ACCELERATING EULERIAN FLUID SIMULATION WITH CONVOLUTIONAL NETWORKS

Jonathan Tompson
Google Brain

Kristofer Schlachter & Pablo Sprechmann & Ken Perlin
New York University

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 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

The dynamics of a large number of physical phenomenon are governed by the incompressible Navier-Stokes equations. In this work, we follow the Eulerian viewpoint for simulating these equations, which approximates quantities on a regular grid (Foster & Metaxas, 1996). Euler methods are able to produce precise results simulating fluids like water or smoke, at the cost of a high computational load.

The most demanding portion of this method is solving the discrete Poisson equation, which enforces the incompressibility condition. Exact solutions can be found using the Preconditioned Conjugate Gradient (PCG) algorithm or via stationary iterative methods such as the Jacobi or Gauss-Seidel methods. A number of numerical methods have been proposed to mitigate this limitation for offline applications, notably multi-grid approximations (McAdams et al., 2010). However, in real-time Jacobi iterations are truncated before reaching convergence, rendering these methods inexact and the obtained velocity fields divergent. A natural approach is to tackle the problem in a data-driven manner, adapting the solver to the specifics of the data of interest. For instance, by operating on a representation of the simulation space of significantly lower dimensionality (Treuille et al., 2006; De Witt et al., 2012). More recently, approaches have been proposed which train black-box machine learning systems to predict the output produced by an exact solver, e.g. using random regression forests (Ladický et al., 2015) or neural networks (Yang et al., 2016) for Lagrangian and Eulerian methods respectively. A major limitation of these 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. 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 3).

The contributions of this work are as follows: *(i)* the learning task can be phrased as a completely unsupervised learning problem; since obtaining ground-truth data is no longer necessary, we can incorporate loss information from a composition of 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

Algorithm 1: 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 $u_t = u_{t-1} - \frac{1}{\rho} \nabla p_t$

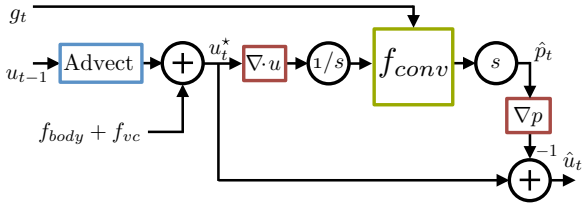

Figure 1: Model architecture.

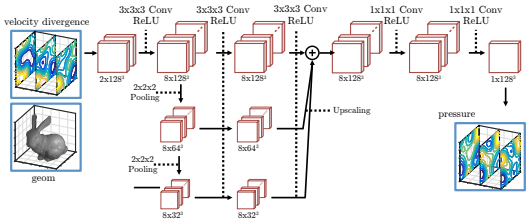

Figure 2: ConvNet Pressure Solve

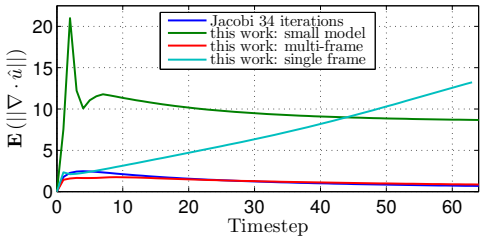

Figure 3: Test-set $\mathbf{E}(\|\nabla \cdot \hat{u}_i\|)$ versus time-step

quantitative improvements. *(iii)* the proposed simulator is stable and permits real-time simulation, showing good generalization properties to unseen settings.

An alternative to our approach is learning an end-to-end mapping that predicts the velocity field directly at each time-step. We argue that our hybrid approach restricts the learning task to a stable projection step, relieving the need for modeling the well understood advection and external body forces and enabling the use of enhancing tools such as vorticity confinement. In addition to the above the technical contributions, we contribute a dataset that can be of interest for people working on real-time simulations and as a benchmarking framework for end-to-end approaches.

2 MODEL

When a fluid has zero viscosity and is incompressible it can be modeled by the Euler equations:

$$\frac{\partial u}{\partial t} = -u \cdot \nabla u - \frac{1}{\rho} \nabla p + f \qquad \text{subject to,} \qquad \nabla \cdot u = 0, \tag{1}$$

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. We numerically calculate all partial derivatives using finite difference (FD) methods on a MAC grid Harlow & Welch (1965). Equations 1 can be solved via the standard operator splitting method described in Algorithm 1. At a high level, step 1 ignores the pressure term ($-\nabla p$ of (1)) to create an advected velocity field, $u_t^\star$, which includes unwanted divergence (see see (Selle et al., 2008) for details), and then step 7 solves for pressure, p, to satisfy the constraint in (1). This produces a divergence free velocity field, u_t. In addition, we use vorticity confinement (Steinhoff & Underhill, 1994) to counteract unwanted numerical dissipation. Step 8 is computationally demanding as it involves solving the Poisson equation: $\nabla^2 p_t = \frac{1}{\Delta t} \nabla \cdot u_t^\star$. Rewriting this 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). After solving for pressure, the divergence free velocity is calculated by subtracting the FD gradient of pressure, $u_t = u_t^\star - \frac{1}{\rho} \nabla p$.

We propose a learned approximate inference mechanism to find fast and efficient solutions to the linear system $Ap_t = b$. The key observation is that, while there is no closed form solution, 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 a deep neural network. A block diagram of our high-level model architecture is depicted in Figure 1, and shows the computational blocks required to calculate $\hat{u}_t$ for a single time-step. The *advect* block is a fixed function unit solving step 1 of Algorithm 1. Then we add the body and vorticity confinement forces and obtain 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 only block with trainable parameters is the ConvNet model.

We define an objective function and formulate the inference solution as an unsupervised machine learning task where the loss function is given by,

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

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. Note that the bottle-neck architecture in the ConvNet avoids obtaining trivial solutions.

The internal structure of the ConvNet architecture is shown in Figure 2. It consists of 5 stages of convolution (spatial or volumetric) and Rectifying Linear layers (ReLU). The convolutional operator itself mimics the local sparsity structure of our linear system. However a single resolution network would have limited context, which limits the network‘s ability to model long-range external forces (such as gravity or buoyancy). As such, we add multi-resolution features to enable modeling long range physical phenomenon, processing each resolution in parallel then upsampling the resultant low resolution features before accumulating them.

3 Results and Analysis

The model of Section 2 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). All tests are performed on a tailored dataset, see Appendix A.

To implement the model of Yang et al. (2016) for comparison, we rephrase their fully-connected architecture as an equivalent, but significantly faster, sliding window model (on a 96x128x96 grid, Yang et al. report 515ms/frame, while our implementation takes 9.4ms/frame). Unfortunately, their loss function fails to learn an accurate projection on our dataset. This is because our divergent velocity frames include gravity and buoyancy terms, which result in a high amplitude, low frequency gradient in the ground-truth pressure. The small 3x3x3 context of the Yang et al. model cannot infer such low frequency output, which dominates the loss function and results in over-training. 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 as our “small model” (i.e. a single resolution with only 3x3x3 context, trained using the loss function, top level architectural improvements and training procedure of this work).

For fair quantitative comparison of output residual, we choose the number of Jacobi iterations (34) to approximately match the FPROP time of our network. PCG is orders of magnitude slower at all resolutions. The “small-model” provides a significant speedup over other methods. The runtime for the PCG, Jacobi, this work, and the “small model” are 2521ms, 47.6ms, 39.9ms and 16.9ms respectively. See Appendix B for details, including timing as a function of resolution in Figure 5.

We simulated a 3D smoke plume using our system and baseline methods (visual results are shown in Appendix C, figures 6 and 7) [1]. Note that this boundary condition is not present in the training set; it is a difficult test of generalization performance. Qualitatively, the PCG and 100-iteration Jacobi solvers and our network produce visually similar results. The “small model”, cannot accurately simulate the large vortex under the plume, and as a result the plume rises too quickly and exhibits density blurring. Similarly the Jacobi method, when truncated early at 34 iterations, introduces implausible high frequency noise and has an elongated shape due to inaccurate modeling of buoyancy. Both ConvNet based methods lose some smoke density inside the arch model due to residual negative divergence at the fluid-geometry boundary. The maximum residual norm was $<$1e-3, 1.235, 1.966, 0.872 for the PCG, Jacobi, small model and this work respectively.

As a test of long-term stability, we record the mean residual norm ($\mathbf{E}\left(\|\nabla \cdot \hat{u}_i\|\right)$) across all samples in our test-set for each frame after the initial condition, shown in Figure 1. Our model outperforms the small model (Yang et al. sizing), and is competitive with Jacobi. We also present the results of our model when a single time-step loss is used; without the multi-frame loss, single time-step accuracy is degraded, and the divergence increases over time as error is accumulated.

[1] Video examples of these experiments can be found in at `http://cims.nyu.edu/~schlacht/CNNFluids.htm`.

REFERENCES

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

Tyler De Witt, Christian Lessig, and Eugene Fiume. 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.

Nick Foster and Dimitri Metaxas. 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.

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

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

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 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.

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 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.

Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 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.

Patrick Min. Binvox utility v1.22. 2016.

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

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

Jiantao Pu and Karthik Ramani. 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.

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

John Steinhoff and David Underhill. 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.

Adrien Treuille, Andrew Lewis, and Zoran Popović. 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.

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

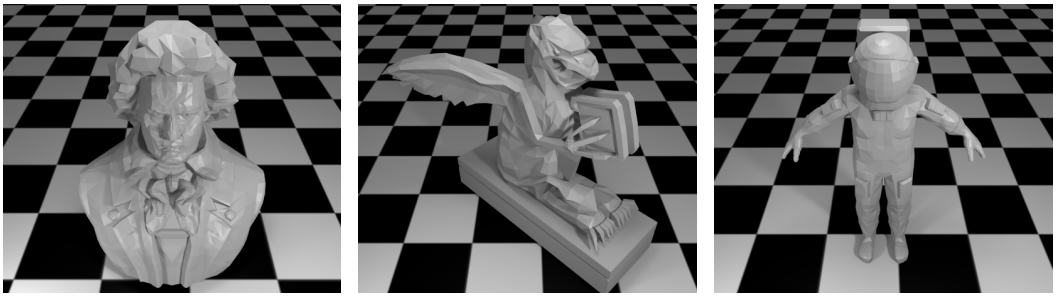

Figure 4: Some of the 3D Model used in our dataset

A Dataset Creation and Model Training

While we do not need ground-truth label information to train the ConvNet model of Section 2, we need a collection of ground-truth pressure solutions to evaluate the precision of our model, and additionally our model does benefit from an efficient sampling of "realistic" initial conditions. That is, the space of all divergent velocity fields is unconstrained, and so our network's generalization performance is improved when using a dataset of natural initial conditions that approximately samples the manifold of real-world fluid simulation states. *To this end, we propose a procedural method to generate a corpus of initial frames for use in training.*

We use synthetic data generated using an offline 3D solver, mantaflow Pfaff & Thuerey - an open-source research library for solving incompressible fluid flow. We then seed this solver with initial condition states generated via a simple 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 (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 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 models are used when generating test samples. Figure 4 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.

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.

With the above initial conditions defined, 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 divergent velocity every 8 frame steps.

Using the above procedure, 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, each 0.8 seconds apart. We use a disjoint set of geometry for the test and training sets to test generalization performance. 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`.

B Details of the Evaluation of the Computational Complexity

Figure 5, shows the computation time of the Jacobi method, the small-model version (with Yang et al. sizing) and this work. This runtime includes the pressure projection steps only: including velocity divergence calculation, the linear system solve, and the velocity update. Note that for

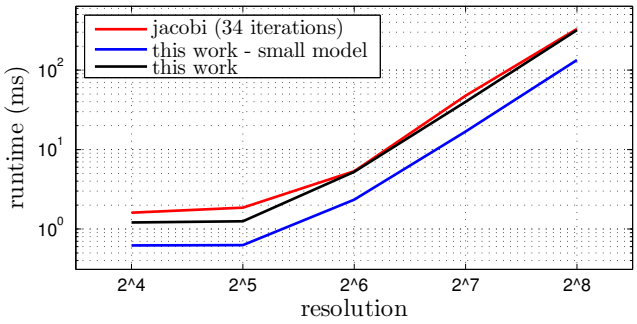

Figure 5: Runtime Vs. grid resolution (PCG omitted for clarity)

fair quantitative comparison of output residual (shown in Section 3 of the paper), we choose the number of Jacobi iterations (34) to approximately match the FPROP time of our network. Since the asymptotic complexity as a function of resolution is the same for Jacobi and our ConvNet, the FPROP times are equivalent. We use an NVIDIA Titan X GPU with 12GB of ram and an Intel Xeon E5-2690 CPU. PCG is orders of magnitude slower at all resolutions and has been left off for clarity. The model of Yang et al. provides a significant speedup over other methods. The runtime for the PCG, Jacobi, this work, and Yang et al. 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.

C QUALITATIVE COMPARISON OF SIMULATIONS

This appendix shows rendered frames for the proposed method as well as baseline alternatives.

Figure 6 shows a rendered frame of our plume simulation (without geometry) for all methods. Note that this boundary condition is not present in the training set and represents an input divergent flow approximately 5 times wider than the largest impulse present during training. It is a difficult test of generalization performance for data-driven methods. Qualitatively, the PCG and Jacobi (with 100 iterations) and our network produce visually similar results. The model of Yang et al., trained using the loss function of this work, cannot accurately simulate the large vortex under the plume, and as a result the plume rises too quickly and exhibits density blurring under the plume itself. Similarly the Jacobi method, when truncated early at 34 iterations, introduces implausible high frequency noise and has an elongated shape due to inaccurate modeling of buoyancy forces.

We also repeat the above simulation with solid cells from the "arch" model held out of our training set. Single frame results for this simulation are shown in Figure 7. Since this scene exhibits lots of turbulent flow, qualitative comparison is less useful. However, the network of Yang et al. 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 model due to negative divergence at the fluid-geometry boundary (specifically at the large flat ceiling), like a result of this wide plume interaction being outside the scope of the training samples.

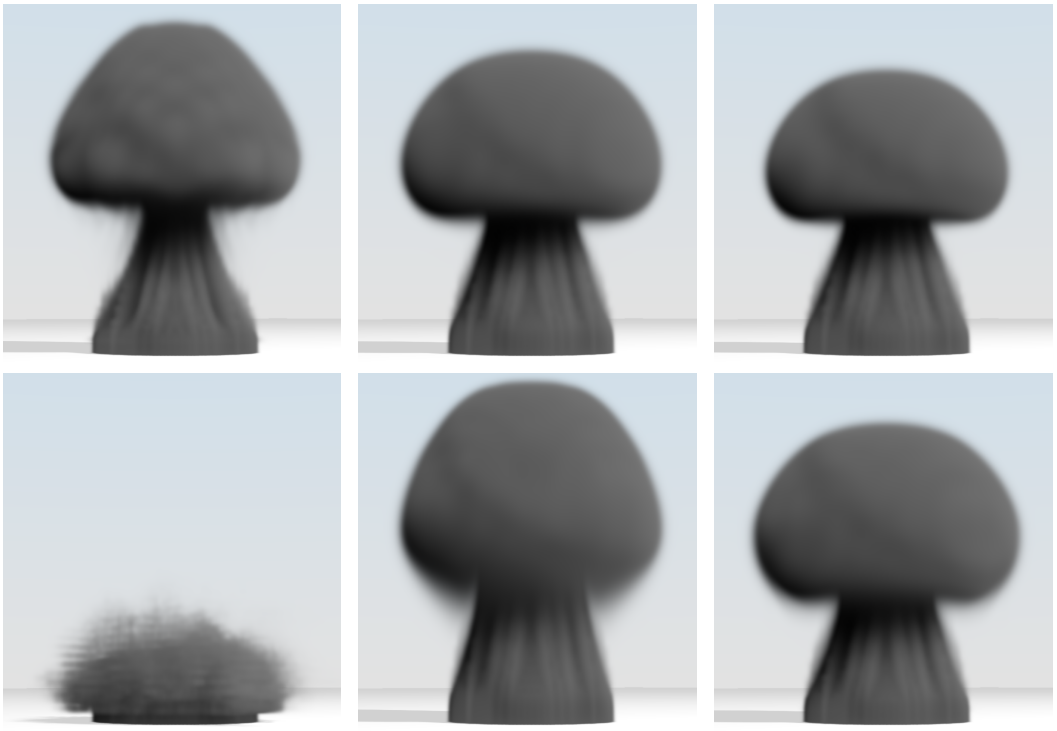

Figure 6: Plume simulation (without vorticity confinement). *Top left*: Jacobi (34 iterations). *Top Middle* Jacobi (100 iterations). *Top Right*: PCG. *Bottom left*: Yang et al. *Bottom middle*: This work - small model. *Bottom Right*: This work.

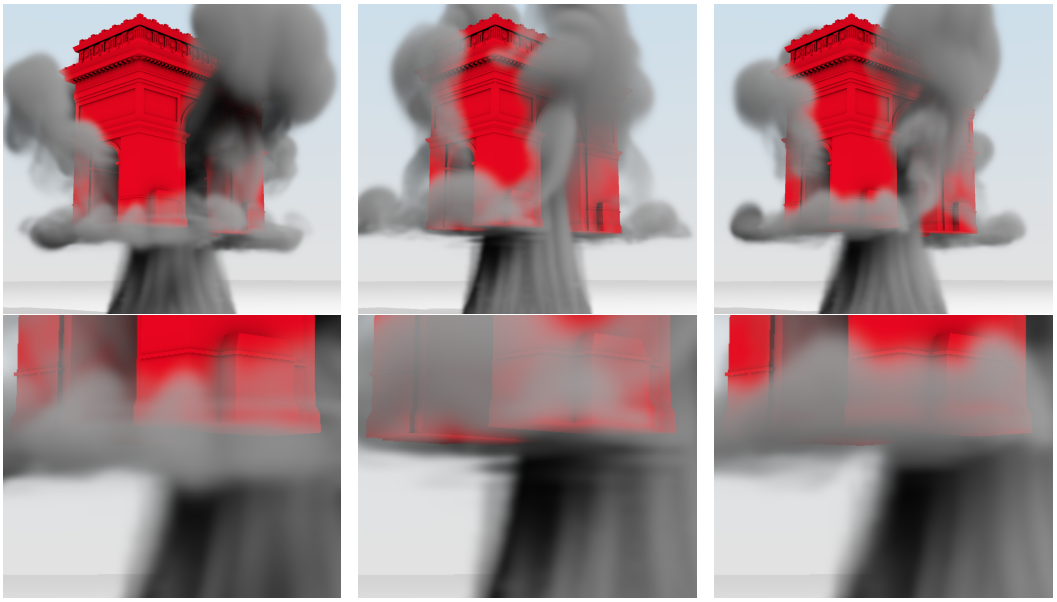

Figure 7: Plume simulation with "Arch" geometry. *Left*: PCG. *Middle* This work - small model. *Right*: This work.