to modify, the learning objective with data augmentation can be summarized as:

$$\begin{cases} \mathbf{u}^* & = G_\mathbf{u}\big(d, \mathrm{M} \odot [\mathbf{s}', \omega', KE']^\top + (1\text{-M}) \odot \mathrm{FlagValues}\big) \\ d^*, s^* & = D_{d,\mathbf{s}}\big(\mathbf{u}^*, \mathrm{M} \odot [\mathbf{s}', \omega', KE']^\top + (1\text{-M}) \odot \mathrm{FlagValues}\big) \\ \mathcal{L}_{G,\mathrm{mod}} & = \|[1, \mathrm{M}] \odot ([d, \mathbf{s}', \omega', KE']^\top - [d^*, s^*, \nabla \times \mathbf{u}^*, \tfrac{1}{2}\mathbf{u}^{*2}]^\top)\| \end{cases} \tag{10}$$

with flag values in the same shape of corresponding conditions and $\odot$ stands for an elementwise product.

3.4 Learning Objectives Summary

To summarize, our approach for conditional generative learning is based on a generator and a discriminator using the same encoder-decoder architecture with switched inputs and outputs. The conditional supervision is established though the cyclic-link between them. Being used in the adversarial training for restorative generation, this conditional supervision further allows us to train via augmented data with modified conditions. We additionally include an L^1 regularization term with strength λ_{l1} for stabilization. The final learning objectives for our generator and discriminator are:

$$\begin{cases} \mathcal{L}_G & = \lambda_{adv}\mathcal{L}_{G,\mathrm{adv}} + \lambda_{mod}\mathcal{L}_{G,\mathrm{mod}} + \lambda_{l1}\|[\mathbf{u}, \mathbf{s}, \omega, \mathrm{KE}]^\top - G(d)\| \\ \mathcal{L}_D & = \mathcal{L}_{D,\mathrm{adv}}. \end{cases} \tag{11}$$

This formulation provides us with an efficient generative model for velocity that supports a range of user controls from density, vorticity, kinetic energy to physical parameters. Most importantly, the network reacts sensitively to all controls.

3.5 Data-sets and Training Summary

Our regular 2D training dataset without obstacles has 200 simulations with a resolution of 256^2. For the initial state, we sample several density sources with random positions and shapes. Regarding physical parameters, the buoyancy force varies from 10^{-4} to 2×10^{-4} and boundary conditions are either closed on all directions or opened along the y-axis. In each of the simulations, the first 60 time steps are skipped as the preparation phase and the following 200 time steps are recorded as data pairs with density and velocity information. The regular 3D dataset is likewise computed with a resolution of 256^3 with a longer preparation phase of 90 time steps. Trained on these datasets according to Eq. 11, our full models in 2D and 3D are able to support a wide range of applications, which will be shown in Sec. 5.

Besides purely buoyant motions contained in the regular datasets, turbulent motion caused by obstacles is also very important for practical fluid simulations. Thus, we prepare datasets in 2D and 3D with static and moving obstacles. Specifically, we first simulate 100 simulations with up to 5 *static* obstacles in regular shapes, namely square and circular shapes, located at random places that could overlay with each other. Based on this 100 simulations, we further assign random velocity to obstacles on each frame to get another dataset with influence from moving obstacles. Using the same training loss and network architectures, we train our obstacle models in 2D and 3D with $G : [d, o_m, o_\mathbf{u}, \underline{\mathbf{s}, \mathrm{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \mathrm{KE}, \omega, \mathbf{u}]^\top$ and $D : [\mathbf{u}, o_m, o_\mathbf{u}, \underline{\mathbf{s}, \mathrm{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \mathrm{KE}, \omega, \underline{d}]^\top$ using additional inputs $o_m, o_\mathbf{u}$, which denote a mask for the obstacle regions and the velocity of the obstacles.

Examples of training data are given in the supplemental video. Before showing various results produced by the four models, in the

Table 1. Evaluations on restorative tests and modification tests of kinetic energy and vorticity. While the performance is close for the restorative tests, our model achieves better control of KE and ω.

	Restoration, $\boldsymbol{s}$ (L1↓)	Restoration, $\boldsymbol{u}$ with ref-$\boldsymbol{s}$ (L1↓)		Restoration, $\boldsymbol{u}$ with self-$\boldsymbol{s}$ (L1↓)	
Models	static, temporal	static, temporal	(LSiM↓) static, temporal	static, temporal	(LSiM↓) static, temporal
MAE	0.128, 0.197	0.059, 0.097	0.107, 0.142	0.071, 0.149	0.131, 0.204
Pix2Pix	0.089, 0.237	0.067, 0.122	0.117, 0.160	0.079, 0.186	0.145, 0.235
cBEGAN	*0.079, 0.134*	**0.055, 0.093**	0.101, 0.136	**0.067, 0.131**	0.125, 0.191
NoMod	0.078, 0.146	0.064, 0.107	**0.094, 0.126**	0.071, 0.156	**0.111, 0.186**
Ours	**0.063, 0.095**	*0.058, 0.094*	*0.101, 0.134*	*0.065, 0.137*	*0.116, 0.192*

KE and ω Modifications				
	M_{aKE}, a =		$M_{a\omega}$, a =	
Models	0.500, 1.000, 2.000, 4.000	$\frac{M_{KE\times4}}{M_{KE\times0.5}}\uparrow$	0.500, 1.000, 2.000, 4.000	$\frac{M_{\omega\times4}}{M_{\omega\times0.5}}\uparrow$
MAE	0.612, 0.947, 1.587, 2.694	*4.827*	0.870, 1.030, 1.216, 1.401	1.61
Pix2Pix	0.563, 0.944, 1.566, 2.626	4.664	1.348, 1.461, 1.736, 2.046	1.518
cBEGAN	0.663, 0.963, 1.558, 2.497	3.766	0.833, 1.071, 1.407, 1.733	*2.08*
NoMod	0.759, 1.004, 1.597, 2.749	3.622	1.003, 1.123, 1.444, 1.782	1.777
Ours	0.588, 0.991, 1.806, 3.480	**5.919**	0.594, 0.804, 1.269, 2.220	**3.737**

Table 2. Evaluations on physical-parameter modifications. Compared to simulation candidates with different conditions, methods should match the true reference, marked with an underline in the table with a closest distance. Our method ranks first for buoyancy and boundary controls in terms of accumulated distances to the true reference.

$\boldsymbol{s}$ Modifications (LSiM↓)						
	Buo × 2.0		Buo × 0.5		**Avg.** to F/T **Buo**	
Models	1.00>1.50>2.00<2.50		0.25>0.50<1.00<2.00		FALSE	**TRUE(↓)**
MAE	0.140, 0.157, 0.203, 0.250		0.186, *0.130*, 0.193, 0.252		0.196	0.167
Pix2Pix	0.171, 0.183, 0.218, 0.255		0.182, 0.143, 0.206, 0.260		0.210	0.181
cBEGAN	0.153, 0.153, 0.194, 0.241		0.181, **0.130**, 0.191, 0.249		0.195	*0.162*
NoMod	0.200, 0.171, *0.176*, 0.202		0.222, 0.159, 0.175, 0.222		0.199	0.168
Ours	0.222, 0.175, **0.159**, 0.176		0.187, 0.135, 0.191, 0.245		0.199	**0.147**
	OpenBnd	Closed Bnd	**Avg.** to F/T **Bnd**		**Total Avg.**	
Models	Closed>Open	Closed<Open	FALSE	**TRUE(↓)**	FALSE	**TRUE(↓)**
MAE	0.246, 0.183	0.150, 0.246	0.246	0.167	0.221	0.167
Pix2Pix	0.227, 0.221	0.162, 0.229	0.228	0.192	0.219	0.186
cBEGAN	0.243, 0.185	*0.145*, 0.237	0.240	0.165	0.217	0.164
NoMod	0.302, *0.163*	0.146, 0.228	0.265	*0.155*	0.232	*0.161*
Ours	0.280, **0.149**	**0.138**, 0.219	0.250	**0.144**	0.224	**0.145**

next section we focus on the regular training without obstacles in 2D to evaluate the quality of the proposed approach with visual results and metrics.

4 EVALUATION SUMMARY

We summarize evaluations of our method in comparison with related work and ablated models to illustrate the effects of $\mathcal{L}_{G,\text{adv}}$ and $\mathcal{L}_{D,\text{adv}}$, as well as the modified conditions via $\mathcal{L}_{G,\text{mod}}$. The following models are used for comparisons: First, a baseline model denoted as *MAE* is trained to reduce the Mean Absolute Error without discriminators using $\mathcal{L}_G = \lambda_{l1}\|[\mathbf{u}, \mathbf{s}, \omega, \text{KE}]^\top - G(d)\|$. This model largely resembles previous work employing supervised approaches [Kim et al. 2019b]. Training the generative part under the adversarial supervision of a classifier-style discriminator, a *Pix2Pix* model is obtained for our case. Furthermore, replacing the regular adversarial training with aforementioned $\mathcal{L}_p = \|\mathbf{u} - D(\mathbf{u}, [d, \mathbf{s}, \omega, \text{KE}]^\top)\|$ and $\mathcal{L}_n = \|\mathbf{u} - D(G(d), [d, \mathbf{s}, \omega, \text{KE}]^\top)\|$, we obtain a model resembling *cBEGAN* [Marzouk et al. 2019]. Correspondingly, training with our adversarial losses of Eq. 8 but without $\mathcal{L}_{G,\text{mod}}$, we get the *NoMod* model. The final model, *Ours*, is additionally trained with $\mathcal{L}_{G,\text{mod}}$, yielding the full formulation of Eq. 11. For all adversarial models, the same dynamic balancing scheme is applied. A regular fluid solver, MantaFlow [Stam 1999; Thuerey and Pfaff 2018], is used as ground truth. In the following, we summarize evaluations in tables, while detailed figures and statistics are presented in the appendix. The supplemental videos are recommended for visual comparisons of the resulting motions.

The evaluations can be divided into *restorative* tests, where density forms the only input and results should match the references, and *modification* tests, where other quantities are modified with the goal to obtain correspondingly varying outputs. In the latter, generated velocities should vary continuously and sensitively in accordance to the changing controls.

Static and temporal restoration accuracy is evaluated using $\mathcal{L}^1$ and LSiM [Kohl et al. 2020] in Table 1 (Restoration). The static error measures outputs across 1200 density inputs with the ground-truth velocity of the reference solver. It quantifies the averaged behavior for single-frame estimations. The temporal error, on the other hand, is measured over the course of 60 frames. For 20 different sequences, we take first density frames as the initial inputs, estimate velocities and advect forward. The averaged difference for 60 frames then quantifies temporal behaviors. The static and temporal errors for the physical parameters $\mathbf{s}$ are likewise calculated.

From Table 1, we can see that MAE model performs poorly for the estimation of $\mathbf{s}$. However, by relying more on density inputs, it still manages to achieve good accuracy for velocity estimation. The performance of all models is close for the restoration of $\mathbf{u}$, with cBEGAN being very close to Ours, both outperforming the other variants. Our final model achieves the best performance in terms of $\mathbf{s}$ estimation, which indicates that our approach yields an improved encoding of physical parameters in the latent space.

As our goal is to move beyond a pure restoration of inputs, it is especially interesting to evaluate how well the different methods support user manipulations. Thus, we evaluate results with KE and ω modifications in Table 1 as well as changes to buoyancy and boundary conditions in Table 2. Changing conditional KE and ω by constant factors, we calculate the ratio between generated fields and the original references. Averaged on 1200 density inputs, our final model yields the best sensitivity for decreased and enhanced KE and ω. For evaluations on sensitivity to physical parameters, we generate results with modified $\mathbf{s}$ and compare them with regard to a series of simulations offered by a regular solver using gradually modified conditions. Our full model ranks first by matching the correct candidate all the time, while others have problems matching tough conditions well, e.g., enhanced buoyancy or open boundary. Our full model has the smallest distances to the true references in average and our NoMod model ranks second, followed by cBEGAN, MAE and Pix2Pix.

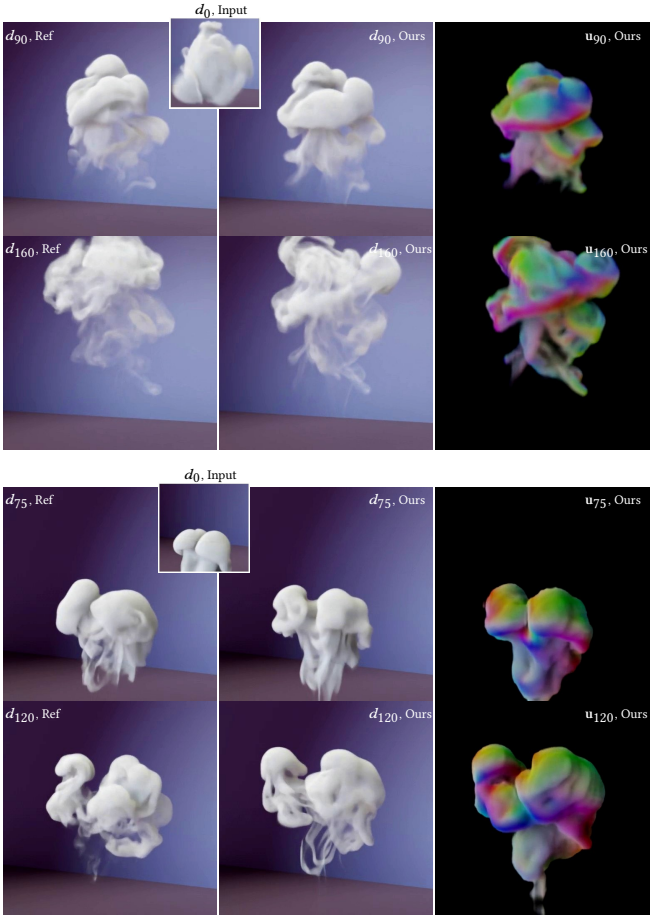


Fig. 8. Regular plume simulations with a resolution of 256^3 generated by our method compared to a reference simulation. Starting from a single density frame as input, our method yields realstic motions that stay close to the reference over the course of more than 100 recurrent evaluation and advection steps.

To summarize, MAE and cBEGAN exhibit similar behavior in general: While cBEGAN slightly improves on all aspects, both of them achieve good velocity restoration accuracy but with entangled controls. Their models are mainly affected by the density and kinetic energy, and are less sensitive to changes of physical parameters and vorticity. Compared to MAE, the Pix2Pix model performs slightly worse for almost all aspects due to mode collapse problems, which is illustrated in the appendix with additional examples. Our adversarial learning with cyclic links balances all conditional inputs significantly better. The NoMod model is more sensitive to modifications of physical-parameters, at the expense of a slightly decreased restoration accuracy and energy control. By adding $\mathcal{L}_{G,mod}$, our final model combines both aspects: it achieves a very good overall performance together with an improved disentanglement of the controls. We further evaluated our method and cBEGAN on a more challenging training task with varying buoyancy directions. Results in the appendix show a consistent conclusion that as a conditional GAN, our method follows the conditions more sensitively and accurately.

5 RESULTS

In the following, we compare our method with the reference fluid solver [Thuerey and Pfaff 2018] for generating rising plume simulations, and show a variety of conditional results achieved with additional user modifications. At last, we extend our model to obstacle interactions and demonstrate its capabilities for generalization with a range of test cases far from the training data. An additional optimization case, leveraging the differentiable nature of the trained model for an inverse problem, can be found in the appendix.

5.1 Smoke Generation from a Single Density Input

Fig. 8 shows two simulations of rising plumes. Starting from a single density volume, our simulation results match the large scale structures of the reference even after 100 time-steps. Although small scale details differ from the references, they rotate and diffuse naturally with realistic motions. Trained on regular simulations, our method generalizes to new, artificial density inputs, e.g., drawings shown in Fig. 9. Using this density-based fluid generation method, users can design the initial state for a simulation with density manipulations, as illustrated in Fig. 2, and modify density in on-going simulations.

5.2 Controlling Results with Physical Quantities

Besides density-based generation, our method supports user controls in terms of KE, ω, as well as physical parameters. When preparing these conditional attributes, users can make use of the self-encoded quantities as a starting point. In Fig. 10, we show modified results of the previous plume scene. Similar to the 2D model, our 3D model responds sensitively when buoyancy increases. Multiplied by a random scalar field in 3D space with values from 1.5 to 4, the enhanced vorticity condition leads to a natural result with more vortices. Conditioned with increasing buoyancy and vorticity, we can generate a series of results that change continuously and sensitively, as shown on the top of Fig. 11. While boundary modification for 2D plumes is displayed at the bottom, we show 2D vorticity control using textures in Fig. 12. As shown in the visualization of velocity and vorticity, the generated velocity closely follows the texture-based modifications.

5.3 Learning Obstacle Influences

Besides purely buoyant motions as discussed above, turbulent motion caused by obstacles is also very important for practical fluid simulations. Based on a dataset with random obstacles in regular shapes, we train our networks to learn their influences on velocity outputs. Using the same training loss and network architectures, we train our $G : [d, o_m, o_\mathbf{u}, \underline{\mathbf{s}, \mathrm{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \mathrm{KE}, \omega, \mathbf{u}]^\top$ and $D : [\mathbf{u}, o_m, o_\mathbf{u}, \underline{\mathbf{s}, \mathrm{KE}, \omega}]^\top \rightarrow [\mathbf{s}, \mathrm{KE}, \omega, \underline{d}]^\top$ with additional inputs o_m, $o_\mathbf{u}$, which denote a mask for the obstacle regions and the velocity of the obstacles.

Our 2D results are shown in Fig. 13. The left scene has obstacles in regular shapes, the middle one uses an irregular heart shape as a generalization test, and the right scene has moving obstacles with concave shapes. The velocity and vorticity visualizations show that obstacles are properly handled in our results. For complex obstacles with very detailed geometry, our model can deviate near small features. However, it presents divergence-free velocity result that

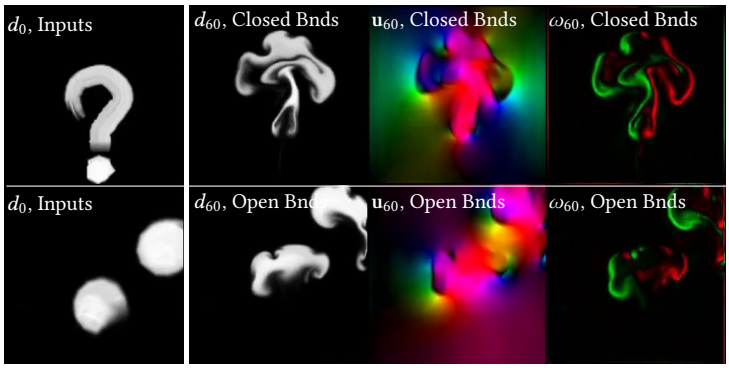


Fig. 9. Our method generalizes well to artificial drawings as inputs.

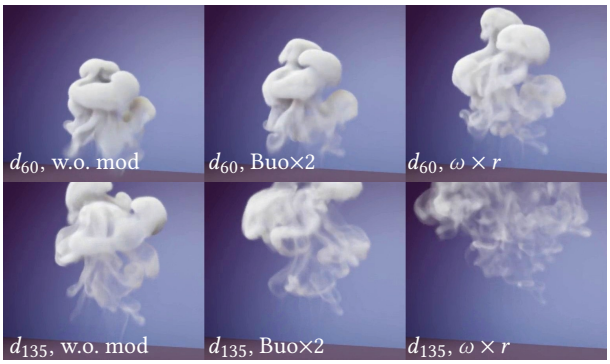


Fig. 10. 3D results with buoyancy and vorticity modifications.

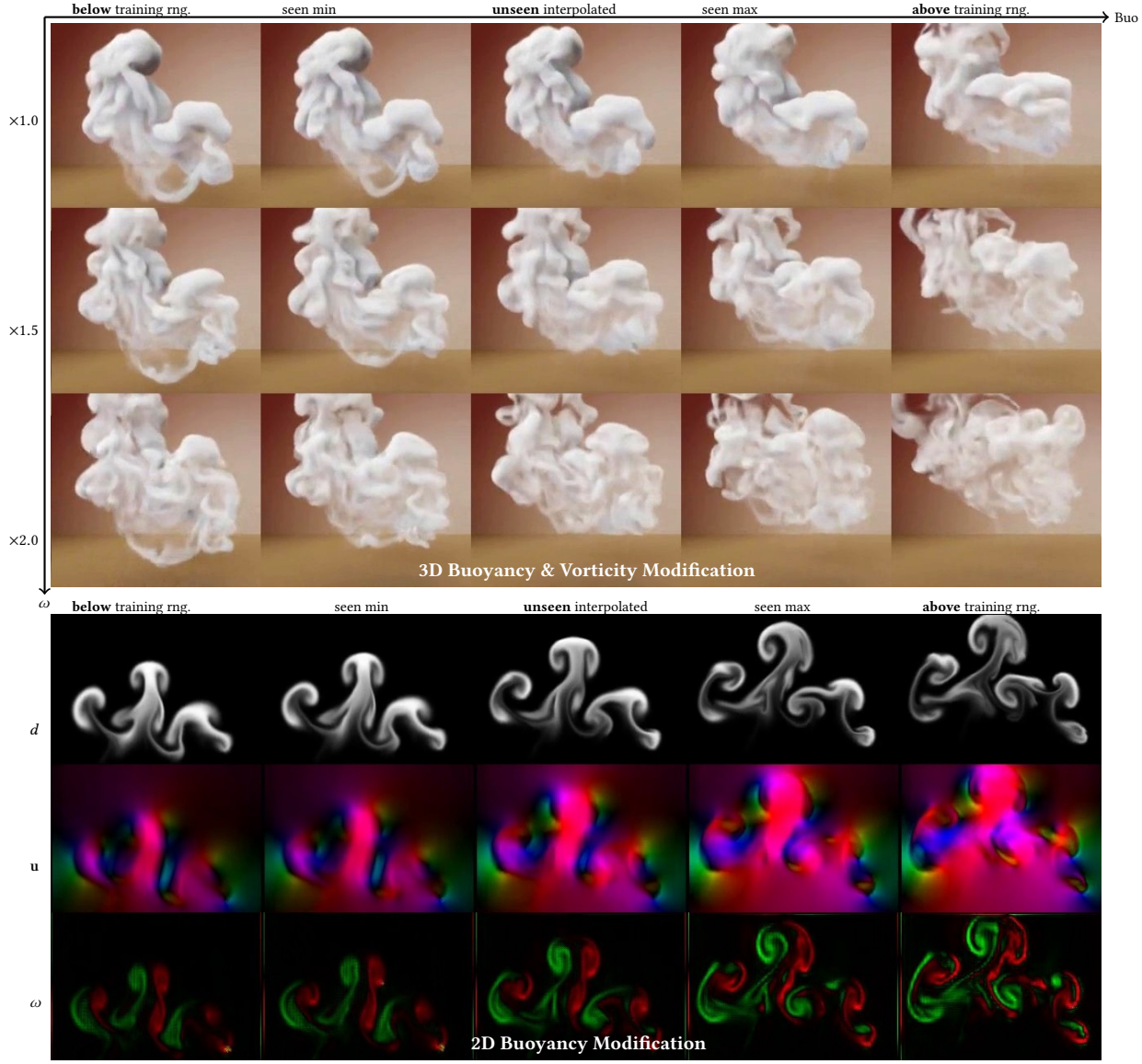


Fig. 11. Our 3D and 2D results change continuously and sensitively with varying controls. Top: we show the 3D case at a resolution of 256^3 with increasing buoyancy along X, and increasing vorticity along Y. Bottom: density, velocity and vorticity results at a resolution of 256^2 when changing buoyancy along X in 2D.

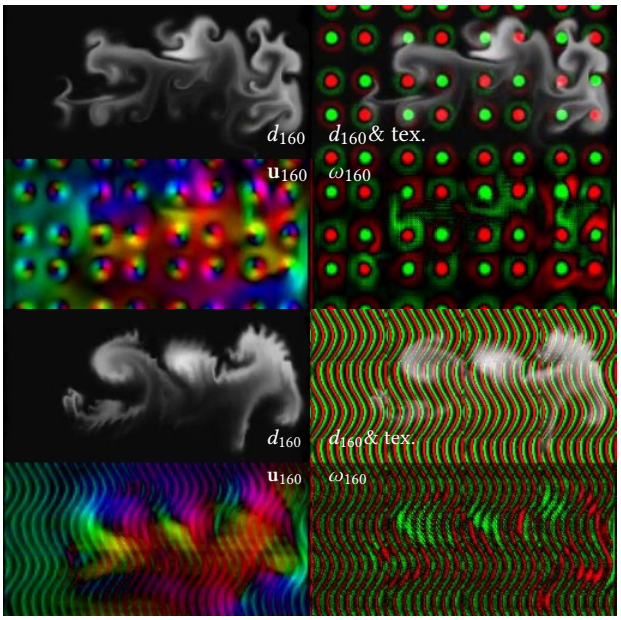


Fig. 12. The results with texture-based vorticity modifications. We show density, velocity, the overlay of density and vorticity texture, and the resulting vorticity.

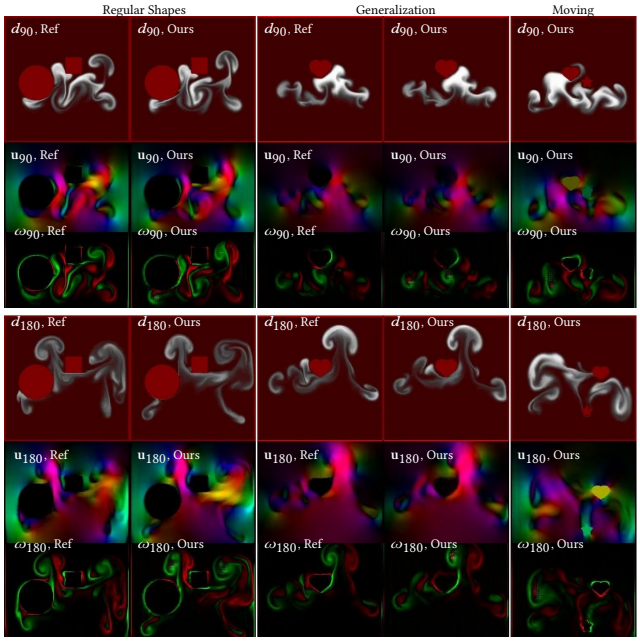


Fig. 13. 2D results with obstacles. Learning from obstacles in regular shapes, our model generalizes well to unseen shapes, and handles moving obstacles successfully.

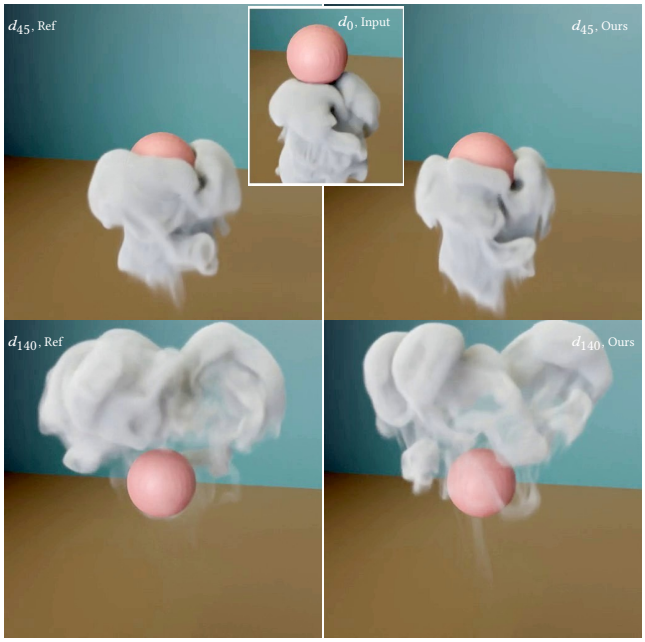


Fig. 14. Based on the initial density frame and the obstacle mask as inputs, the generated velocity manages to respect obstacle boundaries and match the reference well on large scales.

Fig. 15. Users can design simulations by modifying obstacles. Our method generalizes well to obstacles with unseen shapes, e.g., the torus on the left.

corresponds to the overall obstacle settings nicely and efficiently, which is very suitable for user control.

Likewise, 3D results with a spherical obstacle are shown in Fig. 14, and an unseen torus-shaped obstacle is introduced in Fig. 15. A combination of static and moving obstacles (fish) alongside several density sources in the shape of bunnies is shown in Fig. 16.

Our model yields realistic motions for all cases, and requires around 0.7 seconds/frame for simulations with a resolution of 256^3 on Nvidia Tesla v100, which is 10 times faster than the reference

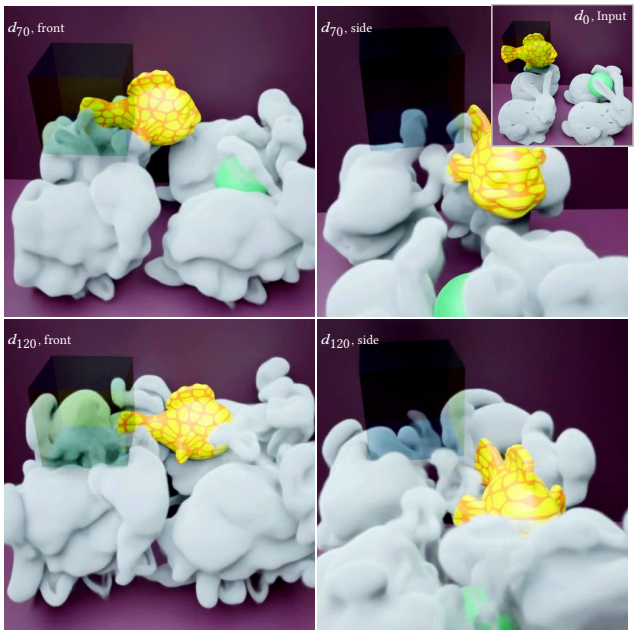


Fig. 16. 3D results with static and moving obstacles. We render the square obstacle transparently for clarity. Simulated with a resolution of 256^3, smoke moves naturally around the fins of the fish, as well as around the transparent square- and green sphere-shaped obstacles.

fluid solver MantaFlow. For a better visual appreciation of our results, the reader is referred to the accompanying media.

5.4 Discussion and Limitations

We have demonstrated that a generative model can be trained to learn and respect a wide range of complex, conditional inputs. However, one limitation of our approach is that the generated result is not guaranteed to closely fulfill all specified goals. Especially for strong modifications of individual parameters, we found that the trained network generates a compromise solution. As shown in Table 1, when enhancing the conditional inputs of KE or ω with a factor of 4, the resulting velocity shows stronger KE or ω, but the resulting sensitivity in terms of $M_{\text{KE}\times 4}$ and $M_{\omega\times 4}$ are less than 4. In the appendix, we test our model on simulations with vorticity confinement to shed more light on this problem and possible solutions, such as enlarging the corresponding sub-networks.

Aiming to provide flexible user controls, our velocity inference depends on the current physical quantities of d, KE, ω and s, without involving historical information. This can lead to ambiguous situations. E.g., when an obstacle abruptly changes it's motion, this change might not yet be reflected in the density configuration. While we believe this is a reasonable compromise, it would be an interesting topic in the future to inject additional historical information as input for the networks.

As a deep-learning algorithm facing a multimodal problem, our result sometimes is a little damped compared to the reference. This is more visible in 3D cases and a larger model with more weights can perform better at the cost of more training resources. Our result can also be less realistic for cases with complex motions around moving or irregular obstacles. Possible improvements can be obtained by using our method for a coarse prediction of the divergence-free velocity, in combination with an efficient post-refinement using a regular solver. Although we offer short run-time, the training time is relatively long. Showing diverse results, we anticipate models trained with our method can support a reasonable range of applications, which would then compensate for the training time.

6 CONCLUSIONS

To the best of our knowledge, we have presented the first density-based velocity generation method via a generative model that learns to represent a large variety of physical behavior. We have demonstrated the learning of meaningful controls via obstacles, physical parameters, kinetic energy and vorticity fields. Using the proposed adversarial learning with cyclic mapping, we achieved highly sensitive and disentangled controls for multiple quantities, as demonstrated in our detailed qualitative and quantitative evaluations. As an efficient tool for velocity control, it would be possible to employ our method for interactive fluid editing by using an optimized model in conjunction with a fast visualization.

With a wide range of results, our method showcases strong generalization and flexibility. While we have focused on density as a visible quantity, a variety of other visual inputs are imaginable to drive simulations: e.g., streamlines or vortex cores. It will be very interesting to extend our method to captured data, e.g., to learn the motion of "dancing droplets" driven by the "Marangoni Effect" from 2D captures. For 3D volumetric smoke data, techniques for density reconstruction from single or multiple views could be helpful. Beyond fluid simulations, we anticipate that our method will provide a good basis for learning other physical problems that require intuitive controls, e.g., cloth, multiphase fluids, or various elasto-plastic materials.

ACKNOWLEDGMENTS

We would like to thank the reviewers and the shepherd for their detailed and constructive comments. We also thank Georg Kohl for instructions on evaluations using LSiM. Christian Theobalt was supported by the ERC Consolidator Grant 4DRepLy (770784). Mengyu Chu was supported by Lise Meitner Postdoctoral Fellowship.

REFERENCES

Thiemo Alldieck, Gerard Pons-Moll, Christian Theobalt, and Marcus Magnor. 2019. Tex2shape: Detailed full human body geometry from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*. 2293–2303.

Amir Arsalan Soltani, Haibin Huang, Jiajun Wu, Tejas D Kulkarni, and Joshua B Tenenbaum. 2017. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1511–1519.

David Berthelot, Thomas Schumm, and Luke Metz. 2017. BEGAN: Boundary Equilibrium Generative Adversarial Networks. arXiv:1703.10717

Robert Bridson. 2015. *Fluid simulation for computer graphics*. CRC press.

Mark Browning, Connelly Barnes, Samantha Ritter, and Adam Finkelstein. 2014. Stylized keyframe animation of fluid simulations. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*. 63–70.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc., Barcelona, 2172–2180.

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics (TOG)* 36, 4 (2017).

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 15–22.

Zahra Forootaninia and Rahul Narain. 2020. Frequency-domain smoke guiding. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–10.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. Curran Associates, Inc., Montreal, 2672–2680.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. *Advances in neural information processing systems* 30 (2017), 5767–5777.

Philipp Holl, Nils Thuerey, and Vladlen Koltun. 2020. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations*.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Fredo Durand. 2019. DiffTaichi: Differentiable Programming for Physical Simulation. In *International Conference on Learning Representations*.

Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. 2018. Multimodal unsupervised image-to-image translation. In *Proceedings of the European conference on computer vision (ECCV)*. 172–189.

Tiffany Inglis, M-L Eckert, James Gregson, and Nils Thuerey. 2017. Primal-Dual Optimization for Fluids. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 354–368.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. 2017. Image-To-Image Translation With Conditional Adversarial Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Hawaii, 1125–1134.

Ondřej Jamriška, Jakub Fišer, Paul Asente, Jingwan Lu, Eli Shechtman, and Daniel Sỳkora. 2015. LazyFluids: appearance transfer for fluid animations. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. 2016. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*. 1–52.

Alexia Jolicoeur-Martineau. 2018. The relativistic discriminator: a key element missing from standard GAN. *arXiv preprint arXiv:1807.00734* (2018).

Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem Rufo. 2019. DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–13.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 8110–8119.

T Kelly, P Guerrero, A Steed, P Wonka, and NJ Mitra. 2018. FrankenGAN: guided detail synthesis for building mass models using style-synchonized GANs. *ACM Transactions on Graphics* 37, 6, Article 216 (November 2018), 14 pages.

Byungsoo Kim, Vinicius C. Azevedo, Markus Gross, and Barbara Solenthaler. 2019a. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Trans. Graph.* 38, 6, Article 188 (Nov. 2019), 11 pages.

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019b. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 59–70.

Theodore Kim, Nils Thuerey, Doug James, and Markus Gross. 2008. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–6.

Georg Kohl, Kiwon Um, and Nils Thuerey. 2020. Learning Similarity Metrics for Numerical Simulations. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5349–5360.

Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2019. Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids. In *Eurographics 2019 - Tutorials*, Wenzel Jakob and Enrico Puppo (Eds.). The Eurographics Association.

Chongyang Ma, Li-Yi Wei, Baining Guo, and Kun Zhou. 2009. Motion field texture synthesis. In *ACM SIGGRAPH Asia 2009 papers*. 1–8.

Pingchuan Ma, Yunsheng Tian, Zherong Pan, Bo Ren, and Dinesh Manocha. 2018. Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. 2017. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 2794–2802.

A. Marzouk, P. Barros, M. Eppe, and S. Wermter. 2019. The Conditional Boundary Equilibrium Generative Adversarial Network and its Application to Facial Attributes. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Budapest, 1–7.

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. 2004. Fluid Control Using the Adjoint Method. *ACM Transactions on Graphics / SIGGRAPH 2004* 23, 3 (Aug. 2004).

Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. arXiv:1411.1784

Joe J Monaghan. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics* 30, 1 (1992), 543–574.

Michael B Nielsen and Robert Bridson. 2011. Guide shapes for high resolution naturalistic liquid simulation. In *ACM SIGGRAPH 2011 papers*. 1–8.

Zherong Pan, Jin Huang, Yiying Tong, Changxi Zheng, and Hujun Bao. 2013. Interactive localized liquid motion editing. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–10.

Zherong Pan and Dinesh Manocha. 2017. Efficient solver for spacetime control of smoke. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. In *NIPS*. 2226–2234.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. 2020. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405* (2020).

Syuhei Sato, Yoshinori Dobashi, Theodore Kim, and Tomoyuki Nishita. 2018. Example-based turbulence style transfer. *ACM Transactions on Graphics (TOG)* 37, 4 (2018).

Lin Shi and Yizhou Yu. 2005. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 229–236.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles A. Sutton. 2017. VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning. In *NIPS*. 3310–3320.

Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 121–128.

Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. 2013. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Tiancheng Sun, Jonathan T Barron, Yun-Ta Tsai, Zexiang Xu, Xueming Yu, Graham Fyffe, Christoph Rhemann, Jay Busch, Paul E Debevec, and Ravi Ramamoorthi. 2019. Single image portrait relighting. *ACM Trans. Graph.* 38, 4 (2019), 79–1.

Simron Thapa, Nianyi Li, and Jinwei Ye. 2020. Dynamic Fluid Surface Reconstruction Using Deep Neural Network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21–30.

Nils Thuerey and Tobias Pfaff. 2018. MantaFlow. *http://mantaflow.com*.

Nils Thürey, Richard Keiser, Mark Pauly, and Ulrich Rüde. 2009. Detail-preserving fluid control. *Graphical Models* 71, 6 (2009), 221–228.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2017. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*. PMLR, 3424–3433.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. 2018. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8798–8807.

Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2019. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer graphics forum*, Vol. 38. 71–82.

Chang Xiao, Peilin Zhong, and Changxi Zheng. 2018. BourGAN: Generative Networks with Metric Embeddings. In *NeurIPS*. 2275–2286.

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 95.

Guowei Yan, Zhili Chen, Jimei Yang, and Huamin Wang. 2020. Interactive liquid splash modeling by user sketches. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–13.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. IEEE, Venice, 2223–2232.

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.

LEARNING MEANINGFUL CONTROLS FOR FLUIDS - APPENDIX

MENGYU CHU, Max Planck Institute for Informatics, SIC, Germany
NILS THUEREY, Technical University of Munich, Germany
HANS-PETER SEIDEL, Max Planck Institute for Informatics, SIC, Germany
CHRISTIAN THEOBALT, Max Planck Institute for Informatics, SIC, Germany
RHALEB ZAYER, Max Planck Institute for Informatics, SIC, Germany

In the following, we first present details on evaluations, Then, we show how to employ our approach for keyframe optimization. Finally, we present concrete examples to illustrate limitations of our method and offer details of training parameters.

A QUALITATIVE AND QUANTITATIVE EVALUATIONS

We evaluate methods in terms of restoration, modification, and generalization of buoyancy directions. While quantitative evaluations are summarized in tables of the main document, we present more details here with qualitative comparisons, discussions, and formulations of applied metrics.

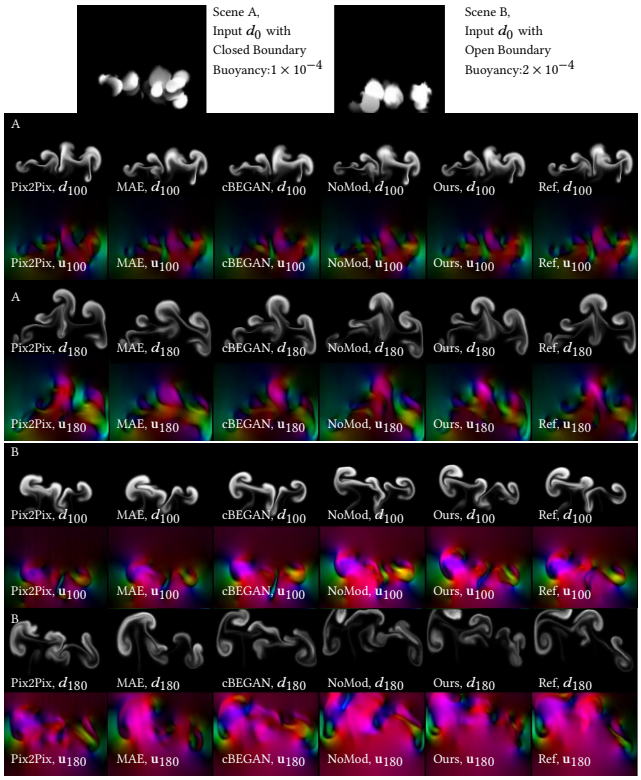


Fig. 17. Restorative results from Pix2Pix, MAE, cBEGAN, NoMod, and our final model. References are shown on the rightmost. Started from the input d_0 on the top, we show the simulation results of frame 100 in the middle and frame 180 on the bottom.

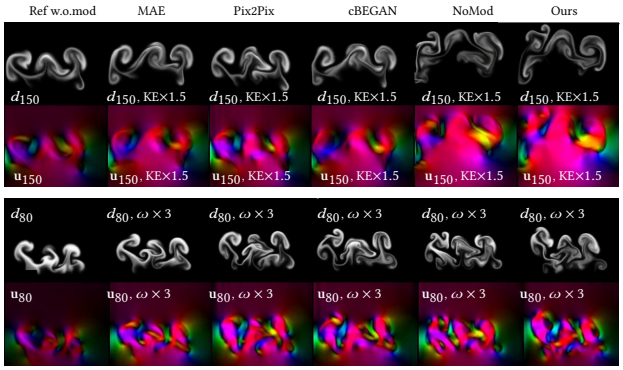


Fig. 18. Results with increased KE (top) and ω (bottom). Compared to the reference on the left, all results are more energetic. While Ours is very responsive to both KE and ω modifications, the NoMod model is not far behind. MAE, Pix2Pix, and cBEGAN are less influenced by the KE modification. MAE ranks the last for ω control, while Pix2Pix and cBEGAN are in between.

Restorative Tests

The goal for the restorative test in Fig. 17 is to restore a sequence of 200 time steps given only a single density frame as input. The results of the aforementioned four models are compared to a reference simulation starting with a non-zero velocity. Starting at the same density input d_0, the simulations gradually deviate from the reference due to the temporal accumulation of errors from the velocity inference step. While results after 100 steps, e.g., d_{100} and $\mathbf{u}_{100}$ are still relatively close for all methods, as shown in Fig. 17, Pix2Pix, MAE, and NoMod show larger deviations for later frames. We notice that the Pix2Pix model has more difficulties with the buoyancy condition. Mixing different modes of buoyancy conditions together, it generates slightly stronger motions for scenes with weak buoyancy like scene A, while much weaker motions are generated for scenes with strong buoyancy, as shown in scene B. Fig. 17 shows the saturation difference of $\mathbf{u}_{180}$ between Pix2Pix and the reference. On frame 180, cBEGAN and Ours still manage to closely match the reference state. This example corresponds to statistics of Table 1, where cBEGAN and Ours achieve lower errors than other models.

Modification Tests

Besides decent restoration accuracy, we demonstrate the sensitivity of our model under modified conditions including kinetic energy, vorticity, buoyancy, and boundary conditions in the following.

KE and ω Modification Tests. The results in Fig. 18 are generated by applying a factor of 1.5 and 3 to the self-encoded KE and ω fields

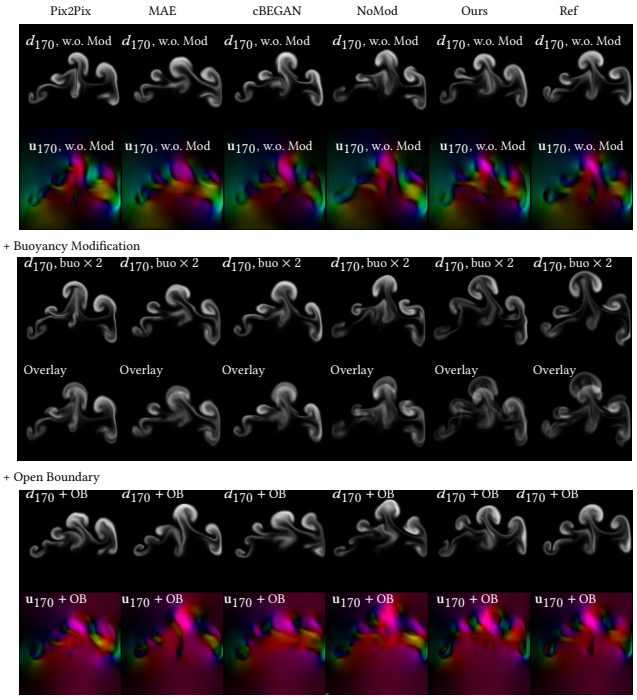


Fig. 19. Results with modified parameters. The original scene has a closed boundary and weak buoyancy. 2nd Row: When increasing the buoyancy, Pix2Pix, MAE, and cBEGAN ignore the modification and repeat their original results. NoMod is influenced slightly. Ours achieves the desired sensitivity. 3rd Row: All models respect the open boundary, however, Ours stays closest to the reference. Others introduce unnecessary deviations.

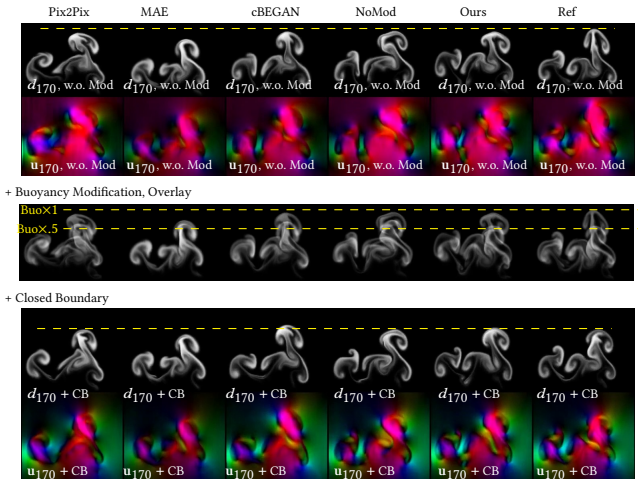


Fig. 20. Results with modified parameters. The original scene has an open boundary and a strong buoyancy. 2nd Row: When decreasing the buoyancy, MAE is barely influenced by the change, while Ours achieves better sensitivity than Pix2Pix, NoMod, and cBEGAN. 3rd Row: All models respect the closed boundary condition. Dashed yellow lines indicate the heights of the reference plumes for clearer comparison.

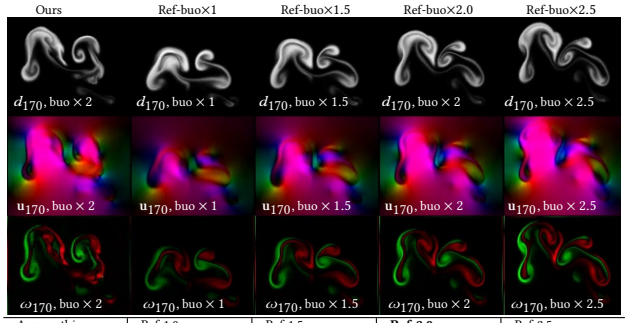
respectively. While NoMod and our full model show strong motions for the increased KE, cBEGAN, Pix2Pix, and MAE largely ignore the modifications. Regarding boosting ω, our full model responds with a large amount of eddies, NoMod is closely behind, cBEGAN and Pix2Pix ranks the third and forth, and MAE is again the least influenced.

Using different factors $a \in \{0.5, 1.0, 2.0, 4.0\}$, we measure the *sensitivity* metric, $M_{a\mathrm{KE}} = \sum \frac{1}{2} G(d, a\mathrm{KE}_r)^2 / \sum \frac{1}{2} G(d)^2$ in Table 1 (KE and ω Modifications) to offer a quantitative evaluation across 1200 density inputs. Here, KE_r stands for the kinetic energy of the reference scene. Likewise, the metric $M_{a\omega} = \sum \|\nabla \times G(d, a\omega_r)\| / \sum \|\nabla \times G(d)\|$ is calculated for vorticity evaluation, with the original ω_r from the reference. When increasing KE or ω, the sensitivity evaluation shows that our method outperforms the others, followed by our adversarial training in NoMod. This is consistent with the qualitative comparison of Fig. 18. Our method likewise yields the best sensitivity in both directions, measured in terms of $\frac{M_{\mathrm{KE}\times 4}}{M_{\mathrm{KE}\times 0.5}}$ and $\frac{M_{\omega\times 4}}{M_{\omega\times 0.5}}$, respectively.

Modifications of **s**. Corresponding to Table 2 of the main paper, we present visual results with modified physical parameters in Fig. 19 and Fig. 20. In Fig. 19, the reference scene in the first row is originally generated with a closed boundary and a buoyancy of 1×10^{-4}. With close restorative accuracy, all results in the first row match the reference well. Changing the buoyancy condition to 2×10^{-4}, the results shown in the second row should spread out more in space. Here, the overlays of the unmodified buoyancy highlight the differences in behavior: While Pix2Pix, MAE, and cBEGAN stay close to their restoration results, ignoring buoyancy changes, the NoMod model yields only a slightly stronger buoyancy. Our full method finally achieves a sensitivity that is very close to the reference.

The results in third row, on the other hand, are generated with a modified boundary condition. With a closed boundary in the original scene, the velocity on the sides of the domain is pointing downwards (as indicated by the green color). After modifying the boundary condition to *open*, the velocity of the reference changes and moves upwards, visualized by a red color. At the same time, we expect the density to rise higher, without overly changing it's local structure. According to the velocity visualizations, all models manage to fulfill the open boundary condition. However, our full model manages to recover the reference with boundary modifications much better than others, indicating a better disentangled control. Similar conclusions can be drawn from Fig. 20, where the scene in the first row originally has an open boundary with a buoyancy of 2×10^{-4}. It is modified by decreasing the buoyancy on the second row and a closed boundary condition in the third row. Our model still presents the most sensitive control, while Pix2Pix, NoMod, and cBEGAN are not far behind. We have observed that lower velocities are typically easier to generate, and hence this modification represents a slightly easier setting than the increased one.

For quantitative evaluations, we compare the modified results with regard to a series of simulations using a regular solver with gradually modified conditions. An example with increased buoyancy is shown in Fig. 21. To measure the distance to simulaiton candidates with buoyancy from 1×10^{-4} to 2.5×10^{-4}, we use $\mathcal{L}^2$, LPIPS [Zhang et al. 2018] and LSiM [Kohl et al. 2020] metrics for this scene over the course of 150 frames. As $\mathcal{L}^2$ is purely local and potentially



Avg. on this scene	Ref-1.0	Ref-1.5	**Ref-2.0**	Ref-2.5
$\mathcal{L}^2 \downarrow$	0.029	0.019	**0.012**	0.014
LPIPS↓	0.200	0.145	**0.113**	0.127
LSiM↓	0.225	0.168	**0.137**	0.160

Fig. 21. Evaluation for increased buoyancy. Compared to references with different buoyancy, our model matches the correct one with closest distances in terms of $\mathcal{L}^2$, LPIPS and LSiM.

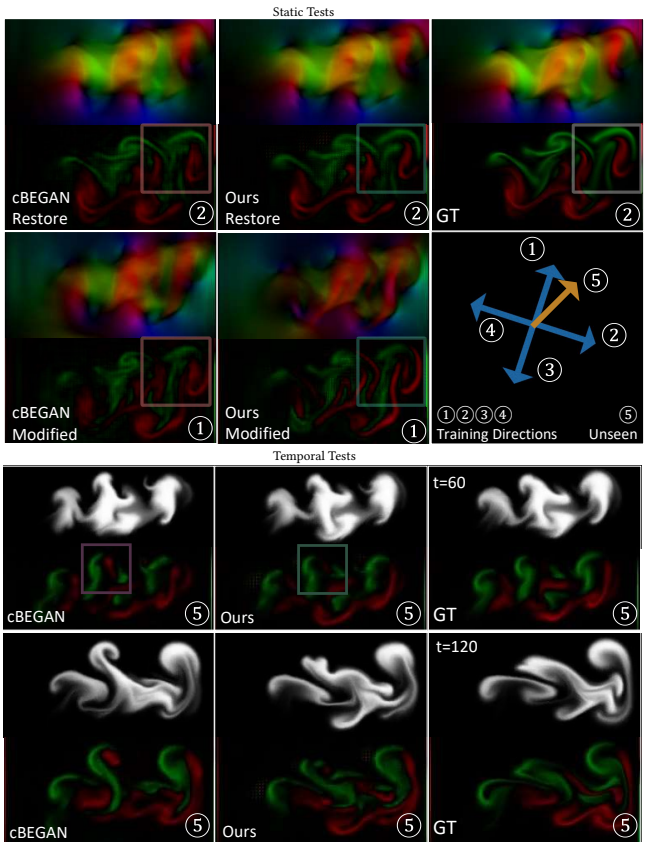
unreliable, we include it only for completeness. LPIPS is widely used for images as a perceptual distance, while LSiM is specialized on numerical simulations. Evaluating them with density fields, their statistics offer a consistent conclusion: Our method matches the correct buoyancy condition of 2.0×10^{-4} with the closest distance, and is least similar to the 1.0×10^{-4} one. Although $\mathcal{L}^2$, LPIPS and LSiM show similar behavior, LSiM offers a clearer separation with larger gaps between candidates. Thus, we focus on LSiM evaluations for the remainder of tests.

Table 2 shows the averaged score for Pix2Pix, MAE, cBEGAN, NoMod, and Ours on 1200 density frames. Our full model ranks first by fulfilling all modifications. While all models work well under easy modifications with weaker buoyancy and closed boundary, differences appear in tough modification tests. In the test with buo×2, while Ours matches the correct reference, NoMod has similar distances to candidates with buo×1.5 and buo×2.0. cBEGAN, MAE, and Pix2Pix perform equally bad, matching false candidates with buo×1.0 and buo×1.5 much better than the correct one. In the open boundary tests, their distances to the true references are likewise much larger in comparison to ours.

Generalization of Buoyancy Directions

In this section, we train and evaluate our method and cBEGAN on a more challenging task with varying buoyancy directions in 2D. By including $(1 + cos\phi, 1 + sin\phi)$ in the latent space of the **s** SubNet, we train the two models on a new dataset. This training dataset contains simulations with four buoyancy directions, i.e. $\phi = 0.4\pi, 1.9\pi, 1.4\pi, 0.9\pi$, shown as blue arrows(①-④) in Fig. 22. On the first row, we show a restoration test where velocity is generated from the density input only. Compared to the ground-truth velocity which has $\phi = 1.9\pi$ (②), our method and cBEGAN behave similarly for the single-frame restoration on the first row. This is also verified by their close static LSiM and L1 errors in the table (**Restoration** of **u**) below. Averaged on 20 sequences with 60 frames each, the temporal LSiM and L1 errors show that our model performs better when used recurrently as a simulator.

On the second row, the direction of $\phi = 0.4\pi$ (①) is used as a modified condition, in addition to the density input. Results of both methods successfully differ from the restoration ones. However,



Restoration of **u**	static LSiM ↓	static L1 ↓	temporal LSiM ↓	temporal L1 ↓	**Modification** Tests	LSiM to True Ref.↓	LSiM to False Candidates.↑
Ours	**0.187**	0.081	**0.209**	**0.102**	Ours	**0.153**	**0.228**
cBEGAN	0.190	**0.080**	0.239	0.112	cBEGAN	0.177	0.213

Fig. 22. Evaluation for varying buoyancy directions.

from their vorticity fields, we can see that ours shows a clear difference, while cBEGAN shows less changed and blurry vorticity. We further evaluate the modified condition task with a temporal test, shown on the last two rows of Fig. 22. Starting from the same initial density setup with a buoyancy direction of $\phi = 0.25\pi$ (⑤), both cBEGAN and Ours are close to the reference for the first 60 time steps. Note that this buoyancy direction is unseen to the models during training and cBEGAN begins to show errors in its vorticity filed after 60 steps. Consequently, in the later time step with $t = 120$, our model follows the reference much better than cBEGAN. This scene is also shown in our supplemental video. On 12 temporal test cases with 200 frames each, we calculate LSiM errors to the true references with $\phi = 0.25\pi$ (⑤) and false candidates with $\phi = 0.05\pi$ or $\phi = 0.45\pi$. The averaged LSiM distances in the table (**Modification** Tests) show that our method can fulfill buoyancy directions more sensitively.

B KEY-FRAME OPTIMIZATIONS

Besides the direct inference of velocity results, our model can be used in optimization tasks due to its inherent differentiability. As a proof of concept, we present an example of a keyframe optimization

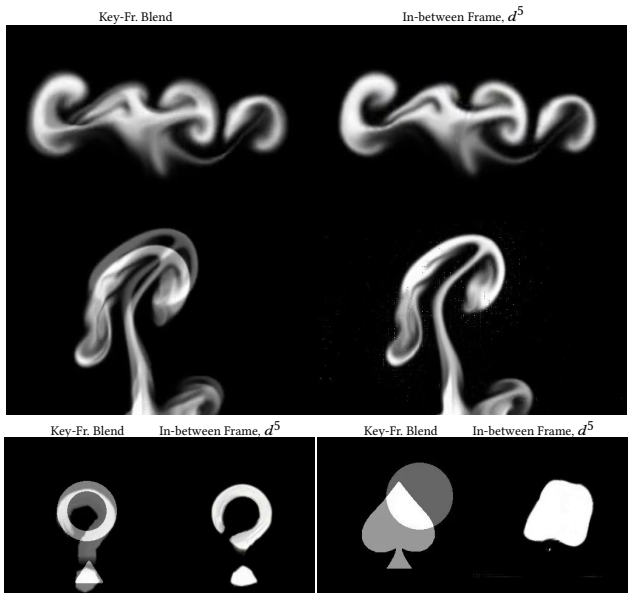


Fig. 23. Tests of keyframe optimizations. From top to bottom and left to right, we test with a normal plume scene, a scene with unknown inflow, a scene with artificial shapes, and a scene with largely changing shapes.

task, with a straightforward objective function and a simple gradient descent algorithm.

Given density configurations at two keyframes as input, we linearly interpolate 9 density frames in between to get the initial density state $ds_0 = [d_0^0, d_0^1, ..., d_0^9, d_0^{10}]$. Here, we use subscripts as iteration steps and the subscripts stand for frame numbers. Taking these 11 frames as density inputs, our velocity inference model is used to calculate their velocities independently, i.e. $\mathbf{u}s_i = [G(d_i^0, \mathbf{s}_i), G(d_i^1, \mathbf{s}_i)..., G(d_i^9, \mathbf{s}_i), G(d_i^{10}, \mathbf{s}_i)]$. We then optimize for the in-between density frames as degrees of freedom, a consistent set of physical parameters for all 11 frames, and a time scale parameter h, i.e. $\{ds_i, \mathbf{s}_i, h\}$, to minimize a short-term goal of

$$\|d_i^{t+1} - \mathcal{A}(d_i^t, hG(d_i^t, \mathbf{s}_i))\| + \|d_i^{t-1} - \mathcal{A}(d_i^t, -hG(d_i^t, \mathbf{s}_i))\|,$$

a long-term goal of

$$\left\| d_i^{10} - \mathcal{A}(d_i^0, \int_{t=1}^{t=9} hG(d_i^t, \mathbf{s}_i)) \right\| + \left\| d_i^0 - \mathcal{A}(d_i^{10}, -\int_{t=1}^{t=9} hG(d_i^t, \mathbf{s}_i)) \right\|.$$

We include a regularization term $\| \frac{\partial hG(d_i^t, \mathbf{s}_i)}{\partial d_i, \mathbf{s}_i, h} \|$. This gradient penalty regularization term helps to stabilize velocity fields.

As shown in Fig. 23 and the supplemental video, we achieve smooth and natural flow with a rising plume scene as input. Optimizing a shallow inflow region below ds_i in addition to $\{ds_i, \mathbf{s}_i, h\}$, our results manages to match keyframes with unknown inflows as well. Applied to artificial shapes, our results transition smoothly, although the motion is driven away from the behaviour of a freely moving plume in order to match the keyframes. Optimizing on the full resolution of 9 density frames, our proof-of-concept implemenation of this optimization is not overly fast: it takes 3000 iterations to converge, spending around 30 minutes for every two keyframes on a Nvidia Tesla v100. A multi-level optimization strategy could be applied in the future to reduce the computation time.

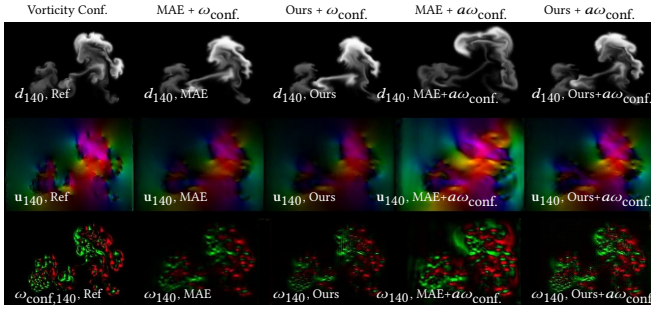


Fig. 24. When taking the ω_{conf} from the vorticity confinement method as an additional condition, the generated velocity contains vorticity (ω_{140}, Ours) that matches the spatial distribution but not the absolute scale of ω_{conf}. With a simple linear scaling, our result could match the vorticity confinement scene much better. As a comparison, MAE doesn't match the spatial distribution and a linear scaling is not helpful for this case.

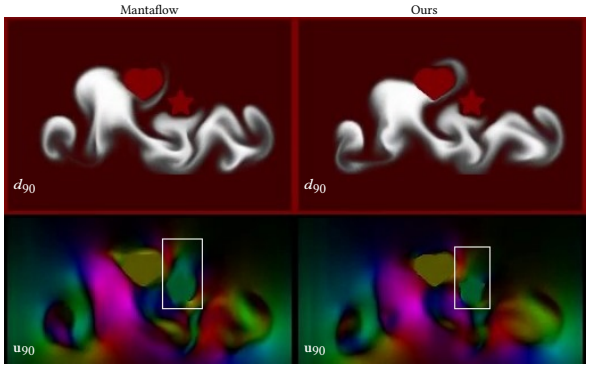


Fig. 25. At locations on the trajectory of the moving obstacle, our velocity output is different to the reference because our model does not have historical information including previous positions and velocities of obstacles.

C FAILURE CASES

In order to test the degree of fulfillment for vorticity control, we apply our model on scenes generated with the vorticity confinement method [Fedkiw et al. 2001], as shown in Fig. 24. Note that simulations with vorticity confinement are very different from our training dataset. Ideally, the modified result with ω_{conf} as a conditional input should resemble the reference scene with vorticity confinement. From Fig. 24 (ω_{140}, Ours), we can see that our result has a reasonable vorticity distribution in space, but the absolute strength is smaller than ω_{conf}, resulting in a different simulation result. By using a simple linearly increased conditional input $a\omega_{\text{conf}}$, with $a_t = \|\omega_{\text{conf},t-1}\| / \|\nabla \times G(d_{t-1}, a_{t-1}\omega_{\text{conf},t-1})\|$, we find that the result can match the reference much better. By contrast, this does not help the MAE model for which the spatial distribution is not matched well. This indicates that the vorticity control of our method is very sensitive to high-frequency features, while, low-frequency features, e.g., the overall strength, can be improved, potentially with a deeper vorticity sub-network.

In Fig. 25, we show our result with regard to a reference simulation with moving obstacles. In this scenes, velocity at the trajectory of a moving obstacles is usually influenced by the historical velocity of the obstacle when passing by. This can cause difficulties for our models when obstacles are moving into the occupied region from empty regions that do not contain density. As shown in the rectangular region of Fig. 25, our result differs from the references for the trajectory of the star-shape obstacle, since our model does not receive the historical information of the obstacle motion. While this could be problematic for accurate velocity inference, it is usually not a big issue for animation purposes, since users are typically less concerned with empty regions in such applications.

D NETWORK ARCHITECTURES AND HYPER-PARAMETERS

In Table 3, we define the notation, specify our network architectures, and give hyper-parameters used for training. Discriminators use almost the same architecture as the generators, except for different numbers of channels for inputs and outputs. The curl operation at the end of generators is also not used in the discriminators. We train 2D adversarial models with the Adam optimizer for 160k iterations. In order to shorten the training time, the first stage with 80k iterations is trained with randomly cropped regions in resolution of 64^2. The second 80k is trained with full resolution of 256^2. Using the Nvidia Tesla v100 GPU with 16G memory, the two stages take 2 and 10 hours receptively. Expanding the 2D network to 3D directly with a spatial resolution of 256^3 would result in a large model requiring huge amounts of GPU memory. Therefore, we add a few convolution and transpose convolution layers at the beginning and end to reduce the spatial resolution of the network, as displayed in Table 3. Based on the reduced size, 3D networks are trained with the Adam optimizer for 80k iterations with an input resolution of 256^3, spending around 180 hours on NVIDIA Quadro RTX 8000 with 48G memory. All training runs use a batch size of 3 and a learning rate of 2×10^{-4}.

Table 3. Notations and Network Architectures

Notation	
spatial resolution in N-D, N =2 or 3	w, which means w^N
bicubic up-/down-scaling by 4	BicUp4/BicDown4
concatenation, point-wise product	$\oplus$, $\odot$
convolution or transposed convolution	$C/CT(\text{input}, \text{reso-lution}, \text{kernel size}, \text{output channel}, \text{stride size})$
instance normalization	InNorm
averaged pool/unpool	pool/unpool (input, target_resolution)

2D Generators	3D Generators
$d \rightarrow d_{in}$;	BicDown4$(d) \rightarrow l_d$;
$o_m \oplus o_{\mathbf{u}} \rightarrow o_{in}$;	$C(d, 256, 7, 8, 2), \rightarrow l_{d0}$;
64 or 256 $\rightarrow$ w;	$l_d + C(l_{d0}, 128, 3, 1, 2), \rightarrow d_{in}$;
	BicDown4$(o_m \oplus o_{\mathbf{u}}) \rightarrow o_{in}$;
	64 $\rightarrow$ w;
without obstacle: $d_{in} \rightarrow l_{in}$; with obstacle: $d_{in} \oplus o_{in} \rightarrow l_{in}$;	
$C(l_{in}, w, 7, 32, 1)$, InNorm, ReLU $\rightarrow l_0$;	
$C(l_0, w, 3, 64, 2)$, InNorm, ReLU $\rightarrow l_1$;	
$C(l_1, w/2, 3, 128, 2)$, InNorm, ReLU $\rightarrow l_2$;	
$ResidualBlock(l_2 + i, w/4) \rightarrow l_{3+i}$ with $i = 0, ..., 5$;	
$CT(l_7, w/4, 3, 64, 2)$, InNorm, ReLU $\rightarrow l_8$;	
$CT(l_8, w/2, 3, 32, 2)$, InNorm, ReLU $\rightarrow l_9$;	
$\nabla \times CT(l_9, w, 7, 3, 1) \rightarrow l_{out}$	$\nabla \times CT(l_9, 64, 7, 3, 3) \rightarrow l_{10}$
	$C(\text{BicUp4}(l_{10}), 256, 7, 8, 1) \rightarrow l_{11}$;
	$C(l_{11}, 256, 3, 3, 1) \rightarrow l_{out}$;

$ResidualBlock(l_2 + i)$	$\mathbf{s}$-$SubNet$
$C(l_{2+i}, \frac{w}{4}, 3, 128, 1)$, InNorm ReLU $\rightarrow t_{2+i}$;	$C(l_x, w/4, 7, 16, 1)$, ReLU $\rightarrow x_1$;
$C(t_{2+i}, w/4, 3, 128, 1)$, InNorm $\rightarrow r_{2+i}$;	pool$(x_1, 8) \rightarrow x_2$;
if i = 0, 1, 2, or 4 : $r_{2+i} + l_{2+i} \rightarrow l_{3+i}$.	flat$(C(x_2, 8, 3, 16, 1)) \rightarrow x_3$;
if i=3:	$C(x_3, 1, 1, 2, 1) \rightarrow \mathbf{s}_{out} \in \mathbb{R}^2$;
r_{2+i}+$l_{2+i} \rightarrow l_x$;	$\mathbf{s}_{out} \oplus \underline{\mathbf{s}} \rightarrow \mathbf{s}_{in}$;
$\mathbf{s}$-$SubNet(l_x) \oplus$KE-$SubNet(l_x) \oplus l_x \rightarrow l_{3+i}$;	
if i=5:	reshape$(C(\mathbf{s}_{in}, 1, 1, 8^N \times 16, 1), 8^N \times 16) \rightarrow x_4$;
r_{2+i}+$l_{2+i} \rightarrow l_w$;	$C(x_4, 8, 7, 16, 1) \rightarrow \mathbf{x}_5$;
ω-$SubNet(l_w) \oplus l_w \rightarrow l_{3+i}$;	unpool$(x_5, w/4) \rightarrow \mathbf{s}$-$SubNet(l_x)$;

KE-$SubNet$	ω-$SubNet$
$C(x_1, w/4, 3, 8, 2)$, InNorm, ReLU $\rightarrow x_6$;	$CT(l_w, w/4, 3, 32, 2)$, InNorm, ReLU $\rightarrow w_1$;
$C(x_6, w/8, 3, 4, 2)$, InNorm $\rightarrow x_7$;	$CT(w_1, w/2, 3, 16, 2)$, InNorm, ReLU $\rightarrow w_2$;
$C(x_7, w/16, 3, 1, 2), \rightarrow \text{KE}_{out} \in \mathbb{R}^{(\frac{w}{16})^{2 \text{ or } 3}}$;	$C(w_2, w, 7, 1 \text{ or } 3, 1) \rightarrow \omega_{out} \in \mathbb{R}^{w^{2 \text{ or } 3} w^3}$;
$\text{KE}_{out} \oplus \underline{\text{KE}} \rightarrow \text{KE}_{in}$;	$\omega_{out} \oplus \underline{\omega} \rightarrow \omega_{in}$;
$C(\text{KE}_{in}, w/16, 3, 2, 1) \rightarrow x_8$;	$C(\omega_{in}, w, 7, 16, 1)$, InNorm, ReLU $\rightarrow w_3$;
$C(x_8, w/16, 3, 1, 1) \rightarrow \mathbf{x}_{10}$;	$C(w_3, w/2, 3, 24, 2)$, InNorm, ReLU $\rightarrow w_4$;
unpool$(x_{10}, w/4) \rightarrow$ KE-$SubNet(l_x)$;	$C(w_4, w/2, 3, 32, 2) \rightarrow \omega$-$SubNet(l_w)$;

2D Parameters	3D Parameters
$\lambda_{adv}, \lambda_{mod}, \lambda_{l1}$ = 0.2, 0.6, 1.0	$\lambda_{adv}, \lambda_{mod}, \lambda_{l1}$ = 0.2, 0.3, 1.0