Unsupervised Deep Learning of Incompressible Fluid Dynamics

Nils Wandel
Department of Computer Science
University of Bonn
wandeln@cs.uni-bonn.de

Michael Weinmann
Department of Computer Science
University of Bonn
mw@cs.uni-bonn.de

Reinhard Klein
Department of Computer Science
University of Bonn
rk@cs.uni-bonn.de

Abstract

Fast and stable fluid simulations are an essential prerequisite for applications ranging from computer aided aerodynamic design of automobiles or airplanes to simulations of physical effects in CGI to research in meteorology. Recent differentiable fluid simulations allow gradient based methods to optimize e.g. fluid control systems in an informed manner. Solving the partial differential equations governed by the dynamics of the underlying physical systems, however, is a challenging task and current numerical approximation schemes still come at high computational costs.

In this work, we propose an unsupervised framework that allows powerful deep neural networks to learn the dynamics of incompressible fluids end to end on a grid-based representation. For this purpose, we introduce a loss function that penalizes residuals of the incompressible Navier Stokes equations. After training, the framework yields models that are capable of fast and differentiable fluid simulations and can handle various fluid phenomena such as the Magnus effect and Kármán vortex streets. Besides demonstrating its real-time capability on a GPU, we exploit our approach in a control optimization scenario.

1 Introduction

Simulating the behavior of fluid-like substances such as liquids (e.g. water, fuel) or gases (e.g. air) is of great importance for numerous applications in entertainment, advertisement, product design, urban planning and meteorology. Examples include the aerodynamic design of vehicles such as automobiles or airplanes, the accurate simulation of fluid phenomena to enrich scenes in computer games or movies, or weather simulations in meteorology. On top of simulating the behavior of fluids, differentiable fluid simulators allow to propagate gradients throughout the simulation. This is a fundamental prerequisite for gradient based methods to more efficiently perform e.g. control tasks in fluid domains [7].

As a full simulation of all interacting molecules within a fluid domain is deemed computationally infeasible, simplifications are needed. Most fluids can be described very precisely by the Navier Stokes equations - a set of non-linear partial differential equations which builds the foundation of most fluid simulations and takes important properties such as viscosity or fluid density into account. Since these Navier Stokes equations do not possess known analytical solutions for most application scenarios, we have to rely on numerical approximation methods. However, such numerical methods

Preprint. Under review.

face several challenging problems: to allow for large simulation timesteps, they must be stable and robust. In order to make use of GPUs they need to be highly parallelizable. Often, large systems of equations have to be solved with iterative methods and trade-offs between computational complexity and accuracy need to be found. On top of that, differentiable fluid simulators have to make sure that no computational steps hinder gradient propagation.

Traditional schemes for fluid simulation include Lagrangian and Eulerian approaches. While Lagrangian methods handle the fluid domain from the perspective of individual particles moving with the velocity field (e.g. smoothed particle hydrodynamics (SPH) [4]), their accuracy is achieved at the cost of simulating the motions of many individual particles. In contrast, Eulerian methods (e.g. [23]) handle fluid fields on a fixed reference frame such as a grid (finite difference methods) or a mesh (finite element methods). However, respective solvers relying on Preconditioned Conjugate Gradient approaches, iterative Jacobi or Gauss-Seidel methods, come at high computational costs. Recently, surrogate modeling with learning based approaches has gained a lot of attention due to short inference times and good generalization capabilities.

To the best of our knowledge, we present the first approach to learn the full incompressible Navier Stokes equations end-to-end on an Eulerian, grid-based representation. For this purpose, we use deep convolutional neural networks to recurrently infer velocity and pressure fields directly from given boundary conditions and the velocity and pressure fields of a previous timestep. In order to train the neural networks, we introduce an unsupervised training framework that produces random boundary configurations and forces the network to find solutions to these boundary conditions using a physically-constrained loss-function based on the residuals of the incompressible Navier Stokes equations. To achieve good performance and stability, we exploit the Helmholtz-decomposition allowing the network to output divergence-free vector fields and introduce pressure regularization for fluids with high Reynolds numbers. This end-to-end approach offers a simple, highly parallelizable, fully differentiable pipeline for computational fluid dynamics and allows to make use of efficient machine learning libraries. The unsupervised framework enables the network to learn intriguing fluid phenomena such as the Magnus effect or Kármán vortex streets without the need for fluid simulation software to generate big ground truth datasets. We make code and data publicly available at `release_upon_acceptance`.

2 Related Work

In literature, several different approaches can be found that aim to approximate the dynamics of PDEs in general and fluids in particular with more efficient, learning-based surrogate models.

Lagrangian methods such as smoothed particle hydrodynamcs (SPH) [4] handle fluids from the perspective of many individual particles that move with the velocity field. Following this approach, learning-based methods using regression forests [12], graph neural networks [15, 13] and continuous convolutions [27] have been developed. In addition, Smooth Particle Networks (SP-Nets) [20] allow for differentiable fluid simulations within the Lagrangian frame of reference. These Lagrangian methods are particularly suitable when a fluid domain exhibits large, dynamic surfaces (e.g. waves or droplets). However, to simulate the dynamics within a fluid domain, Eulerian methods such as finite differences or finite elements [1], that treat the Navier Stokes equations in a fixed frame of reference (e.g. a grid), are usually better suited.

Classical Eulerian approaches such as finite differences methods on marker and cell (MAC) grids date back to the world of Harlow and Welch [6]. The work of Stam [23] is well suited for computer graphics and builds the foundation for many special effects in computer generated imagery but is not accurate enough for most engineering applications as it suffers from numerical dissipation. Following this approach of Stam, recently, Holl et al. [7] implemented a differentiable fluid simulation on a MAC grid to train efficient fluid control algorithms. In the following, we will discern two types of learning-based approaches in the Eulerian frame of reference. *Continuous approaches* learn to map domain coordinates (e.g. x,y,t) directly onto field values (e.g. velocity / pressure). In contrast, *discrete approaches* model PDEs on a grid using for example convolutional neural networks.

Continuous methods allow mesh-free solutions even when considering only a few points during training and work in complex, high-dimensional domains [22, 5, 8], when traditional discretization techniques come impractical. Recent applications focused on flow through porous media [30, 31, 26], fluid modeling [29, 18], turbulence modeling [2, 14] and modeling of molecular dynamics [21].

Training is usually based on a physics constrained loss functions that penalize residuals of the underlying PDEs. Similar to our approach, Raissi et al. [17] used vector potentials to obtain continuous divergence-free velocity fields to approximate the incompressible Navier Stokes equations. Continuous methods can overcome the curse of dimensionality [5] and have been demonstrated to be capable of returning smooth, accurate results for high-dimensional PDEs. However, the learned results are domain-specific and thus cannot be used in interactive scenarios.

Discrete grid-based methods, on the other hand, aim to generalize to new domains without retraining. Among discrete, grid-based approaches, several methods focused on data-driven fluid modeling. Respective approaches include the use of CNNs to learn the Reynolds averaged Navier Stokes equations (RANS) [24] from training data generated with OpenFOAM [16], the use of a data-driven approach to continuously interpolate between training data examples of smoke and fluid simulations [9], and the use of generative adversarial networks [28, 10] to generate temporal consistent smoke simulations at high resolutions. While such methods yield fast simulations, training needs ground truth data and may suffer from issues such as mode-collapse hindering physical accuracy.

Physics-constrained loss functions on the other hand can be directly applied to the output of a network without the need for ground-truth data and thus, reduce memory requirements while being less prone to overfitting. This way, Tompson et al. [25] used CNNs to solve the Poisson equation for a Helmholtz projection step to accelerate a pipeline inspired by Stam's approach [23] for inviscid fluids. Furthermore, Geneva and Zabara [3] formulated a more general framework for physics-constrained loss functions on a square grid and presented results on Burger's equations. In this work, we go a step further and aim to solve the full incompressible Navier Stokes equations end-to-end on a MAC grid using a U-Net [19] and a vector potential for the velocity field.

3 Method

In this section, we provide a detailed description of our approach for unsupervised learning of incompressible fluid dynamics. For this purpose, we first motivate a physics constrained loss function based on the incompressible Navier Stokes equations. Then, we introduce a pressure regularization term for very high Reynolds numbers and explain, how we exploit the Helmholtz decomposition to obtain divergence-free velocity fields to ensure incompressibility within the fluid domain. Finally, we provide details of our discrete differentiable spatio-temporal fluid representation and its use for deep learning based fluid modeling.

3.1 Physics-constrained Loss based on Incompressible Navier Stokes Equations

The goal is to simulate a fluid consisting of a velocity field $\vec{v}$ and a pressure field p within a domain Ω given Dirichlet boundary conditions $\vec{v}_d$ on the boundary of the domain $\partial\Omega$ and initial conditions $\vec{v}^0$ and p^0 for the velocity and pressure field at the beginning of the simulation.

Thus, we aim to solve the incompressible Navier Stokes equations for $\vec{v}$ and p within Ω, which read as follows:

$$\nabla \cdot \vec{v} = 0 \qquad\qquad\qquad\qquad\qquad\qquad \text{incompressibility on } \Omega \qquad (1)$$

$$\rho\dot{\vec{v}} = \rho\left(\frac{\partial\vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\,\vec{v}\right) = -\nabla p + \mu\Delta\vec{v} + \vec{f} \qquad \text{conservation of momentum on } \Omega \qquad (2)$$

Here, ρ describes the fluid density and μ its viscosity. In our case, external forces on the fluid (such as e.g. gravity) can be neglected, so we set $\vec{f} = 0$.

Dirichlet boundary conditions mean, that $\vec{v}$ must match $\vec{v}_d$ at the boundaries:

$$\vec{v} = \vec{v}_d \qquad\qquad\qquad \text{Dirichlet boundary condition on } \partial\Omega \qquad (3)$$

3

Using the residuals of these equations, we can formulate the following divergence and momentum loss terms on Ω as well as a boundary loss term on $\partial\Omega$:

$$L_d = \|\nabla \cdot \vec{v}\|^2 \qquad\qquad \text{on } \Omega \qquad (4)$$

$$L_p = \left\|\rho\left(\frac{\partial\vec{v}}{\partial t} + (\vec{v}\cdot\nabla)\,\vec{v}\right) + \nabla p - \mu\Delta\vec{v} - \vec{f}\right\|^2 \qquad\qquad \text{on } \Omega \qquad (5)$$

$$L_b = \|\vec{v} - \vec{v}_d\|^2 \qquad\qquad \text{on } \partial\Omega \qquad (6)$$

Combining the described loss terms, we obtain the following loss function:

$$L = \alpha L_b + \beta L_d + \gamma L_p \qquad (7)$$

where α, β, γ are hyperparameters that weight the contributions of the different loss terms.

3.2 Pressure Regularization

In the special case of very high Reynolds numbers (see equation 15) and inviscid flows, training becomes unstable as viscous friction cannot dissipate enough energy out of the system anymore. This leads to unrealistic gradients for $\vec{v}$ and p. For such cases, we introduce an additional regularization term

$$L_r = \|\nabla p\|^2 \qquad (8)$$

for the loss function (7) to stabilize training. The intuition behind this regularization term is, that we want to penalize unrealistically high energies of the pressure field.

3.3 Helmholtz Decomposition

A common method to ensure incompressibility of a fluid is to project the flow field onto the divergence-free part of its Helmholtz decomposition. The Helmholtz theorem states that every vector field $\vec{v}$ can be decomposed into a curl-free and a divergence-free part:

$$\vec{v} = \nabla p + \nabla \times \vec{a} \qquad (9)$$

Note, that $\nabla \times (\nabla p) = \vec{0}$ and $\nabla \cdot (\nabla \times \vec{a}) = 0$.

The projection consists of solving the Poisson problem $\nabla\vec{v} = \Delta p$ for p, followed by substracting ∇p from the original flow field. However, solving the Poisson equation on arbitrary domains comes at high computational costs for classical methods and one has to rely e.g. on conjugate gradient methods to approximate its solution.

Here, we propose a different approach and directly try to learn a vector potential $\vec{a}$ with $\vec{v} = \nabla \times \vec{a}$. This ensures that the network outputs a divergence-free velocity field within the domain Ω and allows omitting the divergence loss L_d. In this setting, the two remaining loss terms become:

$$L_p = \left\|\left(\frac{\partial(\nabla\times\vec{a})}{\partial t} + ((\nabla\times\vec{a})\cdot\nabla)(\nabla\times\vec{a})\right) + \nabla p - \mu\Delta(\nabla\times\vec{a}) - \vec{f}\right\|^2 \qquad \text{on } \Omega \qquad (10)$$

$$L_b = \|(\nabla\times\vec{a}) - \vec{v}_d\|^2 \qquad\qquad \text{on } \partial\Omega \qquad (11)$$

In this work, we consider 2D fluid simulations, so only the z-component of $\vec{a}$, a_z, is of interest as v_z and all derivatives with respect to the z-axis are zero:

$$\nabla \times \vec{a} = \begin{pmatrix} \partial_y a_z - \partial_z a_y \\ \partial_z a_x - \partial_x a_z \\ \partial_x a_y - \partial_y a_x \end{pmatrix} = \begin{pmatrix} \partial_y a_z \\ -\partial_x a_z \\ 0 \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ 0 \end{pmatrix} = \vec{v} \qquad (12)$$

3.4 Discrete Spatio-temporal Fluid Representation

Marker and Cell (MAC) grid To solve the Navier Stokes equations, we represent the relation between a_z, v_x, v_y, p in Eulerian coordinates on a 2D staggered marker and cell (MAC) grid (see Figure 1a).Therefore, we discretise time and space as follows:

$$\vec{a}(x,y,t) = \begin{pmatrix} 0 \\ 0 \\ (a_z)^t_{i,j} \end{pmatrix} \; ; \; \vec{v}(x,y,t) = \begin{pmatrix} (v_x)^t_{i,j} \\ (v_y)^t_{i,j} \end{pmatrix} \; ; \; p(x,y,t) = p^t_{i,j} \qquad (13)$$

(a) Layout of Staggered Marker and Cell (MAC) grid.
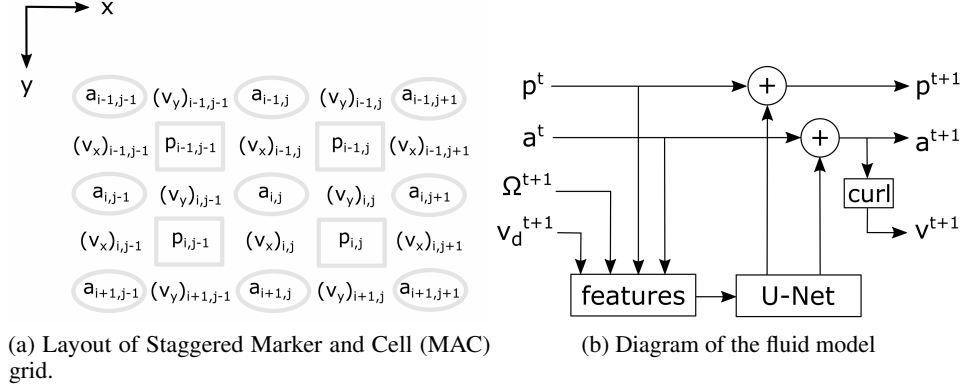
(b) Diagram of the fluid model

Figure 1: MAC grid and diagram of the fluid model

Obtaining gradient, divergence, Laplace and curl operations on this grid with finite differences is straight forward and can be efficiently implemented with convolutions. For detailed descriptions regarding the fully discretized loss-functions, we refer to the supplementary material.

Explicit, Implicit, Implicit-Explicit (IMEX) Loss Formulations The discretization of the time domain is needed to deal with the time-derivative of the velocity field $\frac{\partial \vec{v}}{\partial t}$ in L_p, which becomes:

$$L_p = \left\| \left(\frac{\vec{v}^{t+1} - \vec{v}^t}{dt} + \left(\vec{v}^{t'} \cdot \nabla \right) \vec{v}^{t'} \right) + \nabla p^{t+1} - \mu \Delta \vec{v}^{t'} - \vec{f} \right\|^2 \tag{14}$$

The goal is to take timesteps dt that are as large as possible while maintaining stable and accurate solutions. Stability and accuracy largely depend on the definition of $v^{t'}$. In literature, choosing $v^{t'} = v^t$ is often referred to as explicit integration schemes and frequently leads to unstable behavior. Choosing $v^{t'} = v^{t+1}$ is usually associated with implicit integration schemes and gives stable solutions at the cost of numerical dissipation. Implicit-Explicit (IMEX) schemes, that set $v^{t'} = (v^t + v^{t+1})/2$ are a compromise between both methods and considered to be more accurate but less stable than implicit methods. Especially when propagating gradients throughout the simulation (e.g. for control optimization tasks), training the model beforehand with the implicit definition of $v^{t'}$ returned more stable gradients.

3.5 Fluid Model

We represent the fluid dynamics by a recurrent model that maps the fluid state $p^t, \vec{a}^t$ for timestep t and the domain description $\Omega^{t+1}, \vec{v}_d^{t+1}$ to the fluid state $p^{t+1}, \vec{a}^{t+1}$ of the next timestep. Here, p^t describes the pressure field and $\vec{a}^t$ describes the vector potential of $\vec{v}^t$. For $t = 0$, we can set $p^0 = 0$ and $\vec{a}^0 = \vec{0}$. Ω^{t+1} is a binary mask that contains the domain geometry and is 1 for the fluid domain and 0 everywhere else. For the boundary of the domain, we simply take the inverse of Ω: $\partial\Omega = 1 - \Omega$. $\vec{v}_d^{t+1}$ represents the Dirichlet boundary conditions and contains a velocity field that must be matched by $\vec{v}^{t+1}$ at the domain boundaries. Figure 1b shows a diagram of the fluid model. First, $\left(p^t, \vec{a}^t, \Omega^{t+1}, \vec{v}_d^{t+1} \right)$ is taken to derive a feature representation that comprises $\left(p^t, a^t, \nabla \times a^t, \Omega^{t+1}, \partial\Omega^{t+1}, \Omega^{t+1} \cdot \nabla \times a^t, \Omega^{t+1} \cdot p^t, \partial\Omega^{t+1} \cdot \vec{v}_d^{t+1} \right)$. These features are then fed into a U-Net [19] with a reduced number of channels (the exact network configuration can be found in the supplementary material). The mean of the U-Net output is set to 0 in order to keep p and $\vec{a}$ well defined and prevent drifting offset values. Finally, the output is added to p^t and $\vec{a}^t$ to obtain p^{t+1} and $\vec{a}^{t+1}$ respectively.

3.6 Training Procedure and Implementation Details

Training starts with initializing a pool $\{\Omega_k^0, (v_d)_k^0, (a_z)_k^0, p_k^0\}$ of randomized domains Ω_k^0 and boundary conditions $(v_d)_k^0$ as well as initial conditions for the vector potential and pressure fields that we

both set to zero ($(a_z)_k^0 = \vec{0}$ and $p_k^0 = 0$). The resolution of our training domains was 100x300 grid cells and example-domains of the training pool are shown in the supplementary material.

At each training step, a random mini-batch $\{\Omega_k^t, (v_d)_k^t, (a_z)_k^t, p_k^t\}_{\{k \in \text{minibatch}\}}$ is drawn from the pool and fed into the neural network which is designed to predict the velocity ($\vec{v}_k^{t+1} = \nabla \times \vec{a}_k^{t+1}$) and pressure ($p_k^{t+1}$) fields of the next time step. Based on the physically constrained loss-function (7), we update the weights of the network using the Adam optimizer [11]. At the end of each training step, the pool is updated by replacing the old vector potential and pressure fields $(a_z)_k^t, p_k^t$ by the newly generated ones $(a_z)_k^{t+1}, p_k^{t+1}$.

From time to time, old environments of the training pool are replaced by new randomized environments and the vector potential as well as the pressure fields are reset to 0. This increases the variance of the training pool and helps the neural network to learn "cold starts" from 0-velocity and 0-pressure fields.

Besides the fluid model described above, which we denote as $\vec{a}$-Net in the following, we also trained an ablation model, $\vec{v}$-Net, that directly learns to predict the velocity field without a vector potential. For the implementation of both models, we used the popular machine learning framework Pytorch and trained the models on a NVidia GeForce RTX 2080 Ti where the training converged after about 1 day. The hyperparameters in the loss-function for the $\vec{a}$-Net were $\beta = 1$ and $\gamma = 20$. The reason for choosing a higher weight for the loss term L_b than for L_p was the observation, that errors in L_b can lead to unrealistic flows leaking through boundaries. For the ablation study ($\vec{v}$-Net), we used $\alpha = 100, \beta = 1, \gamma = 0.001$. Here, we had to choose a very high weight for L_d to ensure incompressibility of the fluid, otherwise unrealistic source and sink effects start to appear. For L_b on the other hand, we used a very low weight as the boundary conditions can be trivially learned by the $\vec{v}$-Net. We used these parameter settings for all experiments.

4 Results

To evaluate the potential of our method, we demonstrate its capability to re-produce physical effects such as vortex streets an the Magnus effect. In addition, we provide an analysis of the generalization capability and demonstrate its real-time performance.

4.1 Qualitative Evaluation

Qualitative Analysis of Wake Dynamics The Reynolds number is a dimensionless quantity closely related to qualitative effects in fluid dynamics such as the wake dynamics behind an obstacle. It is defined by:

$$Re = \frac{\rho \, \|\vec{v}\| \, D}{\mu} \tag{15}$$

Here, ρ is the fluid density, $\|\vec{v}\|$ is the fluid speed, D is the diameter of the obstacle, and μ is the viscosity. (We use the units of the grid).

We retrained models for different values of μ and ρ to compare the fluid behavior for a wide range of Reynolds numbers. Figure 2 shows, that the trained models are able to predict the wake dynamics behind an obstacle in good accordance with qualitative expectations from fluid dynamics. As a rule of thumb, for $Re \ll 1$, the flow becomes time-reversible. This can be noticed in Figure 2a by the symmetry of the flow before and after the obstacle and the nearly constant pressure gradient within the pipe. Starting from $Re \approx 10$, the flow is still laminar but a static wake is forming behind the obstacle (see Figure 2b). For Reynolds numbers $Re > \approx 90$, Kármán vortex streets start to appear (see Figure 2c). A Kármán vortex street consists of clock and counterclockwise spinning vortices that are generated at the obstacle and then start moving in a regularly oscillating pattern with the flow. For very large Reynolds numbers or inviscid flows, the flow field becomes turbulent, which can be recognized by the irregular patterns behind the obstacle in Fig 2d.

Magnus effect The Magnus effect appears when a flow interacts with a rotating body. It is widely known e.g. in sports such as soccer or tennis where spin is used to deflect the path of a ball. The reason for the deflection stems from a low pressure field where the surface of the object moves along flow direction and a high pressure field where the object surface moves against the flow. Figure 3a shows, that our models are able to reproduce the Magnus effect around a rotating cylinder.
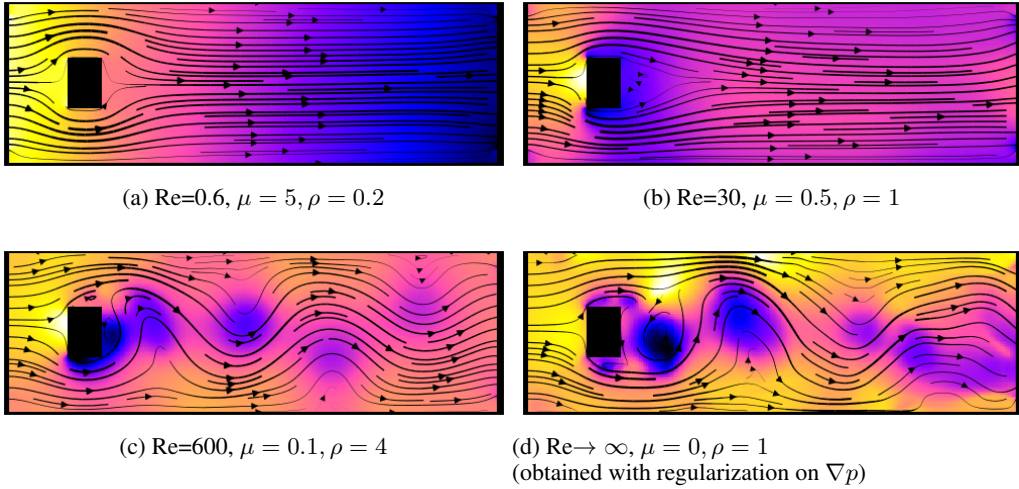
(a) Re=0.6, $\mu = 5, \rho = 0.2$

(b) Re=30, $\mu = 0.5, \rho = 1$

(c) Re=600, $\mu = 0.1, \rho = 4$

(d) Re$\to \infty, \mu = 0, \rho = 1$
(obtained with regularization on ∇p)

Figure 2: After training, our models are able to show correct wake flow dynamics for a wide range of different Reynolds numbers. ($D = 30, \|\vec{v}\| = 0.5$). Streamlines indicate flow direction, linewidth indicates speed and colors represent the pressure field (blue: low pressure / yellow: high pressure).

(a) Magnus effect on a clock-wise turning cylinder.

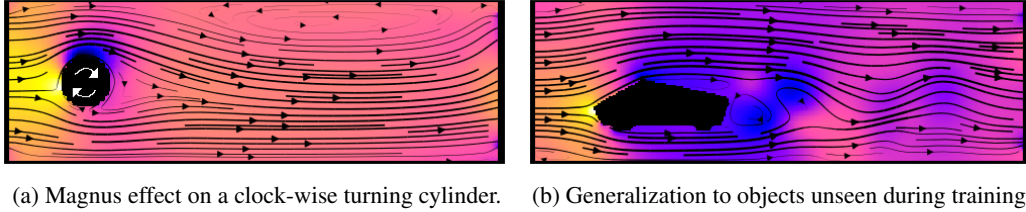(b) Generalization to objects unseen during training.

Figure 3: Our models feature the Magnus effect and generalize to new flow domains.

Analysis of Generalization Capability   We also tested the networks capability to generalize to objects not seen during training. Figure 3b shows the networks capability to meet boundary conditions and return a plausible pressure field for a domain that was not contained in the pool of training domains.

Real-time capability   Since progressing the simulation by a timestep consists merely of one pass through a convolutional neural network, it can be done at comparably low computational costs and can easily be parallelized. This enables for example interactive real-time simulations. We implemented a demo that allows to interact with a fluid by moving obstacles, rotating spheres and changing the flow speed within a pipe (see video in supplementary material). Our method runs at 250 timesteps per second on a 100x300 grid. Therefore, we used a NVidia GeForce RTX 2080 Ti consuming about 860 MB of GPU memory.

4.2   Quantitative Evaluation

The quantitative comparison of different fluid solvers comes with several challenges, as their performance is highly dependent on factors like the geometry of the domain, fluid parameters such as viscosity or density, flow speed or the timestep of the integrator. As benchmarks for fluid simulations on MAC grids are not yet available, we built a simple toy domain on a 100 x 100 grid which simulates a flow around an obstacle within a pipe (similar to Figure 2). More details are provided in the supplementary material. We compared our method ($\vec{a}$-Net) against the recently released open source differentiable fluid simulator PhiFlow [7] and provide an ablation study ($\vec{v}$-Net) to show the performance of our method with and without the use of the Helmholtz decomposition (see Table 1).

First, we compared the computational speed on a CPU and GPU by looking at the number of timesteps that the different methods could integrate per second. The $\vec{v}$-Net as well as the $\vec{a}$-Net were significantly faster than PhiFlow as they do not rely on a conjugate gradient solver but instead use a convolutional neural network that has a fast inference time and can be easily parallelized on a GPU. To provide a fair comparison on L_d, we set the velocity field at the boundaries equal to $\vec{v}_d$. This enables us to compute L_d for the $\vec{a}$-Net architecture on the domain boundaries which would otherwise have zero divergence everywhere. We noticed, that PhiFlow yields significantly better performance on L_d for smaller timesteps (e.g. for $dt = 1 : L_d =$2.5e-5 and for $dt = 0.1 : L_d =$3e-7), however, this further increases its computational burden. Finally, we compared the momentum loss. For both, L_d and L_p, the $\vec{a}$-Net architecture significantly outperformed the more naive $\vec{v}$-Net approach.

Method	TpS on CPU	TpS on GPU	L_d	L_p
PhiFlow	7	-	6.2e-4	-
$\vec{v}$-Net (ours)	**82**	**311**	8.66e-7	4.87e-5
$\vec{a}$-Net (ours)	**82**	**311**	**5.44e-7**	**1.56e-5**

Table 1: Quantitative comparison of timesteps per second (TpS) on CPU / GPU as well as divergence loss and momentum loss for differentiable fluid solvers on a 100x100 grid for timesteps of size $dt = 4$.

4.3   Controlling Frequency of Kármán Vortex Street

Whereas the primary focus of this work is placed on learning incompressible fluid dynamics, we also provide a proof of concept to show the effectiveness of backpropagating gradients through time in order to obtain differentiable fluid simulations with the recurrent $\vec{a}$-Net.

For this purpose, we considered an application scenario, where we aim at controlling the frequency of Kármán vortex streets by changing the flow speed within a pipe. To address this problem, we use an iterative approach where we perform the following major steps within each iteration until we reach convergence:

1. We first simulate a flow at a certain speed (as specified in $\vec{v}_d^t$) within a pipe using the $\vec{a}$-Net and capture the y component of the flow speed $v_y(t)$ behind an obstacle over time.

2. Subsequently, we compute the fast Fourier transform of $v_y(t)$: $V_y(f)$ and introduce a loss function, that rewards strong signals at a given target frequency $\hat{f}$:

$$L = -\int_0^\infty e^{-\frac{(f-\hat{f})^2}{\sigma^2}} \cdot |V_y(f)|^2 df \qquad (16)$$

   We observed that projecting $|V_y(f)|^2$ onto a Gaussian with standard deviation σ centered around $\hat{f}$ returned more stable gradients than directly setting $L = -|V_y(\hat{f})|^2$.

3. This is followed by the computation of the gradients of the loss function with respect to $\vec{v}_d^t$ by backpropagating gradients through time: $\frac{\partial L}{\partial \vec{v}_d^t}$

4. Finally, we optimize the flow speed by updating $\vec{v}_d^t$ with a gradient descent step (e.g. using the Adam optimizer [11].)

A more detailed description can be found in the supplementary material. We want to stress out, that differentiable fluid simulations are limited to scenarios with low Reynolds numbers and laminar flows as in the presence of turbulences, chaotic behavior will lead to exploding gradients which can be explained by the butterfly effect.

5   Discussion and Outlook

In this work, we presented an unsupervised learning scheme for the incompressible Navier Stokes equations. We introduced a fluid model that uses a vector potential to output divergence-free velocity fields. Qualitative results of our trained fluid models are in good accordance with expectations from fluid dynamics for a wide range of Reynolds numbers and generalize to unknown fluid domains. Quantitative assessment showed superior performance for large integration timesteps compared to

PhiFlow and a naive approach that directly predicts the velocity field. Furthermore, our fluid model allows for significantly faster simulations than PhiFlow. We gave a real-time demo and showed how differentiability can be used for optimization tasks in a fluid control scenario.

In the future, this model could be extended to 3D convolutional neural networks and multi-phase domains. On top of Dirichlet boundary conditions, Neumann boundary conditions could be incorporated as well.

Broader Impact

Surrogate models that capture physical relationships are of great importance for simulation tasks and control systems with numerous industrial applications. In addition, the integration of such physics-based constraints into learning techniques allows gaining efficiency, avoiding the need for ground-truth data and reducing the susceptibility regarding overfitting.

As the accuracy of deep surrogate models for grid-based solutions of the Navier Stokes equations will further increase in the future, such fast and differentiable fluid simulations could be applied in a wide variety of research and engineering fields. Researchers could speed up meteorology simulations or investigate the diffusion of aerosols to study the spread of airborne infections such as e.g. Covid-19. Engineers could accelerate development of aerodynamic proof of concepts in simulated wind-tunnels and use differentiable physics for various optimization tasks in fields like flow control and reinforcement learning. In computer generated imagery, this method could speed up physics engines for games or movies to obtain more realistic smoke or water effects.

We do not believe that this work puts anyone at disadvantage. However, if learning-based fluid simulators are used to optimize highly critical tasks, one should keep track of L_d and L_p during inference time and validate the results with well-established classical approaches. As our method directly learns from a physics-constrained loss function, it is not affected by biases in the data.

Acknowledgments and Disclosure of Funding

References

[1] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471 – 483, 1996.

[2] N. Geneva and N. Zabaras. Quantifying model form uncertainty in reynolds-averaged turbulence models with bayesian deep neural networks. *Journal of Computational Physics*, 383:125 – 147, 2019.

[3] N. Geneva and N. Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[4] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[5] P. Grohs, F. Hornung, A. Jentzen, and P. Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

[6] F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids*, 8(12):2182–2189, 1965.

[7] P. Holl, V. Koltun, and N. Thuerey. Learning to control pdes with differentiable physics. *ICLR*, 2020.

[8] Y. Khoo, J. Lu, and L. Ying. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences*, 6(1):1, 2019.

[9] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[10] J. Kim and C. Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. *Journal of Computational Physics*, 406:109216, 2020.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[12] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6), Oct. 2015.

[13] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[14] J. Ling, A. Kurzawski, and J. Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[15] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. K. Yamins. Flexible neural representation for physics prediction. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8813–8824, Red Hook, NY, USA, 2018. Curran Associates Inc.

[16] OpenCFD. *OpenFOAM - The Open Source CFD Toolbox - User's Guide*. OpenCFD Ltd., United Kingdom, 1.4 edition, 11 Apr. 2007.

[17] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019.

[18] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. *arXiv preprint arXiv:1808.04327*, 2018.

[19] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[20] C. Schenck and D. Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Conference on Robot Learning*, pages 317–335, 2018.

[21] M. Schöberl, N. Zabaras, and P.-S. Koutsourelakis. Predictive collective variable discovery with deep bayesian models. *The Journal of Chemical Physics*, 150(2):024109, 2019.

[22] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[23] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999.

[24] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, pages 1–12, 2019.

[25] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3424–3433. JMLR. org, 2017.

[26] R. K. Tripathy and I. Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565 – 588, 2018.

[27] B. Ummenhofer, L. Prantl, N. Thuerey, and V. Koltun. Lagrangian fluid simulation with continuous convolutions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[28] Y. Xie, E. Franz, M. Chu, and N. Thuerey. Tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Trans. Graph.*, 37(4), July 2018.

[29] C. Yang, X. Yang, and X. Xiao. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.

[30] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415 – 447, 2018.

[31] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56 – 81, 2019.