DOI: 10.1111/cgf.13619
EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)
Volume 38 (2019), Number 2

Deep Fluids: A Generative Network for Parameterized Fluid Simulations

Byungsoo Kim[1], Vinicius C. Azevedo[1], Nils Thuerey[2], Theodore Kim[3], Markus Gross[1] and Barbara Solenthaler[1]

[1]ETH Zürich [2]Technical University of Munich [3]Pixar Animation Studios

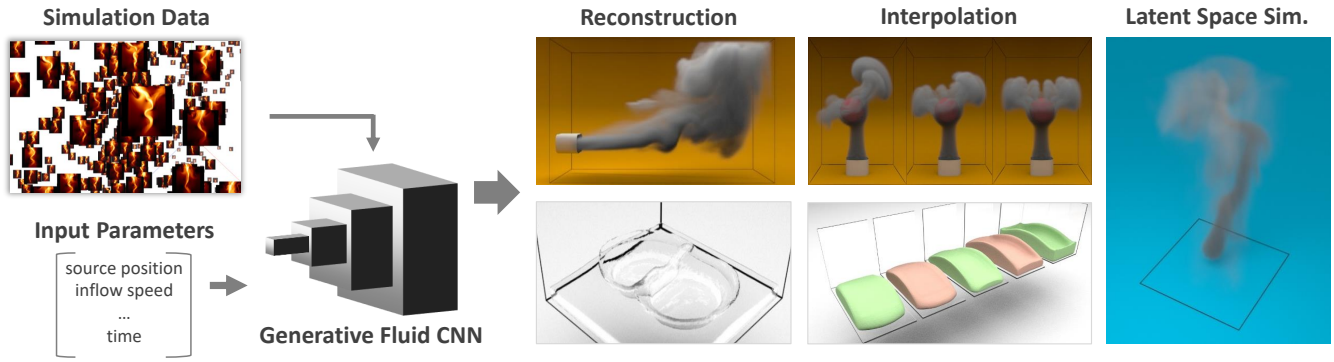


Figure 1: *Our generative neural network synthesizes fluid velocities continuously in space and time, using a set of input simulations for training and a few parameters for generation. This enables fast reconstruction of velocities, continuous interpolation and latent space simulations.*

Abstract

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 re-simulating the data with the underlying CPU solver, while achieving compression rates of up to 1300×.

CCS Concepts

• ***Computing methodologies*** → ***Physical simulation; Neural networks;***

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 [GDG*17], high-resolution faces synthesis [KALL17] and cloud rendering [KMM*17]. In fluid simulation, machine learning techniques have been used to replace [LJS*15], speed up [TSSP17] or enhance existing solvers [XFCT18].

Given the amount of available fluid simulation data, data-driven approaches have emerged as attractive solutions. Subspace solvers [TLP06], fluid re-simulators [KD13] and basis compressors [JSK16] are examples of recent efforts in this direction. However, these methods usually represent fluids using linear basis func-

tions, 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 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 [KD13], 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 [TLP06]. 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 with compression rates that outperform previous work [JSK16] by two orders of magnitude.

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 using the underlying CPU solver for re-simulation.
- 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 method, both when reconstructing samples that have a direct correspondence with the training data and 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 [Lum67], but were introduced to computer graphics by [TLP06] and [GN07]. Since then, improvements have been made to make them modular [WST09], consistent with widely-used integrators [KD13], more energy-preserving [LMH*15] and memory-efficient [JSK16]. A related "Laplacian Eigenfunctions" approach [DLF12] has also been introduced and refined [GKSB15], 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 [LJF16]. Graph-based methods for precomputing liquid motions [Sta14] 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 [LJS*15]. 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 [WBT18], resulting in significant speed-ups of the pressure solver. For a single time step, a CNN-based pressure projection was likewise proposed [TSSP17, YYX16]. In contrast to our 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, [CT17b] synthesized smoke details by looking up pre-computed patches using CNN-based descriptors, while [XFCT18] proposed a GAN for super resolution smoke flows. Other works enhance FLIP simulations with a learned splash model [UHT18], while the deformation learning proposed by [PBT17] 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 [GLI16]. [FGP17] use Generative Adversarial Networks (GANs) [GPAM*14] 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, [UB18] developed a data-driven technique to predict fluid flows around bodies for interactive

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

shape design, while Ma et al. [MTP*18] 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 attention. A few recent examples are [CT17a] using reinforcement learning to reduce the complexity of a quantum many-body problem, [LKT16] employing deep neural networks to synthesize Reynolds average turbulence anisotropy tensors from high-fidelity simulation data, and [PdON18] modeling calorimeter interactions with electromagnetic showers using GANs. GANs have also been employed to generate [RLM*17] and deconvolve [SZZ*17] galaxy images, and reconstruct three-dimensional porous media [MDB17]. 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.

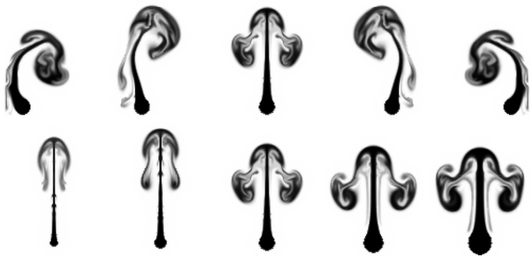


Figure 3: *Different snapshots showing the advected densities for varying smoke source parameters. The top and bottom rows show the variation of the initial position source and width, respectively.*

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 (1 for 2-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 nonlinear nature of the Navier-Stokes equations, these three parameters (i.e. position, width, and time) yield a vastly different set of velocity outputs.

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 [ZGFK17], and perceptual losses [JAFF16, LTH*17], 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

$$L_G(\mathbf{c}) = ||\mathbf{u}_\mathbf{c} - \nabla \times G(\mathbf{c})||_1. \quad (1)$$

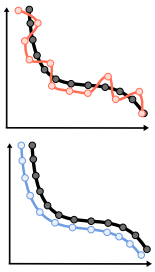
$G(\mathbf{c}) : \mathbb{R}^n \mapsto \mathbb{R}^{H\times W\times D\times G_{\text{dim}}}$ is the network output and $\mathbf{u}_\mathbf{c}$ is a simulation sample from the training data. The curl of the model output

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 $\mathbf{u}_{\mathbf{c}}$.

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

$$L_G(\mathbf{c}) = ||\mathbf{u}_{\mathbf{c}} - G(\mathbf{c})||_1 \tag{2}$$

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. In the upper image derivatives do not match, producing a jaggy reconstructed behavior; in 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 in 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, shearing and divergence information. Similar techniques, as image gradient difference loss [MCL15], 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

$$L_G(\mathbf{c}) = \lambda_{\mathbf{u}} ||\mathbf{u}_{\mathbf{c}} - \hat{\mathbf{u}}_{\mathbf{c}}||_1 + \lambda_{\nabla \mathbf{u}} ||\nabla \mathbf{u}_{\mathbf{c}} - \nabla \hat{\mathbf{u}}_{\mathbf{c}}||_1, \tag{3}$$

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 $\lambda_{\mathbf{u}} = \lambda_{\nabla \mathbf{u}} = 1$ for normalized data (see Section 6.1). 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 [BSM17]. 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

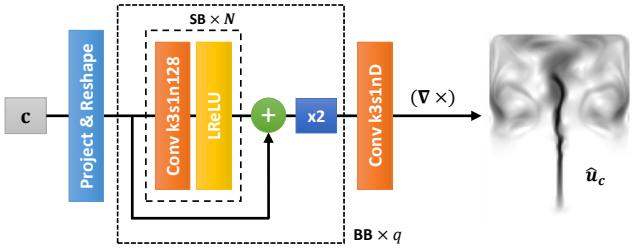


Figure 4: *Architecture of the proposed generative model, subdivided into small (SB) and big blocks (BB). Small blocks are composed of flat convolutions followed by a LReLU activation function. Big blocks are composed of sets of small blocks, an additive skip-connection and an upsampling operation. The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

$d_{\max} = \max(H, W, D)$, q is calculated by $q = \log_2(d_{\max}) - 3$, 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 [MHN13]. We substituted the Exponential Liner Unit (ELU) activation function in the original method from [BSM17] by the LReLU as it yielded sharper outputs when minimizing the L_1 loss function. Additionally, we employ residual skip connections [HZRS16], which are an element-wise sum of feature maps of input and output from each *SB*. While the concatenative skip connections employed by [BSM17] 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 *SB*s 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.

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 add an encoder architecture $G^\dagger(\mathbf{u}) : \mathbb{R}^{H\times W\times D\times V_{\text{dim}}} \mapsto \mathbb{R}^n$ to our generator of Section 3, and combine it with a second smaller network for time integration (Section 4.1), as illustrated in Figure 5. In contrast 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 [KWKT15]. Note that this separation makes the latent space sparser while training, which in turn improves the quality of the reconstruction. For the moving smoke source example in Section 5.2, $n = 16$ 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 [VLL*10], 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 (3), as

$$L_{AE}(\mathbf{u}) = \lambda_{\mathbf{u}}||\mathbf{u_c} - \hat{\mathbf{u}}_{\mathbf{c}}||_1 + \lambda_{\nabla\mathbf{u}}||\nabla\mathbf{u_c} - \nabla\hat{\mathbf{u}}_{\mathbf{c}}||_1 + \lambda_{\mathbf{p}}||\mathbf{p} - \hat{\mathbf{p}}||_2^2, \quad (4)$$

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. Note that the L_2 distance is applied to control parameters unlike vector field outputs, as it is a standard cost function in linear regression. As before, we used $\lambda_{\mathbf{u}} = \lambda_{\nabla\mathbf{u}} = \lambda_{\mathbf{p}} = 1$ for all our normalized examples (Section 6.1). 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 $T(\mathbf{x}_t) : \mathbb{R}^{n+k} \mapsto \mathbb{R}^{n-k}$ takes an input vector $\mathbf{x}_t = [\mathbf{c}_t; \Delta\mathbf{p}_t] \in \mathbb{R}^{n+k}$ which is a concatenation of a latent code $\mathbf{c}_t$ at current time t and a control vector difference between user input parameters $\Delta\mathbf{p}_t = \mathbf{p}_{t+1} - \mathbf{p}_t \in \mathbb{R}^k$. The parameter $\Delta\mathbf{p}_t$ has the same dimensionality k as the supervised part of our latent space, and serves as a transition guidance from latent code $\mathbf{c}_t$ to $\mathbf{c}_{t+1}$. The output of $T(\mathbf{x}_t)$ is the residual $\Delta\mathbf{z}_t$ between two consecutive states. Thus, a new latent code is computed with $\mathbf{z}_{t+1} = \mathbf{z}_t + T(\mathbf{x}_t)$ as seen in Figure 5.

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:

$$L_T(\mathbf{x}_t, ..., \mathbf{x}_{t+w-1}) = \frac{1}{w}\sum_{i=t}^{t+w-1} ||\Delta\mathbf{z}_i - T_i||_2^2, \quad (5)$$

where T_i is recursively computed from t to i. Our window loss Equation (5) 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 rationale behind choosing MLP instead of LSTM is that T is designed to be a navigator on the manifold of the latent space, and we consider these integrations as controlled individual steps rather than physically induced ones. 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 $G, G^\dagger$ 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 $\mathbf{x}_t$; then the latent space integration network computes $\Delta\mathbf{z}_t$, which is used to update $\mathbf{c}_t$ to $\mathbf{c}_{t+1}$. Finally, the generative network G reconstructs the velocity field $\mathbf{u}_{t+1}$ by evaluating $\mathbf{c}_{t+1}$.

Algorithm 1 Simulation with the Latent Space Integration Network

$\mathbf{c_0} \leftarrow G^\dagger(\mathbf{u_0})$
while simulating from t to $t+1$ **do**
  $\mathbf{x}_t \leftarrow [\mathbf{c}_t; \Delta\mathbf{p}_t]$ // $\mathbf{c}_t$ from previous step, $\mathbf{p}$ is given
  $\mathbf{z}_{t+1} \leftarrow \mathbf{z}_t + T(\mathbf{x}_t)$ // latent code inference
  $\mathbf{c}_{t+1} \leftarrow [\mathbf{z}_{t+1}; \mathbf{p}_{t+1}]$
  $\mathbf{u}_{t+1} \leftarrow G(\mathbf{c}_{t+1})$ // velocity field reconstruction
end while

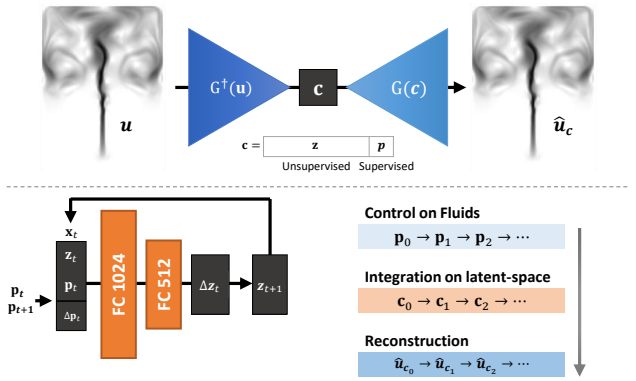


Figure 5: *Autoencoder (top) and latent space integration network (bottom). The autoencoder compresses a velocity field* $\mathbf{u}$ *into a latent space representation* $\mathbf{c}$*, which includes a supervised and unsupervised part (*$\mathbf{p}$ *and* $\mathbf{z}$*). The latent space integration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

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

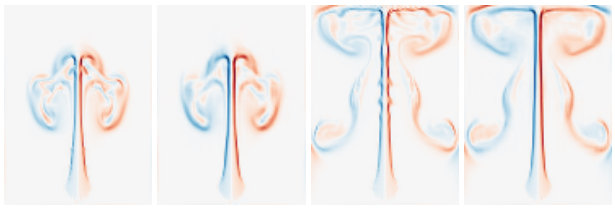


Figure 6: *Vorticity plot of a 2-D smoke simulation with direct correspondences to the training data set for two different times. The RdBu colormap is used to show both the magnitude and the rotation direction of the vorticity (red: clockwise). Our CNN is able to closely approximate ground truth samples (G.t.).*

turbulence synthesis were not applied after the network's reconstruction, but such methods could be added as a post-processing step. We trained our networks using the Adam optimizer [KA15] 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 [LH16], where we use the learning range from [Smi17]. Scene settings, computation times and memory consumptions are summarized in Table 1. Fluid scenes were computed with Mantaflow [TP18] 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.

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 $(1, 1.\overline{33})$. 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 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.

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.

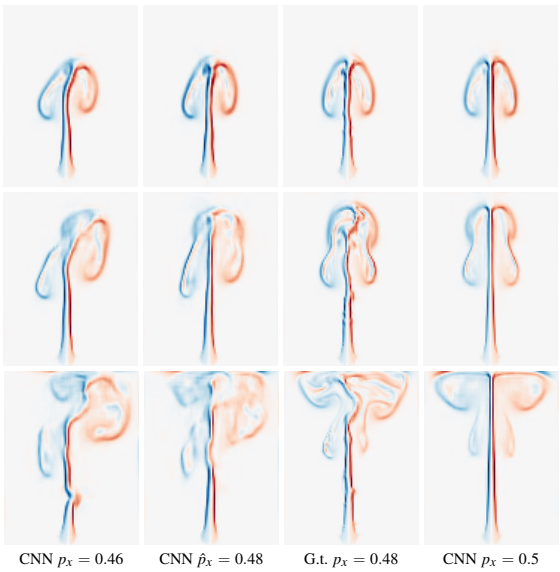


Figure 7: *Vorticity plot of a 2-D smoke simulation showing CNN reconstructions at ground truth correlated positions* $\mathbf{p}_x = 0.46$ *and* $\mathbf{p}_x = 0.5$*, the interpolated result at* $\hat{\mathbf{p}}_x = 0.48$*, and ground truth (G.t.) at* $\hat{\mathbf{p}}_x = 0.48$ *which is not part of the training data set.*

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.

Smoke Inflow and Buoyancy. A collection of simulations with varying inflow speed (columns) and buoyancy (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 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}) 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.

Figure 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 for large differences in the input.*

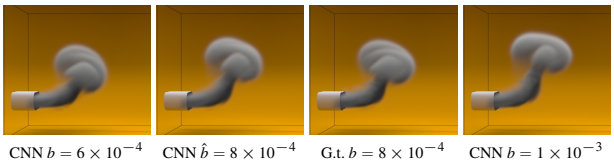


Figure 9: *Reconstructions of the rising plume scene (left and right), reconstruction for an interpolated buoyancy value ($\hat{b} = 8 \times 10^{-4}$) (second image) and the corresponding ground truth (third image).*

Rotating Smoke. We trained our autoencoder and latent space integration network for a smoke simulation with a periodically rotating source using 500 frames as training data. The source rotates in the XZ-plane with a period of 100 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.

Moving Smoke. A smoke source is moved in 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

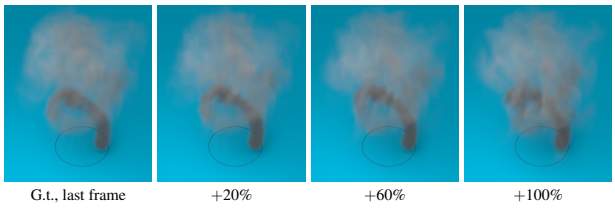


Figure 10: *Time extrapolation results using our latent space integration network. The left image shows the last frame of the ground truth simulation. The subsequent images show results with time extrapolation of +20%, +60% and +100% of the original frames.*

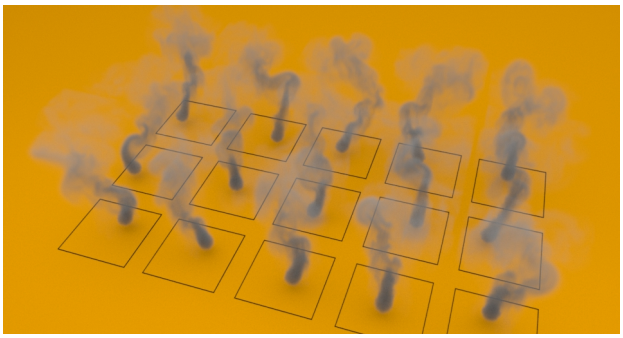


Figure 11: *Example simulations of the moving smoke scene used for training the extended parameterization networks.*

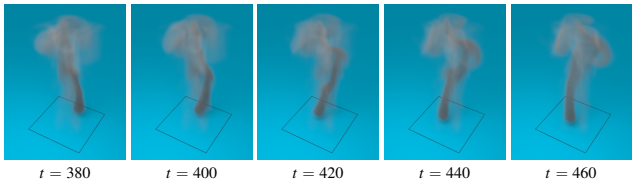


Figure 12: *Different snapshots of a moving smoke source example simulated in the latent space.*

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. 3-D 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 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 $\hat{\theta} = 9^\circ$ and a sphere distance of $\hat{d} = 0.1625$, given two CNN-reconstructed input samples on the left ($\theta = 0^\circ, d = 0.15$) and right ($\theta = 18^\circ, d = 0.175$). Our results demonstrate that the bulk of the

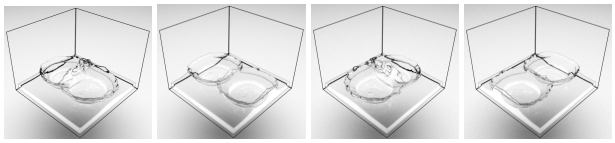


Figure 13: *Training samples for the liquid spheres scene. In total we used* 50 *simulation examples with varying distances and angles.*

Figure 14: *CNN-generated results with parameter interpolation for the liquid spheres example. 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.*

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 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.

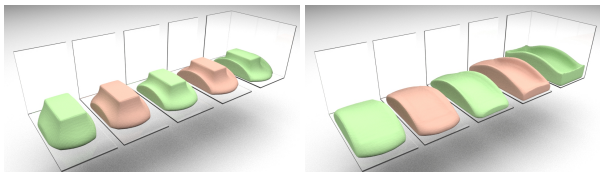


Figure 15: *Snapshots of a CNN reconstructed dam break with different viscosity strengths for two different frames. Green liquids denote correspondences with ground truth ($\mu = 2 \times [10^{-4}, 10^{-3}, 10^{-2}]$, back to front) while pink ones are interpolated ($\hat{\mu} = 2 \times [5^{-3}, 5^{-2}]$,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

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.

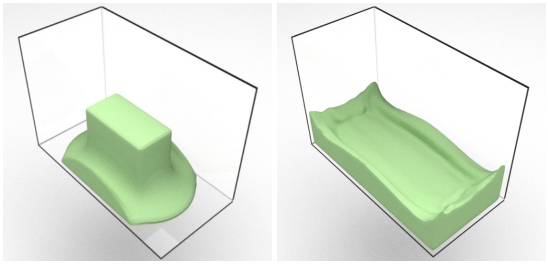


Figure 16: *Reconstruction result using a time varying viscosity strength. In the first few frames the liquid quickly breaks into the container. As the simulation advances, the viscosity increases and the liquid sticks to a deformed configuration.*

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. Scaling tests with various 2-D grid resolutions (64×48, 128×96, 256×192, 512×384) have shown that the training speed scales proportionally with the resolution, while keeping the mean absolute error at a constant level.

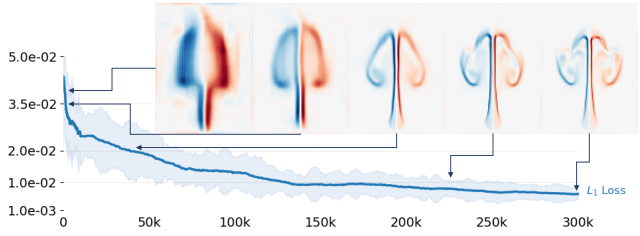


Figure 17: *Convergence plot of the L_1 loss for the 2-D smoke sequence from Figure 7.*

6.2. Performance Analysis

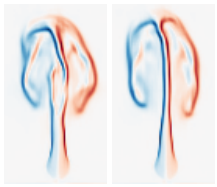
Table 1 summarizes the statistics of all presented examples. In terms of wall-clock time, the proposed CNN approach generates velocity fields up to 700× faster than re-simulating the data with the underlying CPU solver. Some care must be taken when interpreting this number because our Tensorflow network runs on the GPU, while the original Mantaflow code runs on the CPU. Fluid simulations are known to be memory bandwidth-limited [Kim08], 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.

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 [ATW15] or energy-preserving [MCP*09] solvers could potentially be used with our approach, yielding even larger speed-ups.

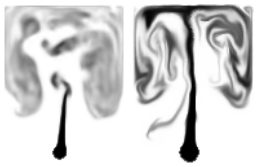
In contrast, computing the linear basis using traditional SVD-based subspace approaches can take between 20 [KD13] and 33 [WST09] hours. The process is non-iterative, so interrupting the computation can yield a drastically inferior result, i.e. the most important singular vector may not have been discovered yet. [SSW*13] reduced the precomputation time to 12 hours, but only by using a 110-node cluster. In contrast, our iterative training approach is fully interruptible and runs on a single machine.

Compression. The memory consumption of our method is at most 30 MB, which effectively compresses the input data by up to 1300×. Previous subspace methods [JSK16] only achieved ratios of 14×, hence our results improve on the state-of-the-art by two orders of magnitude. We have compared the compression ability of our network to FPZIP [LI06], a data reduction technique often used in scientific visualization [MTV17]. FPZIP is a prediction-based compressor using Lorenzo predictor, which is designed for large floating-point data sets in arbitrary dimension. For the two data sets of Section 5.1 and Section 5.2, we reached a compression of 172× and 356×, respectively. In comparison, FPZIP achieves a compression of 4× for both scenes with 16 bits of precision (with a mean absolute error comparable to our method). When allowing for a 6× larger mean absolute error, FPZIP achieves a 47× and 39× compression. I.e., the data is more than 4× larger and has a reduced quality compared to our encoding. Thus, our method outperforms commonly used techniques for compressing scientific data. Details can be found in the supplemental material.

6.3. Quality of Reconstruction and Interpolation



Training Data. Several factors affect the reconstruction and interpolation quality. 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 where only 11 training samples are used (left), instead of the 21 (right) from 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 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 ground truth 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.

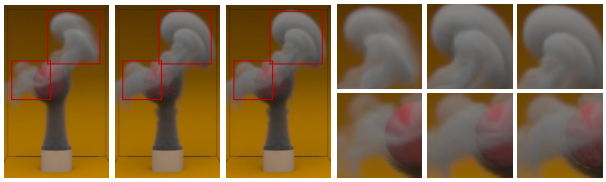


Figure 18: *Comparisons of the results from networks trained on our compressible loss, incompressible loss and the ground truth, respectively. On the right sequence we show the highlighted images from the simulations on the left. We notice that the smoke patterns from the incompressible loss are closer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

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)
Smoke Plume	96×128	21,000	0.033	0.052 [100]	635	2064	12	172	5
Smoke Obstacle	$64 \times 96 \times 64$	6,600	0.491	0.999 [5]	513	31143	30	1038	74
Smoke Inflow	$112 \times 64 \times 32$	3,750	0.128	0.958 [5]	128	10322	29	356	40
Liquid Drops	$96 \times 48 \times 96$	7,500	0.172	1.372 [3]	125	39813	30	**1327**	134
Viscous Dam	$96 \times 72 \times 48$	600	0.984	1.374 [3]	**716**	2389	29	82	100
Rotating Smoke	$48 \times 72 \times 48$	500	0.08	0.52 [10]	308	995	38	26	49
Moving Smoke	$48 \times 72 \times 48$	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.*

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, ghosting arises as features from the closest velocity fields are superimposed [Thu17], and 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.

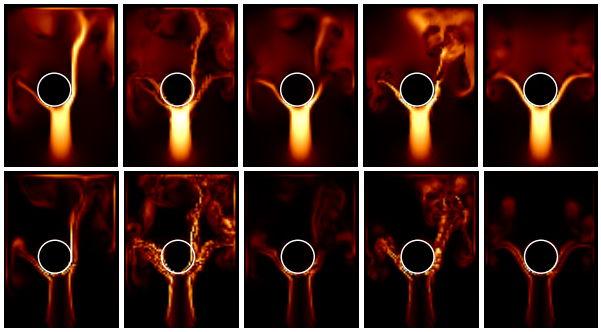


Figure 19: *Slice views of the last row of Fig. 8. The color code represents the velocity (top) and vorticity (bottom) magnitudes. The second column shows a linear interpolation of the input. Despite the absence of any constraints on boundary conditions, our method (third column) preserves the shape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

Liquid-air Interface. Due to the separate advection calculation for particles in our FLIP simulations, smaller splashes can leave the

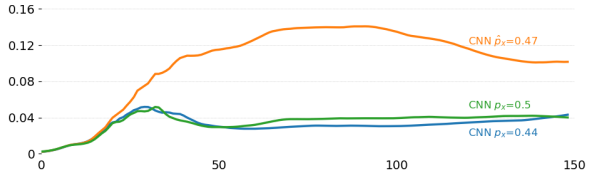


Figure 20: *Mean absolute error plot of velocity penetration for the smoke obstacle example. Though errors in interpolated samples are a bit higher than those of reconstructed samples, they do not exceed 1% of the maximum absolute value of the data set.*

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 smaller velocities than a threshold 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 10% of extrapolation still yielded plausible 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

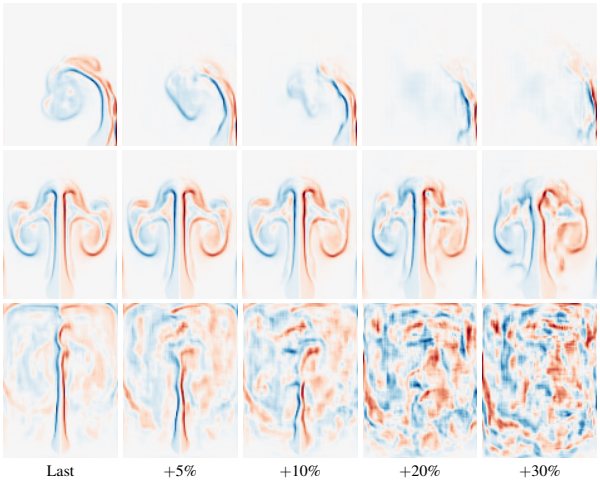


Figure 21: *2-D smoke plume extrapolation results (from t. to b.: position, inflow width, time) where only the generative network is used. Plausible results can be observed for up to 10% extrapolation.*

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 **c**, and there is a possible issue of temporal jittering because of lack of the gradient loss on Equation (5). We provide an extended discussion regarding the reconstruction quality and the latent space size on our supplemental material.

7. Conclusion

We have presented the 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 has a constant evaluation time and is considerably faster (up to 700×) than simulations with the underlying CPU solver, which makes our approach attractive for re-simulation scenarios where input interactions can be parameterized. These performance characteristics immediately suggest applications in games and virtual environments. As high resolution fluid simulations are also known to demand large disk and memory budgets, the compression characteristics of our algorithm (with up to 1300×) render the method attractive for movie productions 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), partial convolutions [LRS*18], joint training with other fields such as SDF, or alternative distance measures to enhance the accuracy for fine structures in the data. Additionally, it is an interesting direction to improve the latent space integration network in various ways, such as using an end-to-end training with the autoencoder or the gradient loss for temporal smoothness.

Generally, combining fluid simulations and machine learning is largely unexplored, and more work is needed to evaluate different architectures and physics-inspired components. Our CNN-based algorithm is the first of its kind in the fluids community, and we believe that the speed, interpolation and compression capabilities can enable a variety of future applications, such as interactive liquid simulations [PBT17] or implementation of fluid databases.

8. Acknowledgments

This work was supported by the Swiss National Science Foundation (Grant No. 200021_168997) and ERC Starting Grant 637014.

References

[ATW15] ANDO R., THUEREY N., WOJTAN C.: A stream function solver for liquid simulations. *ACM Transactions on Graphics 34*, 4 (2015), 53:1–53:9. 9

[BSM17] BERTHELOT D., SCHUMM T., METZ L.: BEGAN: Boundary Equilibrium Generative Adversarial Networks. arXiv:1703.10717. 4

[CT17a] CARLEO G., TROYER M.: Solving the quantum many-body problem with artificial neural networks. *Science 355*, 6325 (2017). 3

[CT17b] CHU M., THUEREY N.: Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Transactions on Graphics 36*, 4 (2017), 1–14. 2

[DLF12] DE WITT T., LESSIG C., FIUME E.: Fluid simulation using Laplacian eigenfunctions. *ACM Trans. Graph. 31*, 1 (2012), 1–11. 2

[FGP17] FARIMANI A. B., GOMES J., PANDE V. S.: Deep Learning the Physics of Transport Phenomena. arXiv:1709.02432. 2

[GDG*17] GUERIN E., DIGNE J., GALIN E., PEYTAVIE A., WOLF C., BENES B., MARTINEZ B.: Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks. *ACM Transactions on Graphics (proceedings of Siggraph Asia 2017) 36*, 6 (2017). 1

[GKSB15] GERSZEWSKI D., KAVAN L., SLOAN P.-P., BARGTEIL A. W.: Basis Enrichment and Solid-fluid Coupling for Model-reduced Fluid Simulation. *Computer Animation and Virtual Worlds 26* (2015). 2

[GLI16] GUO X., LI W., IORIO F.: Convolutional Neural Networks for Steady Flow Approximation. In *Proc. of Knowledge Discovery and Data Mining* (2016), pp. 481–490. 2

[GN07] GUPTA M., NARASIMHAN S. G.: Legendre fluids: a unified framework for analytic reduced space modeling and rendering of participating media. In *Symp. on Computer Anim.* (2007), pp. 17–25. 2

[GPAM*14] GOODFELLOW I., POUGET-ABADIE J., MIRZA M., XU B., WARDE-FARLEY D., OZAIR S., COURVILLE A., BENGIO Y.: Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*. 2014, pp. 2672–2680. 2

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *CVPR* (2016), pp. 770–778. 4

[JAFF16] JOHNSON J., ALAHI A., FEI-FEI L.: Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision* (2016), Springer, pp. 694–711. 3

[JSK16] JONES A. D., SEN P., KIM T.: Compressing Fluid Subspaces. In *Symposium on Computer Animation* (2016), pp. 77–84. 1, 2, 9

[KA15] KINGMA D., ADAM J. B.: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)* (2015), vol. 5. 6

[KALL17] KARRAS T., AILA T., LAINE S., LEHTINEN J.: Progressive Growing of GANs for Improved Quality, Stability, and Variation. arXiv:1710.10196. 1

[KD13] KIM T., DELANEY J.: Subspace fluid re-simulation. *ACM Transactions on Graphics 32*, 4 (2013), 1. 1, 2, 9

[Kim08] KIM T.: Hardware-aware Analysis and Optimization of Stable Fluids. In *Interactive 3D Graphics and Games* (2008), pp. 99–106. 9

[KMM*17] KALLWEIT S., MÜLLER T., MCWILLIAMS B., GROSS M., NOVÁK J.: Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Trans Graph 36*, 6 (2017). 1

[KWKT15] KULKARNI T. D., WHITNEY W. F., KOHLI P., TENENBAUM J.: Deep convolutional inverse graphics network. In *Advances in neural information processing systems* (2015), pp. 2539–2547. 5

[LH16] LOSHCHILOV I., HUTTER F.: SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv:1608.03983. 6

[LI06] LINDSTROM P., ISENBURG M.: Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1245–1250. 9

[LJF16] LU W., JIN N., FEDKIW R.: Two-way Coupling of Fluids to Reduced Deformable Bodies. In *SCA* (2016), pp. 67–76. 2

[LJS*15] LADICKÝ L., JEONG S., SOLENTHALER B., POLLEFEYS M., GROSS M.: Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics 34*, 6 (2015), 1–9. 1, 2

[LKT16] LING J., KURZAWSKI A., TEMPLETON J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics 807* (2016), 155–166. 3

[LMH*15] LIU B., MASON G., HODGSON J., TONG Y., DESBRUN M.: Model-reduced variational fluid simulation. *ACM Transactions on Graphics 34*, 6 (2015), 244. 2

[LRS*18] LIU G., REDA F. A., SHIH K. J., WANG T., TAO A., CATANZARO B.: Image inpainting for irregular holes using partial convolutions. arXiv:1804.07723. 11

[LTH*17] LEDIG C., THEIS L., HUSZÁR F., CABALLERO J., CUNNINGHAM A., ACOSTA A., AITKEN A. P., TEJANI A., TOTZ J., WANG Z., ET AL.: Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR* (2017), vol. 2, p. 4. 3

[Lum67] LUMLEY J. L.: The Structure of Inhomogeneous Turbulent Flows. In *Atmospheric turbulence and radio propagation*. Nauka, 1967, pp. 166–178. 2

[MCL15] MATHIEU M., COUPRIE C., LECUN Y.: Deep multi-scale video prediction beyond mean square error. arXiv:1511.05440. 4

[MCP*09] MULLEN P., CRANE K., PAVLOV D., TONG Y., DESBRUN M.: Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics 28*, 3 (2009), 1. 9

[MDB17] MOSSER L., DUBRULE O., BLUNT M. J.: Reconstruction of three-dimensional porous media using generative adversarial neural networks. *Physical Review E 96*, 4 (2017), 043309. 3

[MHN13] MAAS A., HANNUN A., NG A.: Rectifier nonlinearities improve neural network acoustic models. In *ICML* (2013), vol. 30. 4

[MTP*18] MA P., TIAN Y., PAN Z., REN B., MANOCHA D.: Fluid directed rigid body control using deep reinforcement learning. *ACM Transactions on Graphics (SIGGRAPH Proceedings)* (2018). 3

[MTV17] MEYER M., TAKAHASHI S., VILANOVA A.: Data reduction techniques for scientific visualization and data analysis. *STAR 36*, 3 (2017). 9

[PBT17] PRANTL L., BONEV B., THUEREY N.: Pre-computed Liquid Spaces with Generative Neural Networks and Optical Flow. arXiv:1704.07854. 2, 11

[PdON18] PAGANINI M., DE OLIVEIRA L., NACHMAN B.: Accelerating science with generative adversarial networks: an application to 3d particle showers in multilayer calorimeters. *Physical review letters 120*, 4 (2018), 042003. 3

[RLM*17] RAVANBAKHSH S., LANUSSE F., MANDELBAUM R., SCHNEIDER J. G., POCZOS B.: Enabling dark energy science with deep generative models of galaxy images. In *AAAI* (2017), pp. 1488–1494. 3

[Smi17] SMITH L. N.: Cyclical learning rates for training neural networks. In *Applications of Computer Vision* (2017), pp. 464–472. 6

[SSW*13] STANTON M., SHENG Y., WICKE M., PERAZZI F., YUEN A., NARASIMHAN S., TREUILLE A.: Non-polynomial galerkin projection on deforming meshes. *ACM Trans Graph 32*, 4 (2013), 86:1. 9

[Sta14] STANTON M.: *Data-Driven Methods for Interactive Simulation of Complex Phenomena*. PhD thesis, Carnegie Mellon Univ., 2014. 2

[SZZ*17] SCHAWINSKI K., ZHANG C., ZHANG H., FOWLER L., SANTHANAM G. K.: Generative adversarial networks recover features in astrophysical images of galaxies beyond the deconvolution limit. *Mon Not R Astron Soc.: Letters 467*, 1 (2017), L110–L114. 3

[Thu17] THUEREY N.: Interpolations of smoke and liquid simulations. *ACM Transactions on Graphics (TOG) 36*, 1 (2017), 3. 10

[TLP06] TREUILLE A., LEWIS A., POPOVIĆ Z.: Model reduction for real-time fluids. *ACM Transactions on Graphics 25*, 3 (2006), 826. 1, 2

[TP18] THUEREY N., PFAFF T.: MantaFlow, 2018. *mantaflow.com*. 6

[TSSP17] TOMPSON J., SCHLACHTER K., SPRECHMANN P., PERLIN K.: Accelerating Eulerian fluid simulation with convolutional networks. In *Proceedings of Machine Learning Research* (Aug 2017), vol. 70, pp. 3424–3433. 1, 2

[UB18] UMETANI N., BICKEL B.: Learning three-dimensional flow for interactive aerodynamic design. *ACM Transactions on Graphics (SIGGRAPH Proceedings)* (2018). 2

[UHT18] UM K., HU X., THUEREY N.: Liquid splash modeling with neural networks. *Computer Graphics Forum 37*, 8 (2018), 171–182. 2

[VLL*10] VINCENT P., LAROCHELLE H., LAJOIE I., BENGIO Y., MANZAGOL P.-A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res. 11* (2010), 3371–3408. 5

[WBT18] WIEWEL S., BECHER M., THUEREY N.: Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow. arXiv:1802.10123. 2

[WST09] WICKE M., STANTON M., TREUILLE A.: Modular bases for fluid dynamics. In *ACM SIGGRAPH* (2009). 2, 9

[XFCT18] XIE Y., FRANZ E., CHU M., THUEREY N.: tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 95. 1, 2

[YYX16] YANG C., YANG X., XIAO X.: Data-driven projection method in fluid simulation. *CAVW 27*, 3-4 (2016), 415–424. 2

[ZGFK17] ZHAO H., GALLO O., FROSIO I., KAUTZ J.: Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging 3*, 1 (2017), 47–57. 3