Accelerating Eulerian Fluid Simulation With Convolutional Networks

Jonathan Tompson
Google Inc.
and
Kristofer Schlachter, Pablo Sprechmann, Ken Perlin
New York University

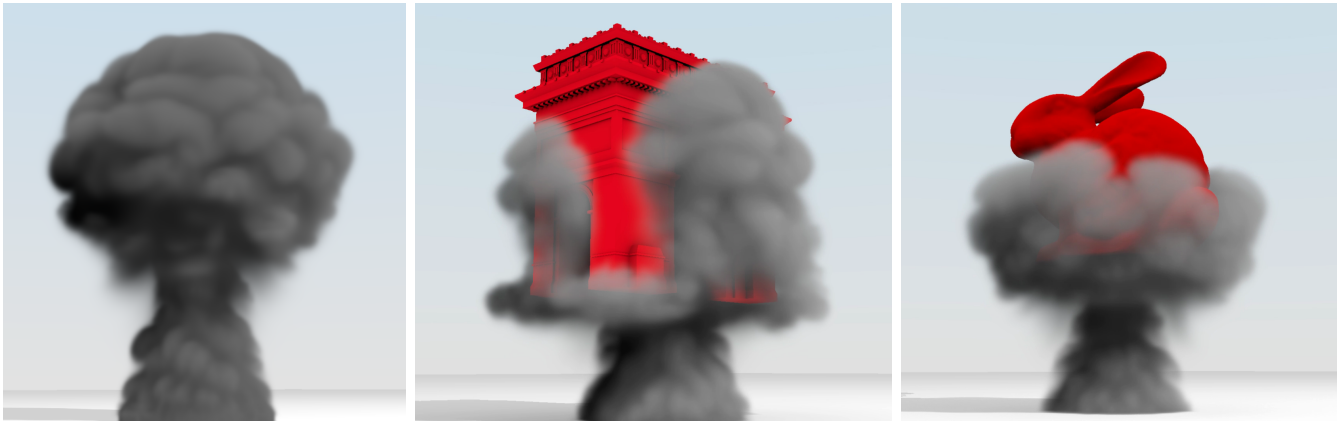


Fig. 1: Smoke simulation using our system - our method is capable of fast and accurate simulation of the Euler Equations for incompressible fluid flow at interactive frame-rates. Videos can be found in the supplemental materials, or at:
`http://cims.nyu.edu/~schlacht/CNNFluids.htm`

Real-time simulation of fluid and smoke is a long standing problem in computer graphics, where state-of-the-art approaches require large compute resources, making real-time applications often impractical. In this work, we propose a data-driven approach that leverages the approximation power of deep-learning methods with the precision of standard fluid solvers to obtain both fast and highly realistic simulations. The proposed method solves the incompressible Euler equations following the standard operator splitting method in which a large, often ill-condition linear system must be solved. We propose replacing this system by learning a Convolutional Network (ConvNet) from a training set of simulations using a semi-supervised learning method to minimize long-term velocity divergence.

ConvNets are amenable to efficient GPU implementations and, unlike exact iterative solvers, have fixed computational complexity and latency. The proposed hybrid approach restricts the learning task to a linear projection without modeling the well understood advection and body forces. We present real-time 2D and 3D simulations of fluids and smoke; the obtained results are realistic and show good generalization properties to unseen geometry.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms

Additional Key Words and Phrases: Navier-Stokes, Euler equations, Computational Fluid Dynamics, Smoke, Machine Learning, Convolutional Networks

1. INTRODUCTION AND RELATED WORK

Real-time simulation of fluid and smoke is a long standing problem in computer graphics. It is increasingly important in visual effects and is starting to make an impact in interactive computer games.

Physics-based simulation has achieved remarkable results animating fluids like water or smoke. However, there is a big gap between pre-computing a simulation for several hours on high-end computers and simulating it in real-time at several frames per second on a single console. Real-time fluid simulations have been possible only under very restricted conditions (with large GPU compute resources for example) or low resolution (and less realistic) settings. Several works have studied different alternatives for mitigating this limitation, as discussed next. However, computational complexity remains one of the most important constraints.

The dynamics of most relevant fluid flows in computer graphics are governed by the incompressible Navier-Stokes equations, which are a set of partial differential equations that must hold throughout a fluid velocity field at any given time. There are two main computational methods for simulating fluid flows from these equations: the Lagrangian viewpoint and the Eulerian viewpoint. The most common Lagrangian approach is the Smoothed Particle Hydrodynamics [Gingold and Monaghan 1977] (SPH) method that approximates continuous quantities using discrete particles. It is a grid-free method in which the coordinates move with the fluid (particles). The second alternative, which is the one we adopt in this work, is Euler's formulation where continuous quantities are approximated on a regular grid [Fedkiw et al. 2001].

Many recent works have addressed the problem of computational performance in fluid simulations. In the context of Lagrangian methods, a position based fluid (PBF) approach has been proposed in [Macklin and Müller 2013]. In this method, all particles are first advected and then projected to satisfy the incompressibility condition. Although this requires an iterative procedure, PBF allows the use of larger time-steps than previous alternatives. To further speed-up the computation, a multi-scale approach was introduced in [Horvath and Solenthaler 2013]. In [Chentanez and Müller 2010] the authors propose a grid-based solution that can produce efficient simulations when restricted to simpler topology.

An alternative approach is to look at data-driven simulations. The main idea of this family of methods is to reduce computation by operating on a representation of the simulation space of significantly lower dimensionality. These representations are obtained by exploiting the structure (or redundancy) in the data. There is a natural compromise between computational cost and the richness of the representation. 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. Finally, the approach most related to the present work is the one introduced in [Ladický et al. 2015]. The authors proposed an adaptation of regression forests for smoothed particle hydrodynamics. Given a set of hand-crafted features, corresponding to individual forces and the incompressibility constraint of the Navier-Stokes equations, the authors trained a regressor for predicting the state of each particle in the following time-step. Experimental results show significant speed up with respect to standard solvers and highly realistic simulations. To the best of our knowledge, the work in [Ladický et al. 2015] is the only previous approach that uses machine learning to learn the fluid dynamics from a set of example simulations.

In this paper, we propose a machine learning based approach for speeding up the most computationally demanding step of a traditional Eulerian solver. We adopt Euler's formulation with numerical simulation following the standard splitting on the incompressible fluid equations. For each time-step, the method sequentially solves the three different component parts of the equation, corresponding to advection, body forces (gravity, buoyancy, etc), and incompressibility. Of these three steps, the most computationally demanding is imposing the incompressibility condition. It is enforced on the grid by solving the discrete Poisson equation and renders the velocity field divergence-free.

The Poisson equation leads to a well-known linear system for which exact solutions can be found through convex optimization techniques. These methods suffer from one main drawback: the iterative nature of the algorithm makes the computational complexity data-dependent.

The main contribution of this work is to state the problem as a regression task. We leverage the power of deep-learning techniques as universal approximating functions to derive an approximate inference mechanism that takes advantage of the statistics of fluid data. Convolutional Networks (ConvNets) are powerful machines particularly well suited for modeling the local correlations of image data. ConvNets are used in most modern computer vision systems, achieving state-of-the-art results in many fundamental problems [Szegedy et al. 2016].

However good the ConvNet approximation might be, it will always result in non-zero divergence residual that can accumulate over time, potentially learning (in the long run) unstable behavior. Our key observation is that, for obtaining realistic simulations, it is more important to produce a low divergence flow (conforming to the boundary conditions) than recovering the exact pressure or velocity. In other words, what we require is visual plausibility rather than obtaining simulations indistinguishable from the real world. We therefore propose a tailored loss function that takes this into consideration. It can be applied in a semi-supervised manner, empirically eliminating long term instability problems. The proposed simulator is stable and can be run with very large time-steps. We present real-time simulations of fluids and smoke in 2D and 3D settings with and without the presence of solid obstacles. The obtained results are realistic and show good generalization properties to unseen data and lead to significant improvements in computational time.

A valid question to ask would be why not to use a ConvNet to learn an end-to-end mapping that predicts the velocity field at each time-step, that is, completely eliminating the physics-based simulation. We argue that the proposed hybrid approach restricts the learning task to a simple projection relieving the need of modeling the well understood advection and external body forces. Furthermore, the proposed method takes advantage of the understanding and modeling power of classic approaches, allowing the use of enhancing tools such as vorticity confinement. On the other hand, while conceptually simpler, learning an end-to-end model would require a larger model (to account for added complexity) and longer training times.

The paper is organized as follows. In Section 2 we briefly introduce the physics-based fluid simulation technique used in this paper. In Section 3 we present the proposed model with the different training regimes and we provide implementation details in Section 4. Experimental results are described in Section 5 and conclusion are drawn in Section 6.

2. FLUID EQUATIONS

Throughout this paper we will drop the viscosity term from the incompressible Navier-Stokes equations that govern fluid motion. When a fluid has zero viscosity and is incompressible it is called *inviscid* and can be modeled by the Euler equations [Batchelor 1967]:

$$\nabla \cdot u = 0 \tag{1}$$

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho}\nabla p + f \tag{2}$$

Where u is the velocity (a 2D or 3D vector field), t is time, p is the pressure (a scalar field), f is external force applied to the fluid body (another vector field) and ρ is fluid density. This formulation has been used extensively in computer graphics applications as it can model a large variety of smoke and fluid effects. Equation 1 is the incompressibility condition and Equation 2 is known as the momentum equation.

We calculate all partial derivatives of Equation 1 and 2 using finite difference (FD) methods on a regularly sampled 2D or 3D grid. In particular, we use central difference approximations for velocity and pressure gradients for internal fluid cells and single sided difference for pressure gradients for cells on the fluid border and adjacent to internal static geometry (or non-fluid cells).

As is standard in the graphics literature [Stam 1999; Fedkiw et al. 2001; Foster and Metaxas 1996; Foster and Metaxas 1997], we numerically solve Equations 1 and 2 by splitting the PDE into an ad-

vection update and "pressure projection" step. Here is an overview of the single time-step, velocity update algorithm:

Algorithm 1 Euler Equation Velocity Update

1: Choose a time-step Δt
2: Advection and Force Update to calculate $u_t^\star$:
3: Advect scalar components through u_{t-1}
4: Self-advect velocity field u_{t-1}
5: Add external forces f_{body}
6: Add vorticity confinement force f_{vc}
7: Pressure Projection to calculate u_t:
8: Solve Poisson eqn to find p_t that satisfies $\nabla \cdot u_t = 0$
9: Apply velocity update $u_t = \Delta u_t + u_{t-1}$
10: Modify velocity to satisfy boundary conditions.

At a high level, Algorithm 1 step 2 ignores the pressure term ($-\nabla p$ of Equation 2) to create an advected velocity field, $u_t^\star$, which includes unwanted divergence, and step 7 solves for pressure, p, to satisfy the divergence constraint of Equation 1 producing a divergence free velocity field, u_t.

It can be shown that an exact solution (within floating point tolerance) of p in step 8, coupled with a semi-Lagrangian advection routine for step 2, results in an unconditionally stable numerical solution [Bridson 2008]. Therefore, Δt in step 1 is somewhat arbitrary, and often heuristics are used to minimize visual artifacts.

For advection of scalar fields (smoke density, temperature, etc) and self-advection of velocity (steps 3 and 4 in Algorithm 1) we use a semi-Lagrangian, RK2 backward particle trace [Stam 1999]. When the backward trace would otherwise sample the input field inside non-fluid cells (or outside the simulation domain), we instead clamp the trace to the edge of the fluid boundary and sample the field at its surface.

Notably absent from the Euler Momentum Equation 2 is the Navier-Stokes term for viscosity ($\nu \nabla \cdot \nabla u$ - where ν is the kinematic viscosity). We choose to ignore this term since our Semi-Lagrangian advection scheme, as well as our ConvNet formulation for solving pressure, introduces unwanted numerical dissipation which acts as viscosity.[1] To counteract this dissipation we use *Vorticity Confinement*, first developed by [Steinhoff and Underhill 1994] and later introduced to the graphics community by [Fedkiw et al. 2001]. Vorticity confinement attempts to reintroduce small-scale detail by detecting the location of vortices in the flow field and then introducing artificial force terms which act to increase rotational motion around these vortices. For our incompressible flow this firstly involves calculating the vorticity, w:

$$w = \nabla \times u$$

Where w is calculated using central difference. We then calculate the per-vertex force, f_{vc}, update of [Fedkiw et al. 2001]:

$$f_{vc} = \lambda h \left(N \times w \right)$$
$$\text{where } N = \frac{\nabla |w|}{\| \nabla |w| \| + \epsilon}$$

The ϵ term is used to guard against numerical precision issues and erroneous division by zero. The scalar constant λ is used to control the amplitude of vorticity confinement (we use $\lambda = 0.05$) and h is the grid size (typically $h = 1$).

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

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

Rewriting the above Poisson 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). Typically the Preconditioned Conjugate Gradient (PCG) method is used to solve the above linear system.

Despite A being symmetric and positive semi-definite, the linear system is often ill-conditioned - even with the use of standard preconditioning techniques - which means that in standard solutions to this system, a large number of PCG iterations must be performed to produce an adequately small residual, and 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 3.

After solving for pressure, the divergence free velocity is calculated by subtracting the FD gradient of pressure, $u_t = u_t^\star - \Delta t \nabla p$, with some modification in the presence of boundary conditions. Firstly, a single sided FD approximation is used for velocity and pressure cells that border on a non-fluid cell. Secondly, when a fluid cell is bordered on a dimension by two non-fluid (solid geometry) neighbor cells, the velocity update for that component is zero.

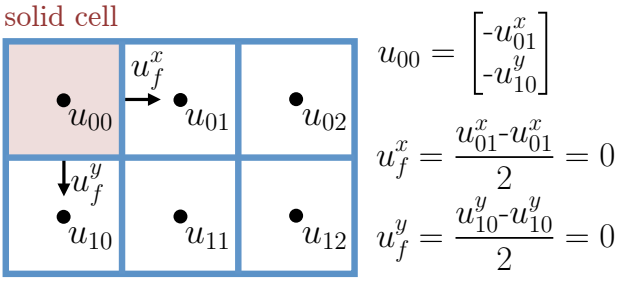


Fig. 2: Setting non-fluid velocity to satisfy boundary constraints

Finally, to satisfy Neumann boundary conditions for internal solid geometry we set the velocity of solid (non-fluid) cells so that the component along the normal of the boundary face is zero (i.e. $\hat{n} \cdot u = 0$). This prevents fluid from entering the solid boundary during advection, which would be a clear violation of the boundary condition. Since we model internal geometry as an occupancy grid (aligned to our 2D or 3D u and p grids), this means that we can simply set the velocity component to the negative of the centered velocity in the adjacent fluid cell. Then, during bilinear or trilinear interpolation of advection step 2, the interpolated velocity at the boundary will be zero. A 2D example of this process is shown in Figure 2.

3. PRESSURE MODEL

The main contribution of this paper is the re-formulation of Algorithm 1 step 8 (pressure projection step) as a data-driven regression problem in which the regression function is given by a tailored

[1] Note that our method does not prohibit the use of viscosity, rather that we find it unnecessary for our application. One can trivially extend Algorithm 1 to include a viscosity component after the velocity advection step 4.

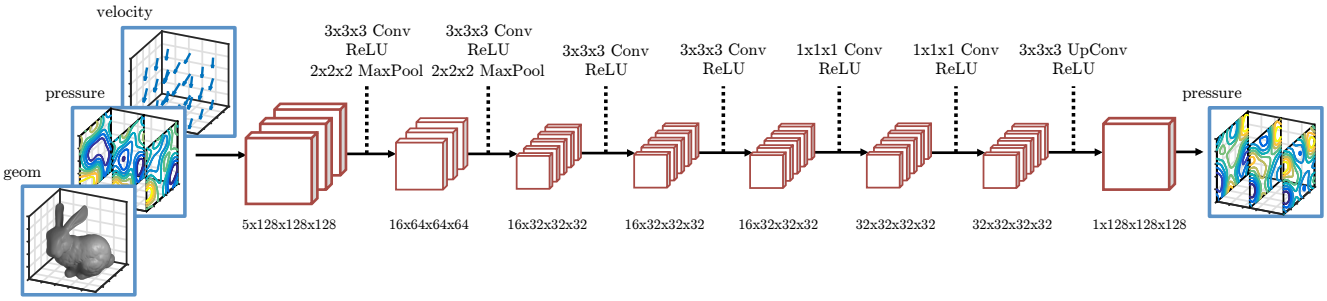


Fig. 3: Convolutional Network Architecture for Pressure Solve

(and highly non-linear) ConvNet. In this section, we describe the proposed architecture and training procedure.

Exactly satisfying the incompressibility condition of the Euler Equations, $\nabla \cdot u = 0$, using a small and fixed amount of compute per frame has been a long standing goal in not only the computer graphics community, but in the wider applied mathematics community as well. In the current formulation, incompressibility is obtained only when the pressure is a solution of the linear system of equations given in Equation 3. In real-time scenarios, however, PCG iterations might have to be truncated before reaching convergence to achieve the required frame rate. The algorithm is not designed to be run in this way and therefore the obtained velocity fields are divergent, which may lead to bad solutions and instability as advection should only be done in a divergence-free velocity field. We argue that truncating the PCG iterations before reaching convergence forgoes treating the incompressibility condition as a hard constraint even if this is not the intention. This is especially true in degenerate cases where $\max(u)$ is large or the matrix in the sparse linear system A is ill-conditioned, which results in the need for a large number of PCG iterations to achieve convergence.

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.

Alternatively, recent approaches have proposed a data-driven approach in which approximate inference mechanisms are learned from the data itself. In the context of sparse inference, recent approaches have studied the used of deep-learning methods to develop real-time approximate inference mechanisms that profit from the specifics of the data on which they are applied [Gregor and LeCun 2010; Sprechmann et al. 2015]. 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. The obtained results show a very significant improvement when compared to truncating the iterations of the exact solvers.

For our problem, we propose a learned approximate inference mechanism to find fast and efficient solutions to the linear system $Ap_t = b$. However, it is difficult to define what an "optimal approximation" to this linear system looks like. The linear system of equations is obtained by combining two conditions that need to be satisfied by the new pressure: p_t has to be such that the new velocity field u_t is divergence free inside the fluid (enforcing the desired boundary conditions) and equal to the input (divergent) velocity minus the corresponding pressure gradient, $u_t = u_{t-1} - \Delta t\, p_t$.

We need to define the conditions for optimality or what compromises we can make when diverging from the true solution. Should we measure only the final velocity divergence (residual) as the goodness measure for our approximate solution? Should we accept solutions that are low divergence but where p_t is far from a true solution (and therefore result in a low-divergence velocity field u_t over a small number of time-steps but is physically inaccurate)? Note that if we only penalize the divergence of the velocity field without imposing any relation to the ground-truth p_t or u_t, the ConvNet can simply learn to shrink the magnitude of the velocity field without imposing the desired structure. For computer graphics applications, likely a good approximation is one which is visually plausible but satisfies the incompressibility constraint as much as possible (and therefore satisfying conservation of mass which, if violated, can lead to visual artifacts over time).

We define a complex, multi-modal objective function and formulate the inference solution as a machine learning task with model parameters collectively represented by c, which we will then optimize to minimize the objective using standard deep-learning techniques. This learning formulation enables engineering a loss function which encodes the terms we care about and is - at the same time - a general enough framework that users can re-define and tailor the loss to their specific application. The loss we propose is the following:

$$f_{obj} = \lambda_p \|p_t - \hat{p}_t\|^2 + \lambda_u \|u_t - \hat{u}_t\|^2 + \lambda_{div} \|\nabla \cdot \hat{u}_t\|^2 \quad (4)$$

Where p_t and u_t are the "ground-truth" pressure and velocity fields respectively, which we obtain using an offline traditional fluid solver (we will cover this in detail in Section 4). $\hat{p}_t$ is the approximate pressure solution to the linear system and $\hat{u}_t$ the approxi-

mate velocity field after the pressure gradient update (using $\hat{p}_t$). The three terms λ_p, λ_u and λ_{div} allow us to control the relative importance of the three loss terms.

To infer $\hat{p}_t$ we use a Convolutional Network architecture parameterized by its weights and biases, c, and whose input is the divergent velocity field $u_t^\star$, a geometry field g_{t-1} (which is an occupancy grid that delineates the cell type for each voxel in our grid: fluid cell or solid cell) and the previous time-step pressure solution p_{t-1}:

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

We then optimize the parameters of this network, c, to minimize the objective f_{obj} using standard deep-learning optimization approaches; namely we use the Back Propagation (BPROP) algorithm [LeCun et al. 1998] to calculate partial derivatives of the loss with respect to the parameters c and use stochastic gradient descent (SGD) to minimize the loss.

A block diagram of our high-level model architecture is depicted in Figure 4, and shows the computational blocks from Algorithm 1 required to calculate $\hat{p}_t$ and $\hat{u}_t$ for a single time-step. The *advect* block is a fixed function unit that encompasses the advection described in Section 2 (Algorithm 1 step 4). After advection, we add the body and vorticity confinement forces to calculate $u_t^\star$ which, along with geometry and p_{t-1}, is fed through a multi-stage ConvNet to produce $\hat{p}_t$. We then calculate the pressure divergence using a fixed-function FD stage, and subtract it from the divergent velocity to produce $\hat{u}_t$. Note that the only block with trainable parameters is the ConvNet model. Furthermore, all blocks in the model are differentiable. This means that we can use BPROP to efficiently calculate loss partial derivatives for optimization.

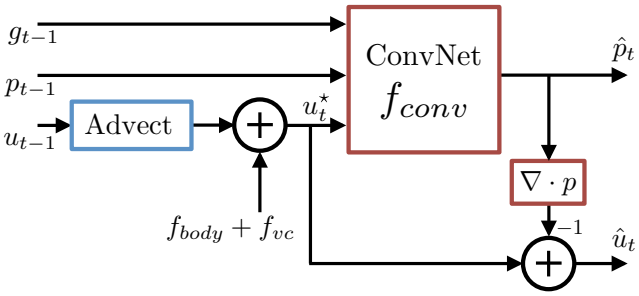


Fig. 4: Pressure Model Architecture

ConvNets (or deep-networks) are extremely well suited to the task of solving the sparse linear system. The convolutional operator itself perfectly encodes 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, exactly mimicking the local inference behavior of the linear system. In addition, modern GPU architectures are extremely efficient at calculating the "embarrassingly parallel" convolutional features of deep-networks. Finally, since many real-time fluid solvers are both Eulerian and GPU based, we believe that our network can act as a drop-in replacement for the PCG solver used in existing systems.

The internal structure of the ConvNet architecture is shown in Figure 3. It consists of 7 stages of convolution, Rectifying Linear layers (ReLU) and Max Pooling. To recover spatial precision lost due to pooling, we use a learned up-sampling layer that incorporates learned linear interpolation into the final convolution layer. We implement this layer by increasing the number of output features of a standard convolution by the interpolation ratio and permuting the output tensor to add the extra elements to the spatial domain (interleaving the feature map pixels).

Note that since we are inferring the solution of a linear system, we can "globally normalize" the input and output pressure and velocity tensors by a constant k (since we are free to apply any arbitrary scale to the linear system). In practice, k is the standard deviation of the input pressure ($k = \text{std}\left(p_{t-1}\right) + \epsilon$), which helps improve SGD convergence. For 2D models we use a spatial convolution layer (a 2D convolution on a 3D input tensor) and for 3D models we use a volumetric convolution layer (a 3D convolution on a 4D input tensor). Since our network consists of only operators applied convolutionally and point-wise non-linearities, and therefore our functional mapping is translationally invariant (with some edge effects), our trained network can scale to any arbitrary domain size (i.e. while we train on inputs of size 64 and 128 in each dimension for the 3D and 2D models respectively, we are not limited to this size domain at inference time).

Note that we do not claim the network architecture of Figure 3 is in any way "optimal" for the task. Its simplicity and relatively small number of trainable parameters allows for fast real-time inference. Alternatively, we have experimented with residual connections [He et al. 2015] and significantly deeper network architectures, which do improve performance but at the cost of additional run-time latency.

When simulating smoke, we do not explicitly satisfy the Dirichlet boundary conditions in our offline simulation (for smoke we surround our simulation domain by an empty air region). Typically "ghost cells" are used to incorporate these boundary conditions into the linear system solution. In our system, we *do not* include ghost cells (or even mark voxels as boundary cells) and instead let the ConvNet learn to apply these boundary conditions by learning edge effects in the functional mapping of the input tensor to pressure output. This means that we must learn a new model if the external boundary changes. This is certainly a limitation of our approach, however it is one that we feel is not severe since most applications use a constant domain boundary.

Unfortunately, with limited learning capacity our ConvNet architecture is *least accurate* on the boundary voxels. As such, additional velocity divergence on the boundary can sometimes cause the simulation to become unstable when a large time-step is used. To counter this, we add a post-processing step that blurs the boundary velocities (i.e. calculates the neighborhood average and sets the boundary voxels to it) increasing numerical dissipation on the border. In practice this completely removes any numerical stability issues and is not visually apparent.

We find that while training our model on $u^\star$ and u pairs from a *single* time-step results in sufficiently low $\nabla \cdot u$, however, divergence can accumulate over long simulations leading to instabilities. We can further improve our ConvNet prediction by adding an additional *semi-supervised* term to our objective function that minimizes long term divergence. We do so by stepping forward the simulation state from p_0 and u_0 of the training set sample, to a small number of time-steps n in the future (to $\hat{p}_n$ and $\hat{u}_n$).[2] Then we calculate $\nabla \cdot \hat{u}_n$ and minimize $\lambda_{div} \|\nabla \cdot \hat{u}_n\|^2$ for this future frame by accumulating its model parameter partial derivatives to the existing calculated gradient of Equation 4. This process is depicted in Figure 5.

[2] when training our model we step forward either $n = 4$ steps, with probability 0.8, or $n = 25$ with probability 0.2.

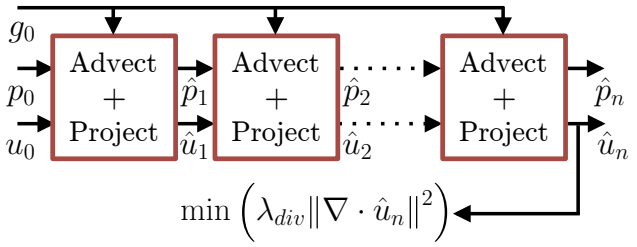


Fig. 5: "Semi-supervised" Velocity Divergence Minimization

Note that we could alternatively perform "positive sample mining" by sending this predicted future frame to our offline solver and using the solver result as a new training sample, however we find that this novel "semi-supervised" scheme is extremely effective and significantly more efficient (since the offline solver is slow, it would become the bottleneck of our model training).

Another alternative would be to consider a fully recurrent neural network (RNN) with non-trainable modules given by the advection step. While this might be a promising idea, it is non-trivial to back-propagate through the semi-Lagrangian procedure. On the other hand, such a model would require much higher memory resources during training and would lead to a slower training procedure. Finally, when moving the simulation state forward we pick a random time-step according to: $\Delta t = 1/30 * (1 + |N(0,1)|)$ (where $N(0,1)$ is a random sample from a Gaussian distribution with mean 0 and variance 1). This helps promote model invariance to simulation time-step size, within some reasonable bound.

4. DATASET CREATION AND MODEL TRAINING

To train the ConvNet model of Section 3, we need example solutions to the sparse linear system described in Section 2. In lieu of real-world fluid data, which is difficult (if not impossible) to collect accurately and at scale, we use synthetic training data generated using an offline 3D solver, mantaflow [Pfaff and Thuerey] - an open-source research library for solving incompressible fluid flow.

The difficulty in generating synthetic data for this task is defining a pseudo-random procedure to create training data which, with a reasonable amount of training samples, sufficiently covers the space of input velocity fields. This is clearly a difficult task as this spans all possible $\mathbb{R}^3$ vector fields and geometry. However, despite this seemingly intractable input space, empirically we have found that a plausible model can be learned by defining a simple procedure to generate inputs 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 (uniformly sampling the wavelet spatial scale and amplitude) and we generate a random seed, which we then use to generate the velocity field.

Next, we generate a geometry (or occupancy) field 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 and Ramani 2006]; 50 models are used only when generating training samples and 50 models are used when generating test samples. Figure 6 shows a selection of these models. Each model is voxelized using the binvox library [Min 2016]. For generating 2D simulation data, we simply take a 2D slice of the 3D voxel grid.

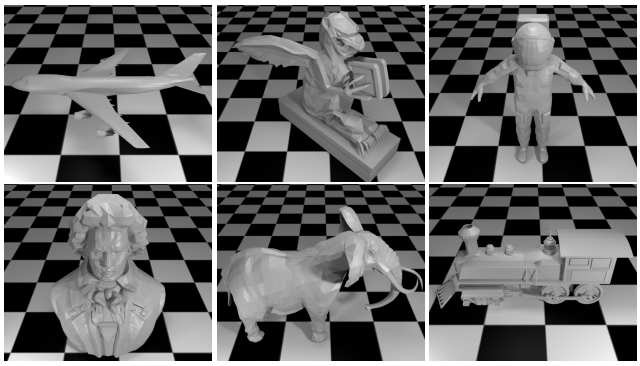


Fig. 6: 3D Model collection (subset) used for training and testing

Finally, we simulate small divergent input perturbations by modeling inflow moving across the velocity field using a collection of emitter particles. We do this by generating a random set of emitters (with random time duration, position, velocity and size) and adding the output of these emitters to the velocity field throughout the simulation. These emitters are procedurally animated to smoothly move around the simulation domain between time-steps. This mimics the sort of divergent inflow one might find due to either user-input or objects moving within the scene.

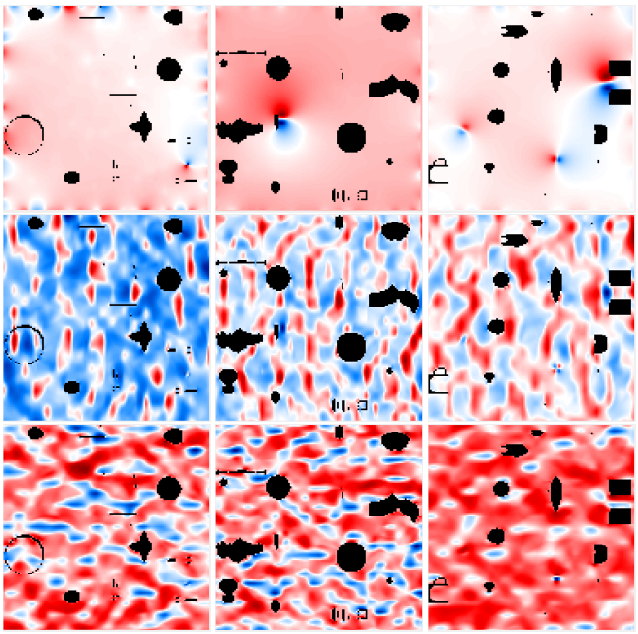
With the above initial conditions defined, we use manta to calculate $u_t^\star$ by advecting the velocity field and adding forces, then we use the built-in PCG routine provided by manta to produce both the output pressure and divergence free velocity, p_t and u_t respectively. This triplet, $(u_t^\star, p_t, u_t)$ is a single training (or test) example. A selection of these examples for our 2D data is shown in Figure 7.

Generating a single sample takes approximately 0.6 and 8.1 seconds per frame, for the 2D and 3D datasets respectively. Using the above procedure, we generate a training set of 80 "simulation runs" (from 80 initial random conditions) and a test set of an additional 80 simulations. Each "simulation run" contains 256 frames, with time-step $1/30$ seconds between each frame. Since adjacent frames have a large amount of temporal coherence, we can decimate the frame set without loss of ConvNet accuracy, and so we save the synthetic frame to disk every 4 frames. This provides a total of 5120 training set and 5120 test set examples, each with a disjoint set of geometry and with different random initial conditions. We will make this dataset public (as well as the code for generating it) for future research use. All materials are located at `http://cims.nyu.edu/~schlacht/CNNFluids.htm`.

5. RESULTS AND ANALYSIS

We implemented the model of Section 3 in the open-source Torch7 [Collobert et al. 2011] scientific computing framework. For fast inference, we use the convolution kernels from the NVIDIA cuDNN V7.5 library [Chetlur et al. 2014], and custom GPU CUDA modules for fixed function advection and FD blocks.

Single-frame inference (i.e. a single forward propagation through the pressure projection model) takes 0.89ms and 21.1ms per sample for the 2D and 3D models respectively, each with 128



(a) Sample A (b) Sample B (c) Sample C

Fig. 7: 2D Training Samples. *Top row*: pressure. *Middle row*: x-component of velocity. *Bottom row*: y-component of velocity. *Black region*: solid geometry.

cells per dimension.[3] This makes our model extremely competitive with state-of-the-art GPU implementations of Jacobi and Gauss-Seidel solvers for pressure.

We believe that run-time performance can be improved significantly using model compression techniques [Lin et al. 2015; Han et al. 2015; Denton et al. 2014], separable convolution kernels, hyper-parameter optimization (for example optimizing our objective function with an additional term for floating point operations) and custom hardware [Movidius ; Google Inc.]. While we will leave these experiments to future work, we believe that our early "poorly-optimized" model is a strong indicator of our network's potential.

Training takes approximate 20 and 48 hours for the 2D and 3D models respectively. We use the ADAM [Kingma and Ba 2014] optimization scheme for fast convergence and a constant learning rate throughout training.

An extremely important implementation detail to ensure proper convergence is that we clamp the L2 magnitude of the gradient to a constant value during training (we use an aggressive value of 1). The distribution of the gradient norm (L2) at the start of training has - unsurprisingly - high mean and low variance. Towards the end of training the mean gets close to zero, as training moves towards the local minima. However the variance actually increases significantly, and this can cause issues for the ADAM optimizer. We attribute high gradient variance to our semi-supervised long-term divergence term; when the ConvNet predicts frames with high divergence (as it has yet to finish training), for a small subset of training frames the divergence will grows significantly over the n steps for our semi-supervised future divergence calculation. This can cause the gradient contributions for these frames to result in optimization instability. Clamping the gradient magnitude - thereby removing large gradient outliers - completely solves this issue and actually improves performance on the test-set.

3D simulation examples of our system are shown in Figure 1. The simulation data was created using our real-time system, while we use the open-source ray-tracer Blender [Blender Foundation] to render the visualization offline. Video examples of the system can be found in the supplemental materials or at `http://cims.nyu.edu/~schlacht/CNNFluids.htm`.

We simulated a 3D smoke plume in both mantaflow and in our system.[4] Single frame results of both experiments can be seen in Figure 8 and the videos can be found in the supplemental materials. Since our buoyancy, vorticity confinement and advection implementations differ significantly to those in mantaflow, quantitative comparison is not appropriate. However, qualitatively we do find that our system is more prone to early smoke dissipation than the full PCG solver and our system lacks some of the high-frequency smoke detail evident in mantaflow (the mantaflow smoke plume has both higher density and structural thickness). We attribute the increased numerical dissipation of our system to lack of model capacity and the network's subsequent (undesirable) tendency to increase dissipation over time in order to minimize long-term divergence. The trade-off of course is that the mantaflow PCG solver takes approximately 43 seconds per frame, whereas our system solver takes approximately 21ms per frame.

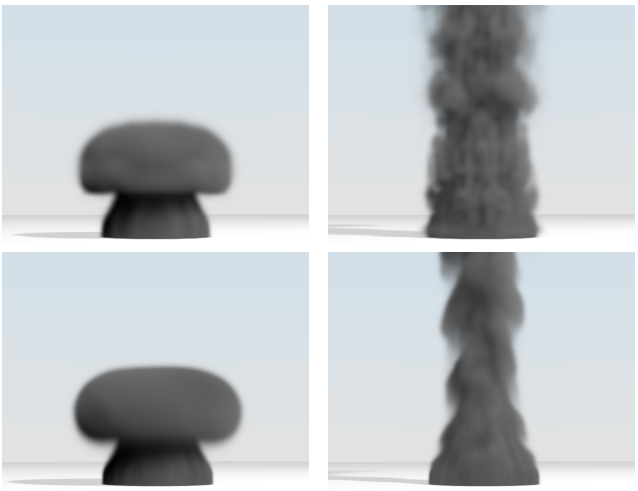


Fig. 8: Plume simulation (without vorticity confinement). *Top row*: mantaflow. *Bottom row*: this work.

For the experiments presented in this work, we use $\lambda = 1$ for all λ values in our objective function. However, adjusting λ_{div} to 0.2

[3] We use an NVIDIA Titan X GPU with 12GB of ram and an Intel Xeon E5-2690 CPU. To reiterate: the quoted run-time is **only** the pressure projection step, and does not include the advection or FD blocks. Our current implementation for the advection block is not optimized and runs on the CPU only, and so is currently the bottleneck of our system by more than an order of magnitude for the 3D model. However, this block is easily implemented on the GPU [Keenan Crane 2007] and can be run at interactive frame-rates.

[4] Note: we did not use vorticity confinement for these simulations as it obfuscates the difference between the two approaches.

(or lower) can help reduce numerical dissipation at the cost of an increase in long-term divergence (and a subsequent softer constraint on conservation of mass). We believe this flexibility is a benefit of our machine learning based approach, where the optimization hyper-parameters can be chosen as per the user application requirements.

Of primary importance to real-time fluid simulation is measuring velocity field divergence over time to ensure stability and correctness. As such, we evaluate our model by seeding a simulation using the velocity field from manta, and by running the simulation for 256 frames; Figure 9 shows $\|\nabla \cdot \hat{u}\|$ for a random sub-set of the Manta Test-set initial conditions. For each frame we record the mean L2

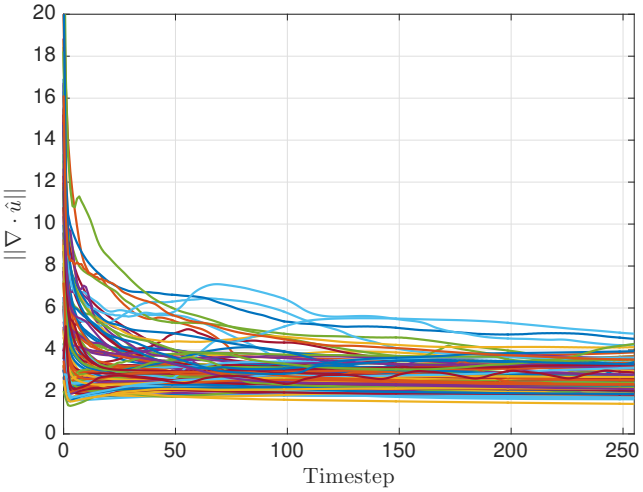


Fig. 9: $\|\nabla \cdot u_t\|$ over time for all test-set initial conditions.

norm of divergence ($\mathbf{E}(\|\nabla \cdot \hat{u}_i\|)$) at each frame index by averaging over the test (and training) set frames (i.e. we average vertical slices of the curves in Figure 9). The result of this experiment is shown in Figure 10. We also indicate the standard-deviation of divergence norm by error bar annotations. As shown, our network is able to generalize to unseen geometry as the test and training performance are equivalent.

Figure 11 shows that training on single frame examples is insufficient in ensuring that divergence stays low over time. Without our semi-supervised "future-divergence" loss term, the mean divergence drops quickly from the initial condition, but errors in the ConvNet projection cause divergence to grow over time. Conversely, when we include our semi-supervised term (minimizing velocity divergence at frame index 5), the divergence drops quickly and stays low through the simulation. In addition, the network is clearly able to generalize to time-steps far beyond the coverage of our semi-supervised term.

One potential limitation with our approach is high memory consumption. For a single frame inference, our 3D model uses 667kB of GPU RAM to store weights, biases and other network parameters (keeping all parameters in GPU RAM to reduce PCI transfers), and requires a peak memory of 42MB to store the input and output tensors for the first 3D convolution layer.[5]

[5]The first convolution layer requires the largest memory pool. All subsequent layers can re-use this memory pool, ping-ponging between the input and output tensor

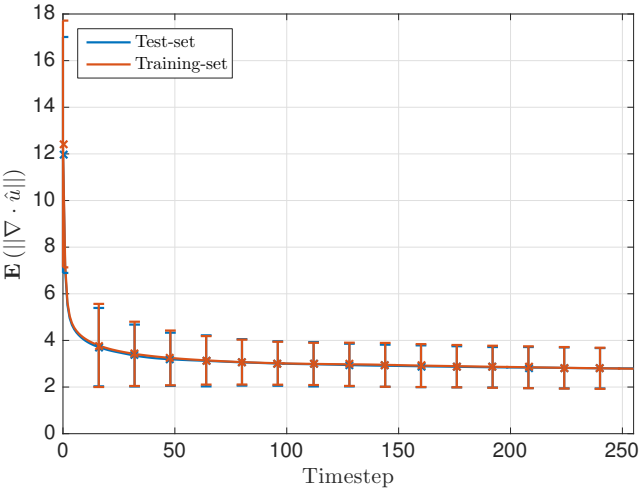


Fig. 10: $\mathbf{E}(\|\nabla \cdot \hat{u}_i\|)$ versus time-step (frame count) for each frame sample in our dataset (error bars represent the standard deviation).

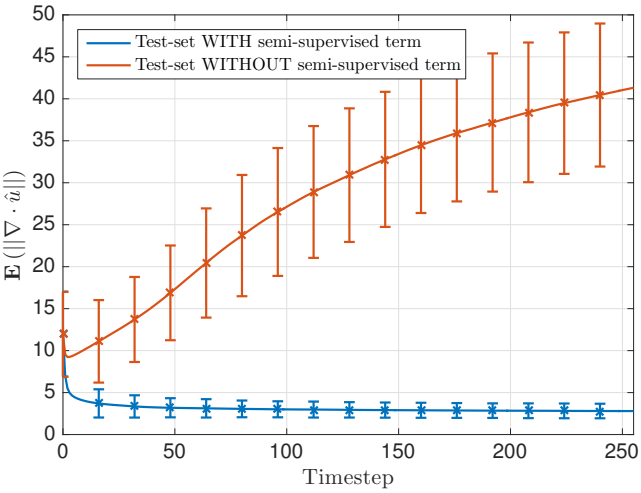


Fig. 11: $\mathbf{E}(\|\nabla \cdot \hat{u}_i\|)$ versus time-step (frame count) with and without including long-term divergence minimization.

A distinct advantage of our machine learning formulation is the ability to trade-off precision for computational load in a controlled and principled way. For instance, our network hyper-parameters can be adjusted to reduce total FLOPS (primarily by reducing the size and number of convolution layers, or increasing pooling) while trading off simulation precision. Inference time for the smaller model takes 21ms sec and results in a test-set criterion error of 0.074, while the larger model takes 42ms sec and results in a test-set criterion error of 0.018. In practice, we find that reducing p and u precision results in loss of high-frequency detail and additional numerical dissipation, but can require less compute.

6. 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 Convolutional Network architecture for approximate inference of the sparse linear system used to enforce Navier-

Stokes incompressibility condition - the so-called "pressure projection" step - of standard Eulerian fluid solvers for computer graphics. We propose a training-loss framework that enables the end user to tailor the ConvNet performance to their particular application. Additionally, the ConvNet architecture is compatible with existing Eulerian-based solvers, facilitating drop-in replacement for existing solutions. While the proposed approach cannot guarantee finding an exact solution to the pressure projection step, it can empirically produce very stable non-divergent velocity fields producing visually plausible results. This approach is a solid alternative to standard solvers in situations where the running time of the algorithm is more important than the exactness of a simulation.

We also present a novel semi-supervised technique to dramatically improve long-term stability by softly enforcing physical constraints on unlabeled training data. We show empirically that our technique is stable over a large test-set corpus of pseudo-random initial conditions and we present fully realized 3D smoke simulations using our system. The system is capable of running on real-time simulations in challenging settings for both 2D and 3D examples with run-times competitive to the state-of-the-art. Furthermore, these results are very promising as the efficient implementation of ConvNets is a central subject of study in the deep learning community. Permanent improvements in terms of running times and memory consumption are reported in the literature. Code, data and videos will be made available at `http://cims.nyu.edu/~schlacht/CNNFluids.htm`.

ACKNOWLEDGMENTS

We would like to thank the NVIDIA Corporation for their donation of a Tesla K40 used during this research, as well as Károly Zsolnai-Fehér for his invaluable advice and insights.

REFERENCES

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

BLENDER FOUNDATION. Bender open-source renderer v2.77a. `http://www.blender.org`.

BRIDSON, R. 2008. *Fluid Simulation for Computer Graphics*. Ak Peters Series. Taylor & Francis.

CHENTANEZ, N. AND MÜLLER, M. 2010. Real-time simulation of large bodies of water with small scale details. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '10. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 197–206.

CHETLUR, S., WOOLLEY, C., VANDERMERSCH, P., COHEN, J., TRAN, J., CATANZARO, B., AND SHELHAMER, E. 2014. cudnn: Efficient primitives for deep learning. *CoRR abs/1410.0759*.

COLLOBERT, R., KAVUKCUOGLU, K., AND FARABET, C. 2011. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

DE WITT, T., LESSIG, C., AND FIUME, E. 2012. Fluid simulation using laplacian eigenfunctions. *ACM Trans. Graph. 31,* 1 (Feb.), 10:1–10:11.

DENTON, E., ZAREMBA, W., BRUNA, J., LECUN, Y., AND FERGUS, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. *CoRR abs/1404.0736*.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. ACM, New York, NY, USA, 15–22.

FOSTER, N. AND METAXAS, D. 1996. Realistic animation of liquids. *Graph. Models Image Process. 58,* 5 (Sept.), 471–483.

FOSTER, N. AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 181–188.

GINGOLD, R. A. AND MONAGHAN, J. J. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society 181,* 3, 375–389.

GIRYES, R., ELDAR, Y. C., BRONSTEIN, A. M., AND SAPIRO, G. 2016. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. *arXiv preprint arXiv:1605.09232*.

GOOGLE INC. Tensor processing unit. `http://cloudplatform.googleblog.com/2016/05/`.

GREGOR, K. AND LECUN, Y. 2010. Learning fast approximations of sparse coding. In *International Conference on Machine Learning (ICML)*. 399–406.

HAN, S., MAO, H., AND DALLY, W. J. 2015. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR abs/1510.00149*.

HE, K., ZHANG, X., REN, S., AND SUN, J. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

HORVATH, C. J. AND SOLENTHALER, B. 2013. Mass preserving multi-scale sph. *Pixar Technical Memo 13-04, Pixar Animation Studios*.

KEENAN CRANE, IGNACIO LLAMAS, S. T. 2007. Real-time simulation and rendering of 3d fluids. *GPU Gems 3*.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27,* 3 (Aug.), 50:1–50:6.

KINGMA, D. AND BA, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LADICKÝ, L., JEONG, S., SOLENTHALER, B., POLLEFEYS, M., AND GROSS, M. 2015. Data-driven fluid simulations using regression forests. *ACM Trans. Graph. 34,* 6 (Oct.), 199:1–199:9.

LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86,* 11, 2278–2324.

LIN, Z., COURBARIAUX, M., MEMISEVIC, R., AND BENGIO, Y. 2015. Neural networks with few multiplications. *CoRR abs/1510.03009*.

MACKLIN, M. AND MÜLLER, M. 2013. Position based fluids. *ACM Transactions on Graphics (TOG) 32,* 4, 104.

MIN, P. 2016. Binvox utility v1.22.

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

OYMAK, S., RECHT, B., AND SOLTANOLKOTABI, M. 2015. Sharp time–data tradeoffs for linear inverse problems. *arXiv preprint arXiv:1507.04793*.

PFAFF, T. AND THUEREY, N. Mantaflow fluid simulator. `http://mantaflow.com/`.

PU, J. AND RAMANI, K. 2006. On visual similarity based 2d drawing retrieval. *Comput. Aided Des. 38,* 3 (Mar.), 249–259.

RAVEENDRAN, K., WOJTAN, C., THUEREY, N., AND TURK, G. 2014. Blending liquids. *ACM Trans. Graph. 33,* 4 (July), 137:1–137:10.

SPRECHMANN, P., BRONSTEIN, A. M., AND SAPIRO, G. 2015. Learning efficient sparse and low rank models. *IEEE transactions on pattern analysis and machine intelligence 37,* 9, 1821–1833.

STAM, J. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 121–128.

STANTON, M., HUMBERSTON, B., KASE, B., O'BRIEN, J. F., FATAHALIAN, K., AND TREUILLE, A. 2014. Self-refining games using player analytics. *ACM Trans. Graph. 33,* 4 (July), 73:1–73:9.

STEINHOFF, J. AND UNDERHILL, D. 1994. 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.

SZEGEDY, C., IOFFE, S., AND VANHOUCKE, V. 2016. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR 2016 Workshop*.

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. *ACM Trans. Graph. 25,* 3 (July), 826–834.