Deep Fluids: A Generative Network for Parameterized Fluid Simulations

BYUNGSOO KIM, ETH Zurich
VINICIUS C. AZEVEDO, ETH Zurich
NILS THUEREY, Technical University of Munich
THEODORE KIM, Pixar Animation Studios
MARKUS GROSS, ETH Zurich
BARBARA SOLENTHALER, ETH Zurich

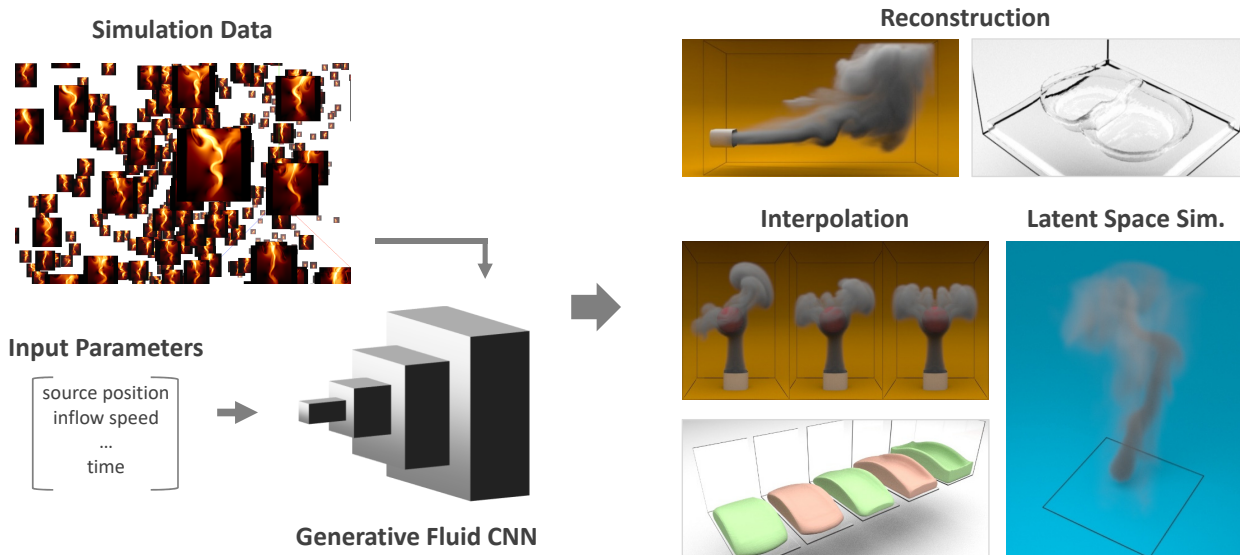


Fig. 1. Our generative neural network can synthesize fluid velocities continuously in space and time, using a set of input simulations for training and a few parameters for generation. This enables fast reconstruction of velocity fields, continuous interpolation and latent space simulations.

This paper presents a novel generative model to synthesize fluid simulations from a set of reduced parameters. A convolutional neural network is trained on a collection of discrete, parameterizable fluid simulation velocity fields. Due to the capability of deep learning architectures to learn representative features of the data, our generative model is able to accurately approximate the training data set, while providing plausible interpolated in-betweens. The proposed generative model is optimized for fluids by a novel loss function that guarantees divergence-free velocity fields at all times. In addition, we demonstrate that we can handle complex parameterizations in reduced spaces, and advance simulations in time by integrating in the latent space with a second network. Our method models a wide variety of fluid behaviors, thus enabling applications such as fast construction of simulations, interpolation of fluids with different parameters, time re-sampling, latent space simulations, and compression of fluid simulation data. Reconstructed velocity fields are generated up to 700× faster than traditional CPU solvers, while achieving compression rates of over 1300×.

Additional Key Words and Phrases: physically-based animation, fluid simulation, deep learning, generative network

ACM Reference Format:
Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2018. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. 1, 1 (June 2018), 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Byungsoo Kim, ETH Zurich, kimby@inf.ethz.ch; Vinicius C. Azevedo, ETH Zurich, vinicius.azevedo@inf.ethz.ch; Nils Thuerey, Technical University of Munich, nils.thuerey@tum.de; Theodore Kim, Pixar Animation Studios, tkim@pixar.com; Markus Gross, ETH Zurich, grossm@inf.ethz.ch; Barbara Solenthaler, ETH Zurich, solenthaler@inf.ethz.ch.

XXXX-XXXX/2018/6-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Machine learning techniques have become pervasive in recent years due to numerous algorithmic advances and the accessibility of computational power. Accordingly, they have been adopted for many applications in graphics, such as generating terrains [Guerin et al. 2017], high-resolution faces synthesis [Karras et al. 2017] and cloud

rendering [Kallweit et al. 2017]. In fluid simulation, machine learning techniques have been used to replace [Ladický et al. 2015], speed up [Tompson et al. 2016] or enhance existing solvers [Xie et al. 2018].

Given the amount of available fluid simulation data, data-driven approaches have emerged as attractive solutions. Subspace solvers [Treuille et al. 2006], fluid re-simulators [Kim and Delaney 2013] and basis compressors [Jones et al. 2016] are examples of recent efforts in this direction. However, these methods usually represent fluids using linear basis functions, e.g., constructed via Singular Value Decomposition (SVD), which are less efficient than their non-linear counterparts. In this sense, deep generative models implemented by convolutional neural networks (CNNs) show promise for representing data in reduced dimensions due to their capability to tailor non-linear functions to input data.

In this paper, we propose the first generative neural network - the Deep Fluids network [anonymous 2018]- that fully constructs dynamic Eulerian fluid simulation velocities from a set of reduced parameters. Given a set of discrete, parameterizable simulation examples, our deep learning architecture generates velocity fields that are incompressible by construction. In contrast to previous subspace methods [Kim and Delaney 2013], our network achieves a wide variety of fluid behaviors, ranging from turbulent smoke to gooey liquids (Figure 1).

The Deep Fluids CNN enhances the state of the art of reduced-order methods (ROMs) in four ways: efficient evaluation time, a natural non-linear representation for interpolation, data compression capability and a novel approach for latent space simulations. Our CNN can generate a full velocity field in constant time, contrasting with previous approaches which are only efficient for sparse reconstructions [Treuille et al. 2006]. Thanks to its 700× speedup compared to regular simulations, our approach is particularly suitable for animating physical phenomena in real-time applications such as games, VR and surgery simulators.

Our method is not only capable of accurately and efficiently recovering learned fluid states, but also generates plausible velocity fields for input parameters that have no direct correspondence in the training data. This is possible due to the inherent capability of deep learning architectures to learn representative features of the data. Having a smooth velocity field reconstruction when continuously exploring the parameter space enables various applications that are particularly useful for the prototyping of expensive fluid simulations: fast construction of simulations, interpolation of fluids with different parameters, and time re-sampling. To handle applications with extended parameterizations such as the moving smoke scene shown in Section 5.2, we couple an encoder architecture with a latent space integration network. This allows us to advance a simulation in time by generating a sequence of suitable latent codes.

Additionally, the proposed architecture works as a powerful compression algorithm for velocity fields, achieving up to 1300× compression rates. This is at least two orders of magnitude greater than previous works [Jones et al. 2016], while simultaneously maintaining a close correspondence to the original data (Figure 2).

To summarize, the technical contributions of our work include:

- The first generative deep learning architecture that fully synthesizes plausible and fully divergence-free 2-D and 3-D fluid simulation velocities from a set of reduced parameters.
- A generative model for fluids that accurately encodes parameterizable velocity fields. Compression rates of 1300× are achieved, as well as 700× performance speed-ups compared to traditional CPU-based fluid solvers.
- An approach to encode simulation classes into a latent space representation through an autoencoder architecture. In combination with a latent space integration network to advance time, our approach allows flexible interactions with flow simulations.
- A detailed analysis of the proposed approach, both when reconstructing samples that have a direct correspondence with the training data set as well as intermediate points in the parameter space.

2 RELATED WORK

Reduced-order Methods. Subspace solvers aim to accelerate simulations by discovering simplified representations. In engineering, these techniques go back decades [Lumley 1967], but were introduced to computer graphics by Treuille et al. [2006] and Gupta and Narasimhan [2007]. Since then, improvements have been made to make them modular [Wicke et al. 2009], consistent with widely-used integrators [Kim and Delaney 2013], more energy-preserving [Liu et al. 2015] and memory-efficient [Jones et al. 2016]. A related "Laplacian Eigenfunctions" approach [De Witt et al. 2012] has also been introduced and refined [Gerszewski et al. 2015], removing the need for snapshot training data when computing the linear subspace. The basis functions used by these methods are all linear however, and various methods are then used to coerce the state of the system onto some non-linear manifold. Exploring the use of non-linear functions, as we do here, is a natural evolution.

One well-known limitation of reduced-order methods is their inability to simulate liquids because the non-linearity of the liquid interface causes the subspace dimensionality to explode. For example, in solid-fluid coupling, usually the fluid is computed directly while only the solid uses the reduced model [Lu et al. 2016]. Graph-based methods for precomputing liquid motions [Stanton 2014] have had some success, but only under severe constraints, e.g. the user viewpoint must be fixed. In contrast, we show that the non-linearity of a CNN-based approach allows it to be applied to liquids as well.

Machine Learning & Fluids. Combining fluid solvers with machine learning techniques was first demonstrated by Ladický et al. [2015]. By employing Regression Forests to approximate the Navier-Stokes equations on a Lagrangian system, particle positions and velocities were predicted with respect to input parameters for a next time step. Regression Forests are highly efficient, but require handcrafted features that lack the generality and abstraction power of CNNs. An LSTM-based method for predicting changes of the pressure field for multiple subsequent time steps has been presented by Wiewel et al. [2018], resulting in significant speed-ups of the pressure solver. For a single time step, a CNN-based pressure projection was likewise proposed [Tompson et al. 2016; Yang et al. 2016]. In contrast to our

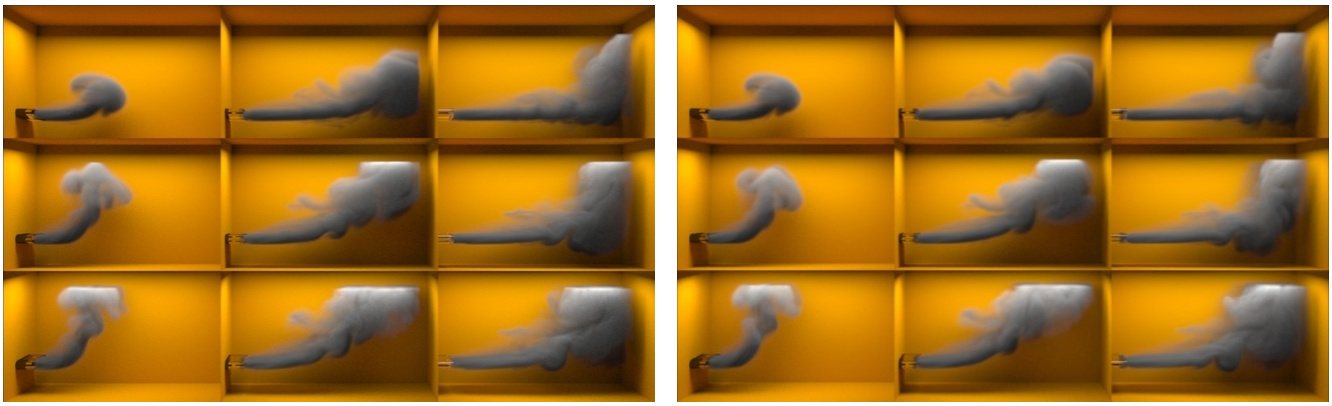


Fig. 2. Ground truth (left) and the CNN-reconstructed results (right) for nine sample simulations with varying buoyancy (rows) and inflow velocity (columns). Despite the strongly varying dynamics of the ground truth simulations, the outputs of our trained model closely reconstruct the reference data.

work, these models only replace the pressure projection stage of the solver, and hence are specifically designed to accelerate the enforcement of divergence-freeness. To visually enhance low resolution simulations, Chu and Thuerey [2017] synthesized smoke details by looking up pre-computed patches using CNN-based descriptors, while Xie et al. [2018] proposed a GAN for super resolution smoke flows. Other works enhance FLIP simulations with a learned splash model [Um et al. 2017], while the deformation learning proposed by [Prantl et al. 2017] shares our goal of capturing sets of fluid simulations. However, it only partially employs CNNs and focuses on signed distance functions, while our work targets the velocity spaces of a broad class of fluid simulations.

Lattice-Boltzmann steady-state flow solutions are recovered by CNN surrogates using signed distance functions as input boundary conditions in [Guo et al. 2016]. Farimani et al. [2017] use Generative Adversarial Networks (GANs) [Goodfellow et al. 2014] to train steady state heat conduction and steady incompressible flow solvers. Their method is only demonstrated in 2-D and the interpolation capabilities of their architecture are not explored. For both methods, the simulation input is a set of parameterized initial conditions defined over a spatial grid, and the output is a single steady state solution. Recently, Umetani and Bickel [2018] developed a data-driven technique to predict fluid flows around bodies for interactive shape design, while Ma et al. [2018] have demonstrated deep learning based fluid interactions with rigid bodies.

Machine Learning & Physics. In the physics community, neural networks and deep learning architectures for approximating, enhancing and modeling solutions to complex physics problems are gaining new attention. A few recent examples are Carleo and Troyer [2016] using a reinforcement learning scheme to reduce the complexity of a quantum many-body problem, and Ling et al. [2016] employing deep neural networks to synthesize Reynolds average turbulence anisotropy tensors from high-fidelity simulation data. Similarly, Hanna et al. [2017] use Regression Forests to enhance local coarse grid patches with a surrogate model, Carrasquilla and Melko [2017] model condensed matter physics with CNNs, and Paganini et al. [2017] model calorimeter interactions with electromagnetic showers using GANs. GANs have also been employed to generate [Ravanbakhsh et al. 2016] and deconvolve [Schawinski et al. 2017] galaxy images, and reconstruct three-dimensional porous media [Mosser et al. 2017]. As we focus on generative networks for known parameterizations, we will not employ learned, adversarial losses. Rather, we will demonstrate that a high quality representation can be learned by constructing a suitable direct loss function.

3 A GENERATIVE MODEL FOR FLUIDS

Fluids are traditionally simulated by solving the inviscid momentum $D\mathbf{u}/Dt = -\nabla p + \mathbf{g}$ and mass conservation $\nabla \cdot \mathbf{u} = 0$ equations, where $\mathbf{u}$ and p are the fluid velocity and pressure, $D\mathbf{u}/Dt$ is the material derivative and $\mathbf{g}$ represents external forces. The viscosity $-\mu\nabla^2\mathbf{u}$ can be included, but simulations for visual effects usually rely on numerical dissipation instead. For a given set of simulated fluid examples, our goal is to train a CNN that approximates the original velocity field data set. By minimizing loss functions with subsequent convolutions applied to its input, CNNs organize the data manifold into shift-invariant feature maps.

Numerical fluid solvers work by advancing a set of fully specified initial conditions. By focusing on scenes that are initially parameterizable by a handful of variables, such as the position of a smoke source, we are able to generate samples for a chosen class of simulations. Thus, the inputs for our method are parameterizable data sets, and we demonstrate that accurate generative networks can be trained in a supervised way.

3.1 Loss Function for Velocity Reconstruction

The network's input is characterized by a pair $[\mathbf{u}_c, \mathbf{c}]$, where $\mathbf{u}_c \in \mathbb{R}^{H\times W\times D\times V_{\text{dim}}}$ is a single velocity vector field frame in V_{dim} dimensions (i.e. $V_{\text{dim}} = 2$ for 2-D and $V_{\text{dim}} = 3$ for 3-D) with height H, width W and depth D, generated using the solver's parameters $\mathbf{c} = [c_1, c_2, ..., c_n] \in \mathbb{R}^n$. For the 2-D example in Figure 3, $\mathbf{c}$ is the combination of x-position and width of the smoke source, and the current time of the frame. Due to the inherent non-linear nature of the Navier-Stokes equations, these three parameters (i.e. position, width, and time) yield a vastly different set of velocity outputs.

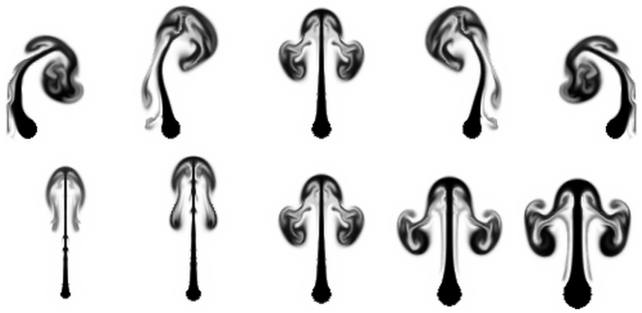
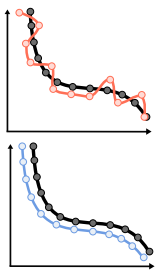


Fig. 3. Different snapshots showing the advected densities for varying smoke source parameters. The top row shows the variation of the initial position, while the bottom row shows the variation of the source width.

For fitting fluid samples, our network uses velocity-parameter pairs and updates its internal weights by minimizing a loss function. This process is repeated by random batches until the network minimizes a loss function over all the training data. While previous works have proposed loss functions for natural images, e.g., L_p norms, MS-SSIM [Zhao et al. 2015], and perceptual losses [Johnson et al. 2016; Ledig et al. 2016], accurate reconstructions of velocity fields have not been investigated. For fluid dynamics it is especially important to ensure conservation of mass, i.e., to ensure divergence-free motion for incompressible flows. We therefore propose a novel *stream function* based loss function defined as

$$L_G(\mathbf{c}) = ||\mathbf{u_c} - \nabla \times G(\mathbf{c})||_1. \tag{1}$$

$G(\mathbf{c}) : \mathbb{R}^n \mapsto \mathbb{R}^{H\times W\times D\times G_{\text{dim}}}$ is the output of our network and $\mathbf{u_c}$ is a simulation sample from the training data set. The curl of the model output is the reconstruction target, and it is *guaranteed* to be divergence-free by construction, as $\nabla \cdot (\nabla \times G(\mathbf{c})) = 0$. Thus, $G(\mathbf{c})$ implicitly learns to approximate a stream function $\Psi_\mathbf{c}$ (i.e. $G_{\text{dim}} = 1$ for 2-D and $G_{\text{dim}} = 3$ for 3-D) that corresponds to a velocity sample $\mathbf{u_c}$. While this formulation is highly suitable for incompressible flows, regions with partially divergent motion, such as extrapolated velocities around the free surface of a liquid, are better approximated with a direct velocity inference. For such cases, we remove the curl term from Equation (1), and instead use $L_G(\mathbf{c}) = ||\mathbf{u_c} - G(\mathbf{c})||_1$, where the output of G represents a velocity field with $G(\mathbf{c}) : \mathbb{R}^n \mapsto \mathbb{R}^{H\times W\times D\times V_{\text{dim}}}$.



In both cases, simply minimizing the L_1 distance of a high-order function approximation to its original counterpart does not guarantee that their derivatives will match. Consider the example shown in the inset image: given a function (black line, gray circles), two approximations (red line, top image; blue line, bottom image) of it with the same average L_1 distances are shown. On the upper image derivatives do not match, producing a jaggy reconstructed behavior; on the bottom image both values and derivatives of the L_1 distance are minimized, resulting in matching derivatives. With a sufficiently smooth data set, high-frequency features of the CNN are on the null space of the L_1 distance minimization and noise may occur. We discuss this further in the supplemental material.

Thus, we augment our loss function to also minimize the difference of the velocity field gradients. The velocity gradient $\nabla : \mathbb{R}^{H\times W\times D\times V_{\text{dim}}} \mapsto \mathbb{R}^{H\times W\times D\times (V_{\text{dim}})^2}$ is a second-order tensor that encodes vorticity and divergence information. Similar techniques, as image gradient difference loss [Mathieu et al. 2015], have been employed for improving frame prediction on video sequences. However, to our knowledge, this is the first architecture to employ gradient information to improve velocity field data. Our resulting loss function is defined as

$$L_G(\mathbf{c}) = \lambda_\mathbf{u}||\mathbf{u_c} - \hat{\mathbf{u}}_\mathbf{c}||_1 + \lambda_{\nabla\mathbf{u}}||\nabla\mathbf{u_c} - \nabla\hat{\mathbf{u}}_\mathbf{c}||_1, \tag{2}$$

where $\hat{\mathbf{u}}_\mathbf{c} = \nabla \times G(\mathbf{c})$ for incompressible flows and $\hat{\mathbf{u}}_\mathbf{c} = G(\mathbf{c})$ for compressible flows, and $\lambda_\mathbf{u}$ and $\lambda_{\nabla\mathbf{u}}$ are weights used to emphasize the reconstruction of either the velocities or their derivatives. In practice, we used $\lambda_\mathbf{u} = \lambda_{\nabla\mathbf{u}} = 1$. The curl of the velocity and its gradients are computed internally by our architecture and do not need to be explicitly included in the data set.

3.2 Implementation

For the implementation of our generative model we adopted and modified the network architecture from [Berthelot et al. 2017]. As illustrated in Figure 4, our generator starts by projecting the initial $\mathbf{c}$ parameters into an m-dimensional vector of weights $\mathbf{m}$ via fully connected layers. The dimension of $\mathbf{m}$ depends on the network output $\mathbf{d} = [H, W, D, V_{\text{dim}}]$ and on a custom defined parameter q. With $d_{\text{max}} = \max(H, W, D)$, q is calculated by $q = \log_2(d_{\text{max}}) - 3$, meaning that the minimum supported number of cells in one dimension is 8. Additionally, we constrain all our grid dimensions to be divisible by 2^q. Since we use a fixed number of feature maps per layer, the number of dimensions of $\mathbf{m}$ is $m = \frac{H}{2^q} \times \frac{W}{2^q} \times \frac{D}{2^q} \times 128$ and those will be expanded to match the original output resolution.

The m-dimensional vector $\mathbf{m}$ is then reshaped to a $[\frac{H}{2^q}, \frac{W}{2^q}, \frac{D}{2^q}, 128]$ tensor. As shown in Figure 4, the generator component is subdivided into small (SB) and big blocks (BB). For small blocks, we perform N (most of our examples used $N = 4$ or 5) flat convolutions followed by Leaky Rectified Linear Unit (LReLU) activation functions [Maas et al. 2013]. We substituted the Exponential Liner Unit (ELU) activation function in the original method from [Berthelot et al. 2017] by the LReLU as it yielded sharper outputs when minimizing the L_1 loss function. Additionally, we employ residual skip connections [He et al. 2015], which are an element-wise sum of feature maps of input and output from each SB. While the concatenative skip connections employed by Berthelot et al. [2017] are performed between the first hidden states and the consecutive maps with doubling of the depth size to 256, ours are applied to all levels of SBs with a fixed size of 128 feature maps. After the following upsample operation, the dimension of the output from a BB after i passes is $[\frac{H}{2^{q-i}}, \frac{W}{2^{q-i}}, \frac{D}{2^{q-i}}, 128]$. Our experiments showed that performing these reductions to the feature map sizes with the residual concatenation improved the network training time without degradation of the final result.

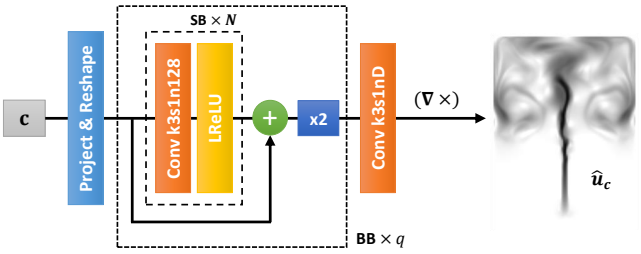


Fig. 4. Architecture of the proposed generative model, subdivided into small (SB) and big blocks (BB). Small blocks are composed of flat convolutions followed by a LReLU activation function. Big blocks are composed of sets of small blocks, an additive skip-connection and an upsampling operation. The output of the last layer has D channels (G_{dim} for incompressible velocity fields, V_{dim} otherwise) corresponding to the simulation dimension.

4 EXTENDED PARAMETERIZATIONS

Scenes with a large number of parameters can be challenging to parameterize. For instance, the dynamically moving smoke source example (Figure 12) can be parameterized by the history of control inputs, i.e., $[\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_t] \rightarrow \mathbf{u}_t$, where $\mathbf{p}_t$ and $\mathbf{u}_t$ represent the smoke source position and the reconstructed velocity field at time t, respectively. In this case, however, the number of parameters grows linearly with the number of frames tracking user inputs. As a consequence, the parameter space would be infeasibly large for data-driven approaches, and would be extremely costly to cover with samples for generating training data.

To extend our approach to these challenging scenarios, we employ an additional encoder architecture $G^{-1}(\mathbf{u}) : \mathbb{R}^{H\times W\times D\times V_{\text{dim}}} \mapsto \mathbb{R}^n$, shown in Figure 5. Conversely to our generative network, the encoder architecture maps velocity field frames into a parameterization $\mathbf{c} = [\mathbf{z}, \mathbf{p}] \in \mathbb{R}^n$, in which $\mathbf{z} \in \mathbb{R}^{n-k}$ is a reduced latent space that models arbitrary features of the flow in an unsupervised way and $\mathbf{p} \in \mathbb{R}^k$ is a supervised parameterization to control specific attributes [Kulkarni et al. 2015]. For the moving smoke source example, $k = 2$ and $\mathbf{p}$ encodes x, z positions used to control the position of the smoke source.

The combined encoder and generative networks are similar to Deep Convolutional autoencoders [Vincent et al. 2010], where the generative network $G(\mathbf{c})$ acts as a decoder. The encoding architecture is symmetric to our generative model, except that we do not employ the inverse of the curl operator and the last convolutional layer. We train both generative and encoding networks with a combined loss similar to Equation (2), as

$$L_{AE}(\mathbf{u}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \hat{\mathbf{u}}_\mathbf{c}||_1 + \lambda_{\nabla\mathbf{u}}||\nabla\mathbf{u_c} - \nabla\hat{\mathbf{u}}_\mathbf{c}||_1 + \lambda_{\mathbf{p}}||\mathbf{p} - \hat{\mathbf{p}}||_2^2, \quad (3)$$

where $\hat{\mathbf{p}}$ is the part of the latent space vector constrained to represent control parameters $\mathbf{p}$, and $\lambda_{\mathbf{p}}$ is a weight to emphasize the learning of supervised parameters. As before, we used $\lambda_{\mathbf{u}} = \lambda_{\nabla\mathbf{u}} = \lambda_{\mathbf{p}} = 1$ for all our examples. With this approach we can handle complex parameterizations, since the velocity field states are represented by the remaining latent space dimensions in $\mathbf{z}$. This allows us to use latent spaces which do not explicitly encode the time dimension as a parameter. Instead, we can use a second latent space integration network that generates a suitable sequence of latent codes.

4.1 Latent Space Integration Network

The latent space only learns a diffuse representation of time by the velocity field states $\mathbf{z}$. Thus we propose a latent space integration network for advancing time from reduced representations. The network $T(\mathbf{x}_{t-1}) : \mathbb{R}^{n+k} \mapsto \mathbb{R}^n$ takes an input vector $\mathbf{x}_{t-1} = [\mathbf{c}_{t-1}; \Delta\mathbf{p}_{t-1}] \in \mathbb{R}^{n+k}$ which is a concatenation of a latent code $\mathbf{c}_{t-1}$ at previous time $t-1$ and a control vector difference between user input parameters $\Delta\mathbf{p}_{t-1} = \mathbf{p}_t - \mathbf{p}_{t-1} \in \mathbb{R}^k$. The parameter $\Delta\mathbf{p}_{t-1}$ has the same dimensionality k as the supervised part of our latent space, and serves as a transition guidance from latent code $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$. The output of $T(\mathbf{x}_{t-1})$ is the residual $\Delta\mathbf{c}_t$ between two consecutive states. Thus, a new latent code is computed with $\mathbf{c}_t = \mathbf{c}_{t-1} + T(\mathbf{x}_{t-1})$.

For improved accuracy we let T look ahead in time, by training the network on a window of w sequential latent codes with an L_2 loss function:

$$L_T(\mathbf{x}_t, ..., \mathbf{x}_{t+w-1}) = \frac{1}{w}\sum_{i=t}^{t+w-1} ||\Delta\mathbf{c}_{i-1} - T(\mathbf{x}_i)||_2^2. \quad (4)$$

Our window loss Equation (4) is designed to minimize not only errors on the next single step integration but also errors accumulated in repeated latent space updates. We found that $w = 30$ yields good results, and a discussion of the effects of different values of w is provided in the supplemental material.

We realize T as a multilayer perceptron (MLP) network. The network consists of three fully connected layers coupled with ELU activation functions. We employ batch normalization and dropout layers with probability of 0.1 to avoid overfitting. Once the networks G, G^{-1} and T are trained, we use Algorithm 1 to reconstruct the velocity field for a new simulation. The algorithm starts from an initial reduced space that can be computed from an initial incompressible velocity field. The main loop consists of concatenating the reduced space and the position update into $\mathbf{x}_{t-1}$; then the latent space integration network computes $\Delta\mathbf{c}_t$, which is used to update $\mathbf{c}_{t-1}$ to $\mathbf{c}_t$. Finally, the generative network G reconstructs the velocity field by evaluating $\mathbf{c}_t$.

Algorithm 1 Simulation with the Latent Space Integration Network

$\mathbf{c}_0 \leftarrow G^{-1}(\mathbf{u}_0)$
while simulating from $t-1$ to t **do**
  $\mathbf{x}_{t-1} \leftarrow [\mathbf{c}_{t-1}; \Delta\mathbf{p}_{t-1}]$
  $\mathbf{c}_t \leftarrow \mathbf{c}_{t-1} + T(\mathbf{x}_{t-1})$
  $\mathbf{u}_t \leftarrow G(\mathbf{c}_t)$
end while

5 RESULTS

In the following we demonstrate that our Deep Fluids CNN can reliably recover and synthesize dynamic flow fields for both smoke and liquids. We refer the reader to the supplemental video for the corresponding animations. For each scene, we reconstruct velocity fields computed by the generative network and advect densities for smoke simulations or surfaces for liquids. Vorticity confinement or turbulence synthesis were not applied after the network's reconstruction, but such methods could be added as a post-processing

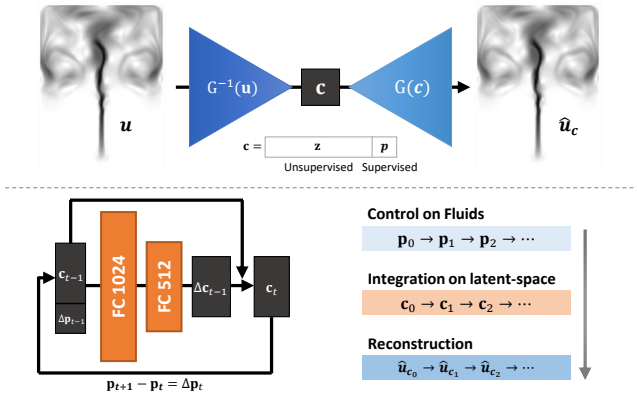


Fig. 5. Autoencoder (top) and latent space integration network (bottom). The autoencoder compresses a velocity field $\mathbf{u}$ into a latent space representation $\mathbf{c}$, which is composed by a supervised and unsupervised part ($\mathbf{z}$ and $\mathbf{p}$, respectively). The latent space integration network finds mappings from subsequent latent code representations $\mathbf{c}_{t-1}$ and $\mathbf{c}_t$.

step. We trained our networks using the Adam optimizer [Kingma and Ba 2014] for 300,000 iterations with varying batch sizes to maximize GPU memory usage (8 for 2-D and 1 for 3-D). For the time network T, we use 30,000 iterations. The learning rate of all networks is scheduled by a cosine annealing decay [Loshchilov and Hutter 2016], where we use the learning range from Smith [2015]. Scene settings, computation times and memory consumptions are summarized in Table 1. Fluid scenes were computed with Mantaflow using an Intel i7-6700K CPU at 4.00 GHz with 32GB memory, and CNN timings were evaluated on a 8GB NVIDIA GeForce GTX 1080 GPU. Networks are trained on a 12GB NVIDIA Titan X GPU. [1]

5.1 2-D Smoke Plume

A sequence of examples which portray varying, rising smoke plumes in a rectangular container is shown in Figure 3, where advected densities for different initial source positions (top) and widths (bottom) are shown. Since visualizing the advected smoke may result in blurred flow structures, we display vorticities instead, facilitating the understanding of how our CNN is able to reconstruct and interpolate between samples present in the data set. Additionally, we use the *hat* notation to better differentiate parameters that do not have a direct correspondence with the ground truth data (e.g., $\hat{\mathbf{p}}_x$ for an interpolated position on the x-axis). Our training set for this example consists of the combination of 5 samples with varying source widths w and 21 samples with varying x positions $\mathbf{p}_x$. Each simulation is computed for 200 frames, using a grid resolution of 96×128 and a domain size of $(1, 1.\overline{33})$. Hence, the network is trained with a total of 21,000 unique velocity field samples.

Reconstruction with Direct Correspondences to the Data Set. To analyze the reconstruction power of our approach, we compare generated velocities for parameters which have a direct correspondence to the original data set, i.e. the ground truth samples. Figure 6 shows

[1] Data sets and source code will be made publicly available.

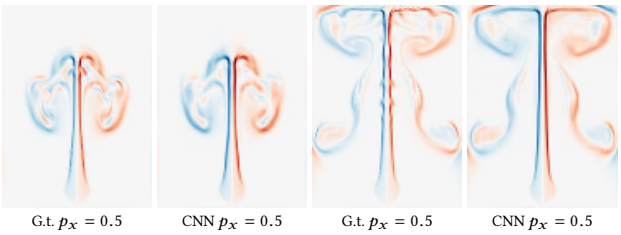


Fig. 6. Vorticity plot of a 2-D smoke simulation with direct correspondences to the training data set for two different times, at fixed position $p_x = 0.5$. Our CNN is able to closely approximate ground truth samples (G.t.).

vorticity plots comparing the ground truth (G.t.) and our CNN output for two different frames. The CNN shows a high reconstruction quality, where coarse structures are almost identically reproduced, and fine structures are closely approximated.

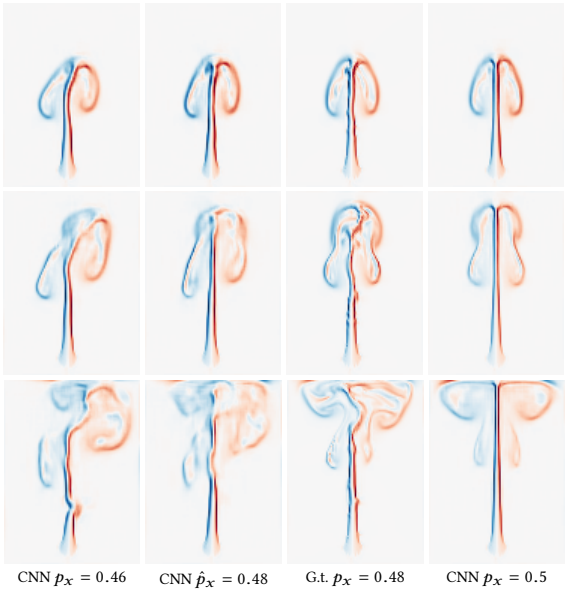


Fig. 7. Vorticity plot of a 2D smoke simulation showing CNN reconstructions at ground truth correlated positions $\mathbf{p}_x = 0.46$ and $\mathbf{p}_x = 0.5$, and the interpolated result at $\hat{\mathbf{p}}_x = 0.48$. The third column shows a ground truth simulation which is not part of the training data set for $\mathbf{p}_x = 0.48$.

Sampling at Interpolated Parameters. We show the interpolation capability of our approach in Figure 7. Left and right columns show the CNN reconstructions at ground truth correlated positions $\mathbf{p}_x = 0.46$ and $\mathbf{p}_x = 0.5$, while the second column shows a vorticity plot interpolated at $\hat{\mathbf{p}}_x = 0.48$. The third column shows the simulated ground truth for the same position. For positions not present in the original data, our CNN synthesizes plausible new motions that are close to ground truth simulations.

5.2 3-D Smoke Examples

Smoke & Sphere Obstacle. Figure 8 shows a 3-D example of a smoke plume interacting with a sphere computed on a grid of size

$64 \times 96 \times 64$. The training data consists of ten simulations with varying sphere positions, with the spaces between spheres centroid samples consisting of 0.06 in the interval $[0.2, 0.8]$. The left and right columns of Figure 8 show the CNN-reconstructed simulations at positions $\mathbf{p}_x = 0.44$ and $\mathbf{p}_x = 0.5$, while the second column presents the interpolated results using our generative network at $\hat{\mathbf{p}}_x = 0.47$. Even with a sparse and hence challenging training data set, flow structures are plausibly reconstructed and compare favorably with ground truth simulations (third column) that were not present in the original data set.

Fig. 8. Interpolated result (second column) given two input simulations (left and right) with different obstacle positions on the x-axis. Our method results in plausible in-betweens compared to ground truth (third column), even in the presence of large discrepancies between the input flow states.

Smoke Inflow and Buoyancy. A collection of simulations with varying inflow speed (different columns) and buoyancy (different rows) is shown in Figure 2 for the ground truth (left) and our generative network (right). We generated 5 inflow velocities (in the range $[1.0, 5.0]$) along with 3 different buoyancy values (from 6×10^{-4} to 1×10^{-3}) for 250 time frames. Thus, the network was trained with 3,750 unique velocity fields. Figure 9 demonstrates an interpolation example for the buoyancy parameter. The generated simulations on the left and right (using a buoyancy of 6×10^{-4} and 1×10^{-3}, respectively) closely correspond to the original data set samples, while the second simulation is reconstructed by our CNN using an interpolated buoyancy of 8×10^{-4}. We show the ground truth simulation on the third image for a reference comparison. Our method recovers structures accurately, and the plume shape matches the reference ground truth.

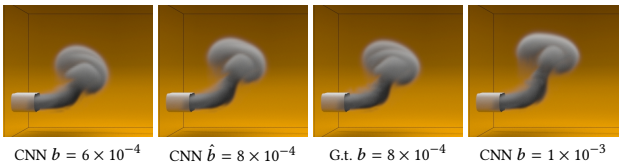


Fig. 9. Results for a rising plume with different buoyancy values. The left and right images show reconstructions for simulations present in the training data set. The second image shows a reconstruction for an interpolated buoyancy value ($\hat{b} = 8 \times 10^{-4}$), while the third image shows the ground truth for comparison.

Rotating Smoke. We trained our autoencoder and latent space integration network for a smoke simulation with a periodically rotating source (period equals to 100 frames) in the XZ-plane for 500 frames. This example is designed as a stress test for extrapolating time using our latent space integration network. In Figure 10, we show that our approach is able to correctly capture the periodicity present in the original data set. Moreover, the method successfully generates another 500 frames, resulting in a simulation that is 100% longer than the original data.

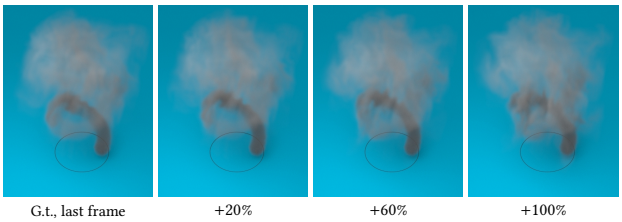


Fig. 10. Simulation results illustrating the effect of time extrapolation using our latent space integration network. The left image shows the last frame of the ground truth simulation. The subsequent images show results with time extrapolation of +20%, +60% and +100% of the original frames.

Moving Smoke. A smoke source is moved on the XZ-plane along a path randomly generated using Perlin noise. We sampled 200 simulations on a grid of size $48 \times 72 \times 48$ for 400 frames - a subset is shown in Figure 11 - and used them to train our autoencoder and latent space integration networks. In Figure 12, we show a moving smoke source whose motion is not part of the training data and was computed by integrating in the latent space. We extrapolate in time to increase the simulation duration by 100% (i.e., 800 frames). The network generates a plausible flow for this unseen motion over the full course of the inferred simulation. Although the results shown here were rendered offline, the high performance of our trained model would allow for interactive simulations.

5.3 3D Liquid Examples

Spheres Dropping on a Basin. We demonstrate that our approach can also handle splashing liquids. We use a setup for two spheres dropping on a basin, which is parameterized by the initial distance of the spheres, as well as by the initial drop angles along the $XZ-$plane relative to the basin. We sample velocity field sequences by combining 5 different distances and 10 different angles; Figure 13 shows

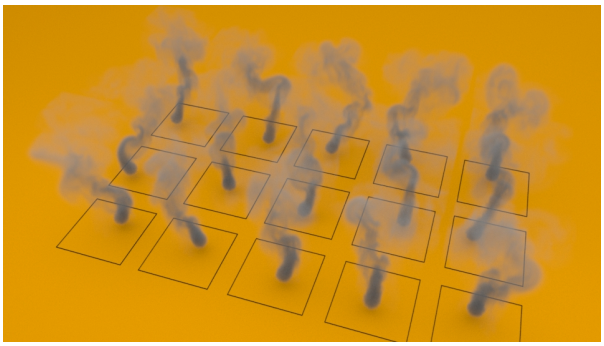


Fig. 11. Example simulations of the moving smoke scene used for training the extended parameterization networks.

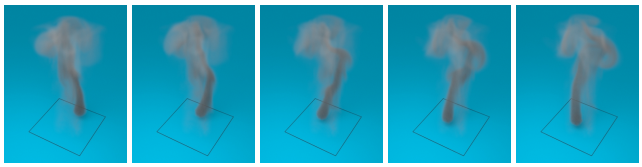


Fig. 12. Different snapshots of a moving smoke source example simulated in the latent space. The smoke rises realistically for the new motion of the smoke source.

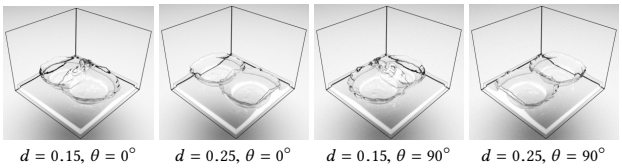


Fig. 13. Training samples for the liquid spheres example, varying distances and angles relative to the basin. We use 5 distances and 10 angles, yielding a total of 50 simulation examples.

4 of the training samples. With 150 frames in time, the network is trained with 7,500 unique velocity fields. We used a single-phase solver and extrapolated velocities from the liquid to the air phase before training (extrapolation size = 4 cells). Figure 14, middle, shows our result for an interpolated angle of $\hat{\theta} = 9°$ and a sphere distance of $\hat{d} = 0.1625$, given two CNN-reconstructed input samples on the left ($\theta = 0°, d = 0.15$) and right ($\theta = 18°, d = 0.175$). Our results demonstrate that the bulk of the liquid dynamics are preserved well. Small scale details such as high-frequency structures and splashes, however, are particularly challenging and deviate from the reference simulations.

Viscous Dam Break. In this example, a dam break with four different viscosity strengths ($\mu = 2\times[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]$) was used to train the network. Our method can reconstruct simulations with different viscosities accurately, and also interpolate between different viscosities with high fidelity. In Figure 15, the CNN-reconstructed, green-colored liquids have direct correspondences in the original

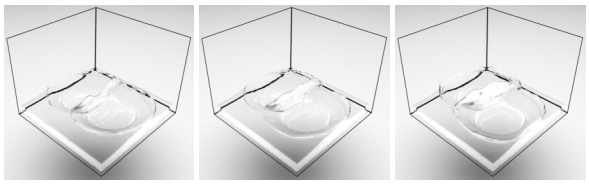


Fig. 14. Parameter interpolation for the liquid spheres example. All three results shown here were generated by our CNN. While the far left and right simulations employ parameter settings that were part of the training data, the middle example represents a new in-between parameter point which is successfully reconstructed by our method.

dataset; the pink-colored simulations are interpolation results between the two nearest green samples. Additionally, it is also possible to increase the viscosity over time as shown in Figure 16. The results show that this works reliably although the original parameterization does neither support time-varying viscosities nor do the training samples represent such behavior.

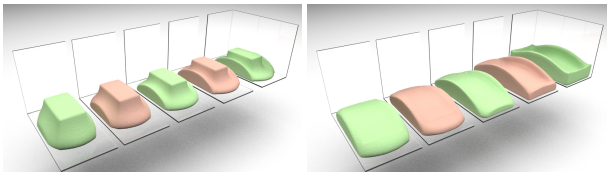


Fig. 15. Snapshots of a CNN reconstructed dam break with different viscosity strengths for two different frames. Green liquids denote correspondences with ground truth ($\mu = 2\times[10^{-4}, 10^{-3}, 10^{-2}]$, back to front) while pink ones are interpolated ($\hat{\mu} = 2\times[5^{-3}, 5^{-2}]$, back to front).

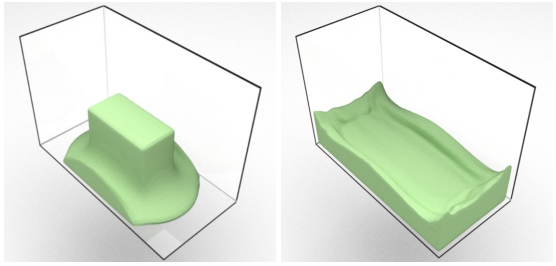


Fig. 16. Reconstruction result using a time varying viscosity strength. On the first few frames the liquid quickly breaks into the container. As the simulation advances, the viscosity increases and the liquid sticks to a deformed configuration.

Slow Motion Fluids. Our supplemental video additionally shows an interesting use case that is enabled by our CNN-based interpolation: the generation of temporally upsampled simulations. Based on a trained model we can create slow-motion effects, which we show for the liquid drop and dam break examples.

6 EVALUATION AND DISCUSSION

6.1 Training

Our networks are trained on normalized data in the range $[-1, 1]$. In case of velocity fields, we normalize them by the maximum absolute value of the entire data set. We found that batch or instance normalization techniques do not improve our velocity fields output, as the highest (lowest) pixel intensity and mean deviation might vary strongly within a single batch. Frames from image-based data sets have a uniform standard deviation, while velocity field snapshots can vary substantially. Other rescaling techniques, such as standardization or histogram equalization, could potentially further improve the training process.

Convergence of the Training. The presented architecture is very stable and all our tests have converged reliably. Training time highly depends on the example and the targeted reconstruction quality. Generally, 3-D liquid examples require more training iterations (up to 100 hours of training) in order to get high quality surfaces, while our smoke examples finished on average after 72 hours of training.

Figure 17 shows a convergence plot of the 2-D smoke example, with training iterations on the x-axis and error on the y-axis. The superimposed images show clearly how quality increases along with training iterations. After about 180,000 iterations, the smoke plume is already reconstructed with good accuracy. This corresponds to roughly 3 hours of training on our hardware.

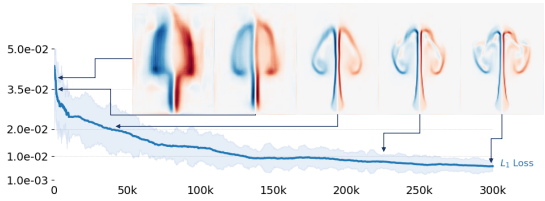


Fig. 17. Convergence plot of the L_1 loss for the 2-D smoke sequence from Figure 7.

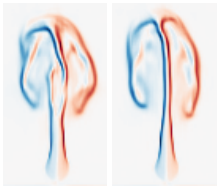
6.2 Performance Analysis

Table 1 summarizes the statistics of all presented examples. In terms of wall-clock time, the proposed CNN approach generates velocity fields up to 700× faster than a standard fluid solver. Some care must be taken when interpreting this number because our Tensorflow network runs on the GPU, while the original Mantaflow code runs on the CPU. Fluid simulations are known to be memory bandwidth-limited [Kim 2008], and the bandwidth discrepancy between a GTX 1080 (320 GB/s) and our Intel desktop (25.6 GB/s) is a factor of 12.5. However, even if we conservatively normalize by this factor, our method achieves a speed-up of up to 58×. Thus, the core algorithm is still at least an order of magnitude faster. To facilitate comparisons with existing subspace methods, we do not include the training time of our CNN when computing the maximum speedup, as precomputation times are customarily reported separately. Instead, we include them in the discussion of training times below. Finally, the memory consumption of our method is at most 30 MB, which effectively compresses the input data by over 1300×. Previous subspace methods [Jones et al. 2016] only achieved ratios of 14×, so our results improve on the state-of-the-art by two orders of magnitude.

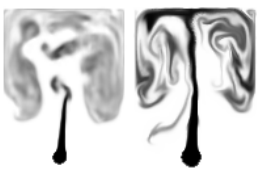
Contrary to traditional solvers, our approach is able to generate multiple frames in time independently. Thus, we can efficiently concatenate CNN queries into a GPU batch, which then outputs multiple velocity fields at once. Adding more queries increases the batch size (Table 1, 5th column, number in brackets), and the maximum batch size depends on the network size and the hardware's memory capacity. Since we are using the maximum batch size possible for each scene, the network evaluation time scales inversely with the maximum batch size supported by the GPU. Due to the inherent capability of GPUs to efficiently schedule floating point operations, the time for evaluating a batch is independent of its size or the size of the network architecture. Additionally, our method is completely oblivious to the complexity of the solvers used to generate the data. Thus, more expensive stream function [Ando et al. 2015] or energy-preserving [Mullen et al. 2009] solvers could potentially be used with our approach, yielding even larger speed-ups.

In contrast, computing the linear basis using traditional SVD-based subspace approaches can take between 20 [Kim and Delaney 2013] and 33 [Wicke et al. 2009] hours. The process is non-iterative, so interrupting the computation can yield a drastically inferior result, i.e. the most important singular vector may not have been discovered yet. Stanton et al. [2013] were able to improve the precomputation time to 12 hours, but only by deploying to a 110-node cluster. In contrast, our iterative training approach is fully interruptible, and runs on a single machine.

6.3 Quality of Reconstruction and Interpolation



Training Data. Several factors affect the reconstruction and interpolation quality of the vector fields. An inherent problem of machine learning approaches is that quality strongly depends on the data used for training. In our case, the performance of our generative model for interpolated positions is sensitive to the input sampling density and parameters. If the sampling density is too coarse, or if the output abruptly changes with respect to the variation of parameters, errors may appear on reconstructed velocity fields. These errors include the inability to accurately reconstruct detailed flow structures, artifacts near obstacles, and especially ghosting effects in the interpolated results. An example of ghosting is shown in the inset image on the left where only 11 training samples are used, instead of the 21 as used in Section 5.1.



Target Quantities. We have also experimented with training directly with density values (inset image, left) instead of the velocity fields (inset image, right). In case of density-trained networks, the dynamics fail to recover the non-linear nature of momentum conservation and artifacts appear. Advecting density with the reconstructed velocity field yields significantly better results. A more detailed discussion

Scene	Grid Resolution	# Frames	Simulation Time (s)	Eval. Time (ms) [Batch]	**Speed Up** (×)	Data Set Size (MB)	Network Size (MB)	**Compression Ratio**	Training Time (h)
2D Smoke Plume (Fig. 6)	96×128	21,000	0.033	0.052 [100]	635	2064	12	172	5
Smoke Obstacle (Fig. 8)	$64 \times 96 \times 64$	6,600	0.491	0.999 [5]	513	31143	30	1038	74
Smoke Inflow (Fig. 9)	$112 \times 64 \times 32$	3,750	0.128	0.958 [5]	128	10322	29	356	40
3D Liquid Drops (Fig. 14)	$96 \times 48 \times 96$	7,500	0.172	1.372 [3]	125	39813	30	**1327**	134
Varying Viscosity (Fig. 15)	$96 \times 72 \times 48$	600	0.984	1.374 [3]	**716**	2389	29	82	100
Rotating Smoke (Fig. 10)	$48 \times 72 \times 48$	500	0.08	0.52 [10]	308	995	38	26	49
Moving Smoke (Fig. 12)	$48 \times 72 \times 48$	80,000	0.08	0.52 [10]	308	159252	38	4191*	49

Table 1. Statistics for training data sets and our CNN. Note that simulation excludes advection and is done on the CPU, while network evaluation is executed on the GPU with batch sizes noted in brackets. In case of liquids, the conjugate gradient residual threshold is set to $1e^{-3}$, while for smoke it is $1e^{-4}$. For the Rotating and Moving Smoke scenes, the numbers for training time and network size include both the autoencoder and latent space integration networks.
* We optimize the network for subspace simulations rather than the quality of reconstruction, so we do not take this number into account when evaluating the maximal compression ratio.

about the quality of the interpolation regarding the number of input samples and discrepancies between velocity and density training is presented in the supplemental material.

Velocity Loss. A comparison between our compressible loss, incompressible functions and a ground truth simulation is shown in Figure 18. The smoke plume trained with the incompressible loss from Equation (1) shows a richer density profile closer to the ground truth, compared to results obtained using the compressible loss.

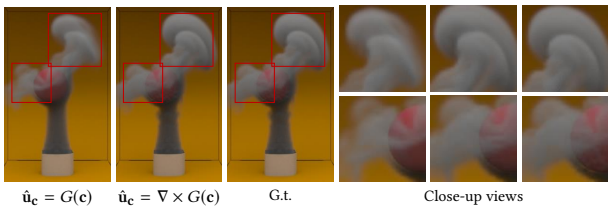


Fig. 18. Comparisons of the results from networks trained on our compressible loss, incompressible loss and the ground truth, respectively. On the right sequence we show the highlighted images from the simulations on the left. We notice that the smoke patterns from the incompressible loss are closer to ground truth simulations.

Boundary Conditions. The proposed CNN is able to handle immersed obstacles and boundary conditions without additional modifications. Figure 19 shows sliced outputs for the scene from Figure 8 which contains a sphere obstacle. We compare velocity (top) and vorticity magnitudes (bottom). The first and last images show the reconstruction of the CNN for $p_\mathbf{x}$ positions that have correspondences in the training data set. The three images in the middle show results from linearly blending the closest velocity fields, our CNN reconstruction and the ground truth simulation, from left to right respectively. In the case of linearly blended velocity fields, the non-penetration constraints for the obstacle are not respected, as velocities are present inside the intended obstacle positions. In Figure 20, we plot the resulting velocity penetration errors. Here we compute the mean absolute values of the velocities inside the voxelized sphere, normalized by the mean sum of the velocity magnitudes for all cells around a narrow band of the sphere. Boundary errors are slightly higher for interpolated parameter regions (orange line in Figure 20), since no explicit constraint for the object's shape is enforced. However, the regularized mean error still accounts for less than 1% of the maximum absolute value of the velocity field. Thus, our method successfully preserves the non-penetration boundary conditions.

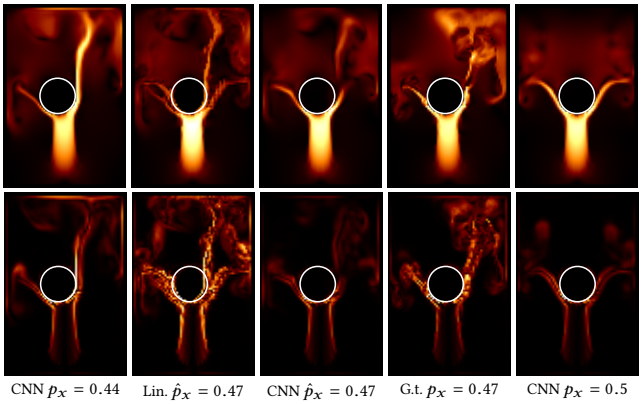


Fig. 19. Slice views of last row in Fig. 8. The color code in each row represents the magnitude of velocity (top) and vorticity fields (bottom). The second column shows a linear interpolation of the input data. Despite the absence of any constraints on boundary conditions, our method (third column) preserves the shape of the original sphere obstacle, and yields significantly better results than the linear interpolation.

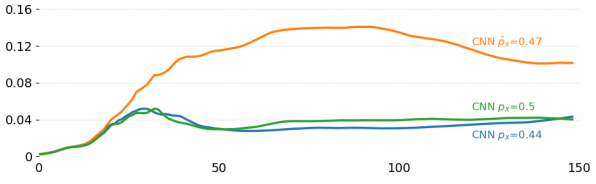


Fig. 20. Mean absolute error plot of velocity penetration for the smoke obstacle example. Though errors in interpolated samples are a bit higher than those of reconstructed samples, they do not exceed 1% of the maximum absolute value of the data set.

Liquid-air Interface. Due to the separate advection calculation for particles in our FLIP simulations, smaller splashes can leave the velocity regions generated by our CNNs, causing surfaces advected by reconstructed velocities to hang in mid-air. Even though the reconstructed velocity fields closely match the ground truth samples, liquid scenes are highly sensitive to such variations. We removed FLIP particles that have small velocities in such regions, which was sufficient to avoid hanging particles artifacts.

6.4 Extrapolation and Limitations

Extrapolation with Generative Model. We evaluated the extrapolation capabilities for the case where only the generative part of our Deep Fluids CNN (Section 3) is used. Generally, extrapolation works for sufficiently small increments beyond the original parameter space. Figure 21 shows an experiment in which we used weights that were up to 30% of the original parameter range ($[-1, 1]$). The leftmost images show the vorticity plot for the maximum value of the range for the position (top), inflow size (middle), and time (bottom) parameters of the 2-D smoke plume example. The rightmost images show the maximum variation of parameters, in which the simulations deteriorate in quality. In practice, we found that up to ca. 10% of extrapolation still yielded plausible results.

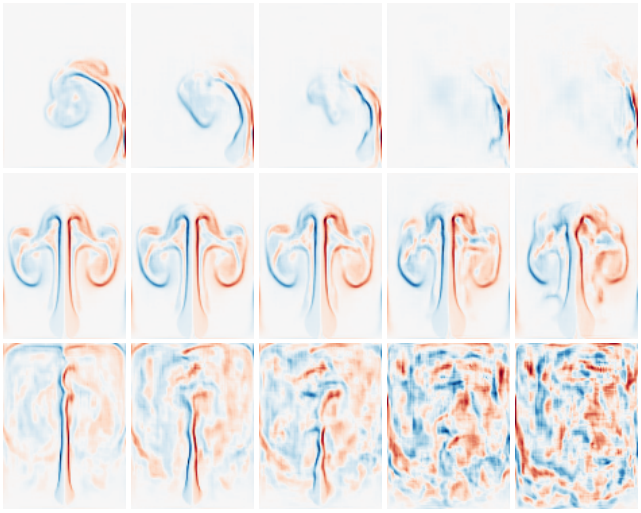


Fig. 21. Extrapolation results of the 2-D smoke plume example (from top to bottom: position, inflow width, time) for the case where only the generative network is used. Plausible results can be observed for up to 10% extrapolation; larger values lead to degraded results.

Limitations. Our Deep Fluids CNN is designed to generate velocity fields for parameterizable scenes. As such, our method is not suitable for reconstructing arbitrary velocity fields of vastly different profiles by reduction to a shared latent representation. As discussed in Section 6.3, there is also no enforcement of physical constraints such as boundary conditions for intermediate interpolated parameters. Thus, the capability of the network to reconstruct physically accurate samples on interpolated locations depends on the proximity of the data samples in the parameter space. Additionally, the reconstruction quality of the autoencoder and latent space integration networks are affected by the size of the latent space $\mathbf{c}$. We provide an extended discussion regarding the reconstruction quality and the latent space size on our supplemental material.

7 CONCLUSION

We have presented a first generative deep learning architecture that successfully synthesizes plausible and divergence-free 2-D and 3-D fluid simulation velocities from a set of reduced parameters. Our results show that generative neural networks are able to construct a wide variety of fluid behaviors, from turbulent smoke to viscous liquids, that closely match the input training data. Moreover, our network can synthesize physically plausible motion when the input parameters are continuously varied to intermediate states that were not present during training. In addition, we can handle complex parameterizations in a reduced latent space, enabling flexible latent space simulations by using a latent space integration network.

Our solver is considerably faster (up to 700×) than traditional CPU solvers, which make CNNs an attractive approach for simulating scenarios where input interactions can be parameterized. These performance characteristics immediately suggest applications in games and virtual environments. Fluid simulations are also known to demand large disk and memory budgets in movie productions, so the compression characteristics of our algorithm (over 1300×) make it appealing in these environments as well.

Our CNN architecture was carefully designed to achieve high quality fluid simulations, which is why the loss function considers both the velocity field and its gradient. Over the course of evaluating many alternatives, we found that the most important factor to simulation quality was the amount of training data. If the data sets are too sparse, artifacts appear, and important flow structures are missing. We address this issue by simply increasing the number of training samples, but in scenarios where data was directly captured or the simulation times are prohibitive, this may not be feasible. Improving the reconstruction quality of interpolated states over sparsely sampled data sets is an open direction for future work.

Overall, we found that the proposed CNN is able to reproduce velocity fields accurately. However, for small-scale details or near discontinuities such as boundary conditions, the network can sometimes smooth out fine flow structures. A possible future research direction is the exploration of generative adversarial networks (GANs) or alternative distance measures to enhance the accuracy for fine structures in the data. Our CNN-based algorithm is the first of its kind in the fluids community, and we believe that the combined speed, interpolation and compression capabilities of our approach can enable a variety of future applications, such as interactive liquid simulations [Prantl et al. 2017] or implementations of fluid databases.

REFERENCES

Ryoichi Ando, Nils Thuerey, and Chris Wojtan. 2015. A stream function solver for liquid simulations. *ACM Transactions on Graphics* 34, 4 (jul 2015), 53:1–53:9. https://doi.org/10.1145/2766935

anonymous. 2018. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. (jun 2018). arXiv:1806.00000

David Berthelot, Thomas Schumm, and Luke Metz. 2017. BEGAN: Boundary Equilibrium Generative Adversarial Networks. (mar 2017). arXiv:1703.10717 http://arxiv.org/abs/1703.10717

Giuseppe Carleo and Matthias Troyer. 2016. Solving the Quantum Many-Body Problem with Artificial Neural Networks. (jun 2016). https://doi.org/10.1126/science.aag2302 arXiv:1606.02318

Juan Carrasquilla and Roger G. Melko. 2017. Machine learning phases of matter. *Nature Physics* 13, 5 (feb 2017), 431–434. https://doi.org/10.1038/nphys4035

Mengyu Chu and Nils Thuerey. 2017. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics* 36, 4 (jul 2017), 1–14. https://doi.org/10.1145/3072959.3073643

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using Laplacian eigenfunctions. *ACM Transactions on Graphics* 31, 1 (jan 2012), 1–11. https://doi.org/10.1145/2077341.2077351

Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. 2017. Deep Learning the Physics of Transport Phenomena. (sep 2017). arXiv:1709.02432 http://arxiv.org/abs/1709.02432

Dan Gerszewski, Ladislav Kavan, Peter-Pike Sloan, and Adam W Bargteil. 2015. Basis Enrichment and Solid-fluid Coupling for Model-reduced Fluid Simulation. *Computer Animation and Virtual Worlds* 26 (2015).

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 27*, Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Eric Guerin, Julie Digne, Eric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoit Martinez. 2017. Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks. *ACM Transactions on Graphics (proceedings of Siggraph Asia 2017)* 36, 6 (2017).

Xiaoxiao Guo, Wei Li, and Francesco Iorio. 2016. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, New York, New York, USA, 481–490. https://doi.org/10.1145/2939672.2939738

Mohit Gupta and Srinivasa G Narasimhan. 2007. Legendre fluids: a unified framework for analytic reduced space modeling and rendering of participating media. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer animation*. 17–25.

Botros N Hanna, Nam T. Dinh, Robert W. Youngblood, and Igor A. Bolotnov. 2017. Coarse-Grid Computational Fluid Dynamic (CG-CFD) Error Prediction using Machine Learning. (oct 2017). arXiv:1710.09105 http://arxiv.org/abs/1710.09105

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. (dec 2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. (mar 2016). arXiv:1603.08155 http://arxiv.org/abs/1603.08155

Aaron Demby Jones, Pradeep Sen, and Theodore Kim. 2016. Compressing Fluid Subspaces. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '16)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 77–84. http://dl.acm.org/citation.cfm?id=2982818.2982830

Simon Kallweit, Thomas Müller, Brian McWilliams, Markus Gross, and Jan Novák. 2017. Deep Scattering: Rendering Atmospheric Clouds with Radiance-Predicting Neural Networks. (sep 2017). https://doi.org/10.1145/3130800.3130880 arXiv:1709.05418

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive Growing of GANs for Improved Quality, Stability, and Variation. (oct 2017). arXiv:1710.10196 http://arxiv.org/abs/1710.10196

Theodore Kim. 2008. Hardware-aware Analysis and Optimization of Stable Fluids. In *Symposium on Interactive 3D Graphics and Games*. 99–106.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics* 32, 4 (jul 2013), 1. https://doi.org/10.1145/2461912.2461987

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. 2015. Deep Convolutional Inverse Graphics Network. (mar 2015). arXiv:1503.03167 http://arxiv.org/abs/1503.03167

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics* 34, 6 (oct 2015), 1–9. https://doi.org/10.1145/2816795.2818129

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2016. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. (sep 2016). arXiv:1609.04802 http://arxiv.org/abs/1609.04802

Julia Ling, Andrew Kurzawski, and Jeremy Templeton. 2016. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics* 807 (2016), 155–166. https://doi.org/10.1017/jfm.2016.615

Beibei Liu, Gemma Mason, Julian Hodgson, Yiying Tong, and Mathieu Desbrun. 2015. Model-reduced variational fluid simulation. *ACM Trans. Graph.* 34, 6 (2015), 244.

Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Warm Restarts. (aug 2016). arXiv:1608.03983 http://arxiv.org/abs/1608.03983

Wenlong Lu, Ning Jin, and Ronald Fedkiw. 2016. Two-way Coupling of Fluids to Reduced Deformable Bodies. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 67–76.

J L Lumley. 1967. The Structure of Inhomogeneous Turbulent Flows. In *Atmospheric turbulence and radio propagation*, A M Yaglom and V I Tatarski (Eds.). Nauka, 166–178.

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 (SIGGRAPH Proceedings)* (2018).

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, Vol. 30.

Michael Mathieu, Camille Couprie, and Yann LeCun. 2015. Deep multi-scale video prediction beyond mean square error. (nov 2015). arXiv:1511.05440 http://arxiv.org/abs/1511.05440

Lukas Mosser, Olivier Dubrule, and Martin J. Blunt. 2017. Reconstruction of three-dimensional porous media using generative adversarial neural networks. (apr 2017). https://doi.org/10.1103/PhysRevE.96.043309 arXiv:1704.03225

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. 2009. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics* 28, 3 (jul 2009), 1. https://doi.org/10.1145/1531326.1531344

Michela Paganini, Luke de Oliveira, and Benjamin Nachman. 2017. Accelerating Science with Generative Adversarial Networks: An Application to 3D Particle Showers in Multi-Layer Calorimeters. (may 2017). arXiv:1705.02355 http://arxiv.org/abs/1705.02355

Lukas Prantl, Boris Bonev, and Nils Thuerey. 2017. Pre-computed Liquid Spaces with Generative Neural Networks and Optical Flow. (apr 2017). arXiv:1704.07854 http://arxiv.org/abs/1704.07854

Siamak Ravanbakhsh, Francois Lanusse, Rachel Mandelbaum, Jeff Schneider, and Barnabas Poczos. 2016. Enabling Dark Energy Science with Deep Generative Models of Galaxy Images. (sep 2016). arXiv:1609.05796 http://arxiv.org/abs/1609.05796

Kevin Schawinski, Ce Zhang, Hantian Zhang, Lucas Fowler, and Gokula Krishnan Santhanam. 2017. Generative Adversarial Networks recover features in astrophysical images of galaxies beyond the deconvolution limit. (feb 2017). https://doi.org/10.1093/mnrasl/slx008 arXiv:1702.00403

Leslie N. Smith. 2015. Cyclical Learning Rates for Training Neural Networks. (jun 2015). arXiv:1506.01186 http://arxiv.org/abs/1506.01186

Matt Stanton, Yu Sheng, Martin Wicke, Federico Perazzi, Amos Yuen, Srinivasa Narasimhan, and Adrien Treuille. 2013. Non-polynomial Galerkin Projection on Deforming Meshes. *ACM Trans. Graph.* 32, 4, Article 86 (July 2013), 14 pages.

Matthew Luchak Stanton. 2014. *Data-Driven Methods for Interactive Simulation of Complex Phenomena*. Ph.D. Dissertation. Carnegie Mellon University.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. (jul 2016). arXiv:1607.03597 http://arxiv.org/abs/1607.03597

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics* 25, 3 (jul 2006), 826. https://doi.org/10.1145/1141911.1141962

Kiwon Um, Xiangyu Hu, and Nils Thuerey. 2017. Liquid Splash Modeling with Neural Networks. (apr 2017). arXiv:1704.04456 http://arxiv.org/abs/1704.04456

Nobuyuki Umetani and Bernd Bickel. 2018. Learning Three-dimensional Flow for Interactive Aerodynamic Design. *ACM Transactions on Graphics (SIGGRAPH Proceedings)* (2018).

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *J. Mach. Learn. Res.* 11 (Dec. 2010), 3371–3408. http://dl.acm.org/citation.cfm?id=1756006.1953039

Martin Wicke, Matt Stanton, and Adrien Treuille. 2009. Modular bases for fluid dynamics. In *ACM SIGGRAPH 2009 papers on - SIGGRAPH '09*. ACM Press, New York, New York, USA, 1. https://doi.org/10.1145/1576246.1531345

Steffen Wiewel, Moritz Becher, and Nils Thuerey. 2018. Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow. (feb 2018). arXiv:1802.10123 http://arxiv.org/abs/1802.10123

You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. 2018. tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow. (jan 2018). arXiv:1801.09710 http://arxiv.org/abs/1801.09710

Cheng Yang, Xubo Yang, and Xiangyun Xiao. 2016. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds* 27, 3-4 (may 2016), 415–424. https://doi.org/10.1002/cav.1695

Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. 2015. Loss Functions for Neural Networks for Image Processing. *CoRR* abs/1511.0 (2015). arXiv:1511.08861 http://arxiv.org/abs/1511.08861