Accelerating Eulerian Fluid Simulation With Convolutional Networks

Jonathan Tompson [1] **Kristofer Schlachter** [2] **Pablo Sprechmann** [2 3] **Ken Perlin** [2]

Abstract

Efficient simulation of the Navier-Stokes equations for fluid flow is a long standing problem in applied mathematics, for which state-of-the-art methods require large compute resources. In this work, we propose a data-driven approach that leverages the approximation power of deep-learning with the precision of standard solvers to obtain fast and highly realistic simulations. Our method solves the incompressible Euler equations using the standard operator splitting method, in which a large sparse linear system with many free parameters must be solved. We use a Convolutional Network with a highly tailored architecture, trained using a novel unsupervised learning framework to solve the linear system. We present real-time 2D and 3D simulations that outperform recently proposed data-driven methods; the obtained results are realistic and show good generalization properties.

1. Introduction

Real-time simulation of fluid flow is a long standing problem in many application domains: from computational fluid dynamics for industrial applications, to smoke and fluid effects for computer graphics and animation. High computational complexity of existing solutions has meant that real-time simulations have been possible under restricted conditions. In this work we propose a data-driven solution to the invicid-Euler equations that is faster than traditional methods, while remaining competitive in long-term simulation stability and accuracy. This work is focused on computer graphics animation as the driving application for our architecture, however the techniques presented here can be extended to more complicated forms of Navier-Stokes (with appropriate modifications that are outside the scope of this work).

[1]Google Brain, Mountain View, USA [2]New York University, New York, USA [3]Google Deepmind, London, UK. Correspondence to: Jonathan Tompson <tompson@google.com>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

The dynamics of a large number of physical phenomenon are governed by the incompressible Navier-Stokes equations, which are a set of partial differential equations that must hold throughout a fluid velocity field for all time steps. There are two main computational approaches for simulating these equations: the Lagrangian methods, that approximate continuous quantities using discrete moving particles (Gingold & Monaghan, 1977), and the Eulerian, methods that approximate quantities on a fixed grid (Foster & Metaxas, 1996). We adopt the latter for this work.

Eulerian methods are able to produce accurate results simulating fluids like water with high compute costs. The most demanding portion of this method is the "pressure projection" step, which satisfies an incompressibility constraint. It involves solving the discrete Poisson equation and leads to a well-known sparse, symmetric and positive-definite linear system. Exact solutions can be found via traditional convex optimization techniques, such as the Preconditioned Conjugate Gradient (PCG) algorithm or via stationary iterative methods, like the Jacobi or Gauss-Seidel methods. PCG exhibits fast asymptotic convergence, however it suffers from high time-constants and is generally not suited to GPU hardware. The Jacobi method is used for real-time applications, however it suffers from poor asymptotic convergence. Additionally, the computational complexity of both these algorithms is strongly *data-dependent* (i.e. boundary condition dependent) and these iterative methods are truncated in real-time to fit within a computational budget.

In this paper, we propose a machine learning based approach for accelerating this linear projection that is fast and whose complexity is *data-independent*. We leverage the power of deep-learning techniques to derive an approximate linear projection. We claim that our machine-learning approach can take advantage of the statistics of fluid data and the local sparsity structure of the discrete Poisson equation to learn an approximate method that ensures long-term stability using a fixed computational budget. The main contributions of this work are as follows:

(i) We rephrase the learning task as an unsupervised learning problem; since ground-truth data is not required we can incorporate loss information from multiple time-steps and perform various forms of non-trivial data-augmentation.

(ii) We propose a collection of domain-specific ConvNet architectural optimizations motivated by the linear system structure itself, which lead to both qualitative and quantitative improvements. We incorporate an in-line normalization scheme, we devise a multi-resolution architecture to better simulate long range physical phenomena and we formulate the high level solver architecture to include a "pressure bottleneck" to prevent trivial solutions of our unsupervised objective.

(iii) Our proposed simulator is stable and is fast enough to permit real-time simulation. Empirical measurements suggest good generalization properties to unseen settings.

(iv) We provide a public dataset and processing pipeline to procedurally generate random ground-truth fluid frames for evaluation of simulation methods.

The paper is organized as follows. We discuss related work in Section 2. In Section 3 we briefly introduce the fluid simulation techniques used in this paper. In Section 4 we present the proposed model. We provide implementation details in Section 5. Experimental results are described in Section 6 and conclusion are drawn in Section 7.

2. Related Work

Recent work has addressed the problem of computational performance in fluid simulations. (McAdams et al., 2010) proposed a multi-grid approach as a preprocessing step of the PCG method. This method can significantly improve performance in large scene settings. However, multi-grid methods are very difficult to implement, hard to parallelize on the GPU, have non-trivial "failure-cases" (e.g. multi-grid complexity resorts to single-resolution complexity for simulations with highly irregular domain boundaries), and the method still requires a iterative optimization procedure with data-dependent complexity.

Some methods propose inexact (but efficient) approximate solutions to the Poisson equation, such as iterated orthogonal projections (Molemaker et al., 2008) or coarse-Grid corrections (Lentine et al., 2010). While these approaches can be competitive in low resolution settings, they are data agnostic: they do not exploit the statistics of the data to be processed. A natural approach is to tackle the problem in a data-driven manner - by adapting the solver to the statistics of the data of interest. The main idea of data-driven methods is to reduce computation by operating on a representation of the simulation space of significantly lower dimensionality. The Galerkin projection transforms the dynamics of the fluid simulation to operations on linear combinations of pre-processed snap-shots (Treuille et al., 2006; De Witt et al., 2012). In (Raveendran et al., 2014) the authors propose generating complex fluid simulations by interpolating a relatively small number of existing pre-processed simulations. The state graph formulation (Stanton et al., 2014) leverages the observation that on simple simulations only a small number of states are visited.

More recently - and most related to this work - some authors have regarded the fluid simulation process as a supervised regression problem. These approaches train black-box machine learning systems to predict the output produced by an exact solver using random regression forests (Ladický et al., 2015) or neural networks (Yang et al., 2016) for Lagrangian and Eulerian methods respectively. Ladický et al., propose an adaptation of regression forests for smoothed particle hydrodynamics. Given a set of hand-crafted features corresponding to individual forces, the authors trained a regressor for predicting the state of each particle in the following time-step. Yang et al. train a patch-based neural network to predict the ground truth pressure given local previous-frame pressure, voxel occupancy, and velocity divergence of an input frame.

A major limitation of existing learning-based methods is that they require a dataset of linear system solutions provided by an exact solver. Hence, targets cannot be computed during training and models are trained to predict the ground-truth output always starting from an initial frame produced by an exact solver, while at test time this initial frame is actually generated by the model itself. This discrepancy between training and simulation can yield errors that can accumulate quickly along the generated sequence. This problem is analogous to that encountered when generating sequences with recurrent neural networks, see (Bengio et al., 2015) and references therein. Additionally, the ConvNet architecture proposed by Yang et al. is not suited to our more general use-case; in particular it cannot accurately simulate long-range phenomena, such as gravity or buoyancy. While providing encouraging results that offer a significant speedup over their PCG baseline, their work is limited to data closely matching the training conditions (as we will discuss in Section 6).

Finally we point out that our work is related to the emergent field of artificial intelligence (AI) referred as intuitive or naive physics. Despite the tremendous progress achieved in AI, agents are still far from achieving common sense reasoning, which is thought to play a crucial role in the development of general AI. Recent works have attempted to build neural models that can make predictions about stability, collisions, forces and velocities from images or videos, or interactions with an environment (e.g. (Lerer et al., 2016)). In this context, having accurate and efficient models for simulating a real world environment could be crucial for developing new learning systems (Hamrick et al., 2016; Fleuret, 2016; Byravan & Fox, 2016). We believe that our work could be used in this context in the more challenging setting of an agent interacting with fluids, see for instance (Kubricht et al., 2016).

3. Fluid Equations

When a fluid has zero viscosity and is incompressible it is called *inviscid*, and can be modeled by the Euler equations (Batchelor, 1967):

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho}\nabla p + f \quad (1)$$

$$\nabla \cdot u = 0 \quad (2)$$

Where u is the velocity (a 2D or 3D vector field), t is time, p is the pressure (a scalar field), f is the summation of external forces applied to the fluid body (buoyancy, gravity, etc) and ρ is fluid density. Equation 1 is known as the momentum equation, it arrises from applying Newton's second law to fluid motion and describes how the fluid accelerates given the forces acting on it. Equation 2 is the incompressibility condition, which enforces the volume of the fluid to remain constant throughout the simulation. For readers unfamiliar with fluid mechanics and it's associated prerequisite topics (multi-variable calculus, finite-difference methods, etc), we highly recommend (Bridson, 2008) as an introductory reference to this material.

We numerically compute all spatial partial derivatives using finite difference (FD) methods on a MAC grid (Harlow & Welch, 1965). The MAC grid representation samples velocity components on the face of voxel cells, and the scalar quantities (e.g. pressure or density) at the voxel center. This representation removes the non-trivial nullspace of central differencing on the standard uniformly sampled grid and it simplifies boundary condition handling, since solid-cell boundaries are collocated with the velocity samples.

Equations 1 and 2 can be solved via the standard operator splitting method, which involves an advection update and a "pressure projection" step. Here is an overview of the single time-step velocity update algorithm:

Algorithm 1 Euler Equation Velocity Update

```
1: Advection and Force Update to calculate $u_t^\star$:
2:    (optional) Advect scalar components through $u_{t-1}$
3:    Self-advect velocity field $u_{t-1}$
4:    Add external forces $f_{body}$
5:    Add vorticity confinement force $f_{vc}$
6:    Set normal component of solid-cell velocities.
7: Pressure Projection to calculate $u_t$:
8:    Solve Poisson eqn, $\nabla^2 p_t = \frac{1}{\Delta t}\nabla \cdot u_t^\star$, to find $p_t$
9:    Apply velocity update $u_t = u_{t-1} - \frac{1}{\rho}\nabla p_t$
```

At a high level, Algorithm 1 step 1 ignores the pressure term ($-\nabla p$ of Equation 1) to create an advected velocity field, $u_t^\star$, which includes unwanted divergence, and then step 7 solves for pressure, p, to satisfy the incompressibility constraint (Equation 2). This produces a divergence free velocity field, u_t. It can be shown that an exact solution of p in step 8, coupled with a semi-Lagrangian advection routine in steps 2 and 3, results in an unconditionally stable numerical solution. In our hybrid apporach we modify step 8 by replacing the exact projection for a *learned* one.

For advection of scalar fields and self-advection of velocity, we perform a semi-Lagrangian backward particle trace using the Maccormack method (Selle et al., 2008). When the backward trace would otherwise sample the input field inside non-fluid cells (or outside the simulation domain), we instead clamp each line trace to the edge of the fluid boundary and sample the field at its surface.

We use vorticity confinement (Steinhoff & Underhill, 1994) to counteract unwanted numerical dissipation, which attempts to reintroduce small-scale detail by detecting the location of vortices in the flow field and then introducing artificial force terms to increase rotational motion around these vortices. This firstly involves calculating the vorticity strength, $w = \nabla \times u$, using central difference and then calculating the per-voxel force, $f_{vc} = \lambda h \left(N \times w\right)$, where, $N = \nabla|w| / \|\nabla|w|\|$, λ controls the amplitude of vorticity confinement, and h is the grid size (typically $h = 1$).

Algorithm 1 step 8 is by far the most computationally demanding component. It involves solving the following Poisson equation:

$$\nabla^2 p_t = \frac{1}{\Delta t}\nabla \cdot u_t^\star \quad (3)$$

Rewriting the above equation results in a large sparse linear system $Ap_t = b$, where A is referred to in the literature as the 5 or 7 point Laplacian matrix (for 2D and 3D grids respectively). Despite A being symmetric and positive semi-definite, the linear system often has a large number of free parameters, which means that with standard iterative solvers a large number of iterations must be performed to produce an adequately small residual. Furthermore, this number of iterations is strongly data-dependent. In this paper, we use an alternative machine learning (and data-driven) approach to solving this linear system, where we train a ConvNet model to infer p_t. The details of this model will be covered in Section 4.

After solving for pressure, the divergence free velocity is calculated by subtracting the FD gradient of pressure, $u_t = u_t^\star - \frac{1}{\rho}\nabla p_t$.

To satisfy slip-condition boundaries at fluid to solid-cell interfaces, we set the velocity of MAC cells so that the component along the normal of the boundary face is equal to the normal component of the object velocity (i.e. $\hat{n} \cdot u_{\text{fluid}} = \hat{n} \cdot u_{\text{solid}}$). The MAC grid representation makes this trivial, as each solid cell boundary is at the sampling location of the velocity grid.

4. Pressure Model

In traditional formulations, incompressibility is obtained only when the pressure is a solution of the linear system of equations given in Equation 3. However, in real-time applications, PCG or Jacobi iterations are truncated before reaching convergence. Therefore the obtained velocity fields are divergent, which may lead to bad solutions (i.e. volume change of smoke or fluid and other visual artifacts) and even instability[1]. As such, truncation forgoes treating the incompressibility condition as a hard constraint even if this is not the intention, and few guarantees can be given for quality of the divergence residual at termination. This is especially true in degenerate cases when the matrix in the sparse linear system A has a large number of free-parameters (for example with highly irregular geometry boundaries).

At the heart of this problem is the ability to predict the run-time of PCG iterations as a function of the required accuracy and the specific data in which it is applied. While there is a vast amount of literature in convex optimization, how data complexity impacts convergence rate of convex solvers is still not well understood (Oymak et al., 2015). Recent works have attempted to shed some light on these questions (Oymak et al., 2015; Giryes et al., 2016). These results show that, given a fixed computational budget (allowing for only a small number of iterations) in projected gradient approaches, it is worthwhile using very inaccurate projections that may lead to a worse solution in the long-term, but are better to use with the given computational constraints. While this line of work is promising, current results only apply to random systems (such as Gaussian maps) and specific types of input data (with local low dimensional structure) in order to characterize the form of the inaccurate projections for a given problem. In the general case, given a fixed computational budget, there is currently no way of guaranteeing a pre-fixed tolerance with respect to the optimality conditions for all possible inputs.

The fundamental observation is that, while there is no closed form solution and a numerical solution might be difficult to compute, the function mapping input data to the optimum of an optimization problem is deterministic. Therefore one can attempt to approximate it using a powerful regressor such as deep neural network. Building upon this observation, another key contribution of this work is that the learning task can be phrased as a completely unsupervised learning problem, if an appropriate ConvNet architecture is used. Instead of using supervised training to directly infer and score the next frame velocity (or pressure) - where the loss would be some distance measure to ground-truth data - we measure the squared L-2 norm of the divergence of the predicted velocity and minimize it directly:

$$\begin{aligned} f_{obj} &= \sum_i w_i \left\{\nabla \cdot \hat{u}_t\right\}_i^2 \\ &= \sum_i w_i \left\{\nabla \cdot \left(u_t^\star - \frac{1}{\rho}\nabla \hat{p}_t\right)\right\}_i^2 \end{aligned} \tag{4}$$

Where $\hat{u}_t$ and $\hat{p}_t$ are the predicted divergence free velocity and pressure fields respectively and w_i is a per-vertex weighting term which emphasizes the divergence of voxels on geometry boundaries:

$$w_i = \max\left(1, k - d_i\right)$$

where d_i is a distance field with value 0 for solid cells, and for fluid cells is the minimum Euclidean distance of each fluid-cell to the nearest solid cell (i.e. a signed distance field encoding of the occupancy grid). Since the fluid-solid border represents a small fraction of the domain (due to the sparse nature of the occupancy grid), without importance weighting it contributes a small fraction of the overall loss-function, and might be ignored by our limited capacity ConvNet. However the effect of this term is not significant and has a minor contribution to the simulation quality.

The Equation 4 formulation has two significant benefits. Firstly, since obtaining ground-truth velocity from expensive iterative methods is no longer necessary, we can perform non-trivial data-augmentation to the input velocity fields. Secondly, we can incorporate loss information from a composition of multiple time-steps without the need of running exact solvers for each frame. With this, we can further improve our ConvNet prediction and long-term stability by adding an additional term to our objective function that minimizes "long-term" divergence. We do so by stepping forward the simulation state from u_0 for each training sample, to a small number of time-steps n in the future (i.e. we calculate $\hat{u}_n$).[2] Then we calculate the scalar value of Equation 4 using this future frame and we add this to the global objective (to be minimized by SGD). This process is depicted in Figure 1.

To infer $\hat{p}_t$ we use a Convolutional Network architecture (f_{conv}) parameterized by its weights and biases, c, and

[1]Advection should only be done in a divergence-free velocity field and typically there are no long-term (or multi-frame) mechanisms to ensure divergence is not accumulated.

[2]When training our model we step forward either $n = 4$ steps, with probability 0.9, or $n = 25$ with probability 0.1. Furthermore we use a random time-step - to promote time-step invariance - according to $\Delta t = 1/30 * (0.203 + |N(0,1)|)$, where $N(0,1)$ is a random sample from a Normal Gaussian distribution.

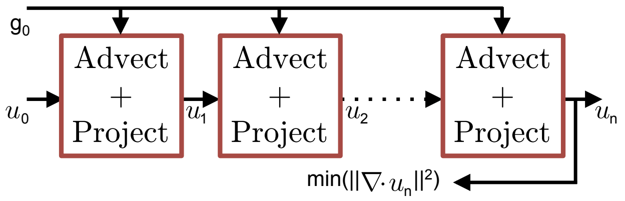

Figure 1. "Long-term" Velocity Divergence Minimization

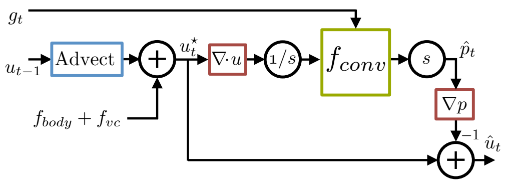

Figure 2. Velocity-Update Architecture

whose input is the divergence of the velocity field, $\nabla \cdot u_t^\star$, and a geometry field g_{t-1} (a Boolean occupancy grid that delineates the cell type for each voxel in our grid: fluid cell or solid cell):

$$\hat{p}_t = f_{conv}\left(c, \nabla \cdot u_t^\star, g_{t-1}\right) \tag{5}$$

We then optimize the parameters of this network, c, to minimize the objective f_{obj} using standard deep-learning optimization approaches; namely we use Back Propagation (BPROP) to calculate all partial derivatives and the ADAM (Kingma & Ba, 2014) optimization algorithm to minimize the loss.

A block diagram of our high-level model architecture is shown in Figure 2, and shows the computational blocks required to calculate the output $\hat{u}_t$ for a single time-step. The *advect* block is a fixed function unit that encompasses the advection step of Algorithm 1. After advection, we add the body and vorticity confinement forces. We then calculate the divergence of the velocity field $\nabla \cdot u_t^\star$ which, along with geometry, is fed through a multi-stage ConvNet to produce $\hat{p}_t$. We then calculate the pressure divergence, and subtract it from the divergent velocity to produce $\hat{u}_t$. Note that the bottle-neck architecture avoids obtaining trivial solutions: the perturbation applied by the CNN to the divergent velocity is restricted to be a conservative vector field (i.e. the gradient field with potential $\hat{p}_t$). Note that the only block with trainable parameters is the ConvNet model and that all blocks are differentiable.

Since the ConvNet (f_{conv}) solves a linear system $Ap_t = b$, we are free to scale the left and right hand sides by some constant s; we calculate the standard deviation of the input velocity field, $s = \text{STD}\left(u_t^\star\right)$, and normalize the input divergence by this scale value. We then undo the output pressure scale by multiplying by the scale reciprocal. By scale normalizing the input the learned network is made globally scale invariant (which helps generalization performance). Recall that while the network is learning a linear projection, the network itself is highly non-linear and so would otherwise be sensitive to input scale.

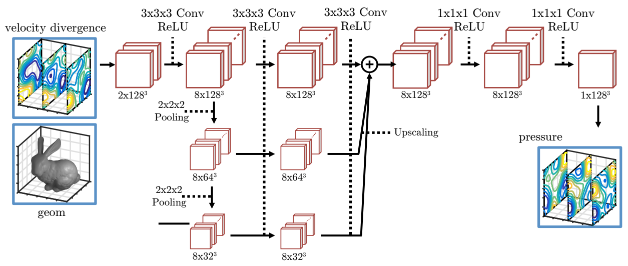

Figure 3. Convolutional Network for Pressure Solve

The internal structure of the ConvNet architecture is shown in Figure 3. It consists of 5 stages of convolution (spatial or volumetric for 2D and 3D respectively) and Rectifying Linear layers (ReLU). The convolutional operator itself mimics the local sparsity structure of our linear system; at each layer, features are generated from local interactions and these local interactions have higher-level global behavior in the deeper layers of the network. However a single resolution network would have limited context which limits the network's ability to model long-range external forces (for example the presence of gravity in a closed simulation domain results in a low frequency pressure gradient). As such, we add multi-resolution features to enable modeling long range physical phenomenon by downsampling the first hidden layer twice (average pooling), processing each resolution in parallel then upsampling (bilinear) the resultant low resolution features before accumulating them.

Note that since our network is fully-convolutional, the size of the domain can be modified at inference time; while we train at 64^3 and 128^2 resolutions for the 3D and 2D models respectively, the network can perform inference on any size domain.

Further improvements can be made to the architecture of Figure 3. We have experimented with residual connections (He et al., 2015), gated convolutions (Dauphin et al., 2016) and significantly deeper network architectures. These techniques do improve accuracy but at the cost of added run-time and latency. Alternatively, run-time can be reduced by replacing full-rank convolution kernels with learned separable convolution stages (i.e. compositions of low-rank convolutions) or by using recent model compression techniques (Lin et al., 2015; Hinton et al., 2015) for moderate increases in output divergence.

Why not to use a ConvNet to learn an end-to-end mapping that predicts the velocity field at each time-step? The chaotic change of velocity between frames is highly unstable and easily affected by external forces and other factors.

We argue that our proposed hybrid approach restricts the learning task to a stable projection step relieving the need of modeling the well understood advection and external body forces. The proposed method takes advantage of the understanding and modeling power of classic approaches, supporting enhancing tools such as vorticity confinement. Having said this, end-to-end models are conceptually simpler and, combined with adversarial training (Goodfellow et al., 2014), have shown promising results for difficult tasks such as video prediction (Mathieu et al., 2015). In that setting, fluid simulation is a very challenging test case and our proposed method represents an important baseline in terms of both accuracy and speed.

5. Dataset Creation and Model Training

Note that while we do not need label information to train our ConvNet, our network's generalization performance improves when using a dataset that approximately samples the manifold of real-world fluid states. *To this end, we propose a procedural method to generate a corpus of initial frames for use in training.*

In lieu of real-world fluid data, we use synthetic data generated using an offline 3D solver, mantaflow (Pfaff & Thuerey). We then seed this solver with initial condition states generated via a random procedure using a combination of *i.* a pseudo-random turbulent field to initialize the velocity *ii.* a random placement of geometry within this field, and *iii.* procedurally adding localized input perturbations. We will now describe this procedure in detail.

Firstly, we use the wavelet turbulent noise of (Kim et al., 2008) to initialize a pseudo-random, divergence free velocity field. At the beginning of each simulation we randomly sample a set of noise parameters (e.g. wavelet spatial scale and amplitude) and we generate a random seed, which we then use to generate the velocity field.

Next we generate an occupancy grid by selecting objects from a database of models and randomly scaling, rotating and translating these objects in the simulation domain. We use a subset of 100 objects from the NTU 3D Model Database (Pu & Ramani, 2006); 50 models are used only when generating training set initial conditions and 50 for test samples. Figure 4 shows a selection of these models. For generating 2D simulation data, we simply take a 2D slice of the 3D voxel grid. Finally, we simulate small divergent input perturbations by modeling inflow moving across the velocity field using a collection of emitter particles of random time duration, position, velocity and size.

With the above initial conditions, we use Manta to calculate $u_t^\star$ by advecting the velocity field and adding forces. We also step the simulator forward 256 frames (using Manta's PCG-based solver), recording the velocity every 8 steps.

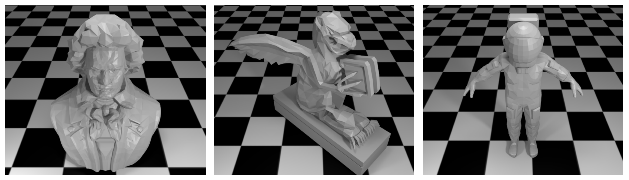
Figure 4. A selection of 3D Models used in our dataset

We generate a training set of 320 "scenes" (each with a random initial condition) and a test set of an additional 320 scenes. Each "scene" contains 32 frames 0.8 seconds apart. We use a disjoint set of geometry for the test and training sets to test generalization performance. The dataset is public, as well as the code for generating it.

During training, we further increase dataset coverage by performing data-augmentation. When stepping forward the simulator to calculate long-term divergence, we randomly add gravity, density and vorticity confinement of varying strengths and with a random gravity vector.

6. Results and Analysis

The model of Section 4 was implemented in Torch7 (Collobert et al., 2011), with two CUDA baseline methods for comparison; a Jacobi-based iterative solver and a PCG-based solver (with incomplete Cholesky L0 preconditioner). For sparse linear system operations, we used primitives from NVIDIA's cuSPARSE and cuBLAS libraries.

To implement the model of (Yang et al., 2016) for comparison, we rephrase their patch-based architecture as an equivalent sliding window model comprised of a 3x3x3 conv and sigmoid stage followed by 3 stages of 1x1x1 convolutions and sigmoid with appropriate feature sizing. Note that this equivalent reimplementation as a ConvNet is significantly faster, as we can make use of the highly optimized convolution implementations from NVIDIA's cudnn library. Yang et al. report 515ms per frame on a 96 x 128 x 96 grid, while our implementation of their model takes only 9.4ms per frame at this resolution.

The supervised loss specified by Yang et al. measures the distance to a ground-truth output pressure (i.e. they train a network in isolation to perform the pressure projection only). For our dataset, this loss does not result in accurate results. When fluid cells are surrounded by solid cells, each connected component of fluid represents and independent linear system, each with an arbitrary pressure offset. As such, we modified the learning procedure of Yang et al. to include a "pressure-normalization" routine which subtracts the mean pressure in each connected-component of fluid cells in the ground-truth pressure frames. This modification enabled SGD to converge when training their model on our dataset. However despite this correction, the model of Yang et al. does not learn an accurate linear projection on our data; our initial condition divergent velocity frames include large gradient and buoyancy terms, which results in a high amplitude, low frequency gradient in the ground-truth pressure. The small 3x3x3 input context of their model is not able to infer such low frequency outputs. Since these loss terms dominate the Yang et al. objective, the network over-trains to minimize it. Likely, this phenomena wasn't evident in Yang et al.'s original results due to the low diversity of their training set, and the high correlation between their evaluation and training conditions (perhaps their intended use-case). The results of their model trained on their objective function can be seen in Figure 6.

By contrast, our unsupervised objective minimizes divergence after the pressure gradient operator, whose FD calculation acts as a high-pass filter. This is a significant advantage; our objective function is "softer" on the divergence contribution for phenomena that the network cannot easily infer. For the remaining experimental results, we will evaluate an improved version of the Yang et al. model, which we call our *"small-model"*[3].

Figure 5 shows the computation time of the Jacobi method, the small-model (sizing of Yang et al.) and this work[4]. Note that for fair quantitative comparison of output residual, we choose the number of Jacobi iterations (34) to approximately match the FPROP time of our network (i.e. to compare divergence with fixed compute). Since the asymptotic complexity as a function of resolution is the same for Jacobi and our ConvNet, the FPROP times are equivalent. PCG is orders of magnitude slower at all resolutions and has been omitted for clarity. The small-model provides a significant speedup over other methods. The runtime for the PCG, Jacobi, this work, and the small-model at 128^3 grid resolution are 2521ms, 47.6ms, 39.9ms and 16.9ms respectively.

Note that with custom hardware (Movidius; Google Inc.), separable convolutions and other architectural enhancements, we believe the runtime of our ConvNet could be reduced significantly. However, we leave this to future work.

We simulated a 3D smoke plume using both our system and baseline methods. The simulation data was created using our real-time system (which supports basic real-time visualization), and the accompanying figures and supplemental videos were rendered offline using Blender (Blender Foundation). Video examples of these experiments can be found in the supplemental materials. Since vorticity confinement tends to obfuscate simulation errors (by adding high frequency detail), Figures 6 and 7 were simulated without it.

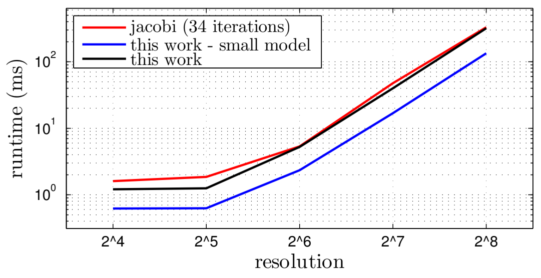

Figure 5. Pressure projection time (ms) versus resolution (PCG not shown for clarity).

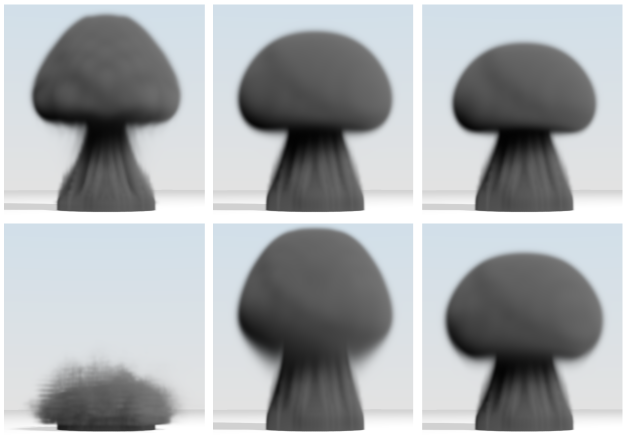

Figure 6. Plume simulation (without vorticity confinement). *Top left*: Jacobi (34 iterations). *Top Middle* Jacobi (100 iterations). *Top Right*: PCG. *Bottom left*: Yang et al. *Bottom middle*: small-model. *Bottom Right*: this work.

Figure 6 shows a rendered frame of our plume simulation (without geometry) for all methods at the same simulation time-step. Note that this boundary condition is not present in the training set and represents an input divergent flow 5 times wider than the largest impulse present during training. It is a difficult test of generalization performance. Qualitatively, the PCG solver, 100 iteration Jacobi solver and our network produce visually similar results. The small-model's narrow receptive field cannot accurately simulate the large vortex under the plume, and as a result the plume rises too quickly (i.e. with too much upward velocity) and exhibits density blurring under the plume itself. The Jacobi method, when truncated early at 34 iterations, introduces implausible high frequency noise and has an elongated shape due to inaccurate modeling

[3]The "small-model" is a single resolution pressure network with only 3x3x3 context and trained using the loss function, top-level architectural improvements and data-augmentation strategies from this work. We include these experiments as a proxy comparison for the original model of Yang et al.

[4]This runtime includes the pressure projection steps only: the velocity divergence calculation, the linear system solve and the velocity update. We use an NVIDIA Titan X GPU with 12GB of ram and an Intel Xeon E5-2690 CPU.

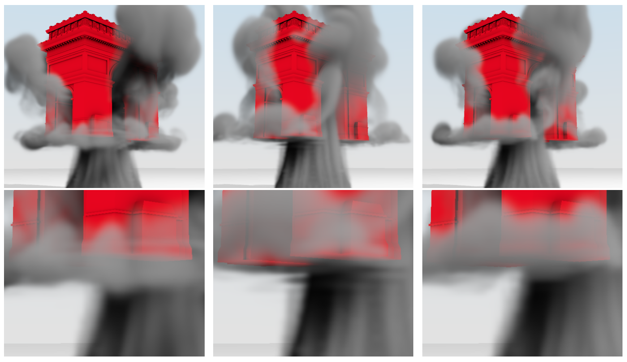

Figure 7. Plume simulation with "Arch" geometry. *Left*: PCG. *Middle* small-model *Right*: this work.

of buoyancy forces. We also repeat the above simulation with solid cells from two models held out of our training set: the "arc de triomphe" model (Pu & Ramani, 2006) and the Stanford bunny (Turk & Levoy, 1994); the single frame results for the arch simulation are shown in Figure 7. Since this scene exhibits lots of turbulent flow, qualitative comparison is less useful. The small-model has difficulty minimizing divergence around large flat boundaries and results in high-frequency density artifacts as shown. Both ConvNet based methods lose some smoke density inside the arch due to negative divergence at the fluid-geometry boundary (specifically at the large flat ceiling).

In addition to comparing divergence using fixed-compute, we also performed the above experiment by fixing divergence to $\max_t\left(||\nabla \cdot u_t||\right) = 0.872$, and measuring compute. For Jacobi to match the divergence performance of our network, it requires 116 iterations and so is $4.1\times$ slower than our network. Since calculating divergence at inference time is fast and efficient, PCG can be used as a fallback method if our system fails with minimal runtime penalty or if the application at hand requires an exact solution.

Table 1 shows the maximum norm L2 linear system residual for each frame over the entire simulation. The maximum residual for the Yang et al. model was greater than 1e5 for both simulations (i.e. it fails on this test case). Interestingly, early termination of the Jacobi method at 34 iterations results in reduced long-term accuracy and visual quality (Figure 6), however the maximum per-frame residual is still relatively low. *This suggests that single frame divergence alone is not a sufficient condition to maintain long-term accuracy, and that the multi-frame error propagation mechanisms are an extremely important (but harder to quantify) factor.*

As a test of long-term stability, we record the mean L2 norm of velocity divergence ($\mathbf{E}\left(||\nabla \cdot \hat{u}_i||\right)$) across all samples in our test-set. The result of this experiment is shown in Figure 8. On our test-set frames, our method outperforms the small-model by a significant margin and is competitive with Jacobi truncated to 34 iterations. Figure 8 also shows the results of our model when a single time-step loss is used. Adding multi-frame components not only improves the divergence residual over multiple time steps as expected (since this is what we are directly minimizing), but additionally the single frame divergence performance is also improved. We attribute this to the fact that these future frames effectively increase dataset diversity, and can be seen as a form of dataset augmentation to improve generalization performance.

	PCG	Jacobi	small-model	this work
No geom	<1e-3	2.44	3.436	2.482
With geom	<1e-3	1.235	1.966	0.872

Table 1. Maximum residual norm throughout our Plume simulation with and without geometry ($\max_t\left(||\nabla \cdot u_t||\right)$)

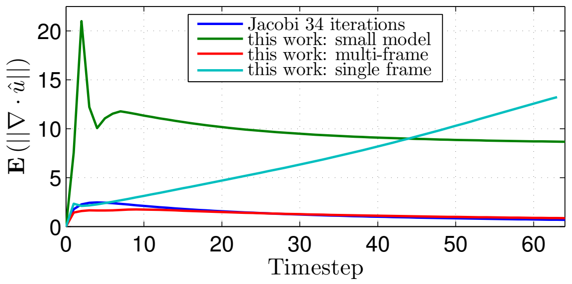

Figure 8. $\mathbf{E}\left(||\nabla \cdot \hat{u}_i||\right)$ versus time-step for each frame sample in our dataset (PCG not shown for clarity).

7. Conclusion

This work proposes a novel, fast and efficient method for calculating numerical solutions to the inviscid Euler Equations for fluid flow. We present a data-driven approach for approximate inference of the sparse linear system used to enforce the Navier-Stokes incompressibility condition - the "pressure projection" step. We propose an unsupervised training-loss, which incorporates multi-frame information to improve long-term stability. We also present a novel and tailored ConvNet architecture, which facilitates drop-in replacement with existing Eulerian-based solvers. While the proposed approach cannot guarantee finding an exact solution to the pressure projection step, it can empirically produce very stable divergence free velocity fields whose runtime and accuracy is better than the Jacobi method (a common technique used for real-time simulation) and whose visual results are comparable to PCG, while being orders of magnitude faster. Code, data and videos are made available at http://cims.nyu.edu/~schlacht/CNNFluids.htm.

References

Batchelor, G.K. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 1967.

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

Blender Foundation. Bender open-source renderer v2.77a. http://www.blender.org.

Bridson, R. *Fluid Simulation for Computer Graphics*. Ak Peters Series. Taylor & Francis, 2008. ISBN 9781568813264. URL https://books.google.com/books?id=gFI8y87VCZ8C.

Byravan, Arunkumar and Fox, Dieter. Se3-nets: Learning rigid body motion using deep neural networks. *arXiv preprint arXiv:1606.02378*, 2016.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

Dauphin, Yann N, Fan, Angela, Auli, Michael, and Grangier, David. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

De Witt, Tyler, Lessig, Christian, and Fiume, Eugene. Fluid simulation using laplacian eigenfunctions. *ACM Trans. Graph.*, 31(1):10:1–10:11, February 2012. ISSN 0730-0301. doi: 10.1145/2077341.2077351. URL http://doi.acm.org/10.1145/2077341.2077351.

Fleuret, François. Predicting the dynamics of 2d objects with a deep residual network. 2016.

Foster, Nick and Metaxas, Dimitri. Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483, September 1996. ISSN 1077-3169. doi: 10.1006/gmip.1996.0039. URL http://dx.doi.org/10.1006/gmip.1996.0039.

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

Giryes, Raja, Eldar, Yonina C, Bronstein, Alex M, and Sapiro, Guillermo. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *arXiv preprint arXiv:1605.09232*, 2016.

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

Google Inc. Tensor processing unit. http://cloudplatform.googleblog.com/2016/05/.

Hamrick, Jessica B, Pascanu, Razvan, Vinyals, Oriol, Ballard, Andy, Heess, Nicolas, and Battaglia, Peter. Imagination-based decision making with physical models in deep neural networks. 2016.

Harlow, Francis H. and Welch, J. Eddie. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Kim, Theodore, Thürey, Nils, James, Doug, and Gross, Markus. Wavelet turbulence for fluid simulation. *ACM Trans. Graph.*, 27(3):50:1–50:6, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360649. URL http://doi.acm.org/10.1145/1360612.1360649.

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

Kubricht, James, Jiang, Chenfanfu, Zhu, Yixin, Zhu, Song-Chun, Terzopoulos, Demetri, and Lu, Hongjing. Probabilistic simulation predicts human performance on viscous fluid-pouring problem. *Retrieved March*, 16:2016, 2016.

Ladický, L'ubor, Jeong, SoHyeon, Solenthaler, Barbara, Pollefeys, Marc, and Gross, Markus. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6):199:1–199:9, October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818129. URL http://doi.acm.org/10.1145/2816795.2818129.

Lentine, Michael, Zheng, Wen, and Fedkiw, Ronald. A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM Transactions on Graphics (TOG)*, volume 29, pp. 114. ACM, 2010.

Lerer, Adam, Gross, Sam, and Fergus, Rob. Learning physical intuition of block towers by example. *arXiv preprint arXiv:1603.01312*, 2016.

Lin, Zhouhan, Courbariaux, Matthieu, Memisevic, Roland, and Bengio, Yoshua. Neural networks with few multiplications. *CoRR*, abs/1510.03009, 2015. URL http://arxiv.org/abs/1510.03009.

Mathieu, Michael, Couprie, Camille, and LeCun, Yann. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

McAdams, Aleka, Sifakis, Eftychios, and Teran, Joseph. A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 65–74. Eurographics Association, 2010.

Molemaker, Jeroen, Cohen, Jonathan M, Patel, Sanjit, and Noh, Jonyong. Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 9–18. Eurographics Association, 2008.

Movidius. Myriad 2 visual processing unit. http://www.movidius.com/.

Oymak, Samet, Recht, Benjamin, and Soltanolkotabi, Mahdi. Sharp time–data tradeoffs for linear inverse problems. *arXiv preprint arXiv:1507.04793*, 2015.

Pfaff, Tobias and Thuerey, Nils. Mantaflow fluid simulator. http://mantaflow.com/.

Pu, Jiantao and Ramani, Karthik. On visual similarity based 2d drawing retrieval. *Comput. Aided Des.*, 38(3):249–259, March 2006. ISSN 0010-4485. doi: 10.1016/j.cad.2005.10.009. URL http://dx.doi.org/10.1016/j.cad.2005.10.009.

Raveendran, Karthik, Wojtan, Chris, Thuerey, Nils, and Turk, Greg. Blending liquids. *ACM Trans. Graph.*, 33 (4):137:1–137:10, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601126. URL http://doi.acm.org/10.1145/2601097.2601126.

Selle, Andrew, Fedkiw, Ronald, Kim, Byungmoon, Liu, Yingjie, and Rossignac, Jarek. An unconditionally stable maccormack method. *J. Sci. Comput.*, 35(2-3), June 2008.

Stanton, Matt, Humberston, Ben, Kase, Brandon, O'Brien, James F., Fatahalian, Kayvon, and Treuille, Adrien. Self-refining games using player analytics. *ACM Trans. Graph.*, 33(4):73:1–73:9, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601196. URL http://doi.acm.org/10.1145/2601097.2601196.

Steinhoff, John and Underhill, David. Modification of the euler equations for vorticity confinement: application to the computation of interacting vortex rings. *Physics of Fluids (1994-present)*, 6(8):2738–2744, 1994.

Treuille, Adrien, Lewis, Andrew, and Popović, Zoran. Model reduction for real-time fluids. *ACM Trans. Graph.*, 25(3):826–834, July 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141962. URL http://doi.acm.org/10.1145/1141911.1141962.

Turk, Greg and Levoy, Marc. Zippered polygon meshes from range images. pp. 311–318, 1994.

Yang, Cheng, Yang, Xubo, and Xiao, Xiangyun. Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds*, 27(3-4):415–424, 2016.