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

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/RhalebZayer/den2vel.

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

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

ACM Reference Format:

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...
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.
While these simulation algorithms yield impressive and realistic velocity samples, providing a different perspective, our work focuses on producing details from textures [Ma et al. 2009; Jamriška et al. 2015], including. Based on a low-resolution simulation, many techniques synthesize geometric forms, but to understand semantic meaning of visible quantities in order to influence velocity in a physically-plausible manner.

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 Thurey 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.

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 Thurey 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 [Kaplayan 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 (eGAN), 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 et al. 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 Pix2Pix2HD [Wang et al. 2018] achieve 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:

\[
\frac{\partial u}{\partial t} + u \cdot \nabla u = -\frac{1}{\rho} \nabla p + \nu \nabla^2 u + f,
\]

where \( u, p \) denote velocity, pressure, and an external force function \( f = f(d, u, s) \) that depends on marker density, velocity and a set of physical parameters \( s \). For simplicity, we have dropped the \( \nu \) 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:

\[
\frac{\partial u_t}{\partial t} = \mathcal{A}(u_{t-1}, u_{t-1}) + f - \frac{1}{\rho} \nabla p,
\]

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

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 \( u_t = \mathcal{T}(u_{t-1}) \), with function \( \mathcal{T} \) derived from Eq. 2. However, we’d instead like to have \( u_t = \mathcal{G}(d_t) \), with an unknown function \( \mathcal{G} \). As mentioned previously, manually deriving the relationship between \( d_t \) and \( u_t \) 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 \( u \). Hence we approach this problem in a data-driven manner: based on a simulation dataset, we aim for finding the most suitable \( u \) 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 \( \{d_t, u_t\}_n \), the dataset defines a non-linear, high-dimensional, and multimodal mapping between \( d \) and \( u \) 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: \( G(d_t; \theta) \approx u_t \). The network improves its variables \( \theta \) during training, until a local minimum is found for chosen loss function \( \mathcal{L}(u_t, G(d_t; \theta)) \). 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 a data-driven algorithm, it is possible to include influence from other quantities, e.g., the kinetic energy (KE) and the vorticity (\( \omega \)), when paired data is provided. Thus, by extending the training dataset to \( \{\{d, s, KE, \omega\}_{t}^n, u_t\}_n \), we train \( \mathcal{G} : \{d, s, KE, \omega\}_{t}^n \rightarrow u \) to learn a multivariate mapping. Denoted with an underline, \( s, KE \) and \( \omega \) 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 allows 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 \( u_t = G(d_t; \theta) \) is summarized as:

\[
d_t = \mathcal{A}(d_{t-1}, u_{t-1}) \rightarrow G(d_t, s_t, KE_t, \omega_t; \theta),
\]

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 \( L^2 \) 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:

\[
\arg \max_{\theta_G} \min_{\theta_D} \mathcal{L}_P(u, d; \theta_D) - \mathcal{L}_n(G(d, \theta_G), d; \theta_D),
\]

where \( \mathcal{L}_P \) is a function of \( D(u, ...) \) measuring the classification distances on the positive side, and \( \mathcal{L}_n \) is a function of \( D(G(d, ...)) \) for
is crucial for an accurate synthesis of the desired output. It is forced to always estimate \( s, \omega, \) and KE as outputs in the latent-space, which is crucial for an accurate synthesis of the desired output \( u. \) 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 \( u \) with the many-to-one mapping between \( (d, u) \) and \( s, \) 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. Specifically, the physical parameters \( s, \) containing buoyancy and boundary conditions, are learned with a \( s \) sub-net, while a coarse kinetic energy and a detailed vorticity are encoded with KE and \( \omega \) sub-networks individually. While we have \( s \) in shape of \( 1 \times 1 \times 2, \) a \( 16 \times 16 \times 1 \) KE, and a \( 256 \times 256 \times 1 \) \( \omega \) 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 \( s, \) we have an encoding network with convolutional layers, pooling layers, and a fully connected layer in the end. KE sub-net uses convolution-transpose layers to decode. Once 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. Specifically, the physical parameters \( s, \) containing buoyancy and boundary conditions, are learned with a \( s \) sub-net, while a coarse kinetic energy and a detailed vorticity are encoded with KE and \( \omega \) sub-networks individually. While we have \( s \) in shape of \( 1 \times 1 \times 2, \) a \( 16 \times 16 \times 1 \) KE, and a \( 256 \times 256 \times 1 \) \( \omega \) 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 \( s, \) 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, \( \omega \) 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 sub-net uses convolution-transpose layers to decode. Once 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. Specifically, the physical parameters \( s, \) containing buoyancy and boundary conditions, are learned with a \( s \) sub-net, while a coarse kinetic energy and a detailed vorticity are encoded with KE and \( \omega \) sub-networks individually. While we have \( s \) in shape of \( 1 \times 1 \times 2, \) a \( 16 \times 16 \times 1 \) KE, and a \( 256 \times 256 \times 1 \) \( \omega \) 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 \( s, \) 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, \( \omega \) 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.

3.1 Latent-Space Control Using Physical Quantities

Our generator network \( G : [d, s, KE, \omega] \rightarrow [s, KE, \omega, u] \) takes several optional inputs, as indicated by an underline. It is forced to always estimate \( s, KE, \) and \( \omega \) as outputs in the latent-space, which is crucial for an accurate synthesis of the desired output \( u. \) This
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:

\[
\begin{align*}
L_{D,\text{adv}} &= L_p(D(u, ...)) - L_n(D(G(d), ...)), \\
L_{G,\text{adv}} &= L_n(D(G(d), ...)).
\end{align*}
\] (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 \( D : u, d \rightarrow (0, 1) \) for Eq. 5 with cross entropy losses as \( L_p \) and \( L_n \), 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 \( D : u, d \rightarrow u \), the discriminator of cBEGAN tries to restore only the ground-truth samples with

\[
L_p = \|u - D(u, d)\| ; \quad L_n = \|u - D(G(d), d)\|.
\] (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 \( u \) is restored, our discriminator \( D : u \rightarrow d \) thus tries to infer the conditioning \( d \) correctly for ground-truth samples via:

\[
L_p = \|d - D(u)\| ; \quad L_n = \|d - D(G(d))\|.
\] (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 \( u \), while the generator should fool the discriminator by keeping features that correspond to the correct conditioning. Trained to minimize \( L_n \) 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 \( s, \omega, \) and KE using bottleneck layers of the discriminator. In this way, we arrive at a generator and discriminator in the form of \( G : [d, s, KE, \omega] \rightarrow [s, KE, \omega, u] \) and \( D : [u, s, KE, \omega] \rightarrow [s, KE, \omega, d] \). Hence, the discriminator can offer conditional supervision on physical quantities according to its self-encoded values.

With

\[
\begin{align*}
L_p &= \|[d, s, \omega, KE]^\top - D(u)\| ; \\
L_n &= \|[d, s, \omega, KE]^\top - D(G(d, s, KE, \omega))\|,
\end{align*}
\] (8)

our discriminator is trained to infer the correct condition of \( [d, s, \omega, KE]^\top \) 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 \( \gamma = \mathbb{E}[L_p(u)]/\mathbb{E}[L_n(G(d))] \). To summarize, the learning objective for the adversarial part of our network is:

\[
\begin{align*}
L_{D,\text{adv}} &= \|[d, s, \omega, KE]^\top - D(u)\| \\
L_{G,\text{adv}} &= \|[d, s, \omega, KE]^\top - D(G(d, s, KE, \omega))\|, \\
k &= \text{MovingAVG} (\gamma)
\end{align*}
\] (9)

During training, the optional inputs for \( u, s, \omega \) 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, \( D : [u, s, KE, \omega]^\top \rightarrow [s, KE, \omega, d]^\top \), a full cBEGAN model can be trained with \( D : [u, d, s, KE, \omega]^\top \rightarrow u \). Compared to the first row of Fig. 6 with models conditioned by density only, a full conditioning via \( (d, s, KE, \omega) \) offers better accuracy. In the rest of our paper, the cBEGAN model and ours refers to the versions with full conditioning attributes.
While this case is handled properly using our cyclic adversarial help of our discriminator. Specifically, we use a loss function as the modified conditions. More specifically, we use wavelet turbu-

tions to the original KE and choose to first synthesize a modified

developed with modified conditions at training time. As the physical quan-
tities are correlated in the training data, the generative model tends
to change in KE according to simulations in the training dataset.

3.3 Control Disentanglement

When receiving density as the only input, the generator first pre-
dicts the physical quantities 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 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 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 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 y(X) and e0 denoting curl noise evaluation and frequency-based weighting, respectively, it can be summarized as:

\[
\begin{align*}
    u_{LR} & = \text{DownSample}(u, 0.25) \\
    u' & = \text{UpSample}(u_{LR}, 4.0) + 2^{-\frac{1}{2}} e_0 \cdot y(X) \\
    \text{KE'}, \omega' & = \frac{1}{2} u'^2, \nabla \times u'
\end{align*}
\]

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 \[|\text{KE'} - \frac{1}{2} \text{G}_u(d, \text{KE'})|^2\] and \[||\omega' - \nabla \times \text{G}_u(d, \omega')||\], the conditioning of s can only be guaranteed with the help of our discriminator. Specifically, we use a loss function \[||s' - \text{D}_s(\text{G}_u(d, s'))||\]. With a binary mask M randomly deciding which conditions to modify, the learning objective with data augmentation can be summarized as:

\[
\begin{align*}
    u^* & = \text{G}_u(d, M \odot [s', \omega', \text{KE'}])^T + (1-M) \odot \text{FlagValues} \\
    d', s' & = \text{D}_d(\text{G}_u(s', \text{KE'})) + (1-M) \odot \text{FlagValues} \\
    \mathcal{L}_{G, \text{mod}} & = \|[1,M] \odot ([d', s', \text{KE'}])^T - [d', s', \nabla \times u', 2u'^2]^T]\|_1
\end{align*}
\]

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 through 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 \[\lambda_{A, T}\] for stabilization. The final learning objectives for our generator and discriminator are:

\[
\begin{align*}
    \mathcal{L}_G & = \lambda_{adv} \mathcal{L}_{G, \text{adv}} + \lambda_{mod} \mathcal{L}_{G, \text{mod}} + \lambda_{r1} ||[u, s, \omega, \text{KE}]^T - G(d)||_1 \\
    \mathcal{L}_D & = \mathcal{L}_{D, \text{adv}}.
\end{align*}
\]

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 simu-
lations 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 \[2 \times 10^{-4}\] 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 \[G : [d, \omega_m, \omega_g, \text{KE}, \omega] \rightarrow [s, \text{KE}, \omega, u]\] and \[D : [u, \omega_m, \omega_g, \text{KE}, \omega] \rightarrow [s, \text{KE}, \omega, d]\] using additional inputs \[\omega_m, \omega_g\] 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.

<table>
<thead>
<tr>
<th>Models</th>
<th>Restoration, s (L1)</th>
<th>Restoration, s with ref-s (L1)</th>
<th>Restoration, s with self-s (L1)</th>
<th>KE and ω Modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static, temporal</td>
<td>static, temporal</td>
<td>static, temporal</td>
<td>KE × 0.5</td>
</tr>
<tr>
<td>MAE</td>
<td>0.128, 0.197</td>
<td>0.059, 0.097</td>
<td>0.107, 0.142</td>
<td>0.071, 0.149</td>
</tr>
<tr>
<td>Pi2Pix</td>
<td>0.099, 0.123</td>
<td>0.067, 0.122</td>
<td>0.117, 0.160</td>
<td>0.079, 0.186</td>
</tr>
<tr>
<td>NoMod</td>
<td>0.078, 0.146</td>
<td>0.064, 0.107</td>
<td>0.092, 0.126</td>
<td>0.071, 0.156</td>
</tr>
<tr>
<td>Ours</td>
<td>0.065, 0.095</td>
<td>0.058, 0.094</td>
<td>0.101, 0.134</td>
<td>0.085, 0.137</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Models</th>
<th>Buo × 2.0</th>
<th>Buo × 0.5</th>
<th>Avg. to F/T Buo</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAE</td>
<td>0.148, 0.157, 0.203, 0.250</td>
<td>0.186, 0.220, 0.193, 0.252</td>
<td>0.196, 0.167</td>
</tr>
<tr>
<td>Pi2Pix</td>
<td>0.123, 0.183, 0.218, 0.255</td>
<td>0.182, 0.143, 0.206, 0.260</td>
<td>0.210, 0.181</td>
</tr>
<tr>
<td>cBEGAN</td>
<td>0.155, 0.153, 0.194, 0.241</td>
<td>0.181, 0.130, 0.191, 0.249</td>
<td>0.195, 0.162</td>
</tr>
<tr>
<td>NoMod</td>
<td>0.200, 0.171, 0.176, 0.202</td>
<td>0.222, 0.159, 0.175, 0.222</td>
<td>0.199, 0.168</td>
</tr>
<tr>
<td>Ours</td>
<td>0.222, 0.175, 0.179, 0.176</td>
<td>0.187, 0.125, 0.191, 0.245</td>
<td>0.199, 0.147</td>
</tr>
</tbody>
</table>

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 ablational models to illustrate the effects of \( L_{G,adv} \) and \( L_{D,adv} \), as well as the modified conditions via \( L_{G,mod} \). 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 \( L_G = \lambda_1 \| [u, s, \omega, KE]^T - G(d) \| \). 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 \( L_G = \| [u - D(u, [d, s, \omega, KE]^T)] \| \) and \( L_H = \| [u - D(G(d), [d, s, \omega, KE]^T)] \| \), we obtain a model resembling cBEGAN [Marzouk et al. 2019]. Correspondingly, training with our adversarial losses of Eq. 8 but without \( L_{G,mod} \), we get the NoMod model. The final model, Ours, is additionally trained with \( L_{G,mod} \), 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 \( L^1 \) 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 s are likewise calculated.

From Table 1, we can see that MAE model performs poorly for the estimation of 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 u, with cBEGAN being very close to Ours, both outperforming the other variants. Our final model achieves the best performance in terms of 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 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.
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 due to 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 $L_{G,mod}$, 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, $\omega$, 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 $G : [d, o_m, o_u, s, KE, \omega]^{\top} \rightarrow [s, KE, \omega, u]^{\top}$ and $D : [u, o_m, o_u, s, KE, \omega]^{\top} \rightarrow [s, KE, \omega, d]^{\top}$ with additional inputs $o_m, o_u$, 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.

ACM Trans. Graph., Vol. 40, No. 4, Article 100. Publication date: August 2021.
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
We have demonstrated that a generative model can be trained to perform better at the cost of more training resources. Our result can be an interesting topic in the future to inject additional historical information as input for the networks.

Aiming to provide flexible user controls, our velocity inference depends on the current physical quantities of $d$, KE, $\omega$ and $s$, without involving historical information. This can lead to ambiguous situations. E.g., when an obstacle abruptly changes its 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
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.

#### 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 \( u_{100} \) 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 \( u_{180} \) 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 \( \omega \) Modification Tests

The results in Fig. 18 are generated by applying a factor of 1.5 and 3 to the self-encoded KE and \( \omega \) fields.
With close restorative accuracy, all results in the first row match $\omega$ (KE and $\omega$ ACM Trans. Graph., Vol. 40, No. 4, Article 100. Publication date: August 2021.

1200 density inputs. Here, KE stands for the kinetic energy of the

$r$ Overlay Overlay Overlay Overlay Overlay Overlay $d$

respective. While NoMod and our full model show strong motions for the increased KE, cBEGAN, Pix2Pix, and MAE largely ignore the modifications. Regarding boosting $\omega$, 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 $a \in \{0.5, 1.0, 2.0, 4.0, 10.0\}$, we measure the sensitivity metric, $M_{\text{ske}} = \frac{1}{2} G(d, aKEr)^2 / \frac{1}{2} G(d)^2$ in Table 1 (KE and $\omega$ 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 $M_{\text{sev}} = \frac{1}{2} \| \nabla \times G(d, a\omega_r) \| / \| \nabla \times G(d) \|$ is calculated for vorticity evaluation, with the original $\omega_r$ from the reference. When increasing KE or $\omega$, 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 $\frac{M_{\text{ske}}}{M_{\text{sev}}}$. 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 $1 \times 10^{-4}$. With close restorative accuracy, all results in the first row match the reference well. Changing the buoyancy condition to $2 \times 10^{-4}$, 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 $2 \times 10^{-4}$. 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 simulaton candidates with buoyancy from $1 \times 10^{-4}$ to $2.5 \times 10^{-4}$, we use $L^2$, LPIPS [Zhang et al. 2018] and LSIM [Kohl et al. 2020] metrics for this scene over the course of 150 frames. As $L^2$ is purely local and potentially

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.

ACM Trans. Graph., Vol. 40, No. 4, Article 100. Publication date: August 2021.
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 one with closest distances in terms of $L^2$, LPIPS and LSiM.

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, from their vorticity fields, we can see that ours shows a clearer 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 $\phi = 2.5\pi$ (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 reference 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
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 \( d_s = [d_i^0, d_i^1, ..., d_i^9, d_i^{10}] \). 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. \( u_{si} = [G(d_i^0, s_i), G(d_i^1, s_i), ..., G(d_i^9, s_i), G(d_i^{10}, s_i)] \). 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. \( \{d_i, s_i, h\} \), to minimize a short-term goal of

\[
\|d_i^{t+1} - A(d_i^t, hG(d_i^t, s_i))\| + \|d_i^{t-1} - A(d_i^t, -hG(d_i^t, s_i))\|,
\]

a long-term goal of

\[
\|d_i^{10} - A(d_i^0, \int_{t=1}^{t=9} hG(d_i^t, s_i))\| + \|d_i^0 - A(d_i^0, -\int_{t=1}^{t=9} hG(d_i^t, s_i))\|.
\]

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

As shown in Fig. 23 and the supplementary video, we achieve smooth and natural flow with a rising plume scene as input. Optimizing a shallow inflow region below \( d_s \) in addition to \( \{d_i, s_i, h\} \), our results manage 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 implementation of this optimization is not overly fast: it takes 3000 iterations to converge, spending around 30 minutes for every two keyframes.

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.

On a Nvidia Tesla v100. A multi-level optimization strategy could be applied in the future to reduce the computation time.

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 \( \omega_{conf} \) as a conditional input should resemble the reference scene with vorticity confinement. From Fig. 24 (\( \omega_{conf}, Ours \)), we can see that our result has a reasonable vorticity distribution in space, but the absolute strength is smaller than \( \omega_{conf} \). Resulting in a different simulation result. By using a simple linearly increased conditional input \( \omega_{conf} \), with \( \omega = \|\omega_{conf,t-1}\| \|\nabla \times G(d_{t-1}, \omega_{conf,t-1})\| \), 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.

Fig. 24. When taking the \( \omega_{conf} \) from the vorticity confinement method as an additional condition, the generated velocity contains vorticity \( (\omega_{conf}, Ours) \) that matches the spatial distribution but not the absolute scale of \( \omega_{conf} \). 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.
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 respectively. 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 $2 \times 10^{-4}$.

<table>
<thead>
<tr>
<th>Notation</th>
<th>2D Generators</th>
<th>3D Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>w, which means $w^N$</td>
<td>BicUp4/BicDown4</td>
<td>BicDown4($d$) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>bicubic up-/down-scaling by 4</td>
<td>$\bigcirc$</td>
<td>C($d$, 256, 7, 8, 2) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>concatenation, point-wise product convolution or transposed convolution</td>
<td>instance normalization</td>
<td>$C(d, 256, 128, 3, 1, 2, )$ $\rightarrow \omega$;</td>
</tr>
<tr>
<td>averaged pool/unpool</td>
<td>pool/unpool</td>
<td>$\omega$ $\rightarrow$ pool/unpool (input, target_resolution)</td>
</tr>
</tbody>
</table>

### Table 3. Notations and Network Architectures

<table>
<thead>
<tr>
<th>Notation</th>
<th>2D Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>w, which means $w^N$</td>
<td>BicUp4/BicDown4</td>
</tr>
<tr>
<td>bicubic up-/down-scaling by 4</td>
<td>$\bigcirc$</td>
</tr>
<tr>
<td>concatenation, point-wise product convolution or transposed convolution</td>
<td>instance normalization</td>
</tr>
<tr>
<td>averaged pool/unpool</td>
<td>pool/unpool</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D Generators</th>
<th>3D Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{D}$ $\rightarrow \mathcal{D}$</td>
<td>BicDown4($d$) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{O} \oplus \mathcal{O}$ $\rightarrow \mathcal{O}$</td>
<td>C($d$, 256, 7, 8, 2) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{R}$ $\rightarrow \mathcal{R}$</td>
<td>$C(d, 256, 128, 3, 1, 2, )$ $\rightarrow \omega$;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResidualBlock($l_i + 1$)</th>
<th>s-SubNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nabla \times \mathcal{C}(l_i, w, 7, 3, 1) \rightarrow \omega$</td>
<td>$\mathcal{C}(\text{BicUp4}(l_0), 256, 7, 8, 1) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C} \mathcal{S} \cup \mathcal{S} \mathcal{S} \mathcal{S} \mathcal{S}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KE-SubNet</th>
<th>$\omega$-SubNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{C} \mathcal{F}(x_1, w, 4, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(\text{BicUp4}(l_0), 256, 7, 8, 1) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C}(x_2, w, 8, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C}(x_3, w, 4, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
</tbody>
</table>

### Table 3. Notations and Network Architectures

<table>
<thead>
<tr>
<th>Notation</th>
<th>2D Generators</th>
<th>3D Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>w, which means $w^N$</td>
<td>BicUp4/BicDown4</td>
<td>BicDown4($d$) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>bicubic up-/down-scaling by 4</td>
<td>$\bigcirc$</td>
<td>C($d$, 256, 7, 8, 2) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>concatenation, point-wise product convolution or transposed convolution</td>
<td>instance normalization</td>
<td>$C(d, 256, 128, 3, 1, 2, )$ $\rightarrow \omega$;</td>
</tr>
<tr>
<td>averaged pool/unpool</td>
<td>pool/unpool</td>
<td>$\omega$ $\rightarrow$ pool/unpool (input, target_resolution)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2D Generators</th>
<th>3D Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{D}$ $\rightarrow \mathcal{D}$</td>
<td>BicDown4($d$) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{O} \oplus \mathcal{O}$ $\rightarrow \mathcal{O}$</td>
<td>C($d$, 256, 7, 8, 2) $\rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{R}$ $\rightarrow \mathcal{R}$</td>
<td>$C(d, 256, 128, 3, 1, 2, )$ $\rightarrow \omega$;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ResidualBlock($l_i + 1$)</th>
<th>s-SubNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\nabla \times \mathcal{C}(l_i, w, 7, 3, 1) \rightarrow \omega$</td>
<td>$\mathcal{C}(\text{BicUp4}(l_0), 256, 7, 8, 1) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C} \mathcal{S} \cup \mathcal{S} \mathcal{S} \mathcal{S} \mathcal{S}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KE-SubNet</th>
<th>$\omega$-SubNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{C} \mathcal{F}(x_1, w, 4, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(\text{BicUp4}(l_0), 256, 7, 8, 1) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C}(x_2, w, 8, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
<tr>
<td>$\mathcal{C}(x_3, w, 4, 3, 3, 2, )$ $\rightarrow \mathcal{R}$</td>
<td>$\mathcal{C}(l_1(11), 256, 3, 1, 3) \rightarrow \omega$;</td>
</tr>
</tbody>
</table>